Wydział Elektroniki
Politechnika Wrocławska

ROZPRAWA DOKTORSKA

Metody ochrony przed kryptoanalizą z uszkodzeniami

mgr. inż. Maciej Nikodem

Promotor: dr hab. inż. Janusz Biernat, prof. PWr.

słowa kluczowe:
kryptoanaliza z uszkodzeniami, AES,
schematy podpisów ElGamala i DSA

krótkie streszczenie:
Praca obejmuje zagadnienia związane z ochroną algorytmu szyfrującego AES i schematów podpisów cyfrowych ElGamal i DSA

Wrocław 2008

... the way to the top of the heap in terms of developing original research is to be a fool, because only fools keep trying. (...) God rewards fools.

Martin Hellman

Spis treści

Rozdział 1

Tematyka i cel pracy

Przedmiotem pracy jest analiza ataków z uszkodzeniami (ang. *fault analysis*), stanowiących istotne zagrożenie bezpieczeństwa algorytmów kryptograficznych. Ataki takie znajdują się obecnie w centrum zainteresowania wielu badaczy na całym świecie. Przyczyną tego, są duże możliwości implementacji ataków z uszkodzeniami, trudność ochrony przed nimi i to, że wszystkie współczesne algorytmy kryptograficzne są na nie podatne.

Szczególną uwagę w pracy poświęcono analizie bezpieczeństwa algorytmu AES oraz schematów podpisów cyfrowych ElGamala i DSA. Dla algorytmu AES zaproponowano nowe rozwiązanie ochronne bazujące na korekcji błędów wprowadzanych przez atakującego. Rozwiązanie to może być zastosowane do ochrony innych algorytmów symetrycznych, gwarantując lepszą ochronę przed atakami z uszkodzeniami niż stosowane dotychczas algorytmy detekcji błędów. Proponowane rozwiązanie nie wymaga powtarzania obliczeń i jest odporne również na te ataki, które mogły być zastosowane w przypadku ochrony algorytmów za pomocą detekcji błędów. Rozwiązania ochronne zaproponowano także dla schematów podpisów cyfrowych ElGamala i DSA. Dotychczas brak było takich rozwiązań, mimo powszechnej świadomości, że oba te algorytmy są podatne na ataki z uszkodzeniami.

1.1 Kryptografia i kryptoanaliza

Jednym z najważniejszych i historycznie najstarszych obszarów zastosowań kryptografii jest szyfrowanie czyli zapis informacji w postaci utajnionej zrozumiałej jedynie dla wybranego odbiorcy — stąd zresztą określenie kryptografia, które wywodzi się od greckich słów *kryptós* — 'ukryty' i *grápho* — 'piszę'. Proste techniki kryptograficzne były stosowane już w starożytności przez Egipcjan, Hebrajczyków i Rzymian. Stosowane wówczas metody polegały na przestawieniach i podstawieniach liter tekstu i mogły być stosunkowo łatwo złamane przez wystarczająco zdeterminowanego przeciwnika. Niski poziom zapewnianego

bezpieczeństwa nie stanowił jednak przeszkody, aby algorytmy tego typu były wykorzystywane powszechnie aż do XV-XVI wieku.

Cechą wspólną wszystkich algorytmów kryptograficznych stosowanych do XIX wieku jest ich mała złożoność, znacząco niższa od współczesnego tym algorytmom stanu wiedzy matematycznej. Jedną z wielu przyczyn tej sytuacji było ręczne realizowanie procedur kryptograficznych oraz niewielka liczba osób, zajmujących się kryptografią. Drugim powodem była względnie duża złożoność znanych wówczas technik łamania algorytmów. W czasach gdy nie znano maszyn liczących prosta analiza częstości występowania liter w szyfrogramie była zadaniem bardzo czasochłonnym. Sytuacja ta spowodowała brak zainteresowania poszukiwaniem nowych rozwiązań zabezpieczających a tym samym zatrzymała rozwój kryptografii.

Jedną z pierwszych i powszechnie stosowanych technik szyfrowania były monoalfabetowe szyfry podstawieniowe. W szyfrach tych każda litera tekstu jawnego zastępowana jest literą innego alfabetu (np. alfabetu z przesuniętymi literami o ustaloną liczbę pozycji). Najbardziej znanym szyfrem tego rodzaju jest szyfr Cezara, w którym każda litera tekstu jawnego jest zastępowana literą alfabetu znajdującą się o trzy pozycje dalej. Szyfry podstawieniowe zostały zaproponowane na początku naszej ery, jednak pierwsze prace dotyczące ich kryptoanalizy pojawiły się dopiero w IX wieku. Prace te, autorstwa arabskiego matematyka al-Kindi, wykorzystywały badanie częstości występowania liter w kryptogramie i języku, z którego pochodził tekst jawny. Na tej podstawie atakujący mógł odgadnąć wykorzystane podstawienia i odczytać kryptogram. Mimo to, w Europie szyfry podstawieniowe były powszechnie stosowane i łamane przez następne 7 stuleci, decydując o losach bitew, wojen a nawet koronowanych głów. Znaczący postęp w dziedzinie szyfrowania nastąpił dopiero w połowie XVI wieku, kiedy Blaise de Vigenére zaproponował szyfr wieloalfabetowy, czyli taki, w którym do wykonania podstawień używa się wielu alfabetów. Rozwiązanie to jest odporne na analizę częstości występowania liter szyfrogramu i pozostało bezpieczną metodą szyfrowania przez następne trzy stulecia. W XIX wieku, Charles Babbage, a kilka lat później Friedrich Kasiski, opracowali metodę analizy szyfrów wieloalfabetowych. Metoda bazująca na poszukiwaniu identycznych ciągów znaków w szyfrogramie pozwalała ustalić długość używanego klucza i podzielić całą wiadomość na mniejsze bloki będące zwykłymi szyfrogramami podstawieniowymi, monoalfabetycznymi [75].

Gwałtowny rozwój kryptografii i kryptoanalizy nastąpił dopiero w pierwszej połowie XX wieku pozostając jednak nadal dostępny tylko dla wąskiej grupy matematyków pracujących dla wojska i agencji rządowych. Rozwój ten był rezultatem licznych konfliktów zbrojnych, wzrostu znaczenia informacji, oraz rozwoju technologii maszyn liczących. Stworzenie pierwszych maszyn szyfrujących, a później komputerów, umożliwiło powierzenie zadań szyfrowania urządzeniom. Umożliwiło to wykorzystanie w algorytmach kryptograficznych bardziej złożonych operacji oraz pozwoliło na przyspieszenie obliczeń. Postęp w technice

umożliwił również zautomatyzowanie i usprawnienie procedur kryptoanalizy. Dzięki postępowi rozbudowywany przez wiele lat aparat matematyczny mógł zostać wreszcie wykorzystany w praktyce. Wszystkie te elementy przyczyniły się do bardzo szybkiego rozwoju kryptografii i kryptoanalizy. Dobrym przykładem przyspieszenia obliczeń uzyskanym za pomocą maszyn liczących może być łamanie szyfru Enigmy. Pierwsza kryptoanaliza tego algorytmu, wykonana przez Mariana Rajewskiego, Jerzego Różyckiego i Henryka Zygalskiego w 1933 roku, wymagała ponad rocznych przygotowań. W tym czasie przeanalizowano ręcznie wszystkie z 105 456 ustawień wirników i spisano wynikające z nich łańcuchy szyfrogramów tych samych liter. Ta sama operacja w 1940 roku, przeprowadzona z użyciem maszyn liczących (tzw. bomb), wymagała już zaledwie tygodnia, a w 1943, z użyciem komputera Colossus, kilka godzin [75].

Rozwój kryptografii po drugiej wojnie światowej był nadal ograniczony do wąskiej grupy organizacji i firm współpracujących z armią i administracją rządową. Sytuacja zaczęła zmieniać się dopiero w latach 60-tych, kiedy firma IBM upowszechniła symetryczne algorytmy Lucifer i DES (ang. *Digital Encryption Standard*). Kluczowym momentem dla rozwoju kryptografii okazały się jednak prace Whitfielda Diffiego, Martina Hellmana, Ronalda Rivesta, Leonarda Adelmana i Adi Shamira, które upowszechniły tą dziedzinę i dały początek kryptografii asymetrycznej [29, 45, 73].

Analizując rozwój kryptografii i kryptoanalizy od czasów starożytnych do XX wieku, łatwo zauważyć, że postęp w tych dziedzinach zawdzięczamy matematykom. Opracowali oni podstawy teoretyczne algorytmów kryptograficznych, sposoby ich opisu, mechanizmy pozwalające na analizę i ocenę ich bezpieczeństwa oraz zasady implementacji w układach cyfrowych. Prace matematyków pozostałyby jednak w sferze teoretycznej, gdyby nie postęp w skonstruowaniu maszyn liczących i komputerów. Bez tych urządzeń masowe zastosowanie kryptografii byłoby zbyt czasochłonne i niemożliwe do wykonania w praktyce. Postęp techniczny, który dokonał się za sprawą takich osób jak Konrad Zuse, Alan Touring czy John von Neuman, spowodował, że technologie kryptograficzne mogły zostać rozbudowane, ulepszone i powszechnie wykorzystane. W rezultacie matematycy zajmujący się kryptografią i kryptoanalizą uzyskali możliwość praktycznego wykorzystania znanego im aparatu matematycznego. Możliwym stało się stworzenie nowych algorytmów kryptograficznych, schematów podpisów cyfrowych i identyfikacji, jak również rozwijanie skutecznych metod kryptoanalizy.

Ogromne możliwości oferowane przez szybko ulepszane komputery spowodowały, że w poszukiwaniu coraz lepszych algorytmów kryptograficznych nie wzięto pod uwagę zagrożeń związanych z ich implementacją. Jest to o tyle dziwne, że w historii kryptoanalizy znane są liczne przypadki, w których niepoprawne wykorzystanie kryptografii umożliwiło złamanie algorytmów. Jednym z takich przykładów jest złamanie szyfru Lorenza stosowanego przez Wermacht pod koniec II wojny światowej. Kryptoanaliza tego algorytmu była możliwa z uwagi na błąd operatora, który tą samą wiadomość zaszyfrował dwukrotnie z użyciem tego

samego klucza, zmieniając jednak nieznacznie jej treść — kilkadziesiąt lat później takie samo działanie atakującego będzie powszechnie określane mianem kryptoanalizy różnicowej.

Rozwój maszyn liczących i komputerów pozwolił na wykorzystanie rozbudowanego aparatu matematycznego, umożliwiając tworzenie i implementowanie złożonych algorytmów kryptograficznych. Przyspieszenie obliczeń pozwoliło skrócić czas potrzebny na łamanie algorytmów kryptograficznych, a tym samym wymusiło poszukiwanie nowych rozwiązań. Czynniki te stymulowały rozwój kryptografii i kryptoanalizy, powodując szybki postęp w obu dziedzinach. Szybki rozwój oraz powszechne przeświadczenie o bezbłędnym wykonywaniu przez układy cyfrowe zaprojektowanych zadań spowodowały, że w analizie bezpieczeństwa algorytmów kryptograficznych pomijano zagadnienia związane z ich implementacją. Znamienne dla prac pojawiających się do lat 90-tych XX wieku są szczegółowe analizy i dowody bezpieczeństwa, sprowadzające problem ataku na algorytm do znanego i trudnego problemu matematycznego, np. problemu faktoryzacji czy logarytmu dyskretnego. Poruszane w tych pracach zagadnienia implementacji algorytmów ograniczały się natomiast do zapewnienia jak największej szybkości działania.

Idealistyczne założenie dotyczące implementacji algorytmów kryptograficznych pozwoliło naukowcom przez wiele lat rozwijać techniki kryptograficzne i udowadniać ich bezpieczeństwo. Dowody te konstruowano poprzez sprowadzanie ataków do problemów trudnych obliczeniowo oraz oszacowywanie czasu potrzebnego na ich rozwiązanie. W analizach tych zapominano jednak o tym, że podobnie jak w przypadku ludzi również maszyny są podatne na błędy, które choć wynikają z całkowicie innych powodów, mogą mieć równie katastrofalne skutki dla bezpieczeństwa. Przez wiele lat nie zauważano również, że informacje na temat działania układów cyfrowych można uzyskać poprzez analizę ulotu elektromagnetycznego (pobieranej mocy, wytwarzanych pól elektromagnetycznych, itp.). Przyczyn pomijania tych zagadnień w analizie i ocenie algorytmów kryptograficznych należy upatrywać w ogromnym udziale matematyków w rozwoju obu dziedzin. W efekcie, większość powstających prac nie poruszała zagadnień implementacji pomijając je w procesie opracowywania i analizy algorytmów kryptograficznych. Sytuacja zmieniła się w latach 90-tych XX wieku, gdy liczne prace zaczęły opisywać znaczenie implementacji algorytmów dla bezpieczeństwa. Pokazały one, że sposób implementacji algorytmu kryptograficznego jest równie istotny co złożoność rozwiązania trudnego problemu matematycznego będącego podstawą bezpieczeństwa.

1.2 Kryptoanaliza z uszkodzeniami

Zagrożenia wynikające z niepoprawnego wykonania algorytmu kryptograficznego były powszechnie znane niemal od początku rozwoju kryptografii i kryptoanalizy. Wiadomo było, że może ono uniemożliwić odszyfrowanie wiadomości, a nawet, tak jak w przypadku szyfru Lorentza, złamać bezpieczeństwo całego algorytmu. Wydaje się, że o problemie błędów zapomniano w momencie upowszechnienia komputerów

i gwałtownego rozwoju kryptografii, zawierzając poprawności wykonania algorytmów realizowanych przez układy cyfrowe.

Sytuacja odmieniła się dopiero w latach 90-tych XX wieku, kiedy po raz pierwszy powszechnie opisano jakie zagrożenia dla bezpieczeństwa algorytmów kryptograficznych wynikają ze sposobu ich implementacji. Zauważono wówczas, że atak na algorytm można przeprowadzić nie przez rozwiązywanie trudnego problemu matematycznego leżącego u jego podstaw, ale drogą analizy oddziaływania urządzenia na otoczenie. Zaobserwowano również, że przeprowadzenie ataku może być prostsze, jeśli przy analizie algorytmu i jego implementacji uwzględni się dodatkowe informacje, będące ubocznym efektem działania układu cyfrowego, np. zmiany poboru mocy przez urządzenie, czas wykonania algorytmu oraz emitowane pole elektromagnetyczne. Te uboczne efekty wykonania algorytmu kryptograficznego tworzą stowarzyszony kanał informacyjny (ang. *side-channel*), dostarczający atakującemu dodatkowej wiedzy na temat działania algorytmu — stąd nazwa *side-channel attacks*.

Efektem pierwszych prac dotyczących ataków na implementacje algorytmów kryptograficznych było pokazanie, że atak taki można uprościć poprzez oddziaływanie na urządzenie w czasie gdy realizuje ono algorytm kryptograficzny. Okazało się wówczas, że wiele z takich oddziaływań jest od wielu lat dobrze znanych i przeanalizowanych, gdyż powszechnie występują one w wielu dziedzinach zastosowań układów elektronicznych. Efektem tych oddziaływań może być przekłamywanie wartości przechowywanych w pamięci i rejestrach układu cyfrowego, zmiana przebiegu wykonania programu czy całkowity reset urządzenia. Badania pokazały, że skutki takich błędów w układach kryptograficznych pozwalają znacznie uprościć przeprowadzenie ataku i umożliwiają odtworzenie kluczy kryptograficznych. Realność tego zagrożenia pokazano dla wszystkich znanych algorytmów kryptograficznych mimo istnienia formalnych dowodów ich bezpieczeństwa.

W publikacjach ostatnich lat wykazano, że dla bezpieczeństwa algorytmów kryptograficznych istotny jest nie tylko problem matematyczny, na którym bazują i jego złożoność obliczeniowa, ale również sposób implementacji oraz oddziaływanie urządzenia/oprogramowania z otoczeniem (m.in. [2, 3, 18, 37, 44, 52, 59]). Zaprezentowano nowe metody ataku polegające nie na czasochłonnym poszukiwaniu rozwiązań problemów trudnych obliczeniowo, lecz na wykorzystaniu właściwości algorytmów i analizie ich przebiegu lub zachowania w przypadku nieznacznej zmiany danych na których operują. Metody te zostały określone mianem ataków z uszkodzeniami (ang. *fault analysis*). W atakach tych pierwszym celem atakującego jest wprowadzenie uszkodzenia do układu cyfrowego. Uszkodzenie takie ma za zadanie spowodować niepoprawny przebieg algorytmu lub zmianę danych, na których algorytm operuje. Błąd taki może być wprowadzony na wiele sposobów. Powszechnie znane są przypadki przekłamywania poszczególnych bitów pamięci RAM w efekcie oddziaływania promieniowania jonizującego [5]. W technice cyfrowej stosowana jest inżynieria

wsteczna (ang. *reverse engineering*) [59], która pozwala uzyskać bezpośredni dostęp do wewnętrznych elementów układów. Wykorzystywane są również wyprowadzenia testowe [59], za pomocą których atakujący może analizować i oddziaływać na wewnętrzne stany układu w czasie realizacji procedur kryptograficznych. Niepoprawne działanie urządzenia można również spowodować za pomocą zakłóceń wprowadzanych do sygnału zegarowego i napięcia zasilania [3, 14]. Zakłócenia takie mogą m.in. powodować pomijanie wykonania niektórych procedur algorytmu i przekłamywanie danych przechowywanych w pamięci. Drugim krokiem w ataku z uszkodzeniami jest obserwacja działania urządzenia i analiza uzyskanych, błędnych kryptogramów. Celem analizy jest uzyskanie częściowej wiedzy o stosowanym kluczu kryptograficznym.

Podatność na kryptoanalizę z uszkodzeniami została wykazana dla niemal wszystkich współcześnie stosowanych algorytmów kryptograficznych. Praca Bihama i Shamira [11] pokazała, że kryptoanaliza z uszkodzeniami rozszerza możliwości zastosowania kryptoanalizy różnicowej i może być zastosowana przeciw wszystkim algorytmom symetrycznym podatnym na ten rodzaj ataku. Dalsze badania [2, 4, 17, 36] nad wykorzystaniem uszkodzeń wprowadzonych do urządzeń i algorytmów kryptograficznych wykazały, że również algorytmy asymetryczne, schematy podpisów cyfrowych i schematy identyfikacji są podatne na ten rodzaj ataku. W rezultacie prowadzonych prac pokazano, że żaden z powszechnie stosowanych algorytmów kryptograficznych nie jest odporny na ataki z uszkodzeniami. Szczegółowo przeanalizowano zarówno implementacje algorytmów symetrycznych (DES, AES, IDEA, Blowfish) [6, 11, 14, 25, 31, 35, 71] jak i asymetrycznych (RSA, ElGamal, DSA)[3, 17, 22, 36, 41, 57, 78, 86]. Udowodniono, że wszystkie te algorytmy mogą zostać złamane, jeśli tylko atakujący potrafi zakłócić dane na których operują, albo zmienić przebieg samego algorytmu. Co więcej, atak taki jest w wielu przypadkach stosunkowo prosty do przeprowadzenia, a wymagane możliwości atakującego i nakład potrzebnych prac nie są duże.

Siłę nowych metod ataku widać bardzo dobrze na przykładzie ataku na algorytm RSA-CRT, który w swoim działaniu wykorzystuje właściwości konwersji odwrotnej według tzw. chińskiego twierdzenia o resztach (CRT) [17]. W ataku tym do odkrycia klucza prywatnego wystarczy, aby atakujący wygenerował zaledwie jeden niepoprawny szyfrogram. Co najważniejsze atak powiedzie się niezależnie od rodzaju wprowadzonego błędu, a istotne jest jedynie, aby błąd wprowadzić do jednej z dwóch operacji potęgowania. Jak zostanie pokazane w dalszej części pracy, wprowadzenie takiego błędu nie nastręcza trudności nawet słabo przygotowanym atakującym. Prostota tego ataku pokazuje, że kryptoanaliza z uszkodzeniami jest potężnym narzędziem potrafiącym złamać bezpieczeństwo nawet najsilniejszych algorytmów kryptograficznych. W pracy [17] pokazano, że również inne algorytmy asymetryczne, takie jak schematy podpisów cyfrowych i schematy identyfikacji, są podatne na atak z uszkodzeniami. W kolejnych pracach [30, 36, 67] zaproponowano rozszerzenie możliwości ataku, upraszczając ich złożoność i umożliwiając zastosowanie do szerszej klasy algorytmów.

Podobne, choć już nie tak spektakularne, możliwości wykorzystania kryptoanalizy z uszkodzeniami istnieją w przypadku algorytmów symetrycznych. Już w pierwszej pracy na ten temat [11] podejmowane zagadnienie przedstawiono w sposób ogólny prezentując podatność wszystkich algorytmów symetrycznych na nowy rodzaj ataku. Autorzy tej pracy zaprezentowali w jaki sposób, atakujący może wykorzystać błędy wprowadzane do algorytmów symetrycznych w celu przeprowadzenia kryptoanalizy różnicowe. Wprowadzenie błędu pozwala bowiem analizować fragment zamiast całości przebiegu algorytmu. W skrajnym przypadku, gdy atakujący potrafi dokładnie kontrolować czas i miejsce wprowadzenia błędu, atak z uszkodzeniami można sprowadzić do ataku różnicowego, w którym różnice zadawane są na wejściu jednej z ostatnich rund algorytmu. Wnioski zaprezentowane w tej pracy zostały w późniejszych latach potwierdzone przez szczegółowe analizy wielu algorytmów symetrycznych. O randze zagrożenia, wprowadzanego przez ataki z uszkodzeniami, może również świadczyć to, że analiza podatności na nie była jednym z kryteriów oceny algorytmów, biorących udział w konkursie na zaawansowany standard szyfrowania AES. Zagadnienie kryptoanalizy z uszkodzeniami poszerzono też o tak zwane ataki z niemożliwymi błędami (ang. *impossible fault analysis*) [12]. Atak tego typu może być zastosowany do złamania algorytmu RC4.

Kolejne prace [38, 49] pokazały, że również szyfry strumieniowe i szyfry wykorzystujące rejestry przesuwne, nie są odporne na ten rodzaj ataku. Podstawowym elementem konstrukcyjnym takich szyfrów są rejestry przesuwne generujące strumień klucza szyfrującego. Umiejętność wprowadzania błędów daje atakującemu możliwość wpływania na działanie tych rejestrów — zakłada się np:, że atakujący może wstrzymać przesuwanie wybranych rejestrów, lub przekłamać niektóre z bitów. Takie oddziaływanie pozwala na zmiany generowanego klucza, obserwację błędnego szyfrogramu i wnioskowanie na jego podstawie o wartościach inicjujących rejestry.

Powyższy opis kryptoanalizy z uszkodzeniami pozwala zauważyć jak duże możliwości dostarcza ona atakującemu. Siłą tej metody jest oddziaływanie na wewnętrzne stany i/lub przebieg algorytmu. Tak więc, atakujący nie musi ograniczać się jedynie do analizy szyfrogramu i wiadomości. Nowa metoda ataku pozwala parametryzować i zmieniać działanie algorytmu przez zakłócenie danych, na których algorytm operuje oraz zmianę wartości rejestrów kontrolujących przebieg algorytmu. Możliwość zmiany przebiegu algorytmu i pominięcia wykonania niektórych operacji, pozwala atakującemu uprościć algorytm kryptograficzny poprzez umożliwienie analizy mniejszej liczby rund. W wyniku tego możliwe staje się przeprowadzenie np: kryptoanalizy różnicowej lub pominięcie procedur kontrolujących poprawność wykonywania obliczeń [84]. Możliwość zastosowania kryptoanalizy z uszkodzeniami wynika natomiast z tego, że istnieje cała gama metod pozwalających atakującemu wprowadzić błędy do układu cyfrowego. Atakujący może ponadto dobierać metodę wprowadzenia błędów, a tym samym wpływać na złożoność całej procedury ataku.

1.3 Ochrona przed kryptoanalizą z uszkodzeniami

Zagrożenia wynikające z kryptoanalizy z uszkodzeniami są dodatkowo potęgowane przez praktyczny brak skutecznych rozwiązań zabezpieczających. Dotyczy to zarówno w miarę dobrze przebadanego algorytmu asymetrycznego RSA oraz innych algorytmów wykorzystujących chińskie twierdzenie o resztach, jak i algorytmów symetrycznych, takich jak AES, RC6 i IDEA. Jeszcze gorzej jest w przypadku pozostałych algorytmów (m.in. schematów podpisów ElGamala, DSA, schematów identyfikacji itp.). Algorytmy te w ogóle nie znalazły się w kręgu zainteresowań naukowców mimo, że są powszechnie stosowane do ochrony informacji i wiadomo, że są podatne na atak z uszkodzeniami.

Do dnia dzisiejszego nie zaproponowano żadnych skutecznych rozwiązań zabezpieczających, które mogłyby być zastosowane do ochrony algorytmu AES. Istniejące rozwiązania mają za cel przeciwdziałanie wprowadzeniu uszkodzeń do układu albo wykrywanie wywołanych błędów. Pierwsza grupa rozwiązań wykorzystuje różnego rodzaju układy sprzętowe, które są odpowiedzialne za kontrolę działania urządzenia i wykrywanie ingerencji z zewnątrz. W przypadku wykrywania błędów zabezpieczenia te mają za zadanie przerwać działanie układu i zapewnić bezpieczeństwo klucza kryptograficznego.

Jednym z pierwszych rozwiązań tego typu była propozycja Bertoniego [7], polegająca na wykorzystaniu bitów parzystości do kontroli przebiegu algorytmu AES. W pracy tej przeanalizowano poszczególne transformacje algorytmu i wyprowadzono reguły pozwalające przewidywać poprawne bity parzystości po kolejnych transformacjach. Umożliwia to porównanie przewidywanych i rzeczywistych bitów parzystości, których różne wartości sygnalizują wprowadzenie błędu. Rozwiązanie zaproponowane przez Bertoniego pozwala wykrywać wszystkie błędy nieparzystej krotności. Dalsze prace prowadzone przez zespół Bertoniego [8] pokazały, że analiza bitów parzystości pozwala na określenie momentu wprowadzenia błędu. Wadą tych rozwiązań jest jednak zapewnianie ochrony tylko dla wąskiej grupy błędów.

Rozszerzeniem prac zespołu Bertoniego jest propozycja Yen i Wu [87], wykorzystująca mechanizm kontrolny CRC (ang. *cyclic redundancy check*) do sprawdzania poprawności przebiegu algorytmu i wykrywania błędów. Zaproponowane rozwiązanie wykorzystuje dobre właściwości algorytmu AES pozwalające znacznie uprościć procedury wyznaczania wartości CRC. Rozwiązanie to posiada znacznie lepsze właściwości niż wcześniejsza propozycja Bertoniego, pozwalające wykryć niemal wszystkie błędy wykorzystywane do ataków na algorytm AES.

Odmienne rozwiązanie zabezpieczające zostało zaproponowane w 2000 roku przez zespół Fernández-Gómez [34]. Wykorzystuje ono technikę równoległego wykrywania błędów CED (ang. *concurrent error detection*), bazującą na właściwościach symetrycznych algorytmów kryptograficznych — liniowości większości wykorzystywanych transformacji i możliwości ich odwrócenia. W kolejnych pracach [50, 51, 53, 55, 79] przedstawiono szczegółowe rozwiązania dla wybranych algorytmów i zdefiniowano trzy rodzaje CED: 1)

na poziomie całego algorytmu kryptograficznego (ang. *algorithm level*), 2) na poziomie pojedynczej rundy algorytmu (ang. *round level*) i 3) na poziomie pojedynczej transformacji algorytmu (ang. *transformation level*). Zaproponowane rodzaje CED różnią się pomiędzy sobą złożonością implementacyjną i opóźnieniem wykrycia błędu. Ich główną zaletą, w porównaniu do rozwiązań bazujących na kodach detekcyjnych [7, 60, 26], jest możliwość wykrywania dowolnych rodzajów błędów.

W pracy [60] zaproponowano ochronę algorytmu AES z wykorzystaniem nieliniowych kodów detekcyjnych — tak zwanych silnych kodów (ang. *robust codes*). W przeciwieństwie do rozwiązań wykorzystujących jeden rodzaj kodu detekcyjnego, w pracy tej zaproponowano użycie różnych algorytmów ochronnych dla każdej transformacji. Postępowanie takie pozwala dopasować metody ochrony do konkretnych transformacji a przez to wypośrodkować pomiędzy skutecznością a narzutem implementacyjnym. Trudnością jest konieczność powiązania wszystkich stosowanych rozwiązań, w celu zagwarantowania ochrony całego algorytmu a nie tylko pojedynczych transformacji.

Rozwiązania bazujące na wykrywaniu błędów, zaproponowane dla AES i innych algorytmów symetrycznych nie gwarantują jednak całkowitego bezpieczeństwa. Zabezpieczenia te nie uniemożliwiają bowiem przeprowadzenia skutecznego ataku, lecz przesuwają obszar ich działania do innych elementów algorytmu. Działanie takie powoduje, że atakujący musi zmodyfikować metodę ataku, wykorzystując inne rodzaje błędów i/lub inne metody ataku.

Jeszcze ciekawiej wygląda sprawa ochrony algorytmu RSA (szczególnie wersji bazującej na chińskim twierdzeniu o resztach). Dla algorytmu tego zaproponowano wiele różnych rozwiązań zabezpieczających [3, 15, 37, 56, 80, 85], z czego spora grupa okazała się nieskuteczna pozostając podatnymi na ataki z uszkodzeniami [68, 78, 86]. W przypadku kilku propozycji ochronnych nie są znane żadne metody przeprowadzenia ataku [37, 56] a w innych [15] ciągle toczy się dyskusja czy proponowane rozwiązanie jest bezpieczne czy nie [16, 78]. Cechą wspólną większości proponowanych rozwiązań jest ich specjalizacja powodująca, że mogą one być wykorzystane do ochrony wąskiej grupy algorytmów kryptograficznych. Wszystkie proponowane do tej pory rozwiązania są również pozbawione formalnych dowodów gwarantowanego poziomu bezpieczeństwa.

Najnowsze artykuły dotyczące ochrony algorytmów asymetrycznych proponują wykorzystanie kodów detekcyjnych do kontroli poprawności przebiegu algorytmu [21, 44]. Wadą tych rozwiązań jest wykorzystanie kodów detekcyjnych nie nadających się do kontroli operacji w arytmetyce modularnej. Z tego względu proponowana ochrona obejmuje jedynie wybrane operacje, przyjmując dodatkowe założenia na temat sposobu ich implementacji. W przypadku [21] ochrona dotyczy pojedynczych operacji mnożenia, wykonywanych w czasie szyfrowania RSA, i zaimplementowanych z wykorzystaniem algorytmu Montgomerego [63]. Wadą tego typu rozwiązań jest brak kompleksowej ochrony całego algorytmu.

W przypadku pozostałych algorytmów asymetrycznych brak jest jakichkolwiek rozwiązań ochronnych mimo, że stwierdzono ich podatność na kryptoanalizę z uszkodzeniami. Tak jest m.in. w przypadku schematów podpisów cyfrowych ElGamala i schematów pokrewnych [4, 30, 36, 65].

Pomimo braku rozwiązań zabezpieczających dla poszczególnych algorytmów kryptograficznych producenci układów kryptograficznych starają się zagwarantować bezpieczeństwo swoich produktów. Zastosowanie znajduje tu zapobieganie atakom poprzez wykorzystanie różnego rodzaju sprzętowych metod ochrony algorytmu i układu kryptograficznego przed wprowadzeniem błędu. Mimo niewątpliwych zalet takiego rozwiązania nie zapewnia ono całkowitej odporności na ataki z uszkodzeniami. Ograniczenie to wynika z ciągłego postępu technologii, miniaturyzacji układów kryptograficznych i silnej zależności ich działania od parametrów otoczenia — temperatury, napięcia zasilania czy pól elektromagnetycznych.

1.4 Teza pracy

Na podstawie analizy metod przeprowadzania ataków z uszkodzeniami oraz istniejących rozwiązań ochronnych, zaobserwowano, że:

1. Wszystkie znane algorytmy kryptograficzne są podatne na kryptoanalizę z uszkodzeniami. Wynikające stąd zagrożenie dla bezpieczeństwa algorytmów jest zróżnicowane i wymaga od atakującego użycia różnorodnych technik. Niemniej jednak, ataki z uszkodzeniami pozwalają uprościć kryptoanalizę i znacząco skrócić czas niezbędny do jej przeprowadzenia.

2. Istniejące rozwiązania zabezpieczające, stosowane do ochrony algorytmów symetrycznych (m.in. AES), nie gwarantują całkowitego bezpieczeństwa. Przeprowadzenie ataku jest możliwe pomimo zaimplementowania w układzie różnorodnych rozwiązań zabezpieczających, których zadaniem jest zapobieganie wprowadzeniu i/lub wykrywanie wprowadzonych błędów.

3. Powszechnie stosowane algorytmy podpisów cyfrowych ElGamala i DSA są pozbawione jakichkolwiek mechanizmów ochronnych. Jest tak pomimo, że ich podatność na atak z uszkodzeniami jest powszechnie znana, a algorytmy te są szeroko wykorzystywane w praktyce. Brak zabezpieczeń pozwala atakującemu wydobyć informację o stosowanym kluczu kryptograficznym na podstawie analizy od kilkuset do kilku tysięcy błędnych szyfrogramów.

4. Symetryczne algorytmy kryptograficzne mogą być skutecznie chronione za pomocą rozwiązań dających możliwość korekcji błędów wprowadzanych do urządzenia i ujednolicających zewnętrzne parametry działania układu realizującego algorytm kryptograficzny (np. czas wykonania i pobór mocy).

5. Asymetryczne algorytmy kryptograficzne mogą być chronione przez zastosowanie procedur rozpraszania błędów wprowadzonych do urządzeń. Rozproszenie takie powinno zagwarantować, że każdy wprowadzony błąd generuje kryptogram nieprzydatny z punktu widzenia atakującego np. poprzez uwikłanie postaci wyniku od wielu czynników utrudniających prześledzenie wpływu wprowadzonego błędu na zmiany wyniku końcowego.

6. Implementacje algorytmów ochronnych powinny być silne zintegrowane z algorytmem kryptograficznym, uniemożliwiając wyłączenie układów ochronnych.

Brak skutecznych rozwiązań ochronnych zmotywował autora do analizy ataków z uszkodzeniami i poszukiwania rozwiązań zabezpieczających. Przeprowadzona analiza istniejących rozwiązań pozwala postawić tezę, że:

Implementacja procedur korekcji błędów dla algorytmu szyfrowania AES
oraz rozpraszania błędów dla schematów podpisów cyfrowych ElGamala i DSA
umożliwia ochronę tych algorytmów przed atakami z uszkodzeniami.

W celu wykazania prawdziwości powyższej tezy, w rozprawie dokonano analizy znanych metod wprowadzania uszkodzeń do układów cyfrowych, istniejących ataków z uszkodzeniami i sposobów ochrony przed nimi.

Analiza metod wprowadzania uszkodzeń pozwoliła określić rodzaje błędów najczęściej występujących w układach cyfrowych. Pozwoliło to na przeanalizowanie i pogrupowanie istniejących metody ochrony przed kryptoanalizą z uszkodzeniami. Na tej podstawie oceniono możliwość praktycznego przeprowadzenia ataków opisywanych w literaturze i wybrano grupę algorytmów do dalszej analizy.

Wybór algorytmu AES był podyktowany tym, że jest on obecnie najpowszechniej stosowanym symetrycznym algorytmem kryptograficznym, a jego podatność na kryptoanalizę z uszkodzeniami została dokładnie przebadana. Dodatkowo analiza istniejących metod ochrony pokazała, że są one niewystarczające. Drugą grupę algorytmów stanowiły schematy podpisów cyfrowych ElGamala i schematy pokrewne. Przegląd literatury w tym zakresie pozwolił zauważyć, że pomimo powszechnego stosowania tych algorytmów oraz świadomości, że są one podatne na kryptoanalizę, nie zaproponowano dla nich żadnych rozwiązań ochronnych.

Szczegółowa analiza ataków z uszkodzeniami oraz istniejących rozwiązań zabezpieczających dla rozważanych algorytmów, pozwoliła zaproponować nowe metody ochrony. Rozwiązania te wykorzystują korekcję i rozpraszanie błędów w celu zagwarantowania, że atakujący nie będzie mógł przeprowadzić ataku z uszkodzeniami. Zaproponowane modyfikacje zostały następnie szczegółowo przeanalizowane. Oszacowano ich złożoność oraz wykazano poprawność działania i zapewniany poziom ochrony.

Rozdział 2

Algorytmy kryptograficzne

Zagadnienie ochrony przed atakami z uszkodzeniami jest rzadko poruszane w literaturze. Istniejące rozwiązania ochronne dotyczą najbardziej popularnych algorytmów symetrycznych (AES, RC6, IDEA) i ograniczają się tylko do wykrywania błędów. W przypadku algorytmów asymetrycznych istniejące rozwiązania mogą być zastosowane wyłącznie do algorytmów bazujących na chińskim twierdzeniu o resztach, przede wszystkim RSA. Ze względu na trudność implementacji wykrywania i korekcji błędów, rozwiązania te wykorzystują rozpraszanie uszkodzeń.

Na podstawie analizy literatury oraz specyfiki algorytmów kryptograficznych można dojść do wniosku, że ochrona przed atakami może być zrealizowana albo przez zastosowanie kodów detekcyjnych albo przez doprowadzenie do rozproszenia uszkodzeń. W pierwszym przypadku możliwa jest kontrola poprawności poszczególnych etapów algorytmu. Metoda ta daje się zastosować w większości algorytmów symetrycznych, które wielokrotnie wykonują sekwencje prostych przekształceń logicznych lub działań odwracalnych w ciałach skończonych. Efektem drugiego podejścia ochronnego powinno być takie zniekształcenie wyniku generowanego przez algorytm, które uniemożliwi lub poważnie utrudni kryptoanalizę. Metoda taka może być użyta do ochrony algorytmów asymetrycznych.

W niniejszej pracy wykazano skuteczność korekcji i rozpraszania błędów w odniesieniu do trzech powszechnie używanych algorytmów kryptograficznych:

- symetrycznego algorytmu szyfrowania AES (ang. *Advanced Encryption Standard*),
- schematu podpisów cyfrowych ElGamala,
- schematu podpisów cyfrowych DSA (ang. *Digital Signature Algorithm*).

W pracy poruszono również zagadnienia ochrony przed atakami na algorytmy DES i RSA. Oba algorytmy zostały przedstawione, w celu zilustrowania omawianych zagadnień oraz ze względu na istnienie szczegółowych analiz tych algorytmów pod kątem podatności na ataki z uszkodzeniami.

Kolejne podrozdziały zawierają najistotniejsze informacje na temat działania wszystkich algorytmów omawianych w pracy.

2.1 Algorytmy symetryczne

W symetrycznych algorytmach kryptograficznych do szyfrowania i deszyfrowania wykorzystywany jest ten sam klucz K. Konieczność ochrony tego klucza powoduje, że algorytmy symetryczne są również określane mianem algorytmów z tajnym/prywatnym kluczem (ang. *secret/private key*). Współczesne algorytmy symetryczne, ze względu na rodzaje operacji wykonywanych w trakcie szyfrowania i deszyfrowania mogą być podzielone na dwie grupy. W przypadku większości algorytmów bazujących na sieci Feistla (m.in. Lucifer, DES) procedury szyfrowania i deszyfrowania są dokładnie takie same a różnica polega jedynie na różnej kolejności używanych kluczy rund K_r (klucze te generowane są na podstawie klucza głównego K). W pozostałych algorytmach symetrycznych, w tym również niektórych algorytmach bazujących na sieci Feistla (np: IDEA, RC5), procedury szyfrowania i deszyfrowania wykorzystują różne transformacje.

Istotnym elementem każdego kryptosystemu symetrycznego jest procedura rozszerzania klucza głównego K (ang. *key schedule*). Służy do zwiększenia długości klucza kryptograficznego stosowanego w algorytmie tak, aby każda runda miała swój własny klucz rundy K_r (ang. *round key*). Nie wnikając w szczegóły takiej procedury można stwierdzić, że w większości kryptosystemów symetrycznych schemat generacji kluczy rund wykorzystuje te same lub bardzo podobne transformacje jak w procedurach szyfrowania i/lub deszyfrowania.

W dalszych opisach algorytmów symetrycznych pominięte zostaną opisy procedur rozszerzania klucza. Uproszczenie to zostało przyjęte dlatego, że procedury te nie będą przedmiotem badań i analizy w ramach pracy doktorskiej.

2.1.1 Szyfr Data Encryption Standard — DES

Algorytm DES jest symetrycznym szyfrem blokowym stworzonym w latach 70. XX wieku. Algorytm ten szyfruje 64-bitowe bloki danych kluczem złożonym z 56 bitów (klucz podawany na wejście algorytmu jest 64-bitowy, ale osiem bitów jest bitami parzystości, stąd efektywna długość klucza wynosi 56 bitów).

DES jest algorytmem zbudowanym w oparciu o sieć Feistla, której konstrukcja zapewnia, że ten sam algorytm może być użyty zarówno do szyfrowania jak i deszyfrowania danych. Podstawowa struktura sieci Feistl'a zakłada, że szyfrowany/deszyfrowany blok danych dzielony jest na połowy — lewą L i prawą R. Jedna z połówek bloku poddawana jest transformacji w funkcji F, która parametryzowana jest kluczem rundy K_r. Wynik funkcji F jest sumowany za pomocą bitowej operacji `XOR` z drugą połówką bloku danych. Ostatnim krokiem jest zamiana połówek bloków miejscami zapewniająca, że w następnej iteracji

algorytmu drugi z bloków będzie poddany transformacji F. Jak już wspomniano, zaletą takiej konstrukcji jest jej odwracalność niezależna od postaci funkcji F. Oznacza to, że dla dowolnej funkcji F (również nieodwracalnej) szyfrowanie i deszyfrowanie przebiega dokładnie według takiego samego schematu.

Algorytm DES składa się z 16 rund, w których szyfrowane dane dzielone są na bloki 32-bitowe L_i, R_i dla $i = 1, 2 \ldots 16$. W każdej rundzie blok R_i podawany jest transformacji F składającej się z sieci permutacji i podstawień uzależnionych dodatkowo od klucza rundy K_r. Wynik funkcji $F(R_i, K_r)$ jest dodawany do bloku L_i za pomocą bitowej operacji `XOR`. Ostatnim krokiem wykonywanym we wszystkich rundach za wyjątkiem końcowej, jest zamiana bloków miejscami. W ten sposób w kolejnej rundzie $L_{i+1} = R_i$, $R_{i+1} = L_i \oplus F(R_i, K_r)$.

2.1.2 Szyfr Advanced Encryption Standard — AES

Zaawansowany standard szyfrowania (ang. *Advanced Encryption Standard*) jest następcą algorytmu DES. Podobnie jak DES jest on symetrycznym szyfrem blokowym, jednak w odróżnieniu od niego nie bazuje na sieci Feistla i nie wykorzystuje sieci permutacyjno-podstawieniowej. Rundy algorytmu AES składają się z czterech transformacji bazujących na działaniach realizowanych w ciele Galois $\mathrm{GF}(2^8)$:

- AddRoundKey — dodanie klucza rundy do szyfrowanych danych,
- MixColumns — przekształcenie kolumny macierzy stanu,
- ShiftRows — przesunięcie cykliczne wierszy macierzy stanu,
- SubBytes — zamiana bajtów macierzy stanu.

Liczba rund AES zależy od rozmiaru klucza K — dla klucza 128-bitowego algorytm wykonuje 10 rund, dla 196-bitowego — 12, a dla 256-bitowego — 14 rund. Pierwszą transformacją wykonywaną w algorytmie jest AddRoundKey po której następują rundy algorytmu. W każdej rundzie, za wyjątkiem ostatniej, wykonywane są wszystkie cztery transformacje w kolejności: SubBytes, ShiftRows, MixColumns i AddRoundKey. W rundzie ostatniej nie jest wykonywana transformacja MixColumns.

Algorytm szyfruje bloki danych o rozmiarze 128 bitów z użyciem klucza o rozmiarze 128, 196 albo 256 bitów. Blok danych reprezentowany jest w algorytmie w postaci macierzy 4×4 nazywanej *Stanem* $\mathbb{S}$. Elementami macierzy $\mathbb{S}$ są bajty $s_{i,j}$ dla $0 \leq i, j \leq 3$, gdzie i oznacza wiersz macierzy a j jej kolumnę. Kolumna stanu $\mathbb{S}$, złożona z czterech bajtów określana jest mianem *Słowa* $\mathbb{W}$ (rys. 2.1).

W algorytmie AES transformacje są wykonywane na elementach stanu $\mathbb{S}$ — pojedynczych bajtach $s_{i,j}$ (transformacje AddRoundKey, SubBytes), słowach $\mathbb{W}_j$(MixColumns) i wierszach (ShiftRows), przekształcając go w stan wyjściowy $\mathbb{S}'$. Wykonywane przekształcenia obejmują dodawanie $\oplus$ i mnożenie $\otimes$ w ciele $\mathrm{GF}(2^8)$ oraz mnożenie wielomianów o współczynnikach w ciele $\mathrm{GF}(2^8)$.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Rysunek 2.1: Macierz stanu $\mathbb{S}$ algorytmu AES.

Elementy ciała rozszerzonego GF(2^8) mogą być zapisywane w postaci wielomianów stopnia mniejszego od 8 i współczynnikach będących elementami ciała prostego — GF(2). Elementami ciała prostego GF(2) są 0 i 1 a dodawanie w tym ciele jest realizowane za pomocą operacji `XOR`. Korzystając z reprezentacji wielomianowej każdy element ciała rozszerzonego GF(2^8) może być zapisany w postaci

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \quad \text{gdzie} \quad a_i = \{0, 1\}\,. \tag{2.1}$$

Skrócony zapis wielomianu wykorzystuje wyłącznie współczynniki wielomianu $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$, pomijając czynniki x^i. Z tego względu elementy ciała GF(2^8) są zazwyczaj reprezentowane w postaci ciągu ośmiu zer i jedynek lub za pomocą cyfr szesnastkowych (taką notację będziemy stosowali w dalszej części pracy). Ze względu na podobieństwo tego zapisu do systemu binarnego powszechnie stosuje się analogię w nazewnictwie używając zamiennie określeń współczynnik i bit oraz element ciała i bajt.

Wielomianowa reprezentacja elementów ciała pozwala łatwo wykonać dodawanie $a(x) \oplus b(x)$. Wynikiem jest wielomian $c(x)$ o współczynnikach będących sumą odpowiednich współczynników wielomianów $a(x)$ i $b(x)$, które dodawane są tak jak elementy ciała GF(2)

$$\begin{aligned} a(x) \oplus b(x) \quad = \quad & (a_7 \oplus b_7)x^7 + (a_6 \oplus b_6)x^6 + (a_5 \oplus b_5)x^5 + (a_4 \oplus b_4)x^4 + \\ & + (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + a_0 \oplus b_0. \end{aligned} \tag{2.2}$$

Dodawanie takie jest realizowane za pomocą operacji `XOR` odpowiadających sobie współczynników $a(x)$ i $b(x)$. W zapisie binarnym jest to równoważne wykonaniu operacji `XOR` dwóch bajtów.

Mnożenie, ze względu na zamkniętość operacji w ciele, jest wykonywane modulo element pierwotny ciała. W przypadku ciał prostych są nimi liczby pierwsze a w przypadku ciał rozszerzonych wielomiany pierwotne. Wielomianem pierwotnym wykorzystywanym w AES jest wielomian

$$p(x) = x^8 + x^4 + x^3 + x + 1, \tag{2.3}$$

który w zapisie szesnastkowym. Mnożenie $a(x) \otimes b(x)$ w GF(2^8) jest wykonywane tak jak mnożenie wielomianów z tą różnicą, że operacje na współczynnikach wykonywane są modulo 2, a uzyskany wynik

jest redukowany modulo $p(x)$. W szczególnym przypadku, mnożenie $a(x)$ przez element ciała 02 daje w wyniku

$$\begin{aligned} a(x) \otimes 02 &= (a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0) \otimes x \bmod p(x) \\ &= (a_7x^8 + a_6x^7 + a_5x^6 + a_4x^5 + a_3x^4 + a_2x^3 + a_1x^2 + a_0x) \bmod p(x) \\ &= (a_6x^7 + a_5x^6 + a_4x^5 + (a_3 \oplus a_7)x^4 + (a_2 \oplus a_7)x^3 + a_1x^2 + (a_0 \oplus a_7)x + 1) \end{aligned} \tag{2.4}$$

Z powyższej zależności wynika, że mnożenie przez element 02 może być zaimplementowane jako przesunięcie binarnej reprezentacji elementu ciała w lewo i opcjonalny `XOR` z 11B, jeśli najstarszy bit był równy 1.

Ponieważ operacja mnożenia jest rozdzielna względem dodawania to każde mnożenie można zapisać jako sumę iloczynów częściowych. Rozdzielność operacji mnożenia względem dodawania pozwala zapisać jeden z czynników w postaci sumy elementów 01, 02, 04, 08, 10, 20, 40, 80 i wyznaczyć iloczyny częściowe (iloczyny drugiego czynnika przez każdy z elementów). Wynik mnożenia jest następnie wyznaczany jako suma iloczynów częściowych. Taka realizacja mnożenia pozawala na prostą implementację wykorzystującą jedynie operacje przesunięcia i dodawania `XOR`, na przykład:

$$a(x) \otimes 63 = a(x) \otimes (40 \oplus 20 \oplus 02 \oplus 01) = (a(x) \otimes 40) \oplus (a(x) \otimes 20) \oplus (a(x) \otimes 02) \oplus a(x). \tag{2.5}$$

Algorytm AES wykorzystuje również operacje wykonywane na wielomianach stopnia mniejszego od 4, których współczynniki są elementami ciała GF(2^8). Operacje realizowane są w sposób analogiczny do opisanych powyżej — współczynniki są mnożone i dodawane tak jak elementy ciała GF(2^8) a uzyskany wielomian jest redukowany modulo $v(x) = x^4 + 01$.

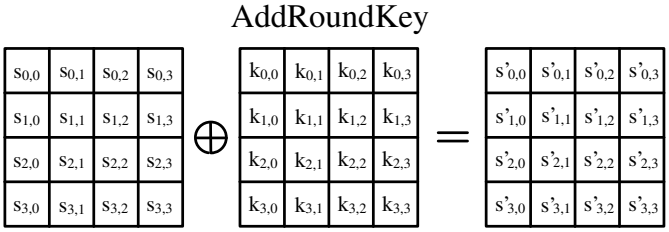
AddRoundKey

Transformacja AddRoundKey polega na dodaniu do stanu algorytmu AES klucza rundy. Klucz rundy jest generowany z klucza głównego na podstawie algorytmu rozszerzania klucza i jest również reprezentowany w postaci macierzy o rozmiarze 4×4. Sama transformacja polega na dodaniu do każdego bajta danych $s_{i,j}$ bajta klucza $k_{i,j}$ za pomocą bitowej operacji `XOR`. Wynikowa macierz stanu jest postaci

$$\mathbb{S}' = \left[s'_{i,j}\right]_{4\times4} \text{ gdzie } s'_{i,j} = s_{i,j} \oplus k_{i,j}. \tag{2.6}$$

Transformacja AddRoundKey jest operacją samoodwrotną co oznacza, że przebiega dokładnie w ten sam sposób w czasie szyfrowania i deszyfrowania.

Implementacja transformacji AddRoundKey wykorzystuje operacje dodawania bitowego modulo 2 (bitowy `XOR`) dla każdej pary $s_{i,j}$, $k_{i,j}$. W przypadku mikroprocesorów (μP) 32 i 64-bitowych transformację tą można zaimplementować odpowiednio jako bitowy `XOR` słów 32 i 64-bitowych.



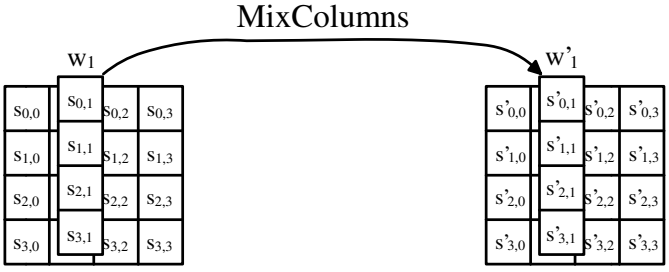
Rysunek 2.2: Transformacja AddRoundKey

MixColumns

Transformacja MixColumns operuje na słowach $\mathbb{W}_j$ macierzy stanu, których elementy są interpretowane jako kolejne współczynniki wielomianu $w(x)$ stopnia 3 nad GF(2^8). Wyznaczony w ten sposób wielomian jest następnie mnożony przez $c(x) = 03x^3 + 01x^2 + 01x + 02$ modulo $v(x) = x^4 + 01$. Mnożenie $w(x) \cdot c(x)$ wymaga wyznaczenia iloczynów każdej pary współczynników obu wielomianów. Iloczyny te są wyznaczane zgodnie z regułami mnożenia w GF(2^8), ponieważ współczynniki wielomianów są elementami tego ciała. Wynikiem mnożenia jest wielomian $w'(x)$ stopnia nie większego niż 3, którego kolejne współczynniki są nowymi elementami słowa wyjściowego $\mathbb{W}'$, stanu $\mathbb{S}'$.

$$\begin{aligned} \mathbb{S}' = [\mathbb{W}']_{1\times 4} \quad \text{gdzie} \quad & \mathbb{W}'_j = \left[s'_{0,j}, s'_{1,j}, s'_{2,j}, s'_{3,j}\right]^T \\ & w'_j(x) = \sum_{i=0}^{3} s'_{i,j} x^i = c(x) w_j(x) \bmod p(x) \ . \\ & w_j(x) = \textstyle\sum_{i=0}^{3} s_{i,j} x^i \end{aligned} \tag{2.7}$$

Transformacja MixColumns nie jest transformacją samoodwrotną. Z tego względu w procesie dekodowania kolejne słowa macierzy $\mathbb{S}$ są mnożone przez wielomian $d(x) = 0\text{b}x^3 + 0\text{d}x^2 + 09x + 0\text{e}$, który jest wielomianem odwrotnym do $c(x)$ modulo $v(x)$ tj. $c(x)d(x) \bmod v(x) = 01$.



Rysunek 2.3: Transformacja MixColumns

Implementacja transformacji bezpośrednio na podstawie zależności (2.7) wymaga wykonania 16 mnożeń w ciele GF(2^8) i redukcji wielomianu 6 stopnia modulo $v(x)$. Ze względu na postać wielomianu $v(x)$ redukcja ta może być uproszczona i zrealizowana za pomocą trzech dodawań w GF(2^8) odpowiednich współczynników wielomianu $c(x) \cdot w_j(x)$. Stąd, transformację MixColumn można wykonać za pomocą

mnożeń i dodawań w ciele GF(2^8). Dodatkowe uproszczenie obliczeń jest możliwe ze względu na odpowiedni dobór współczynników wielomianu $c(x)$. Uproszczenie to staje się widoczne, jeśli iloczyn $c(x) \cdot w(x)_j$ i całą transformację, zapiszemy w postaci iloczynu macierzy i wektora

$$\begin{aligned} w'_j(x) &= c(x)w_j(x) \bmod v(x) \\ &= \left(03x^3 + 01x^2 + 01x + 02\right)\left(s_{0,j}x^3 + s_{1,j}x^2 + s_{2,j}x + s_{3,j}\right) \bmod v(x) \\ &= \left[03s_{0,j}x^6 + (03s_{1,j} \oplus 01s_{0,j})\,x^5 + (03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j})\,x^4\right. \\ &\quad + (03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1})\,x^3 + (01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j})\,x^2 \\ &\quad \left. + (01s_{3,j} \oplus 02s_{2,j})\,x + 02s_{3,j}\right] \bmod v(x) \\ &= + (03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1})\,x^3 + (01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j} \oplus 03s_{0,j})\,x^2 \\ &\quad + (01s_{3,j} \oplus 02s_{2,j} \oplus 03s_{1,j} \oplus 01s_{0,j})\,x + (02s_{3,j} \oplus 03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j}) \end{aligned} \quad , \tag{2.8}$$

skąd

$$\begin{aligned} s'_{0,j} &= 02s_{3,j} \oplus 03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j} \\ s'_{1,j} &= 01s_{3,j} \oplus 02s_{2,j} \oplus 03s_{1,j} \oplus 01s_{0,j} \\ s'_{2,j} &= 01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j} \oplus 03s_{0,j} \\ s'_{3,j} &= 03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1} \end{aligned} \quad . \tag{2.9}$$

Korzystając z tych zależności transformację MixColumns można zapisać w postaci macierzowej

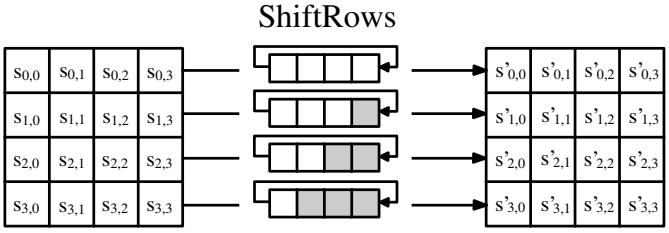
$$\mathbb{S}' = \left[\mathbb{W}'_j\right]_{1\times 4} \text{ gdzie } \mathbb{W}'_j = \begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{3,j} \\ s_{2,j} \\ s_{1,j} \\ s_{0,j} \end{bmatrix}. \tag{2.10}$$

Analiza wzoru (2.10) pozwala zauważyć, że dwa współczynniki o wartości 01 pozwalają w ogóle nie wykonywać ośmiu mnożeń. Mnożenie przez współczynnik 02 można natomiast zaimplementować jako przesuniecie o jedną pozycję bitową w lewo i opcjonalną redukcję modulo moduł 11B, jeśli wysunięty bit miał wartość jeden (2.4). Cztery mnożenia przez współczynnik 03 można zaimplementować jako dodawanie w ciele GF(2^8) wyniku mnożenia przez 01 i 02 (2.5).

ShiftRows

Transformacja ShiftRows powoduje cykliczne przesunięcie wierszy macierzy stanu $\mathbb{S}$ w lewo o różną liczbę pozycji bajtowych. Dla standardu AES rotacje te są wykonywane o 1 pozycję bajtową dla drugiego wiersza macierzy, 2 pozycje dla wiersza trzeciego i 3 dla wiersza czwartego; wiersz pierwszy w ogóle nie jest przesuwany.

$$\mathbb{S}' = \left[s'_{i,j}\right]_{4\times 4} \text{ gdzie } s'_{i,j} = s_{i,j+i \bmod 4} \tag{2.11}$$



Rysunek 2.4: Transformacja ShiftRows

W czasie deszyfrowania operacja ShiftRows przebiega podobnie, z tą tylko różnicą, że wiersze są przesuwane w prawo.

Najszybsze implementacje tej transformacji mają postać permutacji realizowanych za pomocą multiplekserów. W implementacjach programowych standardowym rozwiązaniem jest wykorzystanie rotacji słów 32-bitowych. Jeśli operacje takie nie są dostępne na liście rozkazów konkretnego procesora, to wówczas wykorzystuje się zwykłe przestawienia bajtów.

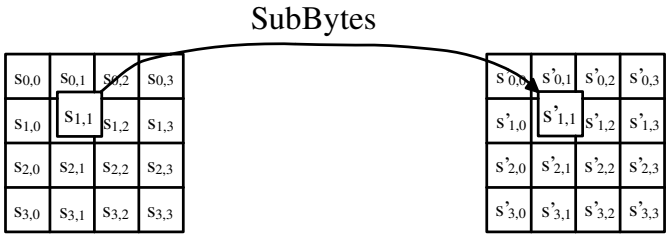
SubBytes

Jest to najbardziej złożona transformacja algorytmu AES. Realizuje ona podstawienia, jednak w przeciwieństwie do takich operacji wykonywanych w DES, nie korzysta z empirycznie utworzonej i przebadanej tablicy podstawień, lecz z tablicy zdefiniowanej za pomocą działań w ciele GF(2^8). Zadaniem operacji SubBytes jest wprowadzenie nieliniowości do przekształceń wykonywanych w czasie szyfrowania. Nieliniowość ta jest uzyskiwania przez wykorzystanie operacji odwrotności multiplikatywnej w GF(2^8), której poddawany jest każdy bajt $s_{i,j}$. Wyznaczona odwrotność jest następnie poddawana transformacji afinicznej składającej się z mnożenia przez F1 modulo $x^8 + 1$ (101) i dodanie stałej o wartości 63. Wynikiem jest wyjściowy bajt $s'_{i,j}$ stanu $\mathbb{S}'$.

$$\mathbb{S}' = \left[s'_{i,j}\right]_{4\times 4} \quad \text{gdzie} \quad s'_{i,j} = (\mathrm{F1} \otimes s_{i,j}^{-1} \oplus 63) \bmod 101. \tag{2.12}$$

Transformacja SubBytes, podobnie do transformacji MixColumns, nie jest transformacją samoodwrotną i w związku z tym deszyfrowanie jest możliwe jeśli zamienione zostaną kolejności operacji wyliczenia odwrotności multiplikatywnej i przekształcenia afinicznego.

W rzeczywistych implementacjach transformacja SubBytes jest rzadko implementowana według zależności (2.12) ze względu na stosunkowo dużą złożoność operacji obliczania odwrotności multiplikatywnej. Pozostałe operacje mogą być zrealizowane za pomocą operacji logicznej `XOR` — przekształcenie afiniczne jest wówczas realizowane jako `XOR` wybranych bitów odwrotności multiplikatywnej. Cała transformacja



Rysunek 2.5: Transformacja SubBytes

może być zapisana w postaci macierzowej:

$$s'_{i,j} = (\mathrm{F1} \otimes s_{i,j}^{-1} \oplus 63) \bmod 101 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} s_{i,j}^{-1\,(7)} \\ s_{i,j}^{-1\,(6)} \\ s_{i,j}^{-1\,(5)} \\ s_{i,j}^{-1\,(4)} \\ s_{i,j}^{-1\,(3)} \\ s_{i,j}^{-1\,(2)} \\ s_{i,j}^{-1\,(1)} \\ s_{i,j}^{-1\,(0)} \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}. \tag{2.13}$$

Ze względu na stosunkowo dużą złożoność obliczeniową algorytmu wyznaczania odwrotności multiplikatywnej większość rzeczywistych implementacji AES tablicuje jej wartości. Rozwiązanie takie standardowo wymaga tablicy o rozmiarze 256×8 bitów (usprawnienia są również możliwe jeśli operacje wykonywane w ciele $\mathrm{GF}(2^8)$ zastąpi się operacjami w innych ciałach, np.: $\mathrm{GF}(\left(2^4\right)^2)$ [77]). Tablicowanie transformacji pozwala na przyspieszenie jej wykonania kosztem zwiększenia rozmiarów wymaganej pamięci. Konieczność przechowywania dwóch tablic Sbox — jednej wykorzystywanej do szyfrowania, drugiej do deszyfrowania, powoduje dodatkowe zwiększenie wymagań pamięciowych.

Wady i zalety algorytmu AES

Z perspektywy bezpieczeństwa niewątpliwą zaletą algorytmu AES jest przejrzystość zasad stanowiących jego podstawę. To co w przypadku DES było dane "z góry" (struktura tablic Sbox) w przypadku AES ma dobrze znane i przebadane podłoże matematyczne, które pozwala na formalną analizę całego algorytmu. Jak zostanie pokazane w dalszej części pracy, pozwala ono również na stosowanie algorytmów kontrolnych łatwo integrowalnych z algorytmem szyfrowania i stosunkowo prostych do analizy.

Od strony implementacyjnej jedną z istotnych zalet algorytmu AES jest jego łatwa adaptowalność do różnych architektur μP oraz układów sprzętowych. W praktyce wszystkie transformacje i operacje wykonywane w algorytmie mogą być zrealizowane za pomocą prostych operacji 8-bitowych i przeszu-

kiwania tablic. Pozwala to na implementację algorytmu zarówno na prostych μP kart inteligentnych jak i w procesorach 32 i 64-bitowych. Różne sposoby implementacji pozwalają na tworzenie zarówno rozwiązań wolnych, o niskich wymaganiach pamięciowych, jak i szybkich, lecz wymagających znacznych rozmiarów pamięci. W literaturze można znaleźć szereg różnych implementacji algorytmu AES od rozwiązań niemal w 100% bazujących na tablicach (pozwalających uzyskać duże przepustowości, lecz wymagających znacznych narzutów implementacyjnych) do rozwiązań bazujących na operacjach arytmetycznych (wymagających niewielkich pamięci, ale za to wolniejszych).

2.2 Algorytmy asymetryczne

2.2.1 Algorytm RSA

Algorytm RSA, zaproponowany w 1978 roku przez Ronalda Rivesta, Adi Shamira i Leonarda Adelmana [73], jest asymetrycznym algorytmem szyfrującym i schematem podpisów. Bezpieczeństwo obu algorytmów wynika z trudności faktoryzacji dużych liczb całkowitych (ang. *integer factorisation*).

Definicja 2.1 (Faktoryzacja liczb całkowitych)**.** Niech N oznacza dodatnią liczbę całkowitą. Faktoryzacja liczby N polega na zapisaniu jej w postaci iloczynu

$$N = \prod_i p_i^{e_i},$$

gdzie p_i są liczbami pierwszymi, a $e_i \geq 1$.

Zaletą algorytmu RSA jest istnienie dwóch kluczy: prywatnego (ang. *private key*) oraz publicznego (ang. *public key*). Informacje zaszyfrowane jednym kluczem mogą być odszyfrowane tylko przy użyciu drugiego klucza z tej samej pary (stąd nazwa algorytmy asymetryczne). Dodatkowo, wyznaczenie dowolnego klucza na podstawie znajomości drugiego z nich jest trudne i wiąże się koniecznością wyznaczenia rozkładu dużej liczby na czynniki pierwsze. Właściwość ta pozwala na udostępnienie klucza publicznego bez narażania bezpieczeństwa klucza prywatnego i całego algorytmu rozwiązując tym samym problem dystrybucji klucza, tak istotny w przypadku algorytmów symetrycznych. Dzięki właściwościom algorytmu RSA każdy, kto zna klucz publiczny, ma możliwość zaszyfrowania wiadomości. Wiadomość tą będzie można jednak odczytać tylko przy użyciu klucza prywatnego z tej samej pary.

Szyfrowanie wiadomości za pomocą klucza prywatnego pozwala natomiast wygenerować szyfrogram, który będzie mógł być odszyfrowany przez dowolną osobę posiadającą klucz publiczny. Umożliwia to odbiorcy weryfikację tożsamości autora wiadomości i tworzenie podpisów cyfrowych — użytkownik wysyłając wiadomość m dołącza do niej jej szyfrogram RSA wygenerowany z użyciem swojego klucza prywatnego.

Odbiorca może użyć klucza publicznego, odszyfrować szyfrogram i porównać uzyskany wynik z przesłaną wiadomością. Zgodność obu elementów potwierdza tożsamość nadawcy.

Generacja kluczy, szyfrowanie i deszyfrowanie RSA

Kluczem prywatnym i publicznym algorytmu RSA są pary liczb $\langle e, N\rangle$ oraz $\langle d, N\rangle$ w których $N = pq$ jest iloczynem dwóch dużych liczb pierwszych p i q, e jest liczbą losową względnie pierwszą z $\phi(N) = (p-1)(q-1)$ a d jej odwrotnością modulo $\phi(N)$ (Alg. 2.1). Warto zauważyć, że wyznaczenie jednego

Algorytm 2.1 Algorytm generacji kluczy RSA

Wyjście: para kluczy prywatny $\langle e, N\rangle$ i publiczny $\langle d, N\rangle$

1: wybierz losowe liczby pierwsze p, q takie, że $p, q > 2^{511}$,

2: wyznacz $N = pq$ oraz $\phi(N) = (p-1)(q-1)$,

3: wylosuj $e < n$ takie, że $\gcd(e, \phi(N)) = 1$,

4: oblicz d spełniające równanie $ed \bmod \phi(N) = 1$,

5: usuń liczby p, q.

klucza na podstawie drugiego z nich (bez znajomości liczb p i q), wymaga dokonania faktoryzacji liczby N. Po wygenerowaniu kluczy i usunięciu liczb p oraz q, wykonanie kroku 3 wymaga ponownego odszukania p, q, co z kolei wymaga faktoryzacji liczby N.

Szyfrowanie wiadomości $m < N$ algorytmem RSA wykonywane jest z użyciem klucza publicznego $\langle e, N\rangle$ i polega na wykonaniu potęgowania m^e modulo N.

Algorytm 2.2 Algorytm szyfrowania RSA

Wejście: wiadomość $m < n$, klucz prywatny $\langle e, N\rangle$

Wyjście: szyfrogram c wiadomości m

1: oblicz $c = m^e \bmod N$.

W analogiczny sposób wykonywane jest deszyfrowanie kryptogramu c z tą jednak różnicą, że wykorzystywany jest klucz prywatny (d, N) — $c^d \bmod N$. Ponieważ $ed \bmod \phi(N) = 1$ to deszyfrowanie szyfrogramu

Algorytm 2.3 Algorytm deszyfrowania RSA

Wejście: szyfrogram $c < N$, klucz publiczny $\langle d, N\rangle$

Wyjście: wiadomość m

1: oblicz $m = c^d \bmod N$.

c rzeczywiście daje w wyniku wiadomość m, mamy bowiem:

$$c^d \bmod N \quad = \quad (m^e)^d \bmod n = m^{ed} \bmod N = m \bmod N = m. \tag{2.14}$$

Szyfrowanie i deszyfrowanie podstawowym algorytmem RSA polega więc na wyznaczeniu potęgi wiadomości/kryptogramu modulo N. Jednym z najszybszych i zarazem prostych algorytmów takiego potęgowania jest algorytm *square and multiply* (Alg. 2.4). Jego zaletą jest mała złożoność obliczeniowa wynosząca $O(log_2^3 n)$ oraz prostota i łatwość implementacji. Niestety, jak zostanie pokazane w dalszej części pracy, algorytm ten jest bardzo podatny na ataki kryptograficzne.

Algorytm 2.4 Potęgowanie *square and multiply*

Wejście: $a < N$ oraz $x < \phi(N)$ którego binarna reprezentacja ma postać $x = \sum_{i=0}^{n} 2^i x_i$ gdzie $x_i = \{0, 1\}$.

Wyjście: $y = a^x \bmod N$

1: $y \leftarrow 1$,
2: **if** $e = 0$ **then** return(y),
3: $A \leftarrow a$,
4: **if** $x_0 = 1$ **then** $y \leftarrow a$,
5: **for** $i = 1$ to n **do**
6: $\quad A \leftarrow A^2 \bmod N$,
7: $\quad$ **if** $x_i = 1$ **then** $y \leftarrow a \cdot y \bmod N$,
8: **end for**

Innym sposobem implementacji RSA jest wykorzystanie chińskiego twierdzenia o resztach (ang. *Chinese Remainder Theorem - CRT*), które pozwala wykonać osobno obliczenia modulo p i q a wynik modulo N obliczyć poprzez wykonanie konwersji odwrotnej.

Definicja 2.2 (Chińskie twierdzenie o resztach)**.** Dla zbioru liczb całkowitych $n_1, n_2, \ldots n_t > 0$, parami względnie pierwszych (tj. $\gcd(n_i, n_j) = 1$ dla $i \neq j$), układ kongurencji:

$$\begin{aligned} a_1 &= x \bmod n_1 \\ a_2 &= x \bmod n_2 \\ &\vdots \\ a_t &= x \bmod n_t, \end{aligned}$$

ma jednoznaczne rozwiązanie x modulo $N = \prod_{i=1}^{t} n_i$. Rozwiązanie to może zostać wyznaczone jako

$$x = \sum_{i=1}^{t} a_i N_i M_i \bmod N \quad \text{gdzie } N_i = \prod_{j=1, j\neq i}^{t} n_i,\ M_i = N_i^{-1} \bmod n_i. \tag{2.15}$$

Korzystając z CRT oraz tego, że w RSA obliczenia są prowadzone modulo $N = pq$ wynika możliwość prowadzenia operacji szyfrowania oddzielnie dla każdego z czynników p, q. Uzyskane w ten sposób

Algorytm 2.5 Algorytm szyfrowania RSA wykorzystujący CRT

Wejście: wiadomość $m < N$, klucz prywatny $\langle e, N\rangle$, tajne liczby pierwsze p, q

Wyjście: szyfrogram c

1: oblicz $c_p = m^{e_p} \bmod p$,

2: oblicz $c_q = m^{e_q} \bmod q$,

3: oblicz $c = CRT(c_p, c_q) = \left[c_p q \left(q^{-1} \bmod p\right) + c_q p \left(p^{-1} \bmod q\right)\right] \bmod N$.

szyfrogramy można następnie złożyć zgodnie z algorytmem konwersji odwrotnej. Korzystając z tej obserwacji szyfrowanie RSA może przebiegać według algorytmu (Alg. 2.5). Zaletą wykorzystania CRT jest zmniejszenie złożoności obliczeniowej całego algorytmu i około czterokrotne przyspieszenie jego działania. Przyspieszenie to wynika z redukcji wykładników potęg do których podnoszona jest wiadomość m — są to wykładniki $e_p = e \bmod (p-1)$ i $e_q = e \bmod (q-1)$. Pewną wadą algorytmu jest konieczność przechowywania wartości p i q, których ujawnienie zagraża bezpieczeństwu algorytmu RSA. Z tego samego powodu rozwiązania tego nie można stosować do szyfrowania wiadomości za pomocą kluczy publicznych.

Algorytm RSA może być wykorzystany zarówno do zapewniania poufności komunikacji — szyfrowania przesyłanych danych, jak i potwierdzenia tożsamości autora wiadomości — składania podpisów cyfrowych. W pierwszym przypadku autor wiadomości szyfruje ją kluczem publicznym odbiorcy. Rozwiązanie to zapewnia, że tylko adresat może odszyfrować i przeczytać wiadomość. W przypadku schematu podpisów cyfrowych, autor wiadomości dołącza do nie jej szyfrogram RSA wygenerowany za pomocą swojego klucza prywatnego. Szyfrogram taki może być odszyfrowany przez każdego odbiorcę, który w ten sposób potwierdza tożsamość nadawcy. Ze względu na bezpieczeństwo podpisów cyfrowych jak i dużą złożoność procedur kryptografii asymetrycznej, podpisem najczęściej jest szyfrogram funkcji skrótu (ang. *hash function*) wyznaczonej dla podpisywanej wiadomości.

Definicja 2.3 (Funkcja skrótu)**.** Niech x oznacza ciąg bitów o dowolnej długości. Funkcja skrótu

$$H(x) : \{0,1\}^* \rightarrow \{0,1\}^k \tag{2.16}$$

przekształca x w ciąg bitowy o stałej długości k.

Kryptograficzna funkcja skrótu, musi ponadto posiadać szereg dodatkowych właściwości zapewniających bezpieczeństwo

1. jednokierunkowość (ang. *one-way*) — oznaczająca, że mając daną wartość funkcji $H(x)$ nie można wyznaczyć jej argumentu x,

2. słaba odporność na kolizje (ang. *weak collision resistance*) — oznaczająca, że jest obliczeniowo

niewykonalne odszukanie takiego argumentu x_2 dla którego wartość $H(x_2)$ jest równa $H(x_1)$ dla znanego $x_1 \neq x_2$,

3. silna odporność na kolizje (ang. *strong collision resistance*) — oznaczająca, że jest obliczeniowo niewykonalne odszukanie dwóch takich $x_1 \neq x_2$ dla których wartości funkcji skrótu są takie same, tj. $H(x_1) = H(x_2)$.

2.2.2 Podpisy cyfrowe ElGamala

Kolejnym algorytmem analizowanym w pracy jest schemat podpisów cyfrowych ElGamala zaproponowany w 1985 roku przez Tahera ElGamala [33]. Podpisy te różnią się znacząco od podpisów RSA zarówno pod względem podstaw teoretycznych, na których bazuje ich bezpieczeństwo jak i możliwych zastosowań. Bezpieczeństwo tych podpisów wynika z trudności obliczania logarytmów dyskretnych.

Definicja 2.4 (Logarytm dyskretny)**.** Niech p oznacza liczbę pierwszą, g będzie generatorem grupy multiplikatywnej $\mathbb{Z}_p^*$ zaś a elementem tej grupy. Dyskretnym logarytmem liczby a o podstawie g nazywamy liczbę $0 \leq x \leq n-1$ taką, że $g^x \bmod p = a$.

W schemacie ElGamala operacje prowadzące do wyznaczenia podpisu prowadzone są w grupie multiplikatywnej $\mathbb{Z}_p^*$, gdzie p jest dużą liczbą pierwszą, mającą od 768 do 1024 bitów. Podobnie jak w RSA i tu istnieje para kluczy: prywatny $\langle p, g, a\rangle$ (często pojęciem klucz prywatny określa się sam element a tego klucza, ze względu na to, że pozostałe dwa elementy są publicznie znane) i publiczny $\langle p, g, y\rangle$. Różnica polega jednak na tym, że klucze te nie mogą być stosowane zamiennie (w RSA dowolny klucz z pary mógł być kluczem publicznym). Generacja kluczy dla podpisów ElGamala przebiega według algorytmu (Alg. 2.6) a uzyskany w ten sposób klucz prywatny może zostać użyty do złożenia podpisu pod wiadomością m. Generacja podpisu wykorzystuje funkcję skrótu $h(x) : \{0,1\}^* \to \mathbb{Z}_p^*$ i przebiega zgodnie

Algorytm 2.6 Generacja kluczy dla schematu podpisów ElGamala

Wyjście: para kluczy: prywatny a — publiczny $\langle p, g, y\rangle$

1: wylosuj liczbę pierwszą p, co najmniej 768-bitową, i znajdź generator g grupy $\mathbb{Z}_p^*$,

2: wylosuj liczbę a, $1 < a \leq p-2$,

3: oblicz $y = g^a \bmod p$.

z algorytmem (Alg. 2.7). Algorytmy (Alg. 2.7) i (Alg. 2.8) przedstawiają podstawowy schemat podpisu ElGamala. Istnieje jednak jego pięć odmian różniących się czwartym krokiem algorytmu podpisu, nazywanym równaniem podpisu (ang. *signing equation*) oraz trzecim krokiem algorytmu weryfikacji.

Równanie podpisu ma ogólną postać $u = av + kw \bmod (p-1)$ gdzie za symbole u, v, w podstawiane są zmienne $h(m), r$ i s. Odpowiednio do przypisania zmiennych mamy do czynienia z jedną z sześciu

Algorytm 2.7 Generacja podpisu ElGamala

Wejście: wiadomość m, klucz prywatny a

Wyjście: podpis $\langle r, s\rangle$

1: wylosuj liczbę pierwszą k taką, że $1 \leq k \leq p-2$ i $gcd(k, p-1) = 1$,

2: oblicz $r = g^k \bmod p$,

3: oblicz $k^{-1} \bmod (p-1)$,

4: oblicz $s = k^{-1}\left(h(m) - ar\right) \bmod (p-1)$.

Algorytm 2.8 Weryfikacja podpisu ElGamala

Wejście: wiadomość m, klucz publiczny $\langle p, g, y\rangle$, podpis $\langle r, s\rangle$

Wyjście: **true** jeśli podpis jest poprawny, **false** w przeciwnym przypadku

1: **if** not $(0 < r < p)$ **then** return(**false**),

2: oblicz $h(m)$,

3: oblicz $v_1 = g^{h(m)} \bmod p$ i $v_2 = y^r r^s \bmod p$,

4: **if** $v_1 = v_2$ **then** return(**true**),

5: return(**false**).

wersji algorytmu ElGamala (tab. 2.1). Wersje te mają odmienne właściwości i cechują się różną złożonością obliczeniową oraz podatnością na ataki z uszkodzeniami [9]. Wymaganiem algorytmu ElGamala jest wykorzystywanie do podpisywania losowej liczby k. Gwarantuje ona, że z prawdopodobieństwem bliskim 1, dwa podpisy pod tą samą wiadomością będą różne (dla $1 \leq k \leq p-2$ losowanego z rozkładu równomiernego prawdopodobieństwo to wynosi $1 - 1/(p-2)$).

Lemat 2.1. Niech $\langle r_i, s_i\rangle$ będzie podpisem ElGamala pod wiadomością m wygenerowanym z użyciem liczby losowej k_i. Prawdopodobieństwo, że dwa podpisy ElGamala $\langle r_1, s_1\rangle$ i $\langle r_2, s_2\rangle$ pod wiadomością m są takie same, wynosi $1 - 1/(p-2)$. Ponadto, jeśli oba podpisy są takie same, to $k_1 = k_2$.

Tabela 2.1: Postaci równania podpisu dla różnych wersji algorytmu ElGamala

	u	v	w	Równanie podpisu	Weryfikacja $v_1 = v_2$
1	$h(m)$	r	s	$h(m) = ar + ks$	$g^{h(m)} = y^r r^s$
2	$h(m)$	s	r	$h(m) = as + kr$	$g^{h(m)} = y^s r^r$
3	s	r	$h(m)$	$s = ar + kh(m)$	$g^s = y^r r^{h(m)}$
4	s	$h(m)$	r	$s = ah(m) + kr$	$g^s = y^{h(m)} r^r$
5	r	s	$h(m)$	$r = as + kh(m)$	$g^r = y^s r^{h(m)}$
6	r	$h(m)$	s	$r = ah(m) + ks$	$g^r = y^{h(m)} r^s$

Dowód. Zauważmy, że równość obu podpisów oznacza, że $r_1 = r_2$ i $s_1 = s_2$. Ponieważ $r_i = g^{k_1} \bmod p$ gdzie g jest generatorem $\mathbb{Z}_p^*$, to pierwsza z równości zajdzie wtedy i tylko wtedy gdy $k_1 = k_2$. Prawdopodobieństwo, że losowo wybrane liczby k_1 i k_2 są sobie równe wynosi zaś $1/(p-2)$. □

Tajność losowej liczby k zapewnia bezpieczeństwo algorytmu ElGamala — ujawnienie k dla dowolnego podpisu $\langle r, s\rangle$ pod znaną wiadomością m pozwala atakującemu na rozwiązanie równania podpisu ze względu na tajny klucz a. Dla bezpieczeństwa schematu podpisów ważne jest również zagwarantowanie losowości liczb k używanych w kolejnych podpisach. Jeśli ta sama liczba k zostanie użyta do generacji dwóch podpisów $\langle r_1, s_1\rangle$ i $\langle r_2, s_2\rangle$ pod wiadomościami m_1 i m_2, wówczas atakujący może z podpisów tych odzyskać użyte k i rozwiązać równanie podpisu. Możliwość ta wynika stąd, że dla tych samych liczb k pierwsze elementy obu podpisów są takie same $r_1 = r_2$. Dalsza analiza równań podpisu dla obu wiadomości pozwala odszukać k

$$s_1 = k^{-1}\left(h(m_1) - ar\right) \bmod (p-1) \tag{2.17}$$

$$s_2 = k^{-1}\left(h(m_2) - ar\right) \bmod (p-1). \tag{2.18}$$

Odejmując stronami równania (2.17) i (2.18) otrzymamy bowiem:

$$s_1 - s_2 = k^{-1}\left(h(m_1) - ar - h(m_2) + ar\right) \bmod (p-1) = k^{-1}\left(h(m_1) - h(m_2)\right) \bmod (p-1)$$

i jeśli tylko $\gcd(s_1 - s_2, p-1) = 1$, to k można wyliczyć jako:

$$k = (s_1 - s_2)^{-1}\left(h(m_1) - h(m_2)\right) \bmod (p-1). \tag{2.19}$$

Wadą schematu ElGamala jest natomiast duży rozmiar generowanego podpisu, który składa się z dwóch liczb, każda mniejsza od p (w RSA podpis jest jedną liczbą). Inną trudnością algorytmu generacji podpisu ElGamala jest konieczność wylosowania k względnie pierwszego z $p-1$ (Alg. 2.7 krok 1). Spełnienie tego warunku jest konieczne ze względu na uzależnienie s od odwrotności multiplikatywnej $k^{-1} \bmod (p-1)$, która istnieje tylko jeśli $\gcd(k, p-1) = 1$.

Lemat 2.2. Mamy daną liczbę pierwszą p i liczbę losową $1 \leq k \leq p-2$ losowaną z rozkładu równomiernego. Prawdopodobieństwo wylosowania liczby k takiej, że $\gcd(k, p-1) = 1$ jest nie większe niż $1/2$.

Dowód. Liczba liczb $1 \leq k \leq p-2$ względnie pierwszych z $p-1$ może być wyznaczona z funkcji Eulera

$$\phi(p-1) = (p-1) \prod_{p_i | (p-1)} \left(1 - \frac{1}{p_i}\right), \tag{2.20}$$

gdzie p_i są dzielnikami pierwszymi $p-1$. Ponieważ $p-1$ jest liczbą parzystą, to $p_1 = 2|(p-1)$ i liczba liczb względnie pierwszych jest równa

$$\phi(p-1) = \frac{p-1}{2} \prod_{p_i|(p-1),\ p_i>2} \left(1 - \frac{1}{p_i}\right) \leq \frac{p-1}{2}. \tag{2.21}$$

Prawdopodobieństwo, że losowo wybrana liczba k jest względnie pierwsza z $p-1$ jest więc równe

$$P\left(\gcd(k, p-1) = 1\right) \leq \frac{\frac{p-1}{2}}{p-1} = \frac{1}{2}. \tag{2.22}$$

To kończy dowód. □

Prawdopodobieństwo wylosowania właściwej liczby k można zwiększyć stosując odpowiednie procedury losowania. Najprostszym rozwiązaniem jest pomijanie w losowaniu liczb parzystych. Może to być zrealizowane poprzez losowanie liczb $1 \leq k_1 \leq (p-1)/2$ i wyznaczanie $k = 2k_1 - 1$. Wylosowane w ten sposób k na pewno nie jest podzielne przez 2. Tak więc prawdopodobieństwo, że k jest względnie pierwsze z $p-1$, jest dwukrotnie większe.

Inne wersje algorytmu ElGamala posiadają podobne właściwości jednak różnią się nieznacznie złożonością obliczeniową. W przypadku wersji 2 i 5 (tab. 2.1) zamiast obliczania odwrotności liczby losowej k wyznaczana jest odwrotność klucza prywatnego a. Odwrotność ta jest stała, przez co może być wyliczona raz i wykorzystywana we wszystkich kolejnych obliczeniach. Rozwiązanie takie upraszcza również algorytm podpisów, eliminując potrzebę obliczania odwrotności modulo $p-1$ na bieżąco oraz losowania odpowiedniego k.

2.2.3 Podpisy cyfrowe DSA

Podpisy DSA (ang. *Digital Signature Algorithm*) są odmianą podpisów ElGamala. Różnica pomiędy tymi schematami polega na tym, że operacje generowania i weryfikacji podpisów DSA są wykonywane w podgrupie rzędu q grupy multiplikatywnej $\mathbb{Z}_p^*$, gdzie p i q są liczbami pierwszymi i $q|(p-1)$. Ponadto w podpisach DSA wykorzystywana jest funkcja skrótu $H(x) : \{0,1\}^* \rightarrow \mathbb{Z}_q$. W 1994 roku odmiana schematu podpisów DSA, nazywana standardem podpisów cyfrowych DSS (ang. *Digital Signature Standard*), została w USA uznana za standard podpisów cyfrowych. Jedyna różnica pomiędzy DSA i DSS polega na tym, że DSS wymaga użycia funkcji skrótu SHA-1 (ang. *Secure Hash Algorithm 1*) podczas gdy, DSA pozwala na pełną dowolność w tym zakresie. Generacja i weryfikacja podpisów DSA przebiega w sposób bardzo zbliżony do schematów ElGamala. Zgodnie z algorytmem generacji kluczy dla schematu DSA (Alg. 2.9) liczba pierwsza p należy do przedziału $\left(2^{511}, 2^{1024}\right)$ a jej zalecany rozmiar to 768 bitów. Liczba q natomiast jest zawsze liczbą 160-bitową.

Algorytm 2.9 Generacja kluczy dla schematu podpisów DSA

Wyjście: para kluczy publiczny a — prywatny $\langle p, q, q, y\rangle$

1: wylosuj 160-bitową liczbę pierwszą q ($2^{159} < q < 2^{160}$),

2: wylosuj $0 \leq t \leq 8$ i liczbę pierwszą p taką, że $2^{511+64t} < p < 2^{512+64t}$ i $q|(p-1)$,

3: wylosuj generator g grupy cyklicznej rzędu q w $\mathbb{Z}_p^*$,

4: wylosuj klucz prywatny a, $1 \leq a \ \leq q-1$,

5: oblicz $y = g^a \bmod p$.

Algorytm 2.10 Generacja podpisu DSA

Wejście: wiadomość m, klucz prywatny a

Wyjście: podpis $\langle r, s\rangle$

1: wylosuj liczbę pierwszą k taką, że $1 \leq k \leq q-1$ i $\gcd(k, q) = 1$,

2: oblicz $r = \left(g^k \bmod p\right) \bmod q$,

3: oblicz $k^{-1} \bmod q$,

4: oblicz $s = k^{-1}\left(H(m) + ar\right) \bmod q$.

Algorytm 2.11 Weryfikacja podpisu DSA

Wejście: wiadomość m, klucz publiczny $\langle p, q, g, y\rangle$, podpis $\langle r, s\rangle$

Wyjście: **true** jeśli podpis jest poprawny, **false** w przeciwnym przypadku

1: **if** not $(0 < r < q)$ or not $(0 < s < q)$ **then** return(**false**),

2: oblicz $w = s^{-1} \bmod q$ i $h(m)$,

3: oblicz $v_1 = wH(m) \bmod q$ i $v_2 = rw \ \bmod q$,

4: **if** $g^{v_1} y^{v_2} = r$ **then** return(**true**),

5: return(**false**).

Zaletą algorytmu DSA, w porównaniu do ElGamala, jest mniejsza złożoność obliczeniowa procedur składania i weryfikacji podpisu. Wynika to stąd, że większość operacji podpisywania i wszystkie operacje weryfikacji podpisu wykonywane są modulo q, którego rozmiar bitowy jest około pięciokrotnie mniejszy niż rozmiar p. Mniejszy rozmiar bitowy powoduje przyspieszenie obliczeń wykonywanych w dwóch ostatnich krokach algorytmu a przez to skrócenie czasu działania algorytmu. Dodatkową zaletą wykonywania obliczeń w podgrupie $\mathbb{Z}_p^*$ rzędu q jest to, że wszystkie losowe liczby k są względnie pierwsze z q. Stąd w procedurze generowania podpisu nie ma żadnych warunków i ewentualnych zapętleń. Algorytm DSA nie ma też wad algorytmu ElGamala wynikających z operowania w pełnej grupie multiplikatywnej [13] zapewniając odporność na część ataków wykorzystujących tzw. boczne wejścia (ang. *trap door*).

Rozdział 3

Ataki typu side-channel

Ataki typu side-channel zostały zaproponowane w drugiej połowie lat 90. XX wieku. Są one przedmiotem badań nowej i ciągle rozwijanej dziedziny kryptologii, w której analizuje się algorytmy kryptograficzne zarówno z punktu widzenia matematyki jak i ich rzeczywistych implementacji programowych i sprzętowych. Prowadzone analizy pokazały, że wcześniejsze badania algorytmów kryptograficznych pod kątem formalnego bezpieczeństwa są niewystarczające ze względu na przyjmowany w nich wyidealizowany model otoczenia. Zauważono wówczas, że matematyczna ocena bezpieczeństwa algorytmów, zakładająca istnienie określonych rodzajów zagrożeń, jest niekompletna i nie odwzorowuje istotnych elementów świata rzeczywistego. Przyczyną problemów okazało się być powszechnie przyjmowane założenie, że układy kryptograficzne bezbłędnie realizują zaimplementowane algorytmy, oraz, że dane nie ulegają przekłamaniu, w czasie ich wykonania.

Na przełomie 1996 i 1997 w pracach Bihama i Shamira [11], Boneha [17, 18] oraz Bao [4] pokazano konsekwencje jakie mogą wynikać z pojawienia się błędów w trakcie obliczeń. W artykułach tych pokazano, że kryptografia jest nauką interdyscyplinarną, w której sama znajomość matematyki może być niewystarczająca do oceny uzyskiwanego poziomu bezpieczeństwa. Od tej pory algorytmy kryptograficzne przestały być traktowane jako czarne skrzynki (ang. *black box*), realizujące złożone algorytmy ściśle według ich definicji, a zaczęły być postrzegane jako urządzenia oddziałujące z otoczeniem. Ten sposób postrzegania spowodował rewolucję w kryptoanalizie. Oprócz istniejących już metod ataku (atak liniowy, różnicowy, urodzinowy, itd.) atakujący pozyskali dodatkowe możliwości analizy przebiegu algorytmu a nawet wpływania na jego wyniki przez oddziaływanie na urządzenie. Nowe możliwości obejmują:

- pomiar parametrów urządzenia w czasie wykonywania obliczeń kryptograficznych, m.in.:
  - pomiar czasu wykonania operacji kryptograficznej,
  - pomiar poboru mocy przez urządzenie,

  - pomiar pól elektromagnetycznych wytwarzanych przez urządzenie,

- oddziaływanie na urządzenie w czasie wykonywania obliczeń poprzez:
  - zmiany napięcia zasilania,
  - zniekształcenia sygnału zegarowego taktującego urządzenie,
  - bombardowanie układu cząstkami naładowanymi i naświetlanie promieniowaniem rentgenowskim lub podczerwonym,
  - oddziaływanie polem elektromagnetycznym.

Wszystkie tego typu ataki są określane w literaturze anglojęzycznej mianem *side-channel attacks*. Ich idea polega na wykorzystaniu w kryptoanalizie informacji na temat działania urządzenia pozyskanych z obserwacji parametrów jego pracy i/lub oddziaływania na urządzenie.

Istniejące podziały ataków side-channel uwzględniają sposoby ingerencji w urządzenie, metodę oddziaływania na urządzenie oraz procedury analizy uzyskanych wyników. Ze względu na sposób ingerencji w urządzenie wyróżniamy trzy grupy ataków:

1. ataki nieinwazyjne (ang. *non-invasive attacks*), w których atakujący nie ma bezpośredniego, fizycznego dostępu do wewnętrznych elementów elektrycznych układu realizującego operacje kryptograficzne a jedynie może mierzyć parametry jego działania, obserwować jego wejścia i wyjścia oraz oddziaływać za pomocą pól elektromagnetycznych czy zakłóceń napięcia zasilania lub sygnału taktującego,

2. ataki inwazyjne (ang. *invasive attacks*), w których atakujący ma możliwość odsłonięcia i bezpośredniego dostępu do wewnętrznych elementów elektrycznych układu (m.in. szyn danych, linii sterujących, komórek pamięci),

3. ataki "półinwazyjne" (ang. *semi-invasive attacks*), w których atakujący nie ma bezpośredniego dostępu do wewnętrznych elementów elektrycznych układu, ale może usunąć z układu wszystkie warstwy ochronne uzyskując dostęp do układu μP.

Zależnie od metody oddziaływania na urządzenie ataki typu side–channel można podzielić na dwie grupy:

1. ataki pasywne (ang. *passive attacks*), w których atakujący jedynie analizuje działanie układu kryptograficznego podając dane na wejścia i odczytując dane z wyjść, bez ingerencji w jego działanie,

2. ataki aktywne (ang. *active attacks*), w których atakujący oprócz zadawania danych wejściowych oddziaływuje na urządzenie w inny sposób (np.: zakłócając napięcie zasilania, sygnał zegarowy, przykładając pola elektromagnetyczne) i analizuje jego reakcję.

Ostatni podział różnicuje ataki z uszkodzeniami ze względu na metodę analizy wyników uzyskanych z obserwacji i ewentualnego oddziaływania na urządzenie. Wyróżniamy dwa rodzaje analizy wyników:

- analiza bezpośrednia (ang. *simple analysis*), w której wykorzystywane są kolejno pojedyncze wyniki uzyskane z obserwacji układu. Gdy w ataku wykorzystywanych jest wiele wyników, każdy z nich jest analizowany oddzielnie, a rezultatem analizy jest uzyskanie częściowej wiedzy o działaniu urządzenia i stosowanym kluczu kryptograficznym,

- analiza różnicowa (ang. *differential analysis*), w której jednocześnie analizowane są pary, trójki lub większe liczby uzyskanych wyników. Zależnie od algorytmu i konkretnego ataku analizie podlegają tylko wyniki błędnego działania algorytmu albo pary błędny–poprawny wynik. W algorytmach symetrycznych analiza różnicowa sprowadza się do takiego samego postępowania jak podczas tradycyjnego ataku różnicowego. W algorytmach asymetrycznych natomiast, analiza polega na wyznaczaniu i porównywaniu różnic między grupami wyników w celu uzyskania informacji o działaniu algorytmu i używanym kluczu kryptograficznym.

Niezależnie od zastosowanej metody ataku, analiza układu kryptograficznego dostarcza dodatkowych informacji na temat działania urządzenia. Co więcej, przebieg algorytmu i parametry pracy urządzenia zależą od używanego klucza kryptograficznego i przetwarzanej wiadomości. Daje to możliwość obserwacji zmian tych parametrów dla różnych kluczy i wiadomości, dostarczając atakującemu dodatkowych informacji o urządzeniu, algorytmie i kluczu. W konsekwencji urządzenie kryptograficzne nie są określane mianem "czarna skrzynka", lecz "szara skrzynka" (ang. *grey box*). Określenie to ma sugerować, że atakujący ma znacznie więcej informacji na temat algorytmu kryptograficznego niż zakładały, stosowane dotychczas, modele matematyczne. W konsekwencji atakujący zyskuje dodatkowe możliwości analizowania działania urządzenia i pozyskiwania tajnych informacji wykorzystywanych do realizacji operacji kryptograficznych.

3.1 Pasywne ataki side–channel

Do pasywnych ataków typu side–channel zaliczamy wszystkie ataki, w których atakujący kontroluje tylko i wyłącznie wejścia urządzenia, podając na nie różne dane i obserwując jego wyjście. Najczęstsze metody ataku to pomiar poboru mocy, pomiar czasu działania i pomiar ulotu elektromagnetycznego. W wyniku każdego z nich atakujący otrzymuje charakterystykę działania urządzenia, pozwalającą wyróżnić etapy jego działania i wnioskować o stosowanym kluczu.

3.1.1 Analiza poboru mocy

Zaprezentowana przez Kochera [58] analiza poboru mocy (ang. *power analysis*) układów kryptograficznych jest jednym z najstarszych rodzajów pasywnych ataków side-channel. Możliwość jej zastosowania wynika ze zróżnicowania poboru mocy w zależności od aktualnego stanu całego układu jak i realizowanych operacji.

Zmiany poboru mocy, choć stosunkowo nieduże ze względu na niskie wartości napięć zasilania V_{CC}, mogą być łatwo mierzone przez pomiar spadku napięcia na niewielkich rezystorach R (około 50Ω) włączonych szeregowo w obwód napięcia zasilającego urządzenie. Pobór mocy jest następnie wyznaczany jako iloraz $P = V_{CC} \cdot \frac{\Delta V}{R}$. Najostrzejszym wymaganiem w całym ataku z pomiarem poboru mocy jest jakość miernika służącego do pomiaru napięcia. Powinien on cechować się dużą graniczną częstotliwością pracy, wysoką rozdzielczością i jak najmniejszym błędem wskazań. W wielu przypadkach wymaganie dotyczące dużej granicznej częstotliwości pracy może zostać złagodzone ze względu na to, że spora część układów kryptograficznych to karty mikroprocesorowe. Karty te są wyposażone w proste mikroprocesory taktowane wolnymi zegarami (rzędu kilkudziesięciu MHz) lub zewnętrznym sygnałem zegarowym. Zewnętrzny sygnał zegarowy pozwala atakującemu obniżyć szybkość działania urządzenia co ułatwia przeprowadzenie ataku.

Analiza poboru mocy jest dzielona na dwie grupy: bezpośrednią analizę poboru mocy SPA (ang. *Simple Power Analysis*) i różnicową analizę poboru mocy DPA (ang. *Differential Power Analysis*).

Bezpośrednia analiza poboru mocy

Idea analizy SPA, zaproponowana przez Kochera w pracy [58], polega na analizie zmian poboru mocy przez urządzenie i powiązaniu ich z operacjami przez nie realizowanymi. Atak taki może ujawnić sekwencję wykonywanych operacji, której znajomość pozwala wnioskować o kluczu kryptograficznym wykorzystanym w algorytmie.

Atak SPA może zostać wykorzystany do analizy algorytmów kryptograficznych, których wykonanie zależy od przetwarzanych danych. W szczególności atak ten można zastosować do:

- **rotacji i permutacji** wykonywanych m.in. w czasie szyfrowania i deszyfrowania algorytmem DES,
- **porównania**, którego rezultatem jest zawsze wykonanie różnych sekwencji instrukcji. Sekwencje te mogą być rozróżnione na charakterystyce SPA, jeśli mają różne charakterystyki poboru mocy,
- **mnożenia**, którego charakterystyka poboru mocy umożliwia uzyskanie informacji o operandach i ich wagach Hamminga,

- **potęgowania** realizowanego za pomocą algorytmów podobnych do *square and multiply*, w których kolejne mnożenia są warunkowane wartościami odpowiednich bitów wykładnika,
- **precyzyjnej lokalizacji poszczególnych kroków szyfrowania**, która jest niezbędna w wielu innych atakach, m.in. w atakach z uszkodzeniami.

Atak SPA jest najbardziej skuteczny w stosunku do algorytmów, w których część instrukcji jest wykonywana tylko przy spełnieniu określonych warunków. W takim przypadku na charakterystyce poboru mocy można zauważyć różnice w zależności od stanu warunku. Przykładem algorytmu, który może być zaatakowany za pomocą SPA jest potęgowanie metodą *square and multiply* (Alg. 2.4), w którym mnożenie jest wykonywane tylko, gdy kolejny bit wykładnika jest równy 1 ($x^{(i)} = 1$). Analiza SPA algorytmu *square and multiply* pozwala na wykrycie iteracji potęgowania, w których mnożenie było wykonane, pozwalając tym samym ustalić, które bity klucza są równe 1 a które 0.

Ponieważ warunkiem przeprowadzenia analizy SPA jest zauważalna zmiana charakterystyki poboru mocy, dlatego też atakom tego typu można dość łatwo zapobiec. Najprostsze metody ochrony to ujednolicenie poboru mocy poprzez wyeliminowanie operacji warunkowych lub ujednolicenie poboru mocy niezależnie od warunku (w przypadku algorytmu *square and multiply* możliwe jest np.: wykonywanie mnożenia za każdym przebiegiem pętli przy czym, jeśli kolejny bit wykładnika ma wartość 0, to rezultat mnożenia jest ignorowany).

Różnicowa analiza poboru mocy

Wad analizy SPA nie ma różnicowa analiza poboru mocy DPA [58], która polega na porównywaniu charakterystyk uzyskanych dla różnych danych. Dzięki wykorzystaniu wielu charakterystyk jednocześnie i analizie różnicowej atakujący ma możliwość zauważenia zmian wynikających ze zmiany danych wejściowych wprowadzonych do urządzenia. Jest to możliwe nawet wtedy, gdy zmiany te są rozmyte w charakterystyce poboru mocy sekwencyjnego wykonania wielu instrukcji.

Jednym z algorytmów do którego analizy zastosowano DPA jest algorytm DES. W algorytmie tym danymi wejściowymi do ostatniej rundy algorytmu są 32 bitowe bloki danych L_{16} i R_{16}. Blok R_{16} poddawany jest przekształceniu F, którego jednym z elementów są podstawienia realizowane za pomocą 8 tablic Sbox, których wyjście jest permutowane i dodawane do bloku L_{16} za pomocą operacji bitowej `XOR`. W analizie DPA atakujący dąży do odzyskania klucza ostatniej rundy algorytmu na podstawie znajomości szyfrogramów i charakterystyk poboru prądu. W tym celu definiowana jest funkcja $D(c, b, K_r)$, której argumentem jest szyfrogram c, indeks szukanego bitu danych b oraz możliwa wartość 6 bitów klucza rundy K_r odpowiadających tablicy Sbox, której wyjście zostało dodane do bitu $L_{16}^{(b)}$. Jeśli klucz K_r został odgadnięty poprawnie, to wartość funkcji jest równa rzeczywistej wartości szukanego bitu. W przeciwnym

przypadku funkcja przyjmuje wartości 0 albo 1 a prawdopodobieństwo, że uzyskany w ten sposób wynik będzie równy rzeczywistej wartości bitu $\mathrm{L}_{16}^{(b)}$, wynosi 1/2.

Rozpoczynając atak DPA, atakujący zbiera m charakterystyk poboru mocy przez urządzenie $T_i[1..k]$ zapamiętując jednocześnie odpowiadające im szyfrogramy c_i. Następnie dla każdego odgadywanego bitu $\mathrm{L}_{16}^{(b)}$ atakujący oblicza charakterystykę różnicową wszystkich charakterystyk T_i

$$\Delta_D[j] = \frac{\sum_{i=1}^{k} D(c_i, b, K_r) T_i[j]}{\sum_{i=1}^{k} D(c_i, b, K_r)} - \frac{\sum_{i=1}^{k} (1 - D(c_i, b, K_r))\, T_i[j]}{\sum_{i=1}^{k} (1 - D(c_i, b, K_r))}. \tag{3.1}$$

Pierwszy składnik we wzorze (3.1) jest średnią wszystkich charakterystyk T_i dla których funkcja D dla odpowiadającego im szyfrogramu c_i i odgadniętej wartość klucza K_r przyjęła wartość bitu $\mathrm{L}_{16}^{(b)}$ równą 1. Drugi składnik jest średnią wszystkich charakterystyk T_i dla których funkcja D wynosi 0.

Jeśli klucz K_r został odgadnięty niepoprawnie, to wówczas funkcja $D(c_i, b, K_r)$ zachowuje się jak funkcja losowa (z pr. 1/2 przyjmuje wartości równe i różne od rzeczywistej wartości szukanego bitu) nieskorelowana z rzeczywistymi wartościami bitu b. W takiej sytuacji prawdopodobieństwo każdej z możliwych wartości bitu $\mathrm{L}_{16}^{(b)}$ jest jednakowe, a różnica średnich powinna dążyć do zera. W praktyce oznacza to, że dla niepoprawnie odgadniętego klucza K_i wartości charakterystyki Δ_D dążą do 0 wraz ze wzrostem liczby użytych charakterystyk

$$\lim_{m \to \infty} \Delta_D[j] = 0. \tag{3.2}$$

W przypadku gdy klucz K_r został odgadnięty poprawnie, wartości wszystkich funkcji $D(c_i, b, K_r)$ dla każdego szyfrogramu c_i będą miały zawsze poprawną wartość co spowoduje, że charakterystyka różnicowa powinna się stabilizować przy rosnącej liczbie pomiarów m. Co więcej, wszelkie zakłócenia pomiarów jak i dane, które nie są skorelowane z funkcją D, nie powodują zakłóceń analizy DPA.

Różnicowa analiza poboru mocy wyższych rzędów

Atakom typu DPA można zapobiegać przez zaburzanie powiązania charakterystyk poboru mocy z danymi przetwarzanymi w algorytmie. Jedna z najprostszych metod polega na rozbiciu operacji na dwa etapy, w których przetwarzane są części informacji (np.: blok B jest zapisywany jako $B = B_1 \oplus B_2$ i operacje wykonywane są oddzielnie dla każdego bloku B_1 i B_2). Po zakończeniu operacji uzyskane wyniki są składane razem tworząc właściwe wyjście algorytmu. To proste rozwiązanie wyklucza możliwość użycia analizy DPA ponieważ pobór mocy zależy od wartości poszczególnych bloków przetwarzanych w algorytmie i nie jest skorelowany z wartościami bloku oryginalnego. Niemniej jednak rozwiązanie takie nie zawsze może być zastosowane i może zostać zaatakowane za pomocą różnicowej analizy poboru mocy drugiego rzędu (ang. *second order DPA*), w której poszukuje się korelacji między parami bitów a uzyskanymi charakterystykami poboru mocy.

Możliwe jest również zastosowanie charakterystyk różnicowych wyższych rzędów (ang. *high order DPA*) jednak złożoność tego typu ataków rośnie szybko z rzędem analizy.

3.1.2 Ataki z pomiarem czasu działania

Ataki z pomiarem czasu działania algorytmu kryptograficznego (ang. *timing attacks*) są kolejnym rodzajem ataków typu side-channel. Zostały one zaproponowane przez Kochera [57], który zaobserwował, że pomiar czasu działania algorytmu dla różnych wiadomości może dostarczać informacji o tajnym kluczu wykorzystywanym przez urządzenie. Dokładne pomiary czasu działania układu są tym prostsze, że niejednokrotnie atakujący ma możliwość kontrolowania sygnału zegarowego i spowalniania jego działania.

Ideę ataku można łatwo zaprezentować na przykładzie kryptosystemu RSA wykorzystującego algorytm *square and multiply*. Podobnie jak w analizie poboru mocy, do ataku wykorzystuje się zależność pomiędzy wartością bitów tajnego klucza a przebiegiem algorytmu. Przeprowadzenie ataku jest możliwe ze względu na dążenie do przyspieszenia działania algorytmu RSA i implementowanie mnożenia według algorytmu Montgomerego [63]. Algorytm ten gwarantuje stały czas wykonania mnożenia za wyjątkiem sytuacji, gdy uzyskany wynik wymaga redukcji modulo. W analizie z pomiarem czasu działania próbuje się wychwycić sytuacje, w których ta redukcja została wykonana, co sygnalizuje wykonanie mnożenia i oznacza, że bit klucza miał wartość 1.

W celu wykrycia kolejnego bitu tajnego klucza, atakujący dobiera wiadomości m_i tak, aby przy wykonaniu mnożenia dla części z nich ($m_i \in M_1$) doszło do redukcji, a dla pozostałych ($m_i \in M_2$) na pewno nie. Po wprowadzeniu wiadomości do urządzenia mierzony jest czas wykonania algorytmu RSA. Jeśli szukany bit klucza ma wartość 1, to czasy wykonania algorytmu dla $m_i \in M_1$ powinny być większe od czasów dla wiadomości ze zbioru M_2. Jeśli zaś wartość bitu jest równa 0, to wówczas czasy te nie powinny się istotnie różnić. Uzyskane w ten sposób wyniki pomiarów są oczywiście obarczone dodatkowymi błędami. Może się bowiem zdarzyć, że dla wiadomości ze zbioru M_2 nie wystąpi redukcja dla szukanego bitu, ale wystąpią redukcje w innych iteracjach potęgowania. Z tego względu do przeprowadzenia ataku wymagana jest duża liczba wiadomości i odpowiadających im czasów wykonania algorytmu (dla RSA z kluczem 512 bitowym konieczne jest od 5 000 do 10 000 par wiadomość–czas wykonania).

Nowe ataki tego typu bazują na analizie procedur predykcji skoków w czasie wykonania algorytmu kryptograficznego przez procesor [1] i mogą również być wykonywane bez bezpośredniego dostępu do komputera czy μP — przykładem jest tu atak na OpenSSL opisany przez Brumleya i Boneha [23].

Ataki typu *timing attacks* można również stosować wobec algorytmów symetrycznych. W pracy Handschuha [43] opisano atak na algorytm RC5, w którym wykorzystano 2^{20} par wiadomość–czas wykonania. Podobny atak, przeciwko algorytmowi AES, został zaproponowany przez Bernsteina [6]. Wykorzystuje on

około 4 000 par szyfrogramów do odtworzenia tajnego klucza. Wadą tego ataku jest jego silne uzależnienie od sposobu implementacji, powodujące, że niektóre implementacje AES są na niego odporne.

3.1.3 Ataki przenikające

Ataki przenikające (ang. *probing attacks*) są jednymi z najprostszych koncepcyjnie ataków na urządzenia kryptograficzne. Polegają one na usunięciu wszystkich warstw ochronnych oraz fizycznych zabezpieczeń układu kryptograficznego (ang. *depackaging*) w celu uzyskania bezpośredniego dostępu do szyn danych, linii sterujących i pamięci [2, 59]. Po uzyskaniu dostępu do elementów układu, atakujący może analizować wewnętrzne stany układu kryptograficznego oraz odczytywać dane przesyłane pomiędzy pamięcią i μP. Aby uprościć pomiar i analizę działania układu atakujący może dodatkowo spowolnić taktowanie układu.

Mimo że wszystkie współcześnie produkowane układy kryptograficzne mają elementy chroniące układ przed atakami przenikającymi, to żadne ze stosowanych zabezpieczeń nie gwarantuje pełnej odporności na ataki [59]. Stosowane powszechnie warstwy pasywacji, mające ochraniać elementy układu przed czynnikami chemicznymi stosowanymi w tzw. inżynierii wstecznej (ang. *reverse engineering*), mogą zostać usunięte za pomocą silnych kwasów. Podobne zagrożenia dotyczą układów wykrywających ingerencję z zewnątrz. Układy te mogą zostać usunięte przy wyłączonym zasilaniu i ponownie podłączone po wyprowadzeniu połączeń niezbędnych do analizy działania układu.

Wadą ataków przenikających jest ingerencja w budowę układu kryptograficznego zwiększająca koszty ataku (mikroskopy, precyzyjne zmiany w strukturze układu) i powodująca pozostawienie zauważalnych śladów.

Techniki usuwania i wyłączania zabezpieczeń fizycznych, stosowane w przypadku ataków przenikających, są również stosowane w atakach z uszkodzeniami. Usuwanie warstw ochronnych w takich atakach ma szczególne znaczenie jeśli do wprowadzenia uszkodzeń wykorzystywane są zakłócenia sygnałów wejściowych układu (napięcie zasilania, sygnał zegarowy), albo promieniowanie małej mocy.

3.2 Aktywne ataki side–channel — ataki z uszkodzeniami

W aktywnych atakach typu side-channel analizowane jest działanie układu kryptograficznego pod wpływem czynników zewnętrznych. W porównaniu do ataków pasywnych oddziaływania te nie są jednak tylko rezultatem podawania na wejście układu różnych danych, ale również skutkiem zmiany innych parametrów określających działanie układu. Ataki aktywne są bardzo często rozszerzeniem ataków pasywnych a wiele z nich wręcz wymaga przeprowadzenia wcześniejszego ataku pasywnego np: w celu usunięcia zabezpieczeń istniejących w układzie.

Celem ataków aktywnych jest zmiana wartości wewnętrznych zmiennych algorytmu kryptograficznego,

analiza reakcji układu oraz zmian w generowanym wyniku. Ponieważ celem ataku jest zmiana poprawnej wartości zmiennej wykorzystywanej w algorytmie na wartość błędną lub zmiana przebiegu algorytmu, to ataki te są określane mianem ataków z uszkodzeniami (ang. *fault analysis*).

Podstawowy schemat ataków z uszkodzeniami wykorzystuje zakłócanie wartości zmiennych, które mogą powodować wygenerowanie błędnego wyniku algorytmów kryptograficznych. Wynik ten jest podstawą analizy mającej na celu odszukanie użytego klucza kryptograficznego. Wśród ataków z uszkodzeniami rozróżnia się:

1. ataki proste (ang. *fault analysis*), w których uzyskane błędne wyniki działania algorytmu są analizowane indywidualnie i na ich podstawie następuje wnioskowanie na temat użytego klucza [3, 18, 30, 36, 78],

2. ataki różnicowe (ang. *differential fault analysis*), w których uzyskane błędne i/lub poprawne wyniki są analizowane grupowo [14, 25, 31, 35].

Istnieje również grupa ataków z uszkodzeniami, w których do przeprowadzenia ataku nie są wykorzystywane ani błędne ani poprawne dane wygenerowane przez urządzenie, a jedynie informacja o tym czy błąd udało się wprowadzić czy nie [2, 84]. Ataki takie mogą być skuteczne nawet wtedy, gdy w urządzeniu zaimplementowano procedury mające na celu wykrywanie i przeciwdziałanie atakom.

Warto zauważyć, że w tej metodzie kryptoanalizy nie jest istotne uszkodzenie wprowadzane do urządzenia a jedynie błąd przez to uszkodzenie wywoływany. Z tego względu w wielu opracowaniach na temat kryptoanalizy z uszkodzeniami pomija się aspekt wprowadzenia uszkodzenia zakładając, że błędy określonego typu można wywołać w układzie. Niejednokrotnie jednak założenia takie są nierzeczywiste, gdyż błędy w sposób istotny zależą od konstrukcji konkretnego układu i rodzaju wprowadzanych uszkodzeń. W konsekwencji prawdopodobieństwo wystąpienia określonych błędów jest bardzo różne, a przeprowadzenie niektórych ataków nierealne. Z tego względu w kolejnym rozdziale przedstawione zostaną różne metody wprowadzania uszkodzeń oraz powodowane przez nie błędy.

3.2.1 Metody wprowadzania uszkodzeń

W proponowanych atakach z uszkodzeniami zakłada się, że do urządzenia kryptograficznego można wprowadzić uszkodzenie powodujące określoną zmianę w przetwarzanych danych lub kluczu kryptograficznym. Przy takim założeniu ataki z uszkodzeniami pozwalają skutecznie wydobyć klucz kryptograficzny. Niestety ich autorzy bardzo często zapominają przeanalizować praktyczne możliwości realizacji ich pomysłów. Nie chcąc popełnić tego samego błędu w tym rozdziale przedstawiony zostanie aktualny stan wiedzy na temat możliwości powodowania uszkodzeń w urządzeniach kryptograficznych i błędów jakie za ich pomocą mogą być wprowadzone do obliczeń.

Promieniowanie jonizujące

Mimo, że problem separacji urządzeń elektronicznych od zewnętrznych zakłóceń jest znany od wielu lat to dopiero w erze lotnictwa i lotów kosmicznych zauważono, że źródłem zakłóceń może być promieniowanie pochodzące z kosmosu. Promieniowanie to ma dużą energię, porównywalną z energią protonów i neutronów przyspieszanych w akceleratorach. Badania prowadzone w przemyśle lotniczym pokazały, że promieniowanie takie może nie tylko wywołać nieniszczące zmiany stanów komórek pamięci, ale także spowodować trwałe uszkodzenia elementów układów elektronicznych. Wszystkie uszkodzenia wywoływane promieniowaniem są określane anglojęzyczną nazwą *single event effects* i są dzielone na cztery rodzaje [5]:

- *single event upsets* (SEU), powodujące przemijające zmiany stanów logicznych,
- *single event latchups* (SEL), powodujące powstawanie dużych prądów pasożytniczych w złączach pnpn w strukturach CMOS — prądy te mogą mieć charakter samopodtrzymujący się i mogą powodować zarówno uszkodzenia przemijające jak i trwałe,
- *single event burnout* (SEB), powodujące trwale uszkodzenia tranzystorów,
- *single event functional interups* (SEFI), powodujące trwale bądź przemijające wyłączenie funkcjonalności układu elektronicznego.

W kryptoanalizie najbardziej przydatne są uszkodzenia typu SEU, które pozwalają atakującemu zmienić dane przechowywane w komórkach pamięci RAM. Zgodnie z wynikami prac prowadzonych przez firmę BOEING [88] oraz informacjami z pracy [40] błędy takie mogą spowodować rekonfigurację urządzeń a nawet zresetowanie układów elektronicznych. Ze względu jednak na postępującą miniaturyzację urządzeń elektronicznych oraz to, że atmosfera ziemska eliminuje większość promieniowania kosmicznego, prawdopodobieństwo takiego przekłamania na powierzchni Ziemi jest niewielkie (zgodnie z [40] pojedyncze błędy zamiany bitu w pamięciach RAM nie zdarzają się częściej niż raz na miesiąc).

Generowanie uszkodzeń typu SEU jest możliwe w warunkach ziemskich [69], jeśli wykorzysta się do tego celu protony albo neutrony przyspieszane w akceleratorach cząstek. Dla większości adwersarzy dostęp do takich urządzeń jest niemożliwy, a koszty ich wykorzystania znaczne. Mniej skuteczną, ale i bardziej dostępną techniką jest zastosowanie źródła złożonego z radioaktywnego pierwiastka Americium (^{241}Am) i Berylu (^{9}Be), które emituje neutrony o dużej energii i jest powszechnie stosowane w badaniach geologicznych.

Należy również pamiętać, że uszkodzenia SEU znacznie łatwiej powodują błędy w pamięciach DRAM (ang. *Dynamic Random Access Memory*) czy SRAM (ang. *Static Random access Memory*) niż układach

EPROM (ang. *Erasable Programmable Read-Only Memory*) i pamięciach EEPROM (ang. *Electrically-Erasable Programmable Read-Only Memory*). Wynika to bezpośrednio z budowy pamięci DRAM i SRAM, gdzie pojedyncze bity danych są zapamiętywane w układzie kondensatora i tranzystorów. Z tego powodu bity mogą zostać zamienione jeśli promieniowanie jonizujące spowoduje naładowanie/rozładowanie kondensatora albo zmieni polaryzację tranzystora. W konsekwencji oznacza to, że w systemach μP znacznie łatwiej przekłamać daną przechowywaną w rejestrach procesora, pamięci podręcznej i pamięci RAM niż pamięciach nieulotnych. Przykładowo, we wspomnianych wcześniej kartach μP klucze kryptograficzne są przechowywane w pamięciach nieulotnych (EEPROM) i dlatego znacznie łatwiej je przekłamać dopiero w momencie, gdy mikroprocesor wykonuje na nich operacje kryptograficzne.

Cząstki α, β i promieniowanie rentgenowskie

Przyczyną wielu uszkodzeń w pierwszych produkowanych układach mikroprocesorowych były ich obudowy wykonane z materiałów zawierających w swoim składzie śladowe ilości pierwiastków promieniotwórczych. Pierwiastki te, będąc źródłem promieniowania jonizującego, podobnie jak promieniowanie kosmiczne, mogą powodować uszkodzenia i błędy. Obecnie, w wyniku doskonalenia procesów wytwarzania układów elektronicznych, prawdopodobieństwo spowodowania błędu przez promieniowanie pochodzące z wnętrza układu jest praktycznie równe 0. Zagrożeniem mogą być jednak zewnętrzne źródła promieniowania.

Mimo że źródła promieniowania α są powszechnie dostępne (wykorzystywane są między innymi w czujnikach przeciwpożarowych), to energia tego promieniowania jest zbyt mała, aby przeniknąć przez obudowy układów i spowodować uszkodzenia. To samo dotyczy promieniowania β, które jest bardzo silnie pochłaniane przez obudowy wykonane z tworzyw sztucznych i metalu.

Inaczej niż w przypadku cząstek α i β wygląda wykorzystanie promieniowania rentgenowskiego. Promieniowanie to, powszechnie wykorzystywane w urządzeniach do prześwietlania bagaży na lotniskach ma za małą energię, aby penetrować obudowy kart μP i układów kryptograficznych. Inaczej jest jednak w przypadku promieniowania o dużej energii (tzw. *twarde promieniowanie*), które może powodować zamiany pojedynczych bitów. Jedyną trudnością w zastosowaniu tego typu promieniowania jest trudność precyzyjnego wprowadzania uszkodzeń. Można więc przyjąć, że promieniowanie to jest zagrożeniem w razie połączenia ataku pasywnego polegającego na rozpakowaniu urządzenia z osłon i bezpośredniego napromieniowywania układu elektronicznego.

Promieniowanie podczerwone

Bardzo prostą metodę ataku na układy elektroniczne zaproponowano w pracy [40] gdzie wprowadzano błędy do układu pamięci RAM komputera PC za pomocą promieniowania podczerwonego. Możliwość

taka wynika stąd, że wszystkie układy elektroniczne mają ściśle określony zakres temperatur, w którym działają poprawnie. Po przekroczeniu tego zakresu w układach zaczynają się pojawiać uszkodzenia i błędy, które prowadzą do niepoprawnego działania. W większości układów μP dopuszczalny zakres temperatur wynosi od około -10°C do około 60°C i może być łatwo przekroczony.

Źródłem promieniowania stosowanym w ataku opisanym w pracy [40] była 50-watową żarówka, którą nagrzewano układ pamięci komputera PC. W temperaturze pomiędzy 80 a 100°C w pamięci zaczęły pojawiać się błędy, których liczność można było dobrać przez odpowiedni dobór temperatury (temperaturę zadawano stosując regulowane napięcia zasilania żarówki). Pozwoliło to tak dobrać parametry ataku, aby w pamięci wprowadzać błędy do kilkunastu słów, przekłamując w każdym około dziesięć bitów.

W warunkach rzeczywistych realizacja takiego ataku jest trudna, tym bardziej jeśli atakujący nie ma bezpośredniego dostępu do układów pamięci i nagrzewa całe urządzenie. W takim przypadku najszybciej będzie się grzał układ μP i może dojść do jego nieodwracalnego uszkodzenia, zanim jeszcze w pamięci zostaną wygenerowane błędy. Precyzja ataku jest kolejnym problemem, szczególnie w atakach na małe układy, stosowane m.in. w kartach mikroprocesorowych.

Zakłócenia napięcia zasilania

Zakłócenia napięcia zasilania są często stosowaną metodą wprowadzania uszkodzeń do układów elektronicznych, ze względu na to, że niemal wszystkie współczesne urządzenia kryptograficzne wymagają zewnętrznych źródeł zasilania (do wyjątków należą tokeny wykorzystywane miedzy innymi do identyfikacji i autoryzacji użytkowników logujących się do systemów komputerowych). Atakujący ma więc pełną kontrolę nad parametrami zasilania i może wykorzystywać jego zmiany do wprowadzania uszkodzeń.

Podobnie jak w przypadku temperatury, tak samo napięcie zasilania układów elektronicznych musi być zawarte w zakresie gwarantującym poprawną pracę. W kartach mikroprocesorowych, zasilanych napięciem $V_{cc} = 5V$ dopuszczalne wahania napięcia wynoszą $\pm 10\% V_{cc}$. Nie oznacza to jednak, że po przekroczeniu tej granicy układ całkowicie przestanie działać. Wręcz przeciwnie, jeśli napięcie różni się nieznacznie od napięć dopuszczalnych, to układ będzie działał nadal jednak część z jego operacji może generować błędne wyniki.

Do takiego zachowania dochodzi bardzo często, gdy w napięciu zasilania pojawiają się gwałtowne i krótkotrwałe zmiany (ang. *spikes*). Zaletą takich zmian, z punktu widzenia kryptoanalizy, jest możliwość wprowadzania błędów zarówno do obliczeń jak i danych przechowywanych w pamięci — szczególnie pamięci DRAM, które muszą być cyklicznie odświeżane w celu podtrzymania danych w nich przechowywanych. Inną zaletą jest możliwość wprowadzania błędów do niemal dowolnie wybranej liczby bitów, w szczególności do pojedynczych bitów danych.

Konsekwencją wprowadzenia tego typu uszkodzeń, oprócz błędów polegających na przekłamywaniu bitów i bajtów danych, może być zmiana przebiegu wykonania algorytmu kryptograficznego. W szczególności atak taki może spowodować zmniejszenie licznika kontrolującego przebieg kolejnych rund algorytmów symetrycznych lub pominięcie wykonania niektórych rozkazów. Mimo, że atak taki może wydawać się mało prawdopodobny to przeprowadzano je już w latach 90. Standardowym przykładem jest odblokowywanie kart dekodujących płatnych telewizji PayTV, których wyłączenie polegało na wprowadzeniu ich w nieskończoną pętlę. Wyjście z tej pętli było możliwe dzięki zastosowaniu układu mikroprocesorowego, który wprowadzał do karty uszkodzenia za pomocą krótkotrwałych zmian napięcia zasilania [3]. Taką samą metodę wykorzystano również przy ataku na schematy podpisów ElGamala i DSA [65], gdzie zakłócenia napięcia zasilania spowodowały wyzerowanie bitów liczby losowej k, a w konsekwencji ujawnienie klucza prywatnego.

Zastosowanie krótkotrwałych skoków napięcia zasilania ma jednak ograniczenia, które powodują, że ich praktyczne zastosowanie jest często trudne. Jak opisano w pracy [3] w większości dotychczasowych ataków tego typu zakłada się, że atakujący ma bardzo dokładną wiedzę na temat przebiegu algorytmu, a układy kryptograficzne mają wyłączone obwody zabezpieczające. Inną trudnością jest wprowadzenie poprawnej zmiany napięcia, która jest opisywana przez 4 parametry czasowe i 3 parametry napięciowe. Odpowiednia synchronizacja ataku oraz dobór poziomów napięć jest niezbędny, aby wprowadzając błędy nie przerwać działania urządzenia.

Zakłócenia sygnału zegarowego

Wykorzystanie zakłóceń sygnału zegarowego do wprowadzania uszkodzeń i powodowania błędów w układach kryptograficznych (ang. *glitch attacks*), zostało po raz pierwszy zaproponowane przez Blömera i Seiferta [14]. Idea ich ataku bazuje na tym, że wszystkie urządzenia elektroniczne powinny działać poprawnie mimo pojawiających się niewielkich zakłóceń sygnału zegarowego (karty mikroprocesorowe interpretują stan wysoki CLK gdy wartość sygnału mieści się w przedziale $[0.7V_{cc}, V_{cc}]$ i gdy niestabilność częstotliwości nie przekracza 9%). Przekroczenie tych granic może spowodować niepoprawne działanie urządzenia i być wykorzystane przez atakującego podobnie jak zakłócenia napięcia zasilania. W szczególności zakłócenia takie można wykorzystać do wprowadzenia uszkodzeń do komórek pamięci i zmiany porządku wykonania programu (pominięcia niektórych instrukcji).

Ataki optyczne

W atakach optycznych wykorzystuje się promieniowanie świetlne widzialne, ultrafioletowe i podczerwone. Takie wysokoenergetyczne promieniowanie może powodować zjawisko fotoelektryczne lub kasować pamięci

EPROM. Warunkiem zastosowania takiego ataku jest rozpakowanie układu kryptograficznego i bezpośrednie napromieniowanie [59]. Stosowane może być zarówno światło lasera, promieniowanie UV a nawet światło zwykłej lampy błyskowej [76].

Zaletą takich ataków jest możliwość precyzyjnego wprowadzania błędów do pojedynczych komórek pamięci. Przykładem może być atak przedstawiony w pracy [76], w którym wykorzystano wskaźnik laserowy, fotograficzną lampę błyskową i mikroskop optyczny. Narzędzia te, w połączeniu z precyzyjnym pozycjonowaniem, pozwoliły wprowadzić do układu pojedyncze uszkodzenia powodujące kasowania i ustawiania pojedynczych bitów w pamięci SRAM. Zaletą tego ataku jest możliwość niemal dowolnego doboru liczby bitów zmienianych przez wprowadzony błąd.

Innym zastosowaniem wysokoenergetycznego promieniowania laserowego jest możliwość precyzyjnego przepalania pojedynczych warstw okrywających układ elektroniczny. Dzięki precyzyjnemu napromieniowaniu układu możliwe jest robienie niewielkich otworów w warstwach ochronnych bez naruszania układów, których zadaniem jest wykryć ewentualne ataki.

Zewnętrzne pole elektromagnetyczne

Zewnętrzne pola elektromagnetyczne są powszechnie uważane za źródło potencjalnych ataków na układy kryptograficzne. Przyczyną uszkodzeń wprowadzanych w ten sposób jest oddziaływanie pola elektromagnetycznego na tranzystory wewnątrz układu. Zaletą wykorzystania pola elektromagnetycznego jest jego przenikanie przez obudowy układów. Ograniczeniem praktycznego zastosowania jest konieczność precyzyjnego oddziaływania na wybrane elementy tak, aby wprowadzać zmiany do pojedynczych bitów i bajtów.

Inna metoda ataku, zaproponowana w 2002 roku przez Quisquatera i Samydea [72], wywodzi się z pasywnej analizy pola elektromagnetycznego wytwarzanego przez układy elektroniczne. W proponowanym ataku wykorzystuje się zmienne pole elektromagnetyczne do wytworzenia prądów wirowych w przewodnikach, znajdujących się wewnątrz układu kryptograficznego. Prądy te mogą wywoływać w urządzeniu błędy na przykład poprzez zmianę liczby aktywnych elektronów w wewnętrznej strukturze tranzystora. Uszkodzenie takie powoduje przesunięcie napięcia progowego tranzystora w ten sposób, że kolejne jego przełączenia są już niemożliwe. Zależnie od rodzaju tranzystora uszkodzenia takie mogą powodować ustalenie wartości komórek pamięci na stałą wartość logiczną 0 lub 1. Użycie prądów wirowych pozwala wprowadzać zarówno błędy przemijające jak i trwałe.

Jak opisano w pracy [72] prądy wirowe mogą być wykorzystane do bardzo precyzyjnego wprowadzania uszkodzeń powodujących ustawienie bądź skasowanie pojedynczych bitów. Podobnie jak ataki optyczne, prądy wirowe są potężnym narzędziem w rękach atakującego, zwłaszcza że przeprowadzenie ataku nie wymaga rozpakowywania układu μP. Trudnością przy przeprowadzeniu ataku jest konieczność dokładnej

znajomości topografii układu kryptograficznego.

Prądy wirowe mogą być również zastosowane w celu rozgrzania układu i stopienia warstw ochronnych bądź wprowadzania uszkodzeń za pomocą temperatury.

Wyprowadzenia testowe

Jeszcze inną metodą ataku na układy kryptograficzne jest wykorzystanie wyprowadzeń testowych do wprowadzania błędów do wnętrza układów. Idea ataku jest kontrowersyjna ze względu na to, że żadne współcześnie produkowane układy kryptograficzne nie mają wyprowadzeń testowych. Wyprowadzenia takie, wykorzystywane w czasie produkcji układu, są trwale niszczone (przepalane) zanim gotowy produkt opuści fabrykę. Działanie takie ma dwie podstawowe wady:

- układy są pozbawione jakiegokolwiek mechanizmu kontroli poprawności ich działania — z jednej strony zwiększa to bezpieczeństwo, z drugiej uniemożliwia użytkownikowi weryfikację poprawności działania układu i wykrycie ewentualnego uszkodzenia. Ograniczenie to jest niewątpliwą wadą, gdyż w wielu przypadkach nawet pojedyncze uszkodzenia mogą spowodować ujawnienie tajnego klucza,
- zniszczenie wyprowadzeń testowych polega na przepaleniu tzw. bezpieczników (ang. *fuses*), będących fragmentami ścieżek wewnątrz układu. Ze względów technicznych oraz bezpieczeństwa pozostałych elementów układu bezpieczniki są zazwyczaj ułożone na brzegach układu. Jeśli atakujący ma możliwości ingerencji w układ, to takie położenie bezpieczników ułatwia ich odszukanie i naprawienie [59].

Zagrożenie powodowane przez brak możliwości testowania poprawności działania układów kryptograficznych powodują, że w ostatnich latach coraz poważniej rozważa się umieszczenie w układach wyprowadzeń testowych [39, 46, 64, 83]. Pociąga to za sobą niebezpieczeństwo, że atakujący będzie mógł wykorzystać takie wyprowadzenia w celu przeprowadzenia ataku.

Jednym z powszechnie stosowanych rozwiązań testujących układy cyfrowe jest tzw. testowanie brzegowe (ang. *boundary scan*), w którym wykorzystuje się dodatkowe elementy pamiętające. Są one połączone w taki sposób, że tworzą rejestr skanujący, w którym każdy z elementów jest dodatkowo podłączony do wejścia albo wyjścia układu cyfrowego. Łańcuch ten jest długim rejestrem przesuwnym, którego każdy element,oprócz standardowego wejścia i wyjścia szeregowego, posiada wejście i wyjście równoległe. Łańcuch skanujący przebiega dookoła wszystkich elementów składowych układu cyfrowego w ten sposób, że sygnały wchodzące i wychodzące z poszczególnych bloków przechodzą przez wejścia i wyjścia równoległe łańcucha. Elementy łańcucha mają możliwość zapamiętywania stanów logicznych pojawiających się wejściach równoległych, wymuszania stanów na wyjściach równoległych oraz szeregowego przesuwania zapamiętanych stanów. Dzięki takiej strukturze łańcuch skanujący pozwala na obserwowanie i kontrolowanie

stanów logicznych wewnątrz układu. Implementacja takiego łańcucha, wraz z maszyną stanów określaną skrótem TAP (ang. *Test Access Port*), kontrolującą działanie układu w czasie testowania, wymaga zaledwie 4 dodatkowych wyprowadzeń w układzie [74].

W pracach [81, 82] pokazano, że łańcuch skanujący można skutecznie wykorzystać do ataku z uszkodzeniami. Przedstawione w tych pracach ataki na algorytm DES składały się z dwóch etapów. W pierwszym atakujący odtwarza strukturę łańcucha skanującego szukając, gdzie w łańcuchu skanującym pojawiają się wybrane bity danych i klucza. Po odtworzeniu struktury łańcucha atakujący przystępuje do drugiego etapu, w którym wprowadza do urządzenia błędne dane i analizuje uzyskiwane wyniki.

Siła ataku wykorzystującego wyprowadzenia testowe polega na możliwości wprowadzenia dowolnych danych do niemal każdego miejsca wewnątrz układu (sterowalność) oraz odczytania stanów tych układów w czasie normalnej pracy (obserwowalność).

Pewne ograniczenia w zastosowaniu łańcuch skanującego do kryptoanalizy z uszkodzeniami wskazano w pracy [74]. Zauważono, że atak tego typu wymaga bardzo precyzyjnego wprowadzenia błędów — m.in. błąd musi być wprowadzony w momencie, kiedy atakowany blok będzie wykonywał operacje z użyciem klucza kryptograficznego, co nie jest trywialne w przypadku bloków wielofunkcyjnych. Dodatkowo ze względu na strukturę łańcucha wprowadzane błędy dotyczą zazwyczaj całego bloku funkcjonalnego układu a nie pojedynczych bitów.

3.2.2 Rodzaje powodowanych błędów

Wszystkie rodzaje aktywnych ataków typu side–channel mają na celu spowodowanie w układzie kryptograficznym uszkodzenia, które objawi się w jako pominięcie wykonania pewnych instrukcji lub w postaci przekłamania danych, na których operuje algorytm.

Jeśli atakujący potrafi precyzyjnie wprowadzić uszkodzenie, to pominięcie niektórych instrukcji nie stanowi dużego problemu. Typowym przykładem może być pominięcie instrukcji skoku warunkowego, którego wykonanie jest uzależnione stanem odpowiedniego bitu w rejestrze statusowym procesora (ang. *status word*). Jeśli atakujący potrafi zmienić pojedynczy bit w odpowiednim momencie, to zmiana przebiegu algorytmu jest sprawą prostą.

Z drugiej strony ataki zmieniające dane, na których operują algorytmy kryptograficzne, pozwalają na zastosowanie mniej precyzyjnych błędów. Wynika to między innymi z tego, że prawdopodobieństwo zresetowania μP w razie atakowania rejestrów czy pamięci jest mniejsze niż wtedy, gdy atakujący próbuje zmienić bity w rejestrze statusowym.

Błędy wprowadzane do układów cyfrowych można formalnie zdefiniować.

Definicja 3.1. Niech x będzie n-bitową zmienną modyfikowaną przez błąd, a $x^{(i)}$ jej i-tym bitem. Błąd

jest funkcją E, która zmiennej x przyporządkowuje wartość błędną $\bar{x}$.

$$E : \{0,1\}^n \rightarrow \{0,1\}^n : x(t) \rightarrow \bar{x}(t + \delta t)$$

Błędna wartość $\bar{x}$ może być zapisana jako wyniki operacji logicznej `AND`, `OR` albo `XOR` wektorów bitowych zmiennej x i błędu e

$$\bar{x} = x \texttt{ AND } e \ \vee \ \bar{x} = x \texttt{ OR } e \ \vee \ \bar{x} = x \texttt{ XOR } e$$

lub równoważnie w postaci zwykłej sumy arytmetycznej $\bar{x} = x + e(x)$, gdzie wartość $e(x)$ zależy zarówno od wektora błędów e, operacji logicznej jak i oryginalnej wartości zmiennej x.

Powyższa definicja jest bardzo ogólna i nie definiuje szczegółowo ani przekształcenia E, ani momentu zaistnienia błędu, ani też czasu jego trwania. W celu precyzyjnego określenia powodowanego błędu konieczne jest dokładne określenie następujących parametrów:

W1: rodzaju powodowanego błędu (ang. *error type*), czyli określenia w jaki sposób pojedynczy bit jest zmieniany przez błąd,

W2: krotności błędu (ang. *number of bits*), czyli liczby bitów zmienionych przez błąd,

W3: miejsca wprowadzenia błędu (ang. *error location*),

W4: momentu wprowadzenia błędu (ang. *error timing*),

W5: czasu trwania błędu (ang. *error duration*) oraz

W6: prawdopodobieństwa wprowadzenia błędu.

Oprócz tych warunków wpływ na powodowane błędy ma również znajomość topografii atakowanego układu. W tym przypadku najczęściej przyjmuje się założenie, że atakujący ma bardzo szczegółową wiedzę na ten temat.

Do typowych rodzajów błędów rozpatrywanych w literaturze należą:

- błędy powodujące zmiany zamierzone:
  - zmiana bitu (ang. *bit-flip*),
  - zablokowanie bitu (ang. *stuck-at*),
  - ustawienie bitu (ang. *bit set/reset*),
- błędy powodujące losowe zamiany bitu (ang. *random error*).

Definicja 3.2 (Zmiana bitu)**.** Niech $x = \left\{x^{(n-1)}, \ldots, x^{(1)}, x^{(0)}\right\}$ będzie zmienną. Błędem typu zmiana bitu nazywamy taki błąd, który zmienia wartość bitu na przeciwną:

$$x^{(i)}(t) \rightarrow \bar{x}^{(i)}(t + \delta t) = x^{(i)}(t) \oplus 1$$

Wprowadzenie błędu tego typu nie wyklucza nadpisania zmiennej i tym samym korekcji błędu. Nie dostarcza ono również informacji na temat uzyskanej wartości błędnego bitu — atakujący wie jedynie, że wartość jest przeciwna do wartości poprawnej.

Definicja 3.3 (Zablokowanie)**.** Niech $x = \left\{x^{(n-1)}, \ldots, x^{(1)}, x^{(0)}\right\}$ będzie zmienną. Błędem typu zablokowanie nazywamy taki błąd, który powoduje trwałe zapamiętanie aktualnej (w momencie zaistnienia błędu) wartości bitu:

$$x^{(i)}(t) \rightarrow \bar{x}^{(i)}(t + \delta t) = x^{(i)}(t)$$

Istotną cechą tego błędu jest jego aspekt czasowy, który oznacza, że błędna wartość bitu nie jest więcej zmieniana, nawet jeśli zmienna x jest nadpisywana. Jest więc to błąd trwały, ale niekoniecznie niszczący — błąd ten może zostać zlikwidowany np: przez całkowity reset urządzenia. Warto zauważyć, że wprowadzając ten błąd, atakujący nie wie jaka jest wartość zablokowanego bitu.

Definicja 3.4 (Ustawienie bitu)**.** Niech $x = \left\{x^{(n-1)}, \ldots, x^{(1)}, x^{(0)}\right\}$ będzie zmienną. Błędem typu ustawienie bitu nazywamy taki błąd, który ustawia wartość bitu na wartość 0 (albo 1) niezależnie od oryginalnej wartości bitu:

$$x^{(i)}(t) \rightarrow \bar{x}^{(i)}(t + \delta t) = \begin{cases} x^{(i)}(t) \texttt{ AND } 0 & \text{jeśli błąd ustawia bit na wartość } 0 \\ x^{(i)}(t) \texttt{ OR } 1 & \text{jeśli błąd ustawia bit na wartość } 1 \end{cases}.$$

Błąd taki jest najczęściej błędem przemijającym. Najistotniejsza różnica w porównaniu do innych rodzajów błędów polega na tym, że wartość błędnych bitów jest znana atakującemu.

Definicja 3.5 (Losowa zamiana bitu)**.** Niech $x = \left\{x^{(n-1)}, \ldots, x^{(1)}, x^{(0)}\right\}$ będzie zmienną. Błędem losowo zamieniającym wartość bitu nazywamy taki błąd, który zmienia wartość bitu w sposób losowy, niezależny od poprawnej wartości bitu:

$$x^{(i)}(t) \rightarrow \bar{x}^{(i)}(t + \delta t) = \{0, 1\}$$

Ten rodzaj błędu jest również stosowany do opisu sytuacji, w której każdy kolejno wprowadzany błąd może być różnego rodzaju (zmiana, zablokowanie albo ustawienie bitu). W praktyce do czynienia z takim błędem mamy w sytuacjach, gdy wiele bitów jest zmienianych jednocześnie. W przypadku, gdy choć jeden bit został zmieniony przez inny rodzaj błędu niż pozostałe, mówimy, że wprowadzony błąd jest błędem losowym.

Drugim parametrem jest krotność błędu, czyli liczba bitów zmienionych przez błąd. W literaturze przyjęło się wprowadzać rozróżnienie na dwie grupy:

- błędy pojedyncze (ang. *single errors*), które dotyczą dokładnie jednego bitu zmiennej. Wprowadzenie błędu tego typu nie oznacza jednak, że atakujący może dowolnie wybrać bit, który będzie zmieniony przez wprowadzany błąd,
- błędy wielokrotne (ang. *multiple errors*), które dzielone są na trzy grupy:
  - błędy bajtowe (ang. *byte errors*), w których zmienianych jest od dwóch do ośmiu bitów jednego bajta. Podobnie jak w przypadku błędów pojedynczych, atakujący może nie mieć możliwości wyboru, które bajty zostaną zmienione a wie jedynie, że błędy będą obejmowały dokładnie jeden bajt zmiennej,
  - błędy wielokrotne (ang. *multiple errors*), w których błędem obarczonych jest wiele bitów znajdujących się w różnych miejscach zmiennej,
  - błędy w ramach jednego słowa (ang. *word errors*), które w niektórych przypadkach są wyodrębniane z grupy błędów wielokrotnych. Z sytuacją taką mamy do czynienia w przypadku ataków na algorytm AES, gdzie rozróżnia się błędy wprowadzone do pojedynczego bitu, bajta, kolumny stanu (słowa 32 bitowego) i całego stanu.

W odniesieniu do miejsca wprowadzenia błędu przyjmuje się, że atakujący może mieć:

- pełną kontrolę (ang. *precise control*) oznaczającą, że atakujący potrafi precyzyjnie określić, które bity zmiennej mają zostać zmienione przez błąd,
- ograniczoną kontrolę (ang. *loose control*), gdy atakujący nie może wybrać konkretnych bitów, ale potrafi tak wprowadzić błędy, aby znajdowały się one w wybranym bajcie, bądź grupie bajtów atakowanej zmiennej,
- brak kontroli (ang. *no control*), gdy atakujący nie ma wpływu na to, które bity zmiennej zostaną zmienione przez wprowadzony błąd.

Podobnie wyróżnia się trzy poziomy kontroli momentu wprowadzenia błędu. Pełna kontrola oznacza, że atakujący ma możliwość wprowadzenia błędu w dowolnym momencie wykonania algorytmu. Ograniczona kontrola pozwala na zgrubny wybór momentu wprowadzenia błędu, a brak kontroli oznacza, że atakujący nie ma żadnej informacji, w którym momencie wykonania algorytmu w urządzeniu wystąpił błąd.

Czas trwania błędu określa czy wprowadzony błąd jest błędem trwałym czy przemijającym.

Prawdopodobieństwo wprowadzenia błędu określa, jaka jest skuteczność atakującego w wywoływaniu błędów w urządzeniu. Rozróżnia się tu prawdopodobieństwa wprowadzenia żądanego błędu, wprowadzenia błędu do wybranej zmiennej i do wybranego bitu tej zmiennej:

- prawdopodobieństwo wprowadzenia żądanego błędu, określa jaka jest skuteczność atakującego we wprowadzaniu błędów o określonym rodzaju i krotności,
- prawdopodobieństwo wprowadzenia błędu do wybranej zmiennej określa jaka jest skuteczność atakującego we wprowadzaniu błędów do określonych zmiennych algorytmu,
- prawdopodobieństwo wprowadzenia błędu do wybranego/wybranych bitów/bajtów zmiennej określa jaka jest skuteczność atakującego we wprowadzaniu błędów do wybranych bitów zmiennych wykorzystywanych w algorytmie.

Wartości powyższych prawdopodobieństw są istotnymi parametrami ataków z uszkodzeniami, szczególnie w kontekście ich praktycznego przeprowadzenia. W literaturze dotyczącej uszkodzeń w układach mikroprocesorowych szacuje się bowiem, że jedynie 50 do 60% wprowadzonych uszkodzeń powoduje błędy. Z tych błędów spora grupa to błędy różne od zamierzonych — wprowadzone do innych niż zamierzone zmienne, czy o innej krotności. Mimo to w pracach dotyczących teoretycznych analiz kryptoanalizy z uszkodzeniami zakłada się powszechnie, że prawdopodobieństwa wprowadzenia żądanego rodzaju błędu i do określonej zmiennej są równe 1. Inaczej jest w przypadku prawdopodobieństwa wprowadzenia błędu do wybranego bitu/bajta o którym zakłada się, że jest zawsze mniejsze od 1. Wynika to z przyjmowanego w kryptoanalizie z uszkodzeniami założenia, że atakujący ma co najwyżej ograniczoną kontrolę nad miejscem wprowadzenia błędu — ma możliwość wybrania zmiennej do której wprowadza błąd, ale nie ma możliwości kontroli, które bity/bajty są zmieniane przez błąd. Wobec tego zakłada się zwykle, że zmiana każdego bitu jest równie prawdopodobna.

Różnice pomiędzy poszczególnymi rodzajami błędów

Zdefiniowane w poprzednim podrozdziale rodzaje błędów różnią się między sobą istotnie. Wynika to z jednej strony z różnych uszkodzeń powodujących poszczególne błędy a z drugiej z różnych prawdopodobieństw ich wprowadzenia. Stopień trudności zależy zarówno od technologii wykonania układów jak i stosowanych metody wprowadzania uszkodzeń, które mają tę cechę, że mogą powodować tylko określone błędy. Listę najważniejszych metod wykorzystywanych do wprowadzenia błędów oraz parametry tych błędów przedstawiono w tabeli 3.1.

Mimo istotnie różnych możliwości wprowadzenia poszczególnych rodzajów błędów w publikacjach z zakresu kryptoanalizy z uszkodzeniami dość często przyjmuje się, że atakujący ma możliwość wprowadzania

Tabela 3.1: Klasyfikacja metody wprowadzania uszkodzeń i powodowanych przez nie błędów w układach kryptograficznych

	rodzaj	krotność	lokalizacja	moment wprowadzenia	czas trwania
promieniowanie jonizujące	ustawienie lub zablokowanie bitu	pojedyncze	ograniczona kontrola	ograniczona kontrola	przemijające lub trwałe
zakłócenia napięcia zasilania	losowy	wielokrotne	brak kontroli	pełna kontrola	przemijające
zakłócenia sygnału zegarowego	zależny od algorytmu	zależny od implementacji	zależny od implementacji	pełna kontrola	przemijające
ataki optyczne	zmiana lub ustawienie bitu	pełna kontrola	pełna kontrola	ograniczona kontrola	przemijające lub trwałe
prądy wirowe	dowolny błąd	ograniczona kontrola	pełna kontrola	pełna kontrola	przemijające
wyprowadzenia testowe	dowolny błąd	ograniczona kontrola	ograniczona kontrola	ograniczona kontrola	przemijające

błędów każdego z rodzajów o żądanych parametrach. Wiele publikacji ogranicza się również do błędów typu zmiana bitu, nie analizując praktycznych możliwości ich wprowadzenia i nie rozważając innych rodzajów błędów. Takie postępowanie jest m.in. wynikiem matematycznego podejścia do problemu kryptoanalizy, które wykorzystuje matematyczny model błędu nie skupiając uwagi na przyczynach jego zaistnienia. Z punktu widzenia opisu matematycznego błędy typu zablokowanie czy zmiana bitu można interpretować jako błędy losowe, w których prawdopodobieństw zamiany bitu zależy od ich oryginalnej wartości. Jeśli więc atakujący potrafi wprowadzać tylko błędy losowe, to wykonując odpowiednio wiele prób uda mu się wprowadzić błąd typu zablokowanie czy zmiana bitu. Takie założenie, mimo że wprost nie pojawia się w publikacjach na temat kryptoanalizy z uszkodzeniami, jest bardzo często przyjmowane. Z drugiej strony błędy losowe mogą być interpretowane jako różne błędy typu zmiana bitu — raz z 0 na 1, innym razem odwrotnie.

Powyższe właściwości wykorzystywanych modeli błędów powodują, że w wielu publikacjach omawiany jest tylko jeden rodzaj błędów. Przykładem mogą być ataki na algorytmy asymetryczne, które niemal zawsze wykorzystują błędy typu zmiana bitu [4, 18, 30, 36]. Przyczyną w tym przypadku jest ich prosty opis matematyczny ułatwiający przeanalizowanie skutków wprowadzenia błędu. Pozwala to w łatwy sposób połączyć opis algorytmu z modelem błędu i przeanalizować jego konsekwencje (analiza propaga-

cji, rozprzestrzeniania błędu czy wykrywania i korekcji) dla bezpieczeństwa algorytmu. Wyjątkiem wśród ataków na algorytmy asymetryczne są ataki na algorytm RSA wykorzystujący w swoim działaniu chińskie twierdzenie o resztach (RSA-CRT), gdzie najczęściej analizowane są błędy losowe [18].

Błędy typu zmiana bitu są również wykorzystywane w przypadku algorytmów symetrycznych [14]. W przypadków tych algorytmów jednak, ze względu na trudność ich kompleksowego opisu za pomocą jednolitych zależności matematycznych, możliwość prostego opisu błędu nie jest najważniejsza. Z tego powodu rozważa się również inne rodzaje błędów, a za wyborem modelu błędu przemawiają inne czynniki takie jak trudność wprowadzenia błędu do rzeczywistego układu kryptograficznego czy podatność algorytmu na ataki klasyczne. Z tych powodów najczęściej rozważanym typem błędów w przypadku algorytmów symetrycznych są błędy losowe [11, 14] ograniczane ewentualnie do pojedynczych bajtów bądź słów [5, 31, 36].

Analiza możliwości wprowadzenia błędów do układu kryptograficznego pozwala ocenić prawdopodobieństwo wystąpienia poszczególnych typów błędów. Najczęściej stosowane metody wprowadzania uszkodzeń — zakłócenia sygnału zegarowego i napięcia zasilania, pozwalają wprowadzić błędy losowe i błędy typu zmiana bitu, dotyczące zazwyczaj wielu bitów. Wprowadzenie błędów pojedynczych wymaga natomiast ataków oddziaływujących bezpośrednio na elementy urządzenia poprzez promieniowanie czy prądy wirowe. Pozwalają one na precyzyjne wprowadzanie błędów jednak wymagają bezpośredniego dostępu do układu. Wiąże się to z koniecznością zdjęcia warstw pasywacji i wyłączeniem zabezpieczeń sprzętowych. Z tego względu, przy przyjmowanych założeniach (np: braku ingerencji w budowę układu), niektóre ataki z uszkodzeniami mogą być trudne w realizacji praktycznej.

3.2.3 Zagrożenia ze strony ataków side-channel

Większość ataków pasywnych wykorzystujących analizy statystyczne algorytmów może być skutecznie utrudniana poprzez wprowadzanie dodatkowych zmiennych randomizujących działanie urządzenia/algorytmu i nie powiązanych ani z szyfrowaną wiadomością ani z kluczem. Standardowym rozwiązaniem jest ujednolicanie charakterystyk poboru mocy i czasów wykonania [37, 57].

Ataki pasywne mogą jednak być wykorzystywane w połączeniu z innymi atakami wspomagając analizę algorytmu i upraszczając wnioskowanie. Przykładem takiego zastosowania może być atak na algorytm RSA analizujący zmiany czasu wykonania w elementarnych operacjach algorytmu czy prognozie skoków wykonywanych przez μP [1].

Ochrona przed atakami aktywnymi jest znacznie trudniejsza. Jest to spowodowane tym, że atakujący może dostosowywać się do aktualnych potrzeb i konkretnych zabezpieczeń. Może nie tylko zmieniać dane wejściowe, ale również parametryzować działanie urządzenia na poziomie jego wewnętrznych układów czy

przekłamywać/zmieniać dane w trakcie wybranych rund algorytmu.

Zagrożenie kryptoanalizą z uszkodzeniami jest również powodowane ogromnymi możliwościami jakie posiada atakujący. Co więcej możliwości te są trudne do formalnego opisania jakiego oczekiwać mogliby matematycy od lat zajmujący się kryptografią i kryptoanalizą. Z tego względu do dziś dnia nie udało się określić uniwersalnych metod analizy algorytmów pod kątem podatności na ataki z uszkodzeniami. Na chwilę obecną algorytmy są analizowane indywidualnie i nie udało się określić uniwersalnych właściwości powodujących, że algorytm kryptograficzny jest, bądź nie jest podatny na ten rodzaj ataku. Ze względu na te trudności ogromna większość istniejących rozwiązań ochronnych to rozwiązania sprzętowe a nie algorytmiczne.

Brak skutecznych rozwiązań ochronnych jest powodem dla którego w pracy doktorskiej zajęto się analizą ataków z uszkodzeniami i poszukiwaniem rozwiązań zabezpieczających.

3.3 Ataki z uszkodzeniami na wybrane algorytmy kryptograficzne

Podatność na ataki z uszkodzeniami wykazano dla wszystkich powszechnie stosowanych algorytmów kryptograficznych [4, 11, 17]. W dalszej części pracy przeanalizujemy szczegółowo ataki na algorytm szyfrowania AES i RSA oraz schematy podpisów cyfrowych ElGamala i DSA. Wybór tych algorytmów był podyktowany istniejącymi, szczegółowymi analizami ich podatności na ataki z uszkodzeniami [3, 9, 14, 16, 21, 25, 30, 31, 36, 35, 37, 57, 65, 67, 70, 71, 85], oraz możliwością pokazania z ich pomocą metod przeprowadzania ataków oraz sposobów ochrony przed nimi.

3.3.1 Atak na algorytm AES

Atak z uszkodzeniami na algorytm AES jest rozwinięciem idei ataku różnicowego (ang. *differential attack*) zaprezentowanego w latach 80. przez Bihama i Shamira. Atak ten polega na analizie par szyfrogramów c_1 i c_2, otrzymanych w wyniku szyfrowania wiadomości m_1 i $m_2 = m_1 \oplus d$, które różnią się między sobą o $D = c_1 \oplus c_2$. W ataku różnicowym wykorzystuje się wiele par wiadomości m_i, m_j i odpowiadających im szyfrogramów c_i, c_j, aby zbadać jak różnice w tekstach jawnych przenoszą się na szyfrogramy. Analiza taka pozwala wywnioskować bity tajnego klucza używanego w algorytmie.

Pomimo że analiza różnicowa jest mocnym narzędziem kryptoanalizy, to wszystkie współczesne algorytmy symetryczne są na nią odporne (podatne na taki atak są tylko algorytmy ze zredukowaną liczbą rund). Wynika to stąd, że w ataku różnicowym atakujący ma możliwość zadawania różnic jedynie poprzez podawanie na wejście algorytmu różnych tekstów jawnych. Trudność w tej sytuacji polega na tym, że różnice są przekształcane przez każdą z rund i prześledzenie ich dla całego algorytmu jest niemożliwe.

Zastosowanie kryptoanalizy z uszkodzeniami eliminuje konieczność analizy różnic w przebiegu całego

algorytmu pozwalając na wprowadzanie różnic do wnętrza algorytmu. Możliwość wstrzyknięcia błędu pozwala zmienić dane pomiędzy kolejnymi rundami lub transformacjami. Z tego względu analiza propagacji różnic staje się prostsza, pozwalając na wykonanie analizy różnicowej i odtworzenie klucza kryptograficznego. Atak taki składa się więc z wprowadzenia różnicy (błędu) w trakcie wykonywania algorytmu i analizy różnicy uzyskanej na wyjściu. Błędne szyfrogramy są w tym przypadku porównywane z szyfrogramami poprawnymi.

W literaturze zaproponowano kilka różnych ataków z uszkodzeniami na algorytm AES. Historycznie pierwszy atak został zaprezentowany przez Blömera i Seiferta [14], którzy założyli, że atakujący jest w stanie wprowadzać błędy do pojedynczych bitów ustawiając wartość wybranego bitu na 0 lub 1. Wprowadzając 128 takich błędów do pośredniego stanu algorytmu AES, po przedostatniej transformacji AddRoundKey, atakujący jest w stanie odtworzyć 128-bitowy klucz algorytmu.

Spostrzeżenia zawarte w pracy [14] zostały dalej rozwinięte przez Dusarta [31], który przyjął bardziej realne założenie o rodzaju wprowadzanego błędu. W pracy tej pokazano, że jeśli atakujący wprowadza losowy błąd do pojedynczego bajta stanu algorytmu AES, przed wykonaniem ostatniej rundy algorytmu, to klucz ostatniej rundy algorytmu może być odtworzony na podstawie analizy poprawnego i kilkunastu błędnych szyfrogramów. Na podstawie klucza ostatniej rundy można następnie odtworzyć klucz algorytmu AES.

Inne rozwiązanie zaproponowano w pracy [25], gdzie analizowano wprowadzanie błędów typu zmiana bajta do kluczy dwóch ostatnich rund. Ze względu na to, że błąd wprowadzony do przedostatniego klucza rundy jest propagowany i rozprzestrzeniany w procedurze rozszerzania klucza, do przeprowadzenia skutecznego ataku wystarcza zaledwie około 30 błędnych szyfrogramów. Na podstawie tych szyfrogramów atakujący jest w stanie wyznaczyć wartości 13 bajtów klucza ostatniej rundy. Pozostałe 3 bajty tego klucza są następnie odgadywane za pomocą przeglądu zupełnego i wyznaczany jest klucz główny algorytmu. Oprócz opisu ataku na algorytm AES w pracy [25] zaproponowano kilka metod ochrony przed opisywanym atakiem. W dwóch z nich wykorzystuje się klucze rund zapisane na stałe w pamięci urządzenia kryptograficznego, zamiast kluczy generowanych na bieżąco, w czasie działania algorytmu. Wadą tych rozwiązań jest zwiększona złożoność pamięciowa. Wady tej nie posiada trzecia propozycja zabezpieczająca, która bazuje na wykorzystaniu kodów z kontrolą parzystości do wykrywania wprowadzanych błędów.

Kolejne modyfikacje ataków opisano w pracach [35, 71]. Wykorzystuje się w nich zarówno błędy wprowadzane do danych jak i klucza zakładając, że zarówno lokalizacja jak i rodzaj tych błędów są losowe. Po raz pierwszy też zaproponowano atak, w którym błędy są wprowadzane do pojedynczych słów stanu algorytmu AES, zamiast pojedynczych bajtów i bitów. Wprowadzając takie błędy do przedostatniej rundy algorytmu, atakujący jest w stanie odszukać klucz szyfrujący z wykorzystaniem 12 par poprawnych

Tabela 3.2: Porównanie metod ataku na algorytm AES

Parametr	[14]	[25]	[31]	[35]	[71]
Rodzaj błędu	ustawienie bitu	zmiana bajta	zmiana bajta	zmiana bitu/bajta	zmiana słowa
Miejsce wprowadzenia	dane, precyzyjne	klucz, dowolne	dane, precyzyjne	dane/klucz dowolne	klucz, dowolne
Moment wprowadzenia	ostatnia runda	dwie ostatnie rundy	ostatnia runda	ostatnia runda	przedostatnia runda
Liczba szyfrogramów	128	< 30	≈ 15	50/250	24
Przegląd zupełny	brak	2^{24}	brak	$2^{32}/2^{16}$	2^{16}

i błędnych szyfrogramów [71].

Zestawienie metod ataku na algorytm AES podano w Tab.3.2.

3.3.2 Atak na algorytmy RSA i RSA-CRT

Podatność na ataki z uszkodzeniami wykazano również dla algorytmu RSA [22, 41, 84]. Atak zaproponowany przez Yena i Joyea [84] zakłada, że RSA jest implementowane z wykorzystaniem algorytmu *square-and-multiply*, w którym mnożenia modulo są realizowanie za pomocą dodawania iloczynów częściowych. Wynik tego mnożenia ma wpływ na generowany kryptogram tylko wtedy, gdy bit wykładnika jest równy 1 (w przeciwnym przypadku wynik jest ignorowany). W takiej sytuacji atakujący ma możliwość wprowadzenia do procedury mnożenia tak zwanych niegroźnych błędów (ang. *safe-errors*), które powodują wygenerowanie niepoprawnego kryptogramu tylko wówczas, gdy bit klucza ma wartość 1 (w przeciwnym przypadku wynik będzie poprawny). Przeprowadzenie takiego ataku jest trudne ponieważ wymaga ono wprowadzenia do układu błędów przemijających, o krótkim czasie trwania. Kolejnym utrudnieniem jest sam przebieg algorytmu *square-and-multiply*, w którym mnożenia modulo nie są wykonywane w czasie każdej iteracji, lecz tylko wówczas, gdy kolejny bit klucza jest równy 1 (mnożenia wykonywane gdy bit klucza ma wartość 0 są ignorowane ponieważ służą one jedynie utrudnieniu przeprowadzenia ataku SPA i DPA). Ponadto wprowadzenie błędu nie spowoduje wygenerowania niepoprawnego kryptogramu RSA, jeśli błąd zostanie wprowadzony do bitów zmiennej nie wykorzystywanych do wyznaczenia iloczynów częściowych. Utrudnienia te ograniczają możliwość praktycznej realizacji ataku, wymagają od atakującego dokładnej znajomości zasad działania układu kryptograficznego oraz możliwość dokładnej obserwacji

przebiegu algorytmu.

W pracach [22, 41] przeanalizowano możliwość przeprowadzenia ataków z uszkodzeniami na urządzenia weryfikujące podpisy cyfrowe RSA, lub deszyfrujące szyfrogramy RSA za pomocą kluczy publicznych $\langle d, N\rangle$. Ponieważ urządzenia te wykorzystują klucze publiczne to nie mogą one działać w oparciu o chińskie twierdzenie o resztach. Weryfikacja polega więc na wykonaniu potęgowania, w trakcie którego atakujący może oddziaływać na urządzenie w celu zaakceptowania przez nie błędnego podpisu. Celem ataku jest takie przekłamanie modułu N, aby błędna wartość $\bar{N}$ była liczbą pierwszą lub łatwą do faktoryzacji i $\gcd(d, \bar{N}) = 1$. Prace [22, 41] pokazały, że takie $\bar{N}$ można uzyskać z prawdopodobieństwem 50% jeśli w module o rozmiarze 1024 bity przekłamywane są 4 bity. Takie samo prawdopodobieństwo dla modułu 2048-bitowego wymaga przekłamywania zaledwie 6 bitów. Jeśli atakujący znajdzie odpowiednią wartość $\bar{N}$, to wówczas, znając klucz publiczny, może wyznaczyć fałszywy klucz prywatny

$$\bar{e} = d^{-1} \bmod \bar{N} \tag{3.3}$$

i wykorzystać go do wygenerowania podpisu. Następnie atakujący żąda od urządzenia weryfikacji tego podpisu, jednocześnie starając się wprowadzić błędy przekłamujące moduł N.

Zaletą tej metody jest możliwość przeprowadzenia wszystkich złożonych operacji (poszukiwania modułu $\bar{N}$ i jego faktoryzacji) w trybie *off-line*. Wadą ataku jest konieczność dostosowania wprowadzanych błędów do konkretnych wartości modułów N i $\bar{N}$. Może to spowodować, że atakujący będzie musiał wprowadzić błędy do wielu, rozproszonych bitów. Pomimo, że algorytm RSA jest podatny na ataki z uszkodzeniami to w literaturze znacznie więcej uwagi poświęca się wersji tego algorytmu, która w swoim działaniu wykorzystuje chińskie twierdzenie o resztach — RSA-CRT (Alg. 2.5). Atak na ten algorytm jest znacznie prostszy niż atak na AES. Wynika to przede wszystkim z tego, że nie wymaga on analizy probabilistycznej, a jedynie wykorzystania matematycznych zależności leżących u podstaw chińskiego twierdzenia o resztach i samego RSA.

Idea ataku wynika z obserwacji, że dowolna zmiana wprowadzona do jednego z obliczeń szyfrogramów c_p albo c_q powoduje, że błąd spowodowany w szyfrogramie c jest wielokrotnością tajnej liczby — odpowiednio q albo p.

Fakt 3.1 (Atak z uszkodzeniami na RSA-CRT [18])**.** Załóżmy, że istnieje poprawny c i błędny $\overline{c}$ szyfrogram RSA-CRT wiadomości m, przy czym szyfrogram $\overline{c}$ powstał przez wprowadzenie błędu do jednego z obliczeń wyznaczających wartości c_p albo c_q (pierwszy albo drugi krok algorytmu). Dysponując parą szyfrogramów atakujący może odszukać tajne czynniki liczby N i złamać bezpieczeństwo algorytmu RSA, niezależnie od wprowadzonego błędu.

Zauważymy, że wprowadzenie błędu do procedury obliczania c_p (albo c_q) powoduje jej błędną war-

tość równą $\bar{c}_p$. W wyniku konwersji odwrotnej, wykonanej zgodnie z chińskim twierdzeniem o resztach, otrzymujemy błędny szyfrogram równy:

$$\begin{aligned}\bar{c} &= CRT(\bar{c}_p, c_q) = \left[\bar{c}_p q \left(q^{-1} \bmod p\right) + c_q p \left(p^{-1} \bmod q\right)\right] \bmod N \\ &= \left[c_p q \left(q^{-1} \bmod p\right) + c_q p \left(p^{-1} \bmod q\right) \pm e(c_p) q \left(q^{-1} \bmod p\right)\right] \bmod N \\ &= \left[c \pm e(c_p) q \left(q^{-1} \bmod p\right)\right] \bmod N.\end{aligned}$$

Ponieważ $e(c_p)$ jest mniejsze od p, dlatego $\gcd\left(e(c_p), p\right) = 1$. Podobnie $q^{-1} \bmod p < p$ więc $\gcd\left(q^{-1} \bmod p, p\right) = 1$ i stąd składnik $e(c_p) q \left(q^{-1} \bmod p\right)$ jest wielokrotnością q i nie jest wielokrotnością p. Oznacza to, że różnica

$$c - \bar{c} = e(c_p) q \left(q^{-1} \bmod p\right) = kq$$

i $\gcd(c - \bar{c}, n) = q$. Ponieważ w powyższej analizie nie poczyniono żadnych założeń na temat błędu wprowadzonego do obliczeń, to atak na algorytm RSA–CRT jest skuteczny dla dowolnego błędu, pod warunkiem, że błąd został wprowadzony tylko do jednego z obliczeń c_p albo c_q.

Jak wynika z powyższej analizy atak na algorytm RSA-CRT jest o wiele bardziej niebezpieczny niż atak na algorytm AES. Jest to konsekwencją znacznie bardziej liberalnego modelu błędu i konieczności wykonania jedynie dwóch szyfrowań — poprawnego i błędnego.

3.3.3 Ataki na algorytmy ElGamala i schematy pokrewne

Atak z uszkodzeniami na algorytm ElGamala jest bardziej podobny do ataku na algorytm AES niż RSA i RSA-CRT. W ataku tym zakładamy, że atakujący ma możliwość wprowadzenia do tajnego klucza a, używanego w algorytmie składania podpisów (Alg. 2.7), błędów typu zmiana bitu [4]. Po wprowadzeniu takiego błędu atakujący analizuje otrzymany szyfrogram szukając bitu, który został zmieniony — atakujący musi więc sprawdzić co najwyżej $2n$ możliwych błędów, gdzie $n = \lceil \log_2 a \rceil$.

Fakt 3.2 (Atak z uszkodzeniami na algorytm podpisów ElGamala [4])**.** Niech p, g, y oznaczają publiczne parametry schematu ElGamala (Alg. 2.7) z kluczem prywatnym a o rozmiarze $n = \lceil \log_2 a \rceil$ bitów. Dysponując błędnym podpisem $\overline{s}$, otrzymanym w wyniku podpisywania wiadomości m z jednoczesnym wprowadzeniem do klucza prywatnego a błędu typu zmiana pojedynczego bitu, możliwe jest odszukanie poprawnej wartość tego bitu.

Zauważmy, że jeśli zgodnie z założeniem, atakujący wprowadzi błąd typu zmiana bitu, to błędna wartość klucza prywatnego wynosi $\overline{a} = a \oplus 2^i = a \pm 2^i$, gdzie i oznacza bit klucza zmieniony przez błąd, a znak $\pm$ określa czy zmiana nastąpiła z 0 na 1 czy odwrotnie (Def. 3.2). Konsekwencją błędnej wartości

klucza prywatnego jest wyznaczenie niepoprawnej wartości podpisu $\overline{s}$, która jest równa:

$$\begin{aligned}\overline{s} &= k^{-1}\left(h(m)-\bar{a}r\right) \bmod (p-1) = k^{-1}\left(h(m)-ar \pm 2^i r\right) \bmod (p-1)\\ &= k^{-1}\left(h(m)-ar\right) \pm 2^i k^{-1} r \bmod (p-1).\end{aligned}$$

W wyniku analizy uzyskanego w ten sposób błędnego podpisu $\left\langle r = g^k \bmod p, \overline{s}\right\rangle$ otrzymujemy:

$$\begin{aligned}r^{\bar{c}} \bmod p &= g^{k\left(k^{-1}(h(m)-ar)\pm 2^i k^{-1} r \bmod (p-1)\right)} \bmod p = g^{h(m)-ar\pm 2^i r \bmod (p-1)} \bmod p\\ &= g^{h(m)} g^{-ar \bmod (p-1)} g^{\pm 2^i r \bmod (p-1)} \bmod p = g^{h(m)} y^{-r} g^{\pm 2^i r \bmod (p-1)} \bmod p.\end{aligned}$$

Występujące w powyższym równaniu czynniki $g^{h(m)} \bmod p$, $y^{-r} \bmod p$ oraz $g^r \bmod p$ mogą być wyznaczone przez atakującego na podstawie znajomości klucza publicznego i podpisu. Posiadając te wszystkie informacje atakujący poszukuje wartości wprowadzonego błędu (wyrażenia $\pm 2^i$) spełniającego równość

$$\frac{r^{\overline{s}}}{g^{h(m)} y^{-r}} \bmod p = \left(g^r\right)^{\pm 2^i} \bmod p. \tag{3.4}$$

Po odszukaniu wartości błędu $\pm 2^i$ spełniającej równanie (3.4) atakujący wie, że zmianie uległ i-ty bit. Poprawna wartość tego bitu to 0, jeśli znak błędu jest dodatni i 1, jeśli znak jest ujemny.

Warto zauważyć, że równanie (3.4) ma niejednoznaczne rozwiązanie tylko w jednym przypadku — gdy $p-1 = 2^{n-1}$. W przeciwnym razie, $(p-1 \neq 2^{n-1})$ nie istnieją takie i, j, że dla $\pm 2^i \neq \pm 2^j$ zachodzi $\pm 2^i = \pm 2^j \bmod (p-1)$. Jeśli bowiem p jest liczbą pierwszą n-bitową, to również $p-1$ jest liczbą n-bitową nie mniejszą niż 2^{n-1} (w przeciwnym razie p nie byłoby n-bitowe albo nie byłoby pierwsze). Oznacza to, że dla dowolnego $i = 0, 1, \ldots, n-1$, $1 \leq 2^i \leq p-1$. Jeśli równanie (3.4) miałoby dwa rozwiązania $\pm 2^i$ i $\pm 2^j$, to musiałyby one spełniać jedną z dwóch zależności

$$\begin{aligned}\pm 2^i &= \pm 2^j \bmod (p-1)\\ \pm 2^i &= \mp 2^j \bmod (p-1)\end{aligned}, \tag{3.5}$$

które są tożsame z

$$\begin{aligned}\pm 2^i \mp 2^j \bmod (p-1) &= \pm\left(2^i - 2^j\right) \bmod (p-1) = 0\\ \pm 2^i \pm 2^j \bmod (p-1) &= \pm\left(2^i + 2^j\right) \bmod (p-1) = 0\end{aligned}. \tag{3.6}$$

Ponieważ $2^i \neq 2^j \leq p-1$ to różnica $2^i - 2^j$ jest zawsze mniejsza od $p-1$ i rożna od zera, a $2^i + 2^j$ jest równe $p-1$ wtedy i tylko wtedy gdy $p-1 = 2^{n-1}$ i $i = j = n-2$. Oznacza to, że istnieje co najwyżej jedna taka para $0 \leq i, j \leq (n-1)$ dla której równanie (3.4) ma niejednoznaczne rozwiązanie.

Przedstawiony atak pozwala atakującemu odszukać poprawną wartość dowolnego bitu zmienionego przez wprowadzony błąd. W mało prawdopodobnym przypadku gdy $p-1 = 2^{n-1}$ (nie występującym, jeśli w schemacie ElGamala wykorzystuje się liczby pierwsze o zalecanym rozmiarze od 768 do 1024 bitów), atakujący może wykorzystać kryptoanalizę z uszkodzeniami do odszukania wszystkich bitów za

Tabela 3.3: Porównanie metod ataku na schematy podpisów ElGamala

Parametr	[4]	[36]
Rodzaj błędu	zmiana bitu	zmiana bajta
Krotność błędu	1	≤ 8
Miejsce wprowadzenia	klucz prywatny	
Moment wprowadzenia	równanie podpisu	
Liczba szyfrogramów	ok. 10 000	ok. 50 000
Przegląd zupełny	nieznane bity	

wyjątkiem najstarszego. Bit ten może być następnie zgadnięty a poprawność zgadywania sprawdzona poprzez weryfikację poprawności generowanych podpisów.

W ataku na algorytm ElGamala zakłada się, że atakujący wprowadza błędy typu zmiana pojedynczego, losowo wybranego bitu klucza prywatnego a. Założenie to oznacza, że atakujący musi wprowadzić co najmniej $n = \lceil \log_2 p \rceil$ błędów, aby poznać wszystkie bity klucza a. W praktyce jednak, ponieważ atakujący wprowadza błędy do losowych bitów, może się zdarzyć, że kolejno wprowadzane błędy zmieniają wartości bitów już znanych atakującemu. Zakładając równe prawdopodobieństwo przekłamania każdego bitu klucza a można wyznaczyć prawdopodobieństwo, że w $m \geq n$ próbach odszukane zostaną wszystkie bity. Prawdopodobieństwo to jest równe jeden minus prawdopodobieństwo zdarzenia przeciwnego — że w m próbach wybrany bit nigdy nie będzie przekłamany. Ponieważ prawdopodobieństwo nieprzekłamania i-tego bitu w m próbach wynosi $(1 - 1/n)^m$ to szukane prawdopodobieństwo jest równe

$$P = 1 - n\left(1 - \frac{1}{n}\right)^m. \tag{3.7}$$

Dla $m = n \ln 2t$ i $t > 1$ prawdopodobieństwo to wynosi

$$P = 1 - n\left(1 - \frac{1}{n}\right)^{n \ln 2t} \geq 1 - n\left(\frac{1}{e}\right)^{\ln 2t} = 1 - n\frac{1}{2t}. \tag{3.8}$$

Dla $t = n$ prawdopodobieństwo to wynosi $1/2$ a $\lim_{t\to\infty} P = 1$.

Metoda przeprowadzenia ataku na algorytm ElGamala została udoskonalona w pracach [36] oraz [65]. Pierwszy z artykułów prezentuje atak na algorytm ElGamala, w którym zamiast błędów typu zmiana pojedynczego bitu wprowadzane są błędy typu zmiana losowo wybranego bajta tajnego klucza. Mimo tej różnicy, scenariusz ataku jest bardzo zbliżony do oryginalnego, zaproponowanego w pracy [4].

Inaczej jest w przypadku ataku proponowanego przez Naccache [65]. Atak ten jest rozszerzeniem wcześniejszych prac dotyczących bezpieczeństwa podpisów cyfrowych ElGamala w przypadku, gdy atakujący posiada częściową informację o liczbach losowych k wykorzystywanych do wygenerowania podpisu [19, 67].

Zaproponowany atak łączy w sobie wykorzystanie kryptoanalizy z uszkodzeniami i poszukiwania rozwiązania problemu najbliższego wektora w kracie (CVP) w celu odszukania tajnego klucza. Atak rozpoczyna się od wprowadzenia błędu typu ustawienie wartości wybranych bitów losowej liczby k. Wprowadzając taki błąd atakujący stara się wyzerować kilka/kilkanaście najmniej znaczących bitów liczby k i tym samym spowodować wygenerowanie poprawnego podpisu przy częściowo znanej wartości liczby losowej. W ten sposób generowanych jest wiele podpisów pod różnymi wiadomościami, a następnie są one wykorzystywane do odszukania tajnego klucza. Zgodnie z [65], jeśli atakujący potrafi wyzerować 8 bitów liczby losowej k, to do skutecznego przeprowadzenia ataku niezbędne jest wygenerowanie zaledwie 27 błędnych podpisów.

Ataki analogiczne do opisanych powyżej, można przeprowadzić na schematy pokrewne z algorytmem ElGamala — DSA, Shnorra i ich wersje wykorzystujące krzywe eliptyczne. Jedyne różnice w tych wypadkach dotyczą sposobu analizy uzyskiwanych błędnych podpisów, której złożoność jest różna dla różnych algorytmów. W przypadku analizowanego w pracy algorytmu DSA różnica dotyczy równania ataku 3.4 i wynika z tego, że atakujący nie zna wartości $r = g^k \bmod p$, gdyż elementem podpisu jest $r = \left(g^k \bmod p\right) \bmod q$. Mimo tej trudności atakujący może przeprowadzić atak odgadując wprowadzony błąd i sprawdzając czy zachodzi równość

$$\left(g^{\overline{s}^{-1}}\right)^{H(m)} \left(y^{\overline{s}^{-1}}\right)^{r} \left(g^{\overline{s}^{-1}}\right)^{\pm 2^i r} \bmod p \bmod q = r. \tag{3.9}$$

Podobnie jak w przypadku ElGamala, znając wartość błędu $\pm 2^i$ dla której zachodzi powyższa równość, atakujący zna postać wprowadzonego błędu a tym samym poprawną wartość jednego bitu klucza prywatnego.

Jeszcze inny atak na schemat DSA przedstawiono w pracy Nguyena i Shparlinskiego [67], gdzie problemu kryptoanalizy sprowadzono do rozwiązania problemu ukrytej liczby (ang. *hidden number problem (HNP)*) [19, 20].

Definicja 3.6 (Problem HNP)**.** Niech $a, b \in \mathbb{Z}_q$ i $a \neq 0$ będą nieznanymi i szukanymi liczbami, a t_i znanym i losowo wybranym elementem grupy Z_q^*. Problem HNP polega na odszukaniu wartości a i b na podstawie znajomości l najmniej znaczących bitów liczb $at_i + b \bmod q$ dla $i = 1, 2, \ldots d$.

Definicję tego problemu można znaleźć w pracach [19, 20] gdzie pokazano, że może on być rozwiązany w czasie wielomianowym pod warunkiem, że znane jest $l = 2\log_2(\log_2 q)$ najmniej znaczących bitów liczb $at_i + b \bmod q$. Prace te zostały rozszerzone przez Nguyena i Shparlinskiego [67] którzy pokazali, że atak na algorytm ElGamala można sprowadzić do rozwiązania problemu HNP. Znajomość l najmniej znaczących bitów liczby losowej k pozwala ją zapisać w postaci $k = 2^l k_1 + k_2$ gdzie k_2 jest znane a $0 \leq k_1 < q/2^l$.

Tabela 3.4: Porównanie metod ataku na schematy podpisów DSA

Parametr	[4]	[36]	[65]
Rodzaj błędu	zmiana bitu	zmiana bajta	ustawienie bajta
Krotność błędu	1	≤ 8	losowa
Miejsce wprowadzenia	klucz prywatny		liczba losowa
Moment wprowadzenia	równanie podpisu		wybór liczby losowej
Liczba szyfrogramów	ok. 1 700	ok. 2 300	27
Przegląd zupełny	nieznane bity		brak

Jeśli dodatkowo $\gcd(s, q) = 1$, to równanie podpisu można przekształcić do postaci

$$\begin{aligned} s &= k^{-1}\left(h(m) + ar\right) \bmod q, \\ s^{-1}ar &= 2^l k_1 + k_2 - s^{-1}h(m) \bmod q, \\ s^{-1}ar2^{-l} &= \left(k_2 - s^{-1}h(m)\right)2^{-l} + k_1 \bmod q. \end{aligned} \tag{3.10}$$

Znajomość najmniej znaczących bitów pozwala atakującemu obliczyć czynniki $t_i = s^{-1}r2^{-l} \bmod q$ i $u = \left(k_2 - s^{-1}h(m)\right)2^{-l} \bmod q$. Mając na uwadze, że $0 \leq k_1 < q/2^l$ możemy napisać

$$at_i - u \bmod q < p/2^l, \tag{3.11}$$

co pozwala nam określić wartość najbardziej znaczących bitów liczby $at_i \bmod q$ (muszą one być takie, aby lewa strona nierówności (3.11) dała w wyniku liczbę, której l najstarszych bitów ma wartość 0). Znajomość wielu t_i oraz l najstarszych bitów liczb $at_i \bmod q$ pozwala odszukać klucz prywatny a poprzez rozwiązanie problemu HNP.

Istotnym wnioskiem z prac dotyczących bezpieczeństwa schematów ElGamala i DSA jest obserwacja, że maskowanie klucza za pomocą liczb losowych może nie być wystarczające dla zapewnienie bezpieczeństwa. Jak zaprezentowano w pracach [19, 65, 67] znajomość części bitów liczby losowej k pozwala na odszukanie maskowanego klucza poprzez rozwiązanie problemu CVP albo HNP.

Istotną różnicą pomiędzy atakami na algorytmy ElGamala i DSA jest liczba błędnych podpisów niezbędnych do przeprowadzenia ataku. Atak na schemat podpisów ElGamala, zaprezentowany w pracy [36], wykorzystujący błędy wprowadzane do bajtów klucza prywatnego, wymaga wygenerowania około 50 000 błędnych podpisów. Analogiczny atak na algorytm DSA wymaga natomiast około 2 300 błędnych podpisów. Zmniejszenie to jest efektem mniejszych rozmiarów liczb używanych w DSA. Innym rezultatem mniejszego rozmiaru klucza prywatnego jest krótsze poszukiwanie wprowadzonego błędu, które wymaga sprawdzenia wszystkich możliwych do wprowadzenia błędów.

Rozdział 4

Ochrona przed atakami z uszkodzeniami

Różnorodność ataków z uszkodzeniami powoduje, że ochrona przed nimi nie jest łatwa i wymaga indywidualnego dostosowania metod ochrony do każdego algorytmu. Wszystkie metody ochrony dążą do jednego z dwóch celów — utrudnienia ataku i/lub uniemożliwienie analizy otrzymanych danych. Utrudnianie ataku polega na obniżaniu możliwości kontrolowania działania urządzenia i przebiegu algorytmu przez atakującego. Jego celem jest uniemożliwienie spełnienia warunków W1-W6 (rozdział 3.2) lub zmniejszenie zdolności atakującego do kontrolowania parametrów ataku z uszkodzeniami. Próby implementacji takich rozwiązań pokazały, że nie pozwalają one zapewnić wystarczającego poziomu bezpieczeństwa. Spowodowało to konieczność poszukiwania nowych metod ochrony i zaowocowało rozwiązaniami, które mają na celu zapewnienie, że w przypadku wprowadzenia uszkodzenia i wywołania błędu, wynik algorytmu będzie nieprzydatny dla atakującego.

Istniejące metody ochrony można podzielić ze względu na rodzaj stosowanej ochrony oraz sposób jej implementacji. Ze względu na pierwsze kryterium wyróżniamy cztery grupy:

- ochronę przed wprowadzeniem błędów do urządzenia (ang. *error avoidance*),
- wykrywanie błędów wprowadzonych do urządzenia (ang. *error detection*),
- korygowanie błędów wprowadzonych do urządzenia (ang. *error correction*),
- rozpraszanie błędów wprowadzonych do urządzenia (ang. *error diffusion*).

Ze względu na sposób implementacji mechanizmów ochronnych rozróżniamy trzy poziomy rozwiązań:

- programowe (ang. *software-based*), w których zabezpieczenia polegają na wykrywaniu i kompensacji błędów i są implementowane w algorytmie,
- sprzętowe (ang. *hardware-based*), w których zabezpieczenia są implementowane sprzętowo, na poziomie bloków funkcjonalnych,

- strukturalne (ang. *transistor-level*), w których zabezpieczenia są implementowane już na poziomie pojedynczych bramek a ich celem jest zapobieganie występowaniu błędów mimo istnienia uszkodzeń.

Konsekwencją implementacji algorytmów zabezpieczających na różnych poziomach jest nie tylko rożna złożoność, ale również uniwersalność i zakres zastosowań. Powszechnie uważa się, że rozwiązania programowe są specjalizowane dla konkretnych algorytmów i pozwalają wykrywać przede wszystkich błędy przemijające. Inaczej jest w przypadku rozwiązań sprzętowych, które są bardziej uniwersalne i mogą służyć zarówno wykrywaniu błędów przemijających jak i trwałych. Wadą tych rozwiązań jest większy koszt ich implementacji w stosunku do rozwiązań programowych. Rozwiązania strukturalne są natomiast powszechnie uważane za najlepsze, gdyż gwarantują poprawne działanie podstawowych elementów składowych układu. Istotną wadą są natomiast duże koszty implementacji, a także ich wąska specjalizacja, pozwalająca zagwarantować bezpieczeństwo konkretnych algorytmów (a nawet konkretnych implementacji) i tylko wobec ataków wykorzystujących wąską grupę specyficznych błędów.

4.1 Ochrona przed wprowadzeniem błędów

Pierwszy poziom ochrony sprzętu kryptograficznego składa się z rozwiązań sprzętowych, których zadaniem jest zminimalizowanie prawdopodobieństwa wystąpienia uszkodzeń w układzie kryptograficznym. Rozwiązania te mają na celu zagwarantowanie, że próby wprowadzenia uszkodzeń za pomocą oddziaływania na urządzenie, skończą się niepowodzeniem. Rozwiązania z tej grupy można podzielić na:

- rozwiązania pasywne, których zadaniem jest zapobieganie występowaniu w układzie zjawisk mogących powodować uszkodzenia i generować błędy. Do grupy tej należą wszelkiego rodzaju osłony i warstwy pasywacji zapobiegające wystawieniu układu na bezpośrednie działanie atakującego oraz układy ochronne łańcucha skanującego zapobiegające jego wykorzystaniu do wprowadzania błędów,

- rozwiązania aktywne, których zadaniem jest wykrywanie zjawisk mogących prowadzić do uszkodzeń i błędów oraz zapobieganie im przez zmianę działania lub reset układu kryptograficznego. Do tej grupy należą sprzętowe układy wykrywania zakłóceń napięcia zegarowego i sygnału zasilania, układy chroniące przed oddziaływaniem polem elektromagnetycznym i uszkodzeniami typu SEU, oraz układy randomizujące sygnał zegarowy.

Rozwiązania sprzętowe, mające na celu zapobieganie wprowadzeniu uszkodzeń do układu kryptograficznego są najlepszą metodą ochrony przed atakami. Do zalet rozwiązań tego typu należy uniwersalność, niezależność od algorytmu kryptograficznego i przetwarzanych danych. Cechy te gwarantują, że informacja o zadziałaniu zabezpieczeń nie wpływa na uproszczenie przeprowadzenia ataku. Jest to zaleta, której

nie posiadają rozwiązania reagujące na błędy już wprowadzone do algorytmów kryptograficznych i która gwarantuje, że nie można przeprowadzić ataków bazujących na analizie czy rozwiązania zabezpieczające zadziałały czy nie [2, 84].

Wadą wszystkich rozwiązań ochronnych tego typu jest ich zależność od technologii. Powszechnie przyjmuje się, że zabezpieczenia chroniące układ przed uszkodzeniami mogą zostać sforsowane lub będą mogły być wyłączone w niedalekiej przyszłości. Po za tym wiele rozwiązań jest zaimplementowanych niedokładnie, pozostawiając możliwość przeprowadzenia ataku [2, 59]. Dlatego też, mimo istotnych zalet, rozwiązania te nie są całkowicie skuteczne i nie gwarantują odporności na ataki z uszkodzeniami.

Ochrona wyprowadzeń testowych

Jak wspomniano w rozdziale 3 jedną z metod wprowadzania błędów do urządzeń kryptograficznych jest wykorzystanie łańcucha skanującego. Łańcuch taki znajduje się we wszystkich mikroprocesorach i układach *system on chip* (SoC) ze względu na to, że około 5% wytwarzanych urządzeń ma wady fabryczne i zanim produkt zostanie dostarczony do odbiorcy musi być wszechstronnie sprawdzony. W standardowych układach elektronicznych układy testowe mogą być wykorzystywane do jego testowania w czasie całego życia układu. W układach kryptograficznych wyprowadzenia te są niszczone zanim układ opuści fabrykę. Postępowanie takie ma na celu zabezpieczenie układu jednak nie gwarantuje ono, że przepalone ścieżki nie zostaną naprawione i wykorzystane do ataku. Przepalenia takie mogą być stosunkowo łatwo naprawione z wykorzystaniem mikroskopu skaningowego lub mikroskopu wykorzystującego skupione wiązki jonów (ang. *focused ion beam - FIB*) [59]. Pewnym zwiększeniem bezpieczeństwa byłoby niszczenie struktury układów testowych a nie tylko ich połączeń z wyjściami układu. Trudnością w tym względzie jest jednak zapewnienie dużej precyzji i nienaruszenie innych elementów układu. Jest to tym trudniejsze, że gęstość upakowania elementów układów cyfrowych ciągle rośnie.

Niszczenie wyprowadzeń testowych ma podstawową wadę — uniemożliwia testowanie układów kryptograficznych w czasie ich normalnej pracy. Wada ta ma poważne konsekwencje dla użytkowników takich układów, którzy ze względu na brak wyprowadzeń testowych nie są w stanie:

- zweryfikować poprawność działania algorytmu na poziomie bardziej szczegółowym niż ocena funkcjonalna na podstawie zadawania wejść i obserwacji wyjść,
- wykryć błędów pojawiających się w urządzeniu,
- sprawdzić poprawności działania poszczególnych bloków funkcjonalnych (np: generatorów liczb losowych) i układów zabezpieczających.

Nie dysponując specjalistycznym sprzętem i nie mogąc zweryfikować poprawność działania, użytkownicy

traktują układ kryptograficzny jak czarną skrzynkę. Muszą zaufać producentowi urządzenia, co jednak naraża ich na zagrożenia związane z różnego rodzaju bocznymi wejściami (ang. *back doors*), czy atakami kleptograficznymi (ang. *kleptographic attacks*), które mogą zostać zaimplementowane przez producenta.

Z tego powodu w ostatnich latach poszukuje się innych rozwiązań, próbujących pogodzić chęć zapewnienia wysokiego poziomu bezpieczeństwa z możliwościami dokładnego przetestowania układu. Niestety oba cele pozostają ze sobą w pewnej sprzeczności, gdyż zapewnienie testowalności układu wymaga możliwości jego dokładnej obserwacji, a to z kolei jest źródłem dodatkowych informacji na temat działania algorytmu, które mogą uprościć przeprowadzenie ataku. Proponowane rozwiązania w tym zakresie zakładają, że łańcuch skanujący implementowany w układach kryptograficznych jest poszerzany o dodatkowe bloki mające zapewnić, że tylko określone osoby będą mogły go wykorzystać [39, 47, 46, 64, 83]. Proponowane w tym zakresie rozwiązania zapewniają bezpieczeństwo wykorzystując tajny klucz w celu:

- przejścia urządzenia w tryb testowy [47, 83] — klucz musi zostać wprowadzony zanim rozpocznie się testowanie układu, a jego weryfikacją zajmuje się rozszerzony moduł TAP. Po poprawnej weryfikacji testowanie przebiega tak jak w standardowym łańcuchu skanującym,

- analizy wyników uzyskanych z procedur testujących [39] — łańcuch skanujący w tym rozwiązaniu rozszerzony jest o układ wyznaczający funkcję MAC (ang. *Message Authentication Code*) dla wiadomości będącej odpowiedzią układu na pobudzenie testowe i klucza oraz o układ ograniczający liczbę cykli testowych. Znajomość klucza jest niezbędna, aby można było przeanalizować uzyskany wynik.

- wyłączenia pseudolosowych zmian wprowadzanych do danych w łańcuchu skanującym [64] — w rozwiązaniu tym każdy wektor testowy zawiera klucz, który powoduje, że cały łańcuch zachowuje się w sposób w pełni deterministyczny. Jeśli klucz jest niepoprawny, to wyjście łańcucha skanującego jest zmieniane zależnie od aktualnego stanu generatora pseudolosowego.

Niestety, żadna z powyższych modyfikacji nie jest rozwiązaniem idealnym. Rozwiązania zaproponowane prze Hélyego [46] wiążą się z dużym narzutem implementacyjnym. Wykorzystanie tak zwanych rejestrów lustrzanych (ang. *mirror key registers*) zaproponowanych przez Yanga [83] zakłada, że ochronę przed atakiem zapewni odseparowanie tajnych danych od rejestru skanującego, co w przypadku wielu algorytmów nie jest prawdą. Rozwiązanie Mukhopadhyaya [64] może natomiast być łatwo złamane jeśli atakujący uzyska dostęp do jednego tylko wektora testowego. Wydaje się, że najlepsze rozwiązanie ochronne połączone z małym narzutem na implementację zapewnia rozwiązane zaproponowane w pracy [39]. W rozwiązaniu tym bezpieczeństwo zapewniane jest za pomocą funkcji haszującej z kluczem (MAC). Dodatkowy układ ograniczający liczbę cykli testowania gwarantuje, że różnego rodzaju ataki na rejestry przesuwne są utrud-

nione. Wadą tego rozwiązania jest jednak pogorszenie zdolności detekcyjnych i wydłużenie procesu analizy odpowiedzi układu na pobudzenie testowe.

Układy siatek czujników

Ponieważ spora liczba ataków z uszkodzeniami wykorzystuje oddziaływania wymagające bezpośredniego dostępu do układu μP (promieniowanie α, β czy promieniowanie świetlne), atakujący musi umieć rozebrać układ usuwając z niego obudowę i warstwy pasywacji. Intuicyjnym rozwiązaniem ochronnym jest więc zastosowanie siatek czujników, których zadaniem jest wykrycie ingerencji w układ μP i skasowanie wrażliwych danych, tak aby zapewnić, że nie zostaną one odzyskane. Rozwiązania takie są jedną z najskuteczniejszych metod ochrony i są powszechnie stosowane w mikroprocesorach i pamięciach przeznaczonych do zastosowań kryptograficznych. Dodatkową zaletą wynikającą z ich stosowania jest utrudnienie analizy topografii układu kryptograficznego znajdującego się poniżej siatki.

Niestety siatki czujników mają swoje wady. Po pierwsze, ponieważ układy detekcyjne wykrywają zwarcia lub przerwy linii, więc pozwalają one wykrywać ingerencję w układ tylko gdy jest on zasilany. Jest to poważną wadą w przypadku kart mikroprocesorowych i innych układów pozbawionych własnego zasilania. Atakujący może w takiej sytuacji odłączyć zasilanie, zniszczyć fragment siatki i uzyskać dostęp do znajdującego się poniżej układu, a następnie zrekonstruować siatkę tak, aby po przywróceniu zasilania, układ działał poprawnie [59]. Inną wadą jest sposób testowania stanu siatki przez μP. W wielu rozwiązaniach stan siatki (wykrycie ingerencji bądź jej brak) jest odwzorowywany na jeden bit słowa statusowego, który jest okresowo sprawdzany przez mikroprocesor. Jak zauważono w pracy [85], oraz mając na względzie możliwości ataków opisane w rozdziale 3, pojedynczy bit słowa statusowego może dość łatwo być zmieniony dając błędne informacje na temat stanu siatki. Istniejące rozwiązania siatek wykrywających ingerencję w układ kryptograficzny pokazują, że w wielu przypadkach najsłabszym ogniwem w łańcuchu zabezpieczeń jest człowiek. W pracy [59] przedstawiono przykład źle zaprojektowanej siatki w mikroprocesorze ST16SF48A, która nie pokrywa zakończeń szyn danych o długości kilku mikrometrów. Tym samym atakujący ma dogodne miejsce do przeprowadzenia ataku.

Czujniki zakłóceń napięcia zasilania i sygnału zegarowego

Powszechnie stosowaną techniką ataku jest wprowadzenie zakłóceń napięcia zasilania i sygnału zegarowego kontrolującego działanie układu. Powoduje to, że wszystkie układy mikroprocesorowe posiadają wbudowane elementy detekcji zmian obu sygnałów powodujące reset urządzenia w przypadku wykrycia potencjalnego ataku.

Innym sygnałem mogącym świadczyć o przeprowadzaniu ataku na urządzenie kryptograficzne jest

obniżenie częstotliwości taktowania układu. Rozwiązanie takie jest bardzo często stosowane w celu zwiększenia precyzji wprowadzenia uszkodzenia i tym samym poprawienie skuteczności ataku. Ochrona przed tego typ działaniem polega na resetowaniu układu kryptograficznego i szyn danych w momencie niewykrycia kolejnego zbocza sygnału zegarowego. Niestety implementacja układów detekcyjnych tego typu jest trudna. Standardowe rozwiązanie, np. górno- i dolnoprzepustowe filtry RC, mogą nie wykryć ataku jeśli zmiany sygnału zegarowego są odpowiednio dobrane lub gdy układy zostaną odłączone [59].

Kontrola poprawności wykonania programu

Potrzeba kontroli poprawności wykonania programu wynika z tego, że wiele ataków z uszkodzeniami stara się doprowadzić do pominięcia wykonania niektórych instrukcji algorytmu. Trudność w kontrolowaniu algorytmu polega jednak na tym, że jego przebieg zależy od szeregu warunków: flag statusowych procesora, danych wejściowych, przerwań itd. Z tego powodu pełna kontrola wymagałaby użycia dodatkowego układu mikroprocesorowego, powielającego funkcjonalność układu podstawowego. Jest to rozwiązanie niepraktyczne i dlatego do sprawdzania poprawności działania stosuje się metody uproszczone, zakładające, że przebieg algorytmu posiada pewne niezmienne cechy.

Typowym sposobem sprawdzania poprawności algorytmu jest analiza częstości występowania rozkazów skoku. Sprawdzanie takie działa przy założeniu, że w poprawnym algorytmie wykonuje się skoki co określony odstęp czasu $t \in [t_0, t_1]$. Jeżeli atakujący spowoduje pominięcie wykonania niektórych instrukcji warunkowych (nie wykonywanie niektórych skoków) lub zwiększenie ich liczby (np. zmieniając liczbę powtórzeń pętli w algorytmie), to układ detekcji sygnalizuje błąd i powoduje reset urządzenia.

Niestety takie zabezpieczenia są trudne w realizacji i silnie zależne od algorytmu. Co więcej również i te zabezpieczenia mogą być dość łatwo wyłączone [59].

Wprowadzanie losowości do wykonania programu

W atakach typu side–channel analizuje się przebieg działania algorytmu, badając jakie operacje zostały wykonane a jakie nie (np: analiza poboru mocy, analiza czasu wykonania i pochodne ataki aktywne). Dość oczywistym sposobem ochrony przed takimi atakami jest ujednolicanie charakterystyk algorytmu. Jest to osiągane poprzez zrównoważenie poboru mocy, ujednolicenie czasu wykonania algorytmu oraz uwikłanie zależności wykonywanych skoków, od przetwarzanych danych i tajnego klucza. Innym rozwiązaniem jest ich uzależnienie od większej liczby parametrów.

Ponieważ ujednolicenie poboru mocy jest trudno osiągalne i prowadzi do zwiększenia sumarycznej mocy pobieranej przez układ, to typowym rozwiązaniem jest randomizowanie poboru mocy. Polega ono na wykonywaniu równolegle z algorytmem kryptograficznym innych operacji, które również są powiązane

z danymi i kluczami używanymi przez algorytm. Celem takiego działania jest wprowadzenie nieregularności do charakterystyki poboru mocy i utrudnienie kryptoanalizy. Niestety zabezpieczenie takie jest skuteczne tylko w przypadku prostej analizy poboru mocy. W analizie różnicowej, ze względu na różne rozkłady sygnału pochodzącego od algorytmu kryptograficznego i układu randomizującego, możliwe jest odseparowanie z charakterystyki poboru mocy poszczególnych składowych. W takim przypadku randomizacja poboru mocy jedynie utrudnia atak, powodując konieczność zebrania większej liczby charakterystyk działania układu kryptograficznego.

Podobne techniki stosuje się w celu zamazania charakterystyk czasowych działania algorytmu. Typowym przykładem jest tu modyfikacja algorytmu *square and multiply*, która zapewnia, że w każdej iteracji pętli wykonywane jest mnożenie — jeśli kolejny bit wykładnika jest równy 1, to mnożone są właściwe zmienne wykorzystywane do wyznaczenia wyniku. Gdy bit ma wartość 0, to wówczas wykonywane jest mnożenie (ang. *dummy multiplication*), którego wynik nie wpływa na wynik działania algorytmu.

Ponadto ujednolicenie charakterystyk poboru mocy, czasu wykonania bądź wprowadzenie losowości utrudnia atakującemu precyzyjne wykonanie ataku z uszkodzeniami. Wynika to stąd, że spora część ataków wykorzystuje charakterystyki do określania miejsca i czasu wprowadzenia uszkodzenia [3, 65]. Jeśli charakterystyki są ujednolicone, to atakujący ma mniejszą możliwość kontrolowania czasu i miejsca wprowadzenia uszkodzenia.

4.2 Wykrywanie błędów wprowadzonych do urządzenia

Ochrona przed wprowadzeniem uszkodzeń do urządzeń kryptograficznych jest pierwszą linią obrony przed atakami z uszkodzeniami. Niestety rozwiązania takie nie gwarantują zadowalającego poziomu bezpieczeństwa i konieczne jest poszukiwanie rozwiązań, które pozwolą zapewnić bezpieczeństwo pomimo występowania w urządzeniu błędów. Jednym z nich jest wykorzystanie kodów detekcyjnych, których zadaniem jest wykrywanie błędów wprowadzanych do danych przetwarzanych w algorytmie kryptograficznym.

Teoria kodów detekcyjnych jest dziedziną bardzo dobrze przebadaną i wiele istniejących w niej rozwiązań jest stosowanych w sposób uniwersalny. Pozornie może się więc wydawać, że znajomość błędów wprowadzanych do urządzeń kryptograficznych i zdolności detekcyjnych poszczególnych kodów, pozwala na prostą implementację algorytmów zapobiegających atakom z uszkodzeniami. Niestety algorytmy detekcyjne były projektowane z myślą o wykrywaniu błędów pojawiających się w kanale transmisyjnym wskutek występujących w takim kanale zakłóceń (np. błędy grupowe (ang. *burst errors*)). Ponadto, założenia przyjmowane dla kanałów transmisyjnych nie obowiązują w układach funkcjonalnych. Do najistotniejszych różnic należy transmitancja kanału transmisyjnego. W przypadku idealnym zakłada się, że transmitancja ta jest równa 1 (idealny kanał przesyła sygnał bez żadnej modyfikacji i bez zniekształceń).

W rzeczywistych kanałach przyjmuje się natomiast, że zniekształcenia sygnału nie uniemożliwiają jego poprawnego odebrania, a błędy występujące w kanale są addytywne (dodają się do przesyłanego sygnału). W przypadku zastosowania kodów detekcyjnych do algorytmów kryptograficznych, kanałem transmisyjnym jest algorytm, którego wyjście jest pewną funkcją sygnału wejściowego. Oznacza to, że nawet jeśli błędy nadal mają charakter addytywny, to na wyjściu kanału są one zależne od zniekształceń wprowadzanych w kanale, realizowanego algorytmu i momentu zaistnienia. Nieliniowy charakter algorytmów kryptograficznych i uzależnienie zdolności detekcyjnej od konkretnego algorytmu komplikuje jeszcze bardziej analizę zdolności detekcyjnych. Z tego względu ochrona układów kryptograficznych, wykorzystująca algorytmy detekcji błędów, jest znacznie bardziej skomplikowana niż w typowych kanałach transmisyjnych. Powoduje to konieczność indywidualnego przeanalizowania każdego algorytmu kryptograficznego.

W literaturze można odnaleźć liczne publikacje na temat detekcji błędów w symetrycznych algorytmach kryptograficznych, pozwalających zapewnić ochronę przed kryptoanalizą z uszkodzeniami [7, 8, 28, 42, 50, 51, 54, 55, 70, 79, 87]. Wszystkie proponowane rozwiązania są dzielone na grupy ze względu na poziom i metodę wykrywania błędów. Ze względu na pierwsze kryterium rozróżniamy trzy rodzaje algorytmów detekcji błędów [55]:

- wykrywające błędy na poziomie całego algorytmu (ang. *algorithm level*),
- wykrywające błędy na poziomie pojedynczej rundy algorytmu (ang. *round level*),
- wykrywające błędy na poziomie pojedynczej transformacji algorytmu (ang. *operation level*).

Zależnie od wybranego poziomu wykrywania, detekcja błędów następuje po zakończeniu całego algorytmu kryptograficznego, zakończeniu pojedynczej rundy albo pojedynczej transformacji. Różnica pomiędzy poszczególnymi rodzajami dotyczy przede wszystkim narzutu implementacyjnego i opóźnienia wykrywania błędów [50].

Ze względu na metodę wykrywania błędów proponowane rozwiązania można podzielić na cztery grupy:

- wykorzystujące odwracalność algorytmów szyfrujących [50, 51, 55],
- bazujące na prostej kontroli parzystości [7, 8, 70, 79],
- bazujące na cyklicznej sumie kontrolnej CRC (ang. *cyclic redundancy check*) [87],
- bazujące na tzw. silnych kodach (ang. *robust codes*) [54] i kodach resztowych [42].

Wykorzystanie odwracalności algorytmów kryptograficznych

Wykrywanie błędów polega tu na sprawdzeniu czy dane wyjściowe algorytmu, rundy lub transformacji (zależnie od poziomu wykrywania błędów) po wprowadzeniu do algorytmu, rundy lub transformacji od-

wrotnej dadzą poprawne dane wejściowe. Jeśli rzeczywista i wyznaczona w ten sposób dana wejściowa jest różna, to jest to sygnałem, że w trakcie wykonania algorytmu (lub algorytmu odwrotnego) wystąpiły błędy [50, 51, 55].

Zaletą takiego rozwiązania jest zdolność wykrywana wszystkich błędów — ponieważ operacje szyfrowania są funkcjami różnowartościowymi i "na", to jednej danej wejściowej odpowiada dokładnie jedna dana wyjściowa i odwrotnie. Dodatkową zaletą jest uniwersalność, pozwalająca na zastosowanie tego rozwiązania w dowolnym algorytmie szyfrującym i dowolnej transformacji. Ograniczeniem dla tej metody nie są również transformacje nieliniowe, dla których zastosowanie kontroli parzystości (lub innych kodów liniowych) pociąga za sobą stosunkowo dużą złożoność implementacyjną. Wadą takich rozwiązań jest natomiast duży narzut implementacyjny wynikający z konieczności implementacji algorytmu szyfrującego i deszyfrującego w jednym urządzeniu. Ograniczeniem zastosowania tego typu rozwiązań jest konieczność istnienia transformacji odwrotnych do wykorzystywanych w algorytmie. W szczególności metoda ta nie może być stosowana do ochrony funkcji haszujących (brak odwrotności) czy algorytmów weryfikacji podpisów cyfrowych (brak informacji o kluczu prywatnym).

Prosta kontrola parzystości

W pracach [7, 8, 70, 79] zaproponowano prostą kontrolę parzystości do wykrywania błędów w implementacjach algorytmów symetrycznych. Proponowane rozwiązania wykorzystują pojedyncze bity parzystości przypisywane do szyfrowanego bloku danych lub jego fragmentów: słów lub bajtów. Przykładem takiego rozwiązania jest zaproponowana przez Bertoniego [7] kontrola parzystości poszczególnych bajtów stanu algorytmu AES. Rozwiązaniu temu zostanie poświęcone więcej uwagi, ponieważ proponowane w niniejszej pracy zabezpieczenie jest rozszerzeniem propozycji Bertoniego.

W artykule [7] zaproponowano, aby każdy bajt $s_{i,j}$ stanu algorytmu AES był rozszerzony o dodatkowy bit parzystości wyznaczany jako

$$p_{i,j} = \bigoplus_{t=0}^{7} s_{i,j}^{(t)}. \tag{4.1}$$

Dodatkowo, dla każdej transformacji algorytmu, zaproponowano metody predykcji parzystości wyniku w oparciu o dane wejściowe i ich parzystość.

W transformacji **AddRoundKey** (2.6) wynik jest wyznaczany jako suma modulo 2 odpowiednich bajtów klucza rundy i bajtów stanu. Z tego względu parzystość po transformacji powinna być równa

$$p'_{i,j} = \bigoplus_{t=0}^{7} {s'}_{i,j}^{(t)} = \bigoplus_{t=0}^{7} \left(s_{i,j}^{(t)} \oplus k_{i,j}^{(t)} \right) = \bigoplus_{t=0}^{7} s_{i,j}^{(t)} \oplus \bigoplus_{t=0}^{7} k_{i,j}^{(t)} = p_{i,j} \oplus \bigoplus_{t=0}^{7} k_{i,j}^{(t)}, \tag{4.2}$$

gdzie $k_{i,j}^{(t)}$ oznacza t-ty bit bajta klucza rundy $k_{i,j}$. Suma $\oplus_{t=0}^{7} k_{i,j}^{(t)}$ jest natomiast parzystością bajta $k_{i,j}$ klucza rundy. Jak wynika z wzoru (4.2) parzystość wyjściowa po transformacji może zostać wyznaczona

na podstawie parzystości bajta wejściowego i parzystości odpowiadającego mu bajta klucza rundy.

W transformacji **MixColumns** (2.7) parzystość wyjściowa może być również opisana w funkcji parzystości wejściowej i danych wejściowych. W przeciwieństwie jednak do innych transformacji predykcja parzystości po transformacji MixColumns przebiega różnie dla różnych wierszy macierzy stanu:

$$\begin{aligned} p_{0,j} &= p_{0,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{0,j} \oplus s^{(7)}_{1,j} \\ p_{1,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{3,j} \oplus s^{(7)}_{1,j} \oplus s^{(7)}_{2,j} \\ p_{2,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s^{(7)}_{2,j} \oplus s^{(7)}_{3,j} \\ p_{3,j} &= p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{3,j} \oplus s^{(7)}_{0,j}. \end{aligned} \tag{4.3}$$

Jak wynika ze wzorów 4.3 przewidywana parzystość wyjściowa zależy od parzystości trzech bajtów wejściowych i najstarszych bitów, dwóch z tych bajtów. Szczegółowe wyprowadzenie tych wzorów można znaleźć w pracy [7].

W transformacji **ShiftRows** (2.11) predykcja parzystości jest najprostsza, ponieważ transformacja ta nie zmienia bajtów stanu, a jedynie obraca je cyklicznie w wierszach. Z tego względu przewidywana parzystość jest równa przesuniętej parzystości wejściowej:

$$p'_{i,j} = p_{i,j+i \bmod 4}. \tag{4.4}$$

Ze względu na nieliniowość transformacji **SubBytes** (2.12) w rozwiązaniu przedstawionym w pracy [7] zastosowano rozszerzoną tablicę Sbox o rozmiarze 512×9 bitów. Wejściem tablicy jest bajt rozszerzony o bit parzystości $s_{i,j} \| p_{i,j}$. Jeśli wejście jest poprawne, to wówczas z tablicy odczytywany jest poprawny wynik transformacji SubBytes wraz z poprawną parzystością — $s'_{i,j} \| p'_{i,j}$. W przeciwnym przypadku, gdy dana wejściowa nie jest poprawna, wynikiem transformacji jest bajt o wartości 0 rozszerzony o niepoprawny bit parzystości:

$$s'_{i,j} \| p'_{i,j} = \begin{cases} \mathrm{Sbox}\left(s_{i,j}\right) \| p\left(\mathrm{Sbox}\left(s_{i,j}\right)\right) & \text{jeśli } p_{i,j} = \bigoplus_{t=0}^{7} s^{(t)}_{i,j} \\ (0000\ 0000 \| 1)_2 & \text{jeśli } p_{i,j} \neq \bigoplus_{t=0}^{7} s^{(t)}_{i,j} \end{cases}, \tag{4.5}$$

gdzie $\mathrm{Sbox}\left(s_{i,j}\right)$ jest wynikiem transformacji SubBytes dla argumentu $s_{i,j}$, a $p(x)$ oznacza bit parzystości dla argumentu x. Efektem takiej modyfikacji jest możliwość rozróżnienia poprawnych i niepoprawnych bajtów podawanych na wejście tablicy Sbox. Pozwala to na detekcję błędów i generowanie poprawnych wyników przekształcenia SubBytes tylko wtedy, gdy bajt wejściowy i jego bit parzystości są poprawne (nie są obarczone błędem o nieparzystej krotności). Efekt taki nie jest możliwy do uzyskania w standardowej tablicy Sbox ponieważ każda dana wejściowa jest poprawnym wejściem tej tablicy. Wadą modyfikacji jest ponad dwukrotne zwiększenie rozmiaru tablicy, która musi teraz przechowywać 512 danych 9-bitowych.

Suma kontrolna CRC

Wykorzystanie sum kontrolnych CRC do wykrywania błędów w algorytmie AES zaproponowali Yen i Wu [87]. Przeanalizowali oni skuteczność wykrywania błędów wprowadzanych do stanu algorytmu AES, za pomocą kodów CRC $(n+1, n)$ dla $n = \{4, 8, 16\}$ bajtów. Rozwiązanie to korzysta z własności algorytmu AES, w szczególności transformacji MixColumns, dla której suma kontrolna wyniku jest taka sama jak suma kontrolna argumentu.

Rozwiązanie zaproponowane w pracy [87] wykorzystuje wielomian generujący $g(x) = x+1$ nad ciałem GF(2^8), niezależnie od przyjętego parametru n. Wyznaczona suma kontrolna CRC jest więc elementem ciała GF(2^8) i może być zapisana w postaci liczby ośmiobitowej. Zaletą takiego doboru wielomianu $g(x)$ jest między innymi proste wyznaczanie wartości CRC.

Lemat 4.1. Reszta z dzielenia wielomianu $w(x) = w_n x^n + w_{n-1}x^{n-1} + \ldots + w_1 x + w_0$ nad ciałem GF(2^8) przez $g(x) = x + 1$ nad GF(2^8) jest równa

$$w(x) \bmod g(x) = \bigoplus_{i=0}^{n} w_i.$$

Dowód. Zauważmy, że dla $a \in$GF(2^8) reszta $a \bmod (x+1) = a$ oraz $ax \bmod (x+1) = a$. Z tego wynika, że $1x \bmod (x+1) = 1$, a więc reszta z dzielenia ax^i dla $i \geq 1$ wynosi:

$$a\ x^i \bmod g(x) = \left(a \prod_{j=1}^{i} x \right) \bmod (x+1) = \left[(a \bmod (x+1)) \prod_{j=1}^{i} (x \bmod (x+1)) \right] \bmod (x+1) = a.$$

Oznacza to, że reszta z dzielenia $w_i x^i \bmod (x+1) = w_i$ dla każdego $i = \{0, \ldots, n\}$ stąd, korzystając z właściwości operacji modulo, mamy $w(x) \bmod (x+1) = \bigoplus_{i=0}^{n} w_i$. □

Jak wynika z lematu 4.1 wartość CRC dla bajtów stanu $s_{i,j}$ może być łatwo wyznaczona za pomocą operacji `XOR`. Złożoność wyznaczenia poprawnej sumy kontrolnej po poszczególnych transformacjach zależy od przyjętej wartości parametru n.

Ze względu na właściwości algorytmu AES [87] wyznaczanie CRC jest najprostsze jeśli $n = 16$, czyli gdy mamy jeden bajt CRC dla całego stanu algorytmu AES. W takiej sytuacji wartości sumy CRC przed i po transformacjach ShiftRows i MixColumns są takie same. Wynika to bezpośrednio z przyjętego parametru n, stałych wartości bajtów $s_{i,j}$ w transformacji ShiftRows i odpowiedniego doboru wag przy transformacji MixColumns. W transformacji AddRoundKey wynikowa wartość CRC jest wyznaczana jako suma `XOR` CRC danych wejściowych i CRC klucza rundy. Transformacja Subbytes, w proponowanym rozwiązaniu, jest natomiast implementowana jako wyznaczenie odwrotności (wykonywane za pomocą tabeli Sbox) oraz wykonanie przekształcenia afinicznego. W takiej sytuacji wynikowe CRC wyznaczane

jest jako suma wartości odczytanych z tablic Sbox, specjalnie zmodyfikowanych dla potrzeb algorytmu — tablice przechowują wartości $s_{i,j} \oplus s_{i,j}^{-1}$ zamiast standardowej wartości $s_{i,j}^{-1}$ [87].

Jeśli $n = \{4, 8\}$, to dodatkowe przewidywanie sumy CRC musi być wykonane dla transformacji ShiftRows. Przewidywanie CRC dla pozostałych transformacji nie ulega istotnym zmianom.

Autorzy pracy [87], proponując wykorzystanie CRC do wykrywania błędów, oszacowali prawdopodobieństwo niewykrycia błędu na około 0.4%, $0.16 \cdot 10^{-2}$% i $0.24 \cdot 10^{-8}$% odpowiednio dla sum CRC (17,16), (9,8) i (5,4). Oszacowanie to jest jednak prawdziwe tylko przy założeniu, że atakujący wprowadza losowe błędy do całego stanu algorytmu AES. Jak pokazano w rozdziale 3.2.1 znacznie bardziej prawdopodobne jest, że atakujący będzie starał się wprowadzić błędy do pojedynczych słów albo bajtów. W takim przypadku prawdopodobieństwo nie wykrycia ataku nie zależy od parametru n i wynosi około $2^{-8} \approx 0.4\%$ dla błędów wprowadzanych do słów i 0% dla błędów wprowadzanych do pojedynczych bajtów (wszystkie takie błędy zostaną wykryte).

Silne kody i kody resztowe

Zastosowanie silnych kodów do ochrony przed kryptoanalizą z uszkodzeniami zaproponowano w pracach [54, 60] jako rozszerzenie standardowych metod ochrony. W rozwiązaniach tych transformacje algorytmu kryptograficznego są dzielone na grupy transformacji liniowych i nieliniowych, po czym każde z nich są chronione na różne sposoby:

- przez wykorzystanie transformacji odwrotnej — dotyczy to ochrony transformacji nieliniowych,
- przez wykorzystanie układów predykcji parzystości i układu porównującego przewidzianą i rzeczywistą parzystość.

W metodzie zaproponowanej przez Karpovskiego [54], w celu zwiększenia skuteczności wykrywania uszkodzeń dla bloków liniowych, wykorzystano tzw. silne kody. Rozwiązanie to wymaga rozszerzenia układu predykcji oraz porównywania parzystości o dwa bloki podnoszenia do sześcianu w ciele GF(2^8). Wadą rozwiązania jest duży narzut implementacyjny, wynikający z konieczności implementacji dodatkowych układów podnoszenia do sześcianu dla każdego bloku sprawdzającego i przewidującego parzystość. Pewną wadą jest również zastosowanie całkowicie różnych metod ochrony dla grup transformacji liniowych i nieliniowych. Rozwiązanie takie wymaga ich wzajemnego powiązania, jeśli wykrywanie błędów ma się odbywać na poziomie rundy albo całego algorytmu. Powiązanie takie wiąże się natomiast z dodatkowymi układami i zwiększeniem narzutu implementacyjnego. Z drugiej strony wykrywanie błędów na poziomie pojedynczych transformacji powoduje zagrożenia atakami wprowadzającymi błędy do wyników pośrednich, przechowywanych w rejestrach układu kryptograficznego. Zaletą omawianego rozwiązania jest

niewątpliwie zwiększenie skuteczności wykrywania błędów przy zastosowaniu sprawdzania wyników po każdej rundzie algorytmu.

Prace, w których proponowano zastosowanie silnych kodów, były przez długi czas ograniczone wyłącznie do ochrony algorytmu AES. W roku 2006 pojawiły się pierwsze prace dotyczące analogicznej ochrony algorytmów bazujących na teorii liczb. Ochrona zaproponowana przez Gunnara [42] wykorzystuje do sprawdzania poprawności działania algorytmów kody resztowe.

Skuteczność istniejących algorytmów detekcji błędów

Wszystkie rozwiązania wykrywające błędy w urządzeniach posiadają dwie niedogodności, które ograniczają ich praktyczne zastosowanie. Pierwsze ograniczenie dotyczy opóźnienia wykrycia błędu wprowadzonego do urządzenia. Opóźnienie to zależy od tego czy zastosowany algorytm wykrywa błędy na poziomie całego algorytmu, rundy czy pojedynczej transformacji. Druga niedogodność jest znacznie bardziej istotna — atak z uszkodzeniami można przeprowadzić nawet jeśli algorytm detekcji działa poprawnie i wykrywa wszystkie błędy [2, 84]. Przyczyną takiego stanu rzeczy jest to, że po wykryciu błędu układ kryptograficzny musi na błąd zareagować. Wydaje się oczywiste, że pierwszą czynnością powinno być przerwanie dalszego wykonywania algorytmu i zgłoszenie błędu użytkownikowi albo powtórzenie obliczeń. Zauważmy jednak, że każda z powyższych reakcji dostarcza atakującemu dodatkowych informacji sygnalizując wykrycie błędu, albo wydłużając czas wykonania algorytmu. Oznacza to, że atakujący może obserwować, kiedy urządzenie wykryje błąd przez niego wprowadzony. Jeśli tylko atakujący potrafi wprowadzać błędy typu ustawienie pojedynczego bitu, lub wnioskować o kluczu bazując na informacji o poprawnym/niepoprawnym przebiegu obliczeń [84], to informacja ta pozwala na przeprowadzenie ataku.

4.3 Korygowanie błędów

Wad algorytmów detekcji nie posiadają algorytmy korygujące błędy, których zaletą jest stały czas wykonania, niezależny od obecności błędu. Dzięki temu algorytmy te nie dostarczają atakującemu dodatkowych informacji i nie ułatwiają w ten sposób ataku na urządzenie kryptograficzne. Kolejną zaletą korygowania błędów jest brak konieczności powtarzania transformacji, rundy lub algorytmu w razie zaistnienia błędów.

Reguła korekcji błędów w algorytmach kryptograficznych jest taka jak w kanałach transmisyjnych i następuje zgodnie z zasadą MLD (ang. *maximum like hood decoding*). Różnica natomiast jest taka sama jak w przypadku detekcji błędów — algorytmy kryptograficzne przekształcają wejściowy blok danych utrudniając w ten sposób analizę i przeprowadzenie korekcji. Trudność wynika również z przyjętej reguły korekcji, która koryguje błędy najbardziej prawdopodobne. W kanałach transmisyjnych, w których błędy wynikają z charakterystyki kanału, nie jest to problemem. Inaczej jest w przypadku kryptoanalizy

z uszkodzeniami, gdzie atakujący może wprowadzać rozmaite rodzaje błędów. Szczęśliwie analiza możliwości wprowadzania błędów do układów kryptograficznych 3.2.1 pokazała, że możliwości atakującego w tym zakresie są dość ograniczone, szczególnie jeśli przeprowadza on ataki nieinwazyjne. Dodatkowo poszczególne algorytmy są podatne na ataki z uszkodzeniami wykorzystujące określone rodzaje błędów. Pozwala to uprościć zadanie korekcji przez ograniczenie go jedynie do korygowania błędów określonego rodzaju, takich, których wprowadzenie jest najbardziej prawdopodobne i/lub najbardziej korzystne z punktu widzenia atakującego.

Mimo przewagi algorytmów korekcji nad algorytmami detekcji błędów do tej pory zaproponowano tylko jedno rozwiązanie tego typu, dla ochrony algorytmu AES. Propozycje te zostały zaprezentowane w pracach autora [26, 27]. Rozwiązanie to pozwala na korygowanie 50% błędów wprowadzanych do jednego bajta stanu oraz detekcję 100% błędów wprowadzanych do jednego bajta i 99.9% błędów wprowadzanych do słowa. Rozwiązanie to będzie szczegółowo zaprezentowane w rozdziale 5.

4.4 Rozpraszanie błędów

Rozpraszanie błędów jest rozwiązaniem, które na pierwszy rzut oka może się wydawać szaleństwem, ponieważ w układach cyfrowych zjawisko takie jest zjawiskiem negatywnym, niepożądanym i od wielu lat minimalizowanym. Idea rozwiązania wynika jednak bezpośrednio z analizy ataków z uszkodzeniami na algorytmy kryptograficzne (rozdział 3.3), w szczególności algorytm RSA-CRT. Analizując ataki z uszkodzeniami można zauważyć, że ich skuteczność zależy w znacznym stopniu od:

- rodzaju wprowadzonego błędu — w algorytmach AES, ElGamal i DSA zakłada się, że błędy będą wprowadzone do jednego bajta (AES) bądź bitu (ElGamal, DSA). W ataku na algorytm RSA–CRT wprowadzony błąd może być dowolny,
- miejsca wprowadzenia błędu — w algorytmie RSA-CRT zakłada się, że błąd jest wprowadzony tylko do jednego obliczenia c_p albo c_q, a w algorytmie ElGamala bezpośrednio do klucza prywatnego a,
- czasu wprowadzenia błędu — w przypadku algorytmu AES zakłada się, że atakujący potrafi wprowadzać błędy do danych i/lub klucza ostatniej i przedostatniej rundy algorytmu.

Jeśli którekolwiek z powyższych założeń nie jest spełnione, to wzrasta złożoność ataku, a jego przeprowadzenie jest utrudnione lub w ogóle niemożliwe.

Powyższe obserwacje prowadzą do wniosku, że sposobem na ochronę algorytmów kryptograficznych jest rozpraszanie wprowadzanych błędów w taki sposób, aby uzyskiwany wynik algorytmu zawierał błędy znacznie utrudniające lub wręcz uniemożliwiające przeprowadzenie analizy.

Typowym przykładem zastosowania rozpraszania uszkodzeń są algorytmy bazujące na chińskim twierdzeniu o resztach, w szczególności algorytm RSA-CRT. Jak pokazano w rozdziale 3.3, atak na algorytm RSA-CRT wymaga od atakującego wprowadzenia dowolnego błędu e do dokładnie jednego z obliczeń c_p albo c_q. Zauważmy, że jeśli atakujący wprowadzi błędy e_p i e_q do obu podpisów, odpowiednio c_p i c_q, to wówczas prawdopodobieństwo skutecznego ataku jest znikome.

Lemat 4.2. Istnieje urządzenie kryptograficzne realizujące szyfrowanie RSA w oparciu o algorytm RSA–CRT (Alg. 2.5), z wykorzystaniem liczb pierwszych p i q. Jeśli, podczas szyfrowania, atakujący wprowadzi do urządzenia błędy w ten sposób, że oba obliczenia c_p i c_q zostaną zmienione przez błędy e_p i e_q, to prawdopodobieństwo skutecznego ataku jest nie większe niż $2\max(\frac{1}{q}, \frac{1}{p})$.

Dowód. Bez utraty ogólności możemy założyć, że $p > q$. Zauważmy, że gdy oba obliczenia wykonywane w kroku 1 i 2 algorytmu RSA–CRT są obarczone błędami (tj. $\bar{c}_p = c_p + e_p, \bar{c}_q = c_q + e_q$), to na podstawie chińskiego twierdzenia o resztach otrzymujemy

$$\begin{aligned} \bar{c} &= CRT(\bar{c}_p, \bar{c}_q) = \left[(c_p + e_p)q\left(q^{-1} \bmod p\right) + (c_q + e_q)p\left(p^{-1} \bmod q\right)\right] \bmod N \\ &= \left[c_p q\left(q^{-1} \bmod p\right) + c_q p\left(p^{-1} \bmod q\right) + e_p q\left(q^{-1} \bmod p\right) + e_q p\left(p^{-1} \bmod q\right)\right] \bmod N \\ &= \left[c + e_p q\left(q^{-1} \bmod p\right) + e_q p\left(p^{-1} \bmod q\right)\right] \bmod N. \end{aligned} \tag{4.6}$$

Stąd mamy, że $\bar{c} - c = \left[e_p q\left(q^{-1} \bmod p\right) + e_q p\left(p^{-1} \bmod q\right)\right] \bmod N$ i atakujący chce wyznaczyć jedną z tajnych liczb jako największy wspólny dzielnik $\gcd(\bar{c} - c, N)$.

Zauważmy, że e_p i e_q są mniejsze od odpowiednio p i q co oznacza, że wyrażenia $e_p q$ i $e_q p$ są zawsze mniejsze od pq. Suma $e_p q\left(q^{-1} \bmod p\right) + e_q p\left(p^{-1} \bmod q\right)$ jest więc mniejsza niż $p^2q + pq^2$. Liczb całkowitych mniejszych niż $p^2q + pq^2$ będących wielokrotnością p jest $pq + q^2$, a będących wielokrotnością q jest $p^2 + pq$. Ponieważ błędy e_p i e_q są losowe to prawdopodobieństwo, że powyższa suma będzie wielokrotnością jednej z tajnych liczb p albo q, wynosi odpowiednio $\frac{pq+q^2}{p^2q+pq^2} = \frac{1}{p}$ i $\frac{p^2+pq}{p^2q+pq^2} = \frac{1}{q}$. Prawdopodobieństwo, że $\gcd(\bar{c} - c, N) \neq 1$ jest więc równe $1 - (1 - \frac{1}{q})(1 - \frac{1}{p}) = \frac{1}{q} + \frac{1}{p} - \frac{1}{pq} \leq 2\max\left(\frac{1}{q}, \frac{1}{p}\right)$. □

Obserwacja ta jest podstawą szeregu rozwiązań zabezpieczających proponowanych w literaturze, których celem jest zagwarantowanie propagacji błędów pojawiających się w obliczeniu c_p do obliczenia c_q i odwrotnie. Zgodnie z lematem 4.2 i dla standardowych rozmiarów liczb p i q prawdopodobieństwo skutecznego ataku w takim przypadku powinno być nie większe niż 2^{-511}.

Bazując na powyższych obserwacjach zaproponowano różne algorytmy propagacji błędów [15, 56, 80, 85]. Różnorodność ta wynikała z dążenia do minimalizacji złożoności obliczeniowej projektowanych algorytmów jak również z tego, że część z proponowanych implementacji okazywała się nieskuteczna, gdyż przesuwając jedynie problem kryptoanalizy do innych obszarów algorytmu [68, 78].

Dobrym przykładem prezentującym ideę rozpraszania błędów są propozycje modyfikacji algorytmu RSA-CRT zaprezentowane w pracy [85]. Zdaniem autorów tej pracy rozwiązanie określane jako CRT-2 (Alg. 4.4) zapewnia ochronę przed atakami z uszkodzeniami. W rozwiązaniu tym tajny klucz e rozdzielany jest na dwie części $e = e_r + r$ takie, że r jest małe, a e_r ma odwrotność modulo $\phi(N)$, t.j. $\gcd(e_r, \phi(N)) = 1$ oraz $d_r \equiv e_r^{-1} \bmod \phi(N)$. Dodatkowo w algorytmie wykorzystywane jet dzielenie całkowitoliczbowe $k_p = \left\lfloor \frac{m}{p} \right\rfloor$ gdzie $k_p p \leq m < (k_p + 1)p$. Analizując algorytm (Alg. 4.1) można zauważyć, że rzeczywiście każdy

Algorytm 4.1 Algorytm szyfrowania RSA-CRT zaproponowany w pracy [85]

Wejście: wiadomość $m < n$, klucz prywatny $\langle e, n\rangle$ gdzie $e = e_r + r$, $\gcd(e_r, \phi(n)) = 1$, $d_r \equiv e_r^{-1} \bmod \phi(n)$ i d_r jest małe oraz tajne liczby pierwsze p, q

Wyjście: szyfrogram c

1: oblicz $k_p = \left\lfloor \frac{m}{p} \right\rfloor$, $k_q = \left\lfloor \frac{m}{q} \right\rfloor$,

2: oblicz $c_p = m^{e_r} \bmod p$,

3: oblicz $c_q = m^{e_r} \bmod q$,

4: oblicz

$$T = \left\lfloor \frac{\left(c_p^{d_r} \bmod p + k_p p\right) + \left(c_q^{d_r} \bmod q + k_q q\right)}{2} \right\rfloor,$$

5: oblicz $c = CRT(c_p, c_q) T^r \bmod N$.

błąd wprowadzony do obliczeń c_p albo c_q powoduje błędną wartość wyrażenia T — zwanego w literaturze czynnikiem rozpraszania błędów (ang. *error diffusion term*). W ostatnim kroku algorytmu czynnik T jest przemnażany przez wynik konwersji odwrotnej wykonanej zgodnie z CRT:

$$c = \left[c_p q \left(q^{-1} \bmod p\right) + c_q p \left(p^{-1} \bmod q\right)\right] T^r \bmod N. \tag{4.7}$$

Tym samym T powoduje, że w razie wystąpienia błędów (np: w c_p) uzyskiwany błędny wynik $\bar{c}$ jest równy

$$\bar{c} = \left[\overline{T}^r \bar{c}_p q \left(q^{-1} \bmod p\right) + \overline{T}^r c_q p \left(p^{-1} \bmod q\right)\right] \bmod N. \tag{4.8}$$

Obliczając różnicę pomiędzy poprawnym i błędnym szyfrogramem otrzymamy:

$$\begin{aligned} \bar{c} - c &= \left[\overline{T}^r \bar{c}_p q \left(q^{-1} \bmod p\right) + \overline{T}^r c_q p \left(p^{-1} \bmod q\right) - c_p q \left(q^{-1} \bmod p\right) - c_q p \left(p^{-1} \bmod q\right)\right] \bmod n \\ &= \left[\left(\overline{T}^r - 1\right) c_p q \left(q^{-1} \bmod p\right) + \left(\overline{T}^r - 1\right) c_q p \left(p^{-1} \bmod q\right) + eq \left(q^{-1} \bmod p\right)\right] \bmod n. \end{aligned} \tag{4.9}$$

Zauważając, że T jest zawsze mniejsze od $N = pq$ i prowadząc analizę podobną jak w dowodzie lematu 4.2 można dojść do wniosku, że przy wprowadzeniu losowego błędu do obliczeń c_p albo c_q prawdopodobieństwo skutecznego ataku wynosi odpowiednio $\frac{1}{p}$ i $\frac{1}{q}$.

Rozwiązanie zaproponowane w pracy [85], wbrew sugestiom autorów, nie zapewnia jednak skutecznej ochrony przed kryptoanalizą. Jest tak dlatego, że atakujący może wprowadzać błędy także do innych

elementów algorytmu. Zauważmy, że jeśli atakujący wprowadzi błąd e do liczb k_p albo k_q, pozwalając jednocześnie na poprawne wykonanie obliczeń c_p i c_q, to w ostatnim kroku algorytmu (Alg. 4.1) wyznaczony zostanie błędny szyfrogram $\bar{c}$. Szyfrogram ten jest obarczony błędem będącym wielokrotnością jednej z tajnych liczb p (jeśli błąd był wprowadzony do k_p) albo q (w przeciwnym razie). Zaproponowane rozwiązanie jest więc tylko częściowym rozwiązaniem problemu i przesunięciem problemu kryptoanalizy.

Lemat 4.3 (Atak z uszkodzeniami na algorytm RSA–CRT-2)**.** Mając dany poprawny c i dwa błędne $\bar{c}_1$, $\bar{c}_2$ szyfrogramy RSA, wiadomości m, wygenerowane za pomocą algorytmu (Alg. 4.1), przy wprowadzeniu błędu do k_p albo k_q, można wyznaczyć tajne dzielniki p, q liczby N.

Dowód. Bez utraty ogólności można założyć, że błąd e został wprowadzony do liczby k_p a jej nowa, błędna wartość jest równa $\overline{k_p} = k_p + e$. Wówczas błędna wartość $\overline{T}$ wyznaczanego w czwartym kroku algorytmu szyfrowania jest równa

$$\begin{aligned} \overline{T} &= \left\lfloor \frac{(c_p^{d_r} \bmod p + \overline{k_p} p) + (c_p^{d_r} \bmod q + k_q q)}{2} \right\rfloor \\ &= \left\lfloor \frac{(c_p^{d_r} \bmod p + k_p p) + (c_p^{d_r} \bmod q + k_q q) + ep}{2} \right\rfloor = T + \left\lfloor \frac{ep}{2} \right\rfloor . \end{aligned} \tag{4.10}$$

Wzór (4.10) można przepisać oddzielnie dla przypadków gdy ep jest parzyste ($e = 2e_0$) i nieparzyste ($e = 2e_0 + 1$):

$$\overline{T} = \begin{cases} T + \frac{2e_0 p}{2} & \text{dla } ep \text{ parzystego} \\ T + \frac{2e_0 p}{2} + \frac{p-1}{2} & \text{w przeciwnym razie} \end{cases} . \tag{4.11}$$

Zauważając, że $p - 1$ jest parzyste i upraszczając zapisy możemy podstawić $t = e_0 p$ i $p_0 = \frac{p-1}{2}$.

Błędna wartość $\overline{T}$ powoduje, że w ostatnim kroku algorytmu wyznaczany jest wynik szyfrowania równy (wszystkie operacje modulo N):

$$\begin{aligned} \bar{c} &= \mathrm{CRT}\,(c_p, c_q)\overline{T}^r = \begin{cases} \mathrm{CRT}\,(c_p, c_q)(T+t)^r = \mathrm{CRT}\,(c_p, c_q) \sum_{i=0}^{r} \binom{r}{i} T^{r-i} t^i \\ \mathrm{CRT}\,(c_p, c_q)(T+t+p_0)^r = \mathrm{CRT}\,(c_p, c_q) \sum_{i=0}^{r} \binom{r}{i} (T+p_0)^{r-i} t^i \end{cases} \\ &= \begin{cases} \mathrm{CRT}\,(c_p, c_q) \left[T^r + \sum_{i=1}^{r} \binom{r}{i} T^{r-i} t^i \right] \\ \mathrm{CRT}\,(c_p, c_q) \left[(T+p_0)^r + \sum_{i=1}^{r} \binom{r}{i} (T+p_0)^{r-i} t^i \right] \end{cases} \\ &= \begin{cases} \mathrm{CRT}\,(c_p, c_q) \left[T^r + \sum_{i=1}^{r} \binom{r}{i} T^{r-i} t^i \right] \\ \mathrm{CRT}\,(c_p, c_q) \left[T^r + \sum_{i=1}^{r} \binom{r}{i} T^{r-i} {p_0}^i + \sum_{i=1}^{r} \binom{r}{i} (T+p_0)^{r-i} t^i \right] \end{cases} \\ &= \begin{cases} c + \mathrm{CRT}\,(c_p, c_q) \sum_{i=1}^{r} \binom{r}{i} T^{r-i} t^i & \text{dla } ep \text{ parzystego} \\ c + \mathrm{CRT}\,(c_p, c_q) \left[\sum_{i=1}^{r} \binom{r}{i} T^{r-i} p_0^i + \sum_{i=1}^{r} \binom{r}{i} (T+p_0)^{r-i} t^i \right] & \text{w przeciwnym razie} \end{cases} . \end{aligned} \tag{4.12}$$

Różnica $\bar{c} - c$ jest więc równa:

$$\bar{c} - c = \begin{cases} \mathrm{CRT}\,(c_p, c_q) \sum_{i=1}^{r} \binom{r}{i} T^{r-i} t^i & \text{dla } ep \text{ parzystego} \\ \mathrm{CRT}\,(c_p, c_q) \left[\sum_{i=1}^{r} \binom{r}{i} T^{r-i} p_0^i + \sum_{i=1}^{r} \binom{r}{i} (T+p_0)^{r-i} t^i \right] & \text{w przeciwnym razie} \end{cases} . \tag{4.13}$$

Każdy ze składników sum $\sum_{i=1}^{r}\binom{r}{i}T^{r-i}t^i$ i $\sum_{i=1}^{r}\binom{r}{i}(T+p_0)^{r-i}t^i$ jest wielokrotnością t, a więc i wielokrotnością p, a składnik $\sum_{i=1}^{r}\binom{r}{i}T^{r-i}p_0^i$ jest stały dla ustalonej wiadomości m. Jeśli atakujący wprowadzi błąd taki, że ep jest parzyste, to, pomijając mało prawdopodobną sytuację $q|(\overline{c}-c)$, różnica $\overline{c}-c$ jest wielokrotnością p i $\gcd(\overline{c}-c,N)=p$ pozwala na przeprowadzenie ataku. W przeciwnym razie (ep jest nieparzyste) atakujący musi uzyskać dwa błędne szyfrogramy $\overline{c}_1 \neq \overline{c}_2$ tej samej wiadomości m, obarczone błędami e_1 i e_2. Różnica tych szyfrogramów jest równa

$$\begin{aligned}
\overline{c}_1-\overline{c}_2 &= \mathrm{CRT}\,(c_p,c_q)\left[\sum_{i=1}^{r}\binom{r}{i}T^{r-i}p_0^i+\sum_{i=1}^{r}\binom{r}{i}(T+p_0)^{r-i}{t_1}^i\right]+\\
&\quad-\mathrm{CRT}\,(c_p,c_q)\left[\sum_{i=1}^{r}\binom{r}{i}T^{r-i}p_0^i+\sum_{i=1}^{r}\binom{r}{i}(T+p_0)^{r-i}{t_2}^i\right]\\
&= \mathrm{CRT}\,(c_p,c_q)\left[\sum_{i=1}^{r}\binom{r}{i}(T+p_0)^{r-i}{t_1}^i-\sum_{i=1}^{r}\binom{r}{i}(T+p_0)^{r-i}{t_2}^i\right], \qquad (4.14)
\end{aligned}$$

i w każdym jej składniku występuje t_1 albo t_2, każde będące wielokrotnością p. Oznacza to, że $\gcd(\overline{c}_1-\overline{c}_2,N)=p$ i atakujący może przeprowadzić atak.

Do przeprowadzenia ataku wystarczające są więc trzy szyfrogramy — poprawny c i dwa błędne $\overline{c}_1$, $\overline{c}_2$ wygenerowane dla tej samej wiadomości m. □

Jak pokazuje przykład algorytmu CRT-2, implementacja algorytmów rozpraszania błędów nie jest prosta. W celu zapewnienia wysokiego poziomu bezpieczeństwa powinna ona spełniać następujące wymagania:

- zapewniać odporność na błędy przemijające i trwałe wprowadzane do różnych elementów algorytmu kryptograficznego,
- w razie wystąpienia błędów generować wyniki utrudniające bądź uniemożliwiające przeanalizowanie i odtworzenie zależności błędu od wykorzystywanego klucza kryptograficznego czy przetwarzanej wiadomości,
- być silnie zintegrowana z algorytmem kryptograficznym.

W ochronie przed kryptoanalizą z uszkodzeniami szczególne znaczenie ma silna integracja rozwiązania z algorytmem. Chodzi o to, aby atakujący nie był w stanie wyłączyć procedur ochronnych lub pominąć operacji odpowiedzialnych za sprawdzanie poprawności działania czy rozpraszanie błędów. Integracja powinna zapewniać, że w przypadku prób wyłączenia obwodów sprawdzających uzyskiwany wynik działania algorytmu będzie losowy bez prostych do analizy związków z używanym kluczem czy wiadomością.

Zaletą stosowania tego typu rozwiązań ochronnych jest niezależność od rodzaju wprowadzanych błędów. Ograniczeniem w zastosowaniach jest natomiast trudność analizy algorytmu i propagacji błędów oraz

konieczność projektowania mechanizmów propagacji błędów dla każdego rodzaju algorytmów oddzielnie. Z tych powodów gruntownie przebadano algorytm RSA i RSA-CRT, znacznie mniej uwagi poświęcając algorytmom symetrycznym i innym algorytmom asymetrycznym (m.in. ElGamal, DSA). Propozycja mechanizmów propagacji błędów dla schematów podpisów ElGamala i DSA zostanie zaprezentowana w rozdziale 6.

Rozdział 5

Ochrona algorytmu AES

W niniejszym rozdziale przedstawiona jest propozycja ochrony algorytmu AES przed kryptoanalizą z uszkodzeniami. Propozycja wykorzystuje kontrolę parzystości w celu zagwarantowania poprawności generowanych szyfrogramów. Cel ten jest osiągany poprzez rozszerzenie wejściowej macierzy stanu o bajty i bity parzystości. Pozwalają one wyznaczyć przewidywaną parzystość po kolejnej transformacji. Na tej podstawie, oraz znając rzeczywistą parzystości po transformacji, wyznaczana jest macierz korygująca

$$\mathbb{C} = [c_{i,j}]_{4\times 4}\,. \tag{5.1}$$

Wyznaczona macierz $\mathbb{C}$ jest następnie dodawana, za pomocą operacji `XOR`, do macierzy stanu $\mathbb{S}'$ wyznaczonej w wyniku wykonania transformacji algorytmu AES. Dodanie macierzy $\mathbb{C}$ pozwala na korekcję ewentualnych błędów zaistniałych w trakcie wykonywania transformacji

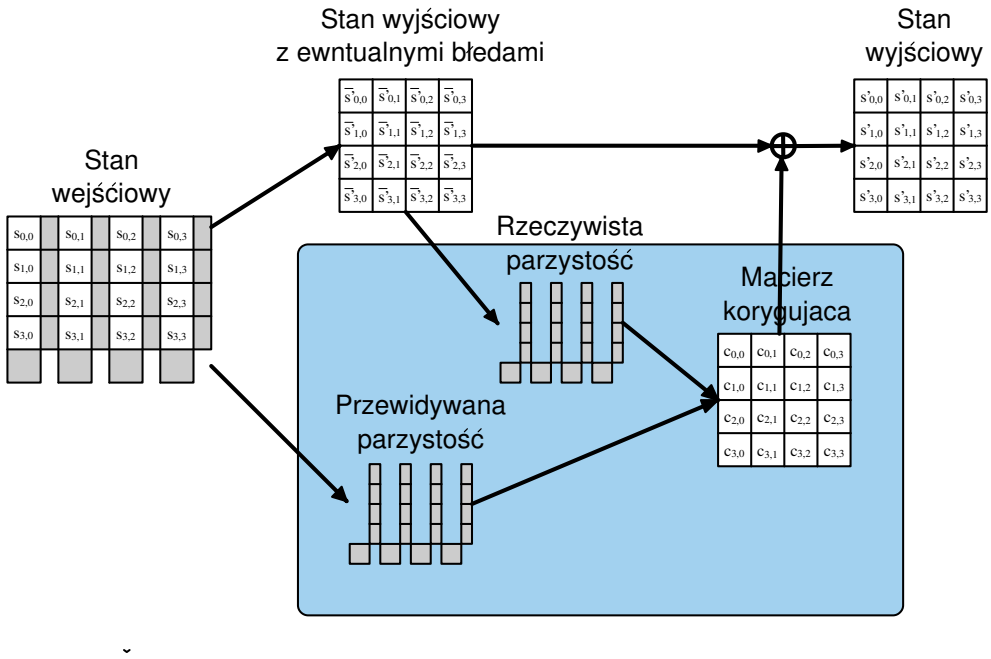
$$\mathbb{S} = \mathbb{S}' \oplus \mathbb{C} = \left[s'_{i,j} \oplus c_{i,j}\right]_{t\times 4} \tag{5.2}$$

Jeśli do urządzenia nie wprowadzono błędów, to wszystkie procedury korekcji przebiegają tak samo, a jedynie elementy macierzy korygującej mają wartość 0. Schemat proponowanego rozwiązania przedstawia rysunek 5.1.

5.1 Model błędów

Na podstawie przeprowadzonej analizy błędów wykorzystywanych w kryptoanalizie z uszkodzeniami algorytmu AES (rozdział 3.3) w dalszej części niniejszego rozdziału będziemy przyjmować założenie, że atakujący wprowadza błędy losowe do pojedynczych bajtów macierzy stanu algorytmu AES.

Definicja 5.1 (Błędy wprowadzane do algorytmu AES)**.** Niech $\mathbb{S}$ i $\mathbb{W}_j$ oznaczają odpowiednio macierz stanu algorytmu AES i j-tą kolumnę tej macierzy. Atakujący, w trakcie wykonywania dowolnej transformacji



Rysunek 5.1: Schemat rozwiązania ochronnego dla algorytmu AES.

algorytmu AES, może wprowadzić uszkodzenie powodujące losowy błąd w pojedynczym, dowolnie wybranym bajcie $s_{i,j}$ kolumny $\mathbb{W}_j$.

Model błędu wprowadzanego do algorytmu AES	
rodzaj błędu	dowolny
liczba zmienionych bitów	losowa ≤ 8
miejsce wprowadzenia błędu	pełna kontrola — atakujący może wybrać bajt macierzy stanu do którego wprowadza błąd
moment wprowadzenia	pełna kontrola — atakujący może wybrać transformację po której wprowadza błąd
czas trwania	błędy przemijające

Zgodnie z przyjętym modelem błędu atakujący ma pełną kontrolę nad momentem wprowadzenia błędu i może je wprowadzać zarówno w trakcie wykonywania konkretnej transformacji jak i pomiędzy transformacjami. W sprzętowych implementacjach algorytmu AES wprowadzanie uszkodzeń w trakcie wykonywania transformacji jest często trudne. Trudności zależą od sposobu implementacji algorytmu i są największe dla implementacji wykorzystujących tablicowanie w celu przyspieszenia działania algorytmu. W takiej sytuacji atakujący może jedynie przekłamywać adresy w tablicach lub zmieniać dane wejściowe do transformacji, mając na celu wprowadzenie określonych błędów. Wspólną cechą wszystkich implementacji

jest natomiast przechowywanie wyników pośrednich transformacji i rund, co stwarza dogodną możliwość przeprowadzenia ataku.

Osobne założenie dotyczy poprawności procedur generacji kluczy rund. Zgodnie z przyjętym modelem błędów atakujący wprowadza przekłamania do macierzy stanu algorytmu AES, podczas gdy procedura rozszerzania klucza i sam klucz rundy nie są obarczone błędem.

Bazując na przyjętym modelu błędu można zauważyć, że atakujący może jednocześnie wprowadzić najwyżej cztery błędy, do różnych kolumn macierzy stanu.Wprowadzone błędy można więc zapisać w postaci macierzy

$$\mathbb{E} = [e_{i,j}]_{4\times 4}\,. \tag{5.3}$$

5.2 Predykcja bitów i bajtów parzystości

Proponowana metoda korygowania uszkodzeń bazuje na kontroli parzystości poszczególnych kolumn macierzy stanu $\mathbb{S}$ algorytmu AES. Standardowa kolumna $\mathbb{W}_j$ tej macierzy jest rozszerzana o dodatkowe bity parzystości $p_{i,j}$ przypisane do każdego bajta $s_{i,j}$ danych i o bajt parzystości p_i przypisany do całej kolumny $\mathbb{W}_j$.

Definicja 5.2 (Bity i bajty parzystości)**.** Niech $\mathbb{S}, \mathbb{W}_j, s_{i,j}$ i $s_{i,j}^{(k)}$ oznaczają odpowiednio stan algorytmu AES, j-tą kolumnę stanu, i-ty element (bajt) tej kolumny oraz k-ty bit elementu. Bity parzystości $p_{i,j}$ przypisane do poszczególnych bajtów $s_{i,j}$ są wyznaczane jako

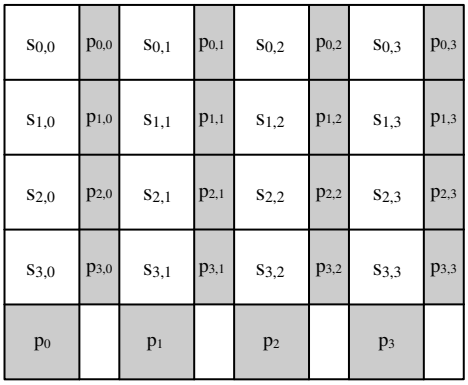
$$p_{i,j} = \bigoplus_{k=0}^{7} s_{i,j}^{(k)}. \tag{5.4}$$

Bajt parzystości p_j dla kolumny $\mathbb{W}_j$ jest zdefiniowany jako

$$p_j = \left(p_j^{(7)} \| p_j^{(6)} \| p_j^{(5)} \| p_j^{(4)} \| p_j^{(3)} \| p_j^{(2)} \| p_j^{(1)} \| p_j^{(0)}\right) \text{ gdzie } p_j^{(k)} = \bigoplus_{i=0}^{3} s_{i,j}^{(k)}. \tag{5.5}$$

Zgodnie z definicją (Def. 5.2) każda kolumna, interpretowana jako macierz o rozmiarze 4×8 bitów, jest rozszerzona o dodatkową kolumnę i wiersz złożony z bitów parzystości. Powstały w ten sposób kod detekcyjny jest znany pod nazwą kodu macierzowego (ang. *matrix code*).

Odległość kodowa kodu macierzowego, $d = 3$, umożliwia wykrycie wszystkich błędów podwójnych i korekcję wszystkich błędów pojedynczych. Właściwości te, w aspekcie ataków z uszkodzeniami na algorytm AES (rozdział 3.3), mogą się wydawać nieprzydatne, jednak jak pokazano w pracach autora [10, 26, 27] kod ten pozwala na wykrywanie wszystkich rodzajów błędów wykorzystywanych w atakach z uszkodzeniami na AES [14, 25, 31, 35] oraz korygowanie większości z nich. Zaletą kodu macierzowego zastosowanego do algorytmu AES jest stosunkowo mały narzut implementacyjny. Zastosowanie bitów



Rysunek 5.2: Macierz stanu algorytmu AES z bitami i bajtami parzystości.

parzystości stowarzyszonych z każdym bajtem macierzy stanu $\mathbb{S}$ algorytmu AES zostało zaproponowane m.in. w pracach [7, 8, 70, 79]. Najdokładniejszy opis wykorzystania dodatkowych bitów parzystości można znaleźć w pracy [7], w której szczegółowo przeanalizowano zmiany bitów parzystości po każdej transformacji. W pracy tej zaproponowano metody przewidywania parzystości wyjściowej poszczególnych bajtów stanu dla każdej transformacji. Predykcja ta wykonywana jest na podstawie znajomości macierzy stanu przed transformacją oraz wejściowych bitów parzystości odpowiadających poszczególnym bajtom.

Predykcja bitów parzystości dla transformacji AddRoundKey

Zgodnie z definicją transformacji AddRoundKey (2.6) wyjściowy bajt stanu $\mathbb{S}$ jest wyznaczany jako suma `XOR` bajta stanu wejściowego i odpowiadającego mu bajta klucza rundy: $s'_{i,j} = s_{i,j} \oplus k_{i,j}$. W takim przypadku parzystość bajta $s'_{i,j}$ jest równa

$$p'_{i,j} = p\left(s'_{i,j}\right) = p\left(s_{i,j} \oplus k_{i,j}\right) = p_{i,j} \oplus p\left(k_{i,j}\right). \tag{5.6}$$

Oznacza to, że przewidywane bity parzystości można obliczyć w oparciu o znajomość parzystości bajtów stanu przed transformacją i parzystości odpowiadających im bajtów klucza rundy.

Predykcja bitów parzystości dla transformacji ShiftRows

Transformacja ShiftRows realizuje cykliczne przesunięcie wierszy macierzy stanu pozostawiając bez zmiany wartości poszczególnych bajtów macierzy $\mathbb{S}$. Z tego względu przewidywane bity parzystości są dokładnie takie same jak bity parzystości przed transformacją

$$p'_{i,j} = p_{i,j+i \bmod 4}. \tag{5.7}$$

Predykcja bitów parzystości dla transformacji MixColumns

Transformacja MixColumns operuje na całej kolumnie macierzy stanu, zmieniając wszystkie bajty tej kolumny. Ze względu na dobór parametrów tej transformacji przewidywanie bitów parzystości jest stosunkowo proste i zależy od bitów parzystości przypisanych do całej kolumny przed transformacją i najbardziej znaczących bitów 2 z 4 elementów tej kolumny (wyprowadzenie zależności pozwalających na predykcję parzystości można znaleźć w pracy [7]):

$$\begin{aligned} p'_{0,j} &= p_{0,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{0,j} \oplus s^{(7)}_{1,j} \\ p'_{1,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{3,j} \oplus s^{(7)}_{1,j} \oplus s^{(7)}_{2,j} \\ p'_{2,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s^{(7)}_{2,j} \oplus s^{(7)}_{3,j} \\ p'_{3,j} &= p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{3,j} \oplus s^{(7)}_{0,j}. \end{aligned} \quad .$$

Zgodnie z wzorem (5.8) przewidywanie bitów parzystości po transformacji MixColumns można wykonać w oparciu o parzystości danych wejściowych i najbardziej znaczące bity tych danych.

Predykcja bitów parzystości dla transformacji SubBytes

Ze względu na to, że transformacja SubBytes jest przekształceniem nieliniowym, predykcja parzystości na podstawie zależności logicznych jest skomplikowana, a jej implementacja nieefektywna. Z tego względu w wielu pracach (m.in. [7, 79]) zaproponowano rozbudowanie tablicy S-box standardowo wykorzystywanej do realizacji transformacji. W proponowanym rozwiązaniu wejściem do rozbudowanej tablicy XS-box jest wektor 9 bitowy składający się z 8 bitów danych — bajta macierzy stanu $s_{i,j}$, oraz ich bitu parzystości — $p_{i,j}$. Jeśli bit parzystości jest poprawny (tj. $p_{i,j} = \bigoplus_{k=0}^{7} s^{(k)}_{i,j}$), to wyjściem jest poprawna wartości $s'_{i,j}$ oraz poprawny bit parzystości $p'_{i,j}$. Jeśli $p_{i,j}$ jest niepoprawne, to wyjściem jest $s'_{i,j}$ oraz $\overline{p'_{i,j}}$. Przewidywana wartość bitów parzystości jest więc równa

$$p'_{i,j} = \begin{cases} \bigoplus_{k=0}^{7} s'^{(k)}_{i,j} & \text{dla } p_{i,j} = \bigoplus_{k=0}^{7} s^{(k)}_{i,j} \\ 1 \oplus \bigoplus_{k=0}^{7} s'^{(k)}_{i,j} & \text{dla } p_{i,j} \neq \bigoplus_{k=0}^{7} s^{(k)}_{i,j} \end{cases} . \tag{5.8}$$

Zastosowanie bajtów parzystości stowarzyszonych z każdą kolumną $\mathbb{W}_j$ macierzy stanu $\mathbb{S}$ algorytmu AES zaproponowano w pracach [10, 26, 27], gdzie podano metody przewidywania parzystości wyjściowej poszczególnych kolumn stanu dla każdej transformacji. Predykcja ta wykonywana jest na podstawie znajomości macierzy stanu przed transformacją oraz wejściowych bajtów parzystości odpowiadających poszczególnym kolumnom. Zaprezentowanie możliwości realizacji predykcji bajtów parzystości wmaga wcześniejszego udowodnienia właściwość transformacji MixColumns polegającej na zachowaniu parzystości w kolumnie macierzy stanu.

Lemat 5.1 (Parzystość kolumny stanu w transformacji MixColumns)**.** Niech $\mathbb{W}_j = [s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}]^T$ oznacza j–tą kolumnę stanu AES, której parzystość jest równa p_j. Parzystość j–tej kolumny po poprawnie wykonanej transformacji MixColumns nie zmienia się.

Dowód. Transformacja MixColumns jest zdefiniowana wzorem 2.7, który, korzystając z właściwości operacji w ciele GF(2^8), można przekształcić do postaci:

$$\begin{aligned}
w'_j(x) &= c(x)w_j(x) \bmod p(x) \\
&= \left(03x^3 + 01x^2 + 01x + 02\right)\left(s_{0,j}x^3 + s_{1,j}x^2 + s_{2,j}x + s_{3,j}\right) \bmod p(x) \\
&= \left[03s_{0,j}x^6 + (03s_{1,j} \oplus 01s_{0,j})\,x^5 + (03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j})\,x^4\right. \\
&\quad + (03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1})\,x^3 + (01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j})\,x^2 \\
&\quad \left. + (01s_{3,j} \oplus 02s_{2,j})\,x + 02s_{3,j}\right] \bmod p(x) \\
&= + (03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1})\,x^3 + (01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j} \oplus 03s_{0,j})\,x^2 \\
&\quad + (01s_{3,j} \oplus 02s_{2,j} \oplus 03s_{1,j} \oplus 01s_{0,j})\,x + (02s_{3,j} \oplus 03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j})
\end{aligned} \tag{5.9}$$

Zgodnie z powyższym przekształceniem nowe elementy wyjściowej kolumny $\mathbb{W}'_j$ są równe:

$$\begin{aligned}
s'_{0,j} &= 03s_{3,j} \oplus 01s_{2,j} \oplus 01s_{1,j} \oplus 02s_{0,1} \\
s'_{1,j} &= 01s_{3,j} \oplus 01s_{2,j} \oplus 02s_{1,j} \oplus 03s_{0,j} \\
s'_{2,j} &= 01s_{3,j} \oplus 02s_{2,j} \oplus 03s_{1,j} \oplus 01s_{0,j} \\
s'_{3,j} &= 02s_{3,j} \oplus 03s_{2,j} \oplus 01s_{1,j} \oplus 01s_{0,j}.
\end{aligned} \quad .$$

Mając na uwadze, że $03 = 02 \oplus 01$ oraz $a \oplus a = 0$, możemy wyznaczyć wyjściowy bajt parzystości j–tej kolumny:

$$\begin{aligned}
p'_j &= \bigoplus_{i=0}^{3} s'_{i,j} = \bigoplus_{i=0}^{3} (03 \oplus 01 \oplus 01 \oplus 02)\, s_{i,j} \\
&= \bigoplus_{i=0}^{3} s_{i,j} = p_j.
\end{aligned} \tag{5.10}$$

To kończy dowód. □

Twierdzenie 5.1 (Predykcja bajtów parzystości)**.** Niech p_j oznacza bajt parzystości j–tej kolumny $\mathbb{W}_j = [s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j}]^T$ macierzy stanu $\mathbb{S}$ algorytmu AES. Wyjściowy bajt parzystości p'_j, j–tej kolumny macierzy stanu po transformacji, może być wyznaczony na podstawie znajomości bajta parzystości p_j i danych wejściowych $s_{i,j}$.

Dowód. Poprawność twierdzenia udowodnimy oddzielnie dla każdej transformacji algorytmu AES.

Predykcja bajtów parzystości dla transformacji AddRoundKey. Zgodnie z definicją transformacji

AddRoundKey (2.6) każdy wyjściowy bajt stanu $\mathbb{S}$ jest wyznaczany jako $s'_{i,j} = s_{i,j} \oplus k_{i,j}$. W takim przypadku parzystość kolumny $\mathbb{W}'_j$ po transformacji wynosi:

$$p'_j = p\left(\mathbb{W}'_j\right) = \bigoplus_{i=0}^{3} (s_{i,j} \oplus k_{i,j}) = \bigoplus_{i=0}^{3} s_{i,j} \oplus \bigoplus_{i=0}^{3} k_{i,j} = p_j \oplus \bigoplus_{i=0}^{3} k_{i,j}. \tag{5.11}$$

Zgodnie z wzorem (5.11) wyjściowy bajt parzystości dla j–tej kolumny — p'_j, po transformacji AddRoundKey, może być wyznaczony jako suma `XOR` wejściowego bajta parzystości p_j i odpowiadającego mu bajta parzystości j–tej kolumny macierzy klucza — $\bigoplus_{i=0}^{3} k_{i,j}$.

Predykcja bajtów parzystości dla transformacji ShiftRows. Zgodnie z definicją tej transformacji (2.11), kolejne wiersze macierzy stanu $\mathbb{S}$ są przesuwane cyklicznie o stałą liczbę pozycji. Zgodnie z definicją tylko pierwszy wiersz macierzy stanu nie jest przesuwany, podczas gdy kolejne są przesuwane o 1, 2 i 3 pozycje bajtowe. Mając na uwadze, że $p_j = \bigoplus_{i=0}^{3} s_{i,j}$ (5.5), wyjściowy bajt parzystości p'_j można zapisać jako

$$p'_j = p\left(\mathbb{W}'_j\right) = \bigoplus_{i=0}^{3} s'_{i,j} = \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} = \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} \oplus p_j \bigoplus_{i=0}^{3} s_{i,j} = p_j \bigoplus_{i=1}^{3} \left(s_{i,j} \oplus s_{i,j+i \bmod 4}\right). \tag{5.12}$$

Zgodnie z powyższym wzorem wyjściowy bajt parzystości po transformacji ShiftRows może zostać wyznaczony na podstawie wejściowego bajta parzystości i danych wejściowych.

Predykcja bajtów parzystości dla transformacji MixColumns. Zgodnie z definicją tej transformacji przekształca ona całą kolumnę macierzy stanu $\mathbb{W}_j$ do nowej postaci $\mathbb{W}'_j$. Opisuje to przekształcenie (2.7), która nie zmienia parzystości kolumny macierzy (Lemat 5.1). Z tego względu przewidywanie parzystości po transformacji jest trywialne

$$p'_j = p_j. \tag{5.13}$$

Predykcja bajtów parzystości dla transformacji SubBytes. Transformacja SubBytes jest nieliniowym przekształceniem pojedynczych bajtów macierzy stanu $\mathbb{S}$. Ponieważ bajt parzystości p'_j jest sumą bajtów $s'_{i,j}$ dla $i = 0, 1, 2, 3$, to p'_j jest nieliniową funkcją 4 bajtów $s_{i,j}$ wejściowej macierzy stanu. Z tego względu, gdyby do przewidywania parzystości p'_j chcieć zastosować tablicowanie, podobnie jak zaproponowano w pracy [7] dla bitów parzystości, to rozmiar tej tablicy wynosiłby $2^{40} \times 2^{40}$ (uwzględniając 32 bity danych $s_{i,j}$ i 8 bitów bajta parzystości p_j). Taki rozmiar tablicy nie jest akceptowalny i z tego względu w pracach [10, 26, 27] zaproponowano wykonanie predykcji parzystości w oparciu o algebraiczne zależności opisujące transformacje SubBytes.

Zgodnie z definicją transformacji (2.12) wynikowy bajt jest równy

$$s'_{i,j} = a\ s_{i,j}^{-1} \oplus 63. \tag{5.14}$$

a odwrotności multiplikatywne są wyznaczane w ciele GF(2^8). Wyjściowy bajt parzystości jest więc równy

$$p'_j = \bigoplus_{i=0}^{3} s'_{i,j} = \bigoplus_{i=0}^{3} \left(a\ s_{i,j}^{-1} \oplus 63 \right). \tag{5.15}$$

Mając na względzie, że $\bigoplus_{i=0}^{3} a\ s_{i,j}^{-1} = a\ \bigoplus_{i=0}^{3} s_{i,j}^{-1}$ oraz $63 \oplus 63 = 0$ otrzymujemy

$$p'_j = a \left(\bigoplus_{i=0}^{3} s_{i,j}^{-1} \right). \tag{5.16}$$

Ostatnie równanie może być uzależnione od parzystości wejściowej p_j jeśli zapiszemy $s_{0,j} = p_j \bigoplus_{i=1}^{3} s_{i,j}$. Wówczas wyjściowy bajt parzystości może być zapisany w postaci

$$p'_j = a \left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j} \right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1} \right). \tag{5.17}$$

Zgodnie ze wzorem (5.17) wyjściowy bajt parzystości może być wyznaczony za pomocą wejściowego bajta parzystości i danych wejściowych j-tej kolumny macierzy stanu.

Z równań (5.11), (5.12), (5.13) i (5.17) wynika, że wyjściowy bajt parzystości p'_j można wyznaczyć na podstawie danych wejściowych $s_{i,j}$ i wejściowego bajta parzystości p_j. To kończy dowód. □

Bity i bajty parzystości przypisane do bajtów i kolumn macierzy stanu oraz algorytmy ich predykcji dla każdej transformacji AES pozwalają porównywać przewidziane bity/bajty z rzeczywistą parzystością wyników kolejnych transformacji. Porównanie parzystości przewidzianej i rzeczywistej pozwala wykryć, zlokalizować i skorygować wprowadzone błędy.

5.3 Lokalizacja wprowadzonych błędów

Lokalizacja błędów wprowadzonych do macierzy stanu algorytmu AES jest wykonywana oddzielnie dla każdej kolumny macierzy stanu. Postępowanie takie wynika z zastosowania do ochrony algorytmu kodu macierzowego obejmującego każdą kolumnę macierzy stanu. Zgodnie z analizą ataków z uszkodzeniami na algorytm AES (rozdział 3.3) będziemy dalej zakładali, że atakujący wprowadza błędy do pojedynczych bajtów kolumny macierzy stanu.

Lokalizacja wprowadzonego błędu składa się z dwóch etapów:

1. określenia maski błędu na wyjściu transformacji — na podstawie bajtów parzystości przypisanych do kolumn macierzy stanu, możliwe jest wyznaczenie maski błędu, która zmieniła stan $\mathbb{S}$ po transformacji,

2. określenia, które bajty kolumny macierzy stanu po transformacji zostały zmienione przez wprowadzony błąd — jest to możliwe na podstawie znajomości bitów parzystości przypisanych do bajtów macierzy stanu.

Obie procedury opisane w dalszej części rozdziału są ze sobą powiązane. Dla transformacji SubBytes możliwość określenia maski błędu wymaga znajomości, które słowo kolumny stanu zostało zmienione przez błąd. Podobnie jest dla MixColumns, gdzie określenie słowa zmienionego przez błąd wymaga wcześniejszego określenia maski błędu na podstawie bajtów parzystości.

Znajomość przewidywanych i rzeczywistych bajtów parzystości przypisanych do kolumn macierzy stanu pozwala określić, które bity bajtów należących do kolumny stanu zostały zmienione przez wprowadzony błąd. W celu uproszczenia dalszej analizy przydatne jest zbadanie jak zmieniają się rzeczywiste bajty parzystości p_j^R po kolejnych transformacjach.

Zgodnie z definicją bajta parzystości (Def. 5.2) mamy

$$p_j^R = \bigoplus_{i=0}^{3} s'_{i,j} \tag{5.18}$$

gdzie $s'_{i,j}$ oznacza wartość słowa i, j po transformacji. Uzależniając rzeczywistą parzystość od wartości bajtów macierzy stanu sprzed transformacji $s_{i,j}$ i korzystając z twierdzenia 5.1 otrzymamy:

$$\begin{aligned}
p_j^R &= \bigoplus_{i=0}^{3} (s_{i,j} \oplus k_{i,j}) && \text{dla transformacji AddRoundKey} \\
p_j^R &= \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} && \text{dla transformacji ShiftRows} \\
p_j^R &= \bigoplus_{i=0}^{3} s_{i,j} && \text{dla transformacji MixColumns} \\
p_j^R &= a \bigoplus_{i=0}^{3} s_{i,j}^{-1} && \text{dla transformacji SubBytes.}
\end{aligned}$$

Dzięki możliwości wyznaczenia przewidywanej parzystości po każdej transformacji p'_j (Lemat 5.1) można ją porównać z parzystością rzeczywistą i określić jaki błąd został wprowadzony.

Twierdzenie 5.2 (Określenie maski wprowadzonego błędu)**.** Niech $\mathbb{W}_j$, p_j i $s_{i,j}$ oznaczają odpowiednio j-te słowo macierzy stanu, parzystość tego słowa oraz i-ty element słowa, a p'_j i p_j^R przewidywaną i rzeczywistą parzystość $\mathbb{W}_j$ po transformacji. Jeśli atakujący wprowadza błąd $e_{k,j}$ do pojedynczego, losowo wybranego bajta $s_{k,j}$, słowa $\mathbb{W}_j$, to na podstawie porównania rzeczywistej i przewidywanej parzystości po transformacji można określić jaki błąd został wprowadzony.

Dowód. Podobnie jak poprzednio dowód przeprowadzimy oddzielnie dla każdej transformacji.

Transformacja AddRoundKey. W razie wprowadzenia błędu $e_{k,j}$ przewidywana i rzeczywista wartość bajta parzystości po transformacji AddRoundKey wynoszą odpowiednio:

$$p'_j = p_j \bigoplus_{i=0}^{3} k_{i,j} = \bigoplus_{i=0}^{3} (s_{i,j} \oplus k_{i,j}), \tag{5.19}$$

$$p_j^R = \bigoplus_{i=0}^{3} (s_{i,j} \oplus k_{i,j}) \oplus e_{k,j}. \tag{5.20}$$

Wyznaczając sumę `XOR` obu parzystości otrzymamy wartość błędu $e_{k,j}$ wprowadzonego do j-tej kolumny macierzy stanu

$$e_{k,j} = p'_j \oplus p_j^R. \tag{5.21}$$

Transformacja ShiftRows. Dla transformacji ShiftRows rzeczywista i przewidywana parzystość wynoszą

$$p'_j = \begin{cases} p_j \bigoplus_{i=1}^{3} (s_{i,j} \oplus s_{i,j+i \bmod 4}) & \text{dla } k = 0 \\ p_j \bigoplus_{i=1}^{3} (s_{i,j} \oplus s_{i,j+i \bmod 4}) \oplus e_{k,j} & \text{dla } k = 1,2,3 \end{cases}, \tag{5.22}$$

$$p_j^R = \begin{cases} \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} \oplus e_{k,j} & \text{dla } k = 0 \\ \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} & \text{dla } k = 1,2,3 \end{cases}. \tag{5.23}$$

Pamiętając, że $p_j = \bigoplus_{i=0}^{3} s_{i,j}$ suma $p'_j \oplus p_j^R$ wynosi:

$$\begin{aligned} p'_j \oplus p_j^R &= \begin{cases} p_j \bigoplus_{i=1}^{3} (s_{i,j} \oplus s_{i,j+i \bmod 4}) \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} \oplus e_{k,j} & \text{dla } k = 0 \\ p_j \bigoplus_{i=1}^{3} (s_{i,j} \oplus s_{i,j+i \bmod 4}) \oplus e_{k,j} \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} & \text{dla } k = 1,2,3 \end{cases} \\ &= p_j \bigoplus_{i=1}^{3} (s_{i,j} \oplus s_{i,j+i \bmod 4}) \bigoplus_{i=0}^{3} s_{i,j+i \bmod 4} \oplus e_{k,j} \\ &= p_j \bigoplus_{i=1}^{3} s_{i,j} \bigoplus_{i=1}^{3} s_{i,j+i \bmod 4} \oplus s_{0,j} \bigoplus_{i=1}^{3} s_{i,j+i \bmod 4} \oplus e_{k,j} \\ &= e_{k,j}. \end{aligned} \tag{5.24}$$

Warto zauważyć, że taki sam wynik otrzymamy wtedy, gdy atakujący wprowadzi jednocześnie błędy do innych kolumn stanu. Właściwość taka wynika z tego, że bajty z innych kolumn — $s_{i,j+i \bmod 4}$, występują w obu zależnościach definiujących rzeczywisty i przewidywany bajt parzystości po transformacji ShiftRows. Tym samym dzięki obliczeniu $p'_j \oplus p_j^R$ błędy wprowadzone do innych kolumn niż j zostaną zamaskowane, i nie wpłyną na wyznaczanie maski błędu dla tej kolumny.

Transformacja MixColumns. Na podstawie lematu 5.1, rzeczywisty i przewidywany bajt parzystości

są równe

$$p'_j = p_j, \tag{5.25}$$

$$p_j^R = \bigoplus_{i=0}^{3} s_{i,j} \oplus e_{k,j}. \tag{5.26}$$

Suma XOR obu parzystości jest więc równa

$$p'_j \oplus p_j^R = p_j \bigoplus_{i=0}^{3} s_{i,j} \oplus e_{k,j} = e_{k,j}. \tag{5.27}$$

Transformacja SubBytes. W transformacji SubBytes rzeczywista i przewidywana parzystość wynoszą

$$\begin{aligned} p'_j &= \begin{cases} a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1}.\right) & \text{dla } k=0 \\ a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j} \oplus e_{k,j}\right)^{-1} \bigoplus_{i=1, i\neq k}^{3} s_{i,j}^{-1} \oplus (s_{k,j} \oplus e_{k,j})^{-1}\right) & \text{dla } k=1,2,3 \end{cases} \\ &= \begin{cases} a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1}.\right) & \text{dla } k=0 \\ a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1} \oplus e'_{k,j} \oplus e'_{1,j}\right) & \text{dla } k=1,2,3 \end{cases}, \end{aligned} \tag{5.28}$$

$$p_j^R = a \bigoplus_{i=0, i\neq k}^{3} s_{i,j}^{-1} \oplus (s_{k,j} \oplus e_{k,j})^{-1} = a \bigoplus_{i=0}^{3} s_{i,j}^{-1} \oplus e'_{k,j}. \tag{5.29}$$

We wzorach (5.28) i (5.29) czynniki $e'_{k,j}$ i $e'_{1,k}$ oznaczają błędy zniekształcające wyjściowe słowa $s'_{k,j}$ i $s'_{1,j}$. Błąd $e'_{k,j}$ jest bezpośrednim efektem wprowadzenia błędu $e_{k,j}$ i nieliniowości transformacji SubBytes

$$\text{SubBytes}\,(s_{k,j} \oplus e_{k,j}) = \text{SubBytes}\,(s_{k,j}) \oplus a\ e'_{k,j}. \tag{5.30}$$

Błąd $e'_{1,j}$ jest natomiast konsekwencją zaproponowanej metody przewidywania parzystości (5.17) i występującego w niej czynnika $\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1}$, który dla każdego błędu wprowadzonego do bajta $s_{k,j}$ dla $k = 1,2,3$ przyjmuje wartość $(s_{1,j} \oplus e_{k,j})^{-1}$. Przewidywana parzystość jest więc dodatkowo obarczona błędem $e'_{1,j}$ równym

$$\text{SubBytes}\,(s_{1,j} \oplus e_{k,j}) = \text{SubBytes}\,(s_{1,j}) \oplus a\ e'_{1,j}. \tag{5.31}$$

Suma XOR obu parzystości wynosi więc

$$\begin{aligned} p'_j \oplus p_j^R &= \begin{cases} a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1}\right) \oplus a\ \bigoplus_{i=0}^{3} s_{i,j}^{-1} \oplus e'_{k,j} & \text{dla } k=0 \\ a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1} \oplus e'_{k,j} \oplus e'_{1,j}\right) \oplus a\ \bigoplus_{i=0}^{3} s_{i,j}^{-1} \oplus e'_{k,j} & \text{dla } k=1,2,3 \end{cases} \\ &= \begin{cases} a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \oplus s_{0,j}^{-1} \oplus e'_{k,j}\right) & \text{dla } k=0 \\ a\left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j}\right)^{-1} \oplus s_{0,j}^{-1} \oplus e'_{1,j}\right) & \text{dla } k=1,2,3 \end{cases} \\ &= \begin{cases} a\ e'_{k,j} & \text{dla } k=0 \\ a\ e'_{1,j} & \text{dla } k=1,2,3 \end{cases}. \end{aligned} \tag{5.32}$$

Jak można zauważyć, na podstawie analizy wzoru (5.32), suma XOR przewidywanej i rzeczywistej parzystości pozwala bezpośrednio wyznaczyć maskę błędu tylko wtedy, gdy błąd był wprowadzony do pierwszego bajta $s_{0,j}$, j-tej kolumny stanu. W pozostałych przypadkach (błąd wprowadzony do bajtów $s_{1,j}, s_{2,j}$ albo $s_{3,j}$) suma XOR określa maskę błędu na wyjściu transformacji, która pozwala na korekcję błędu o wartości $e_{1,j}$ wprowadzonego do bajta $s_{1,j}$. Informacja ta nie jest bezpośrednio przydatna do korekcji jednak na podstawie znajomości bajtów $s_{i,j}$ z j-tej kolumny macierzy stanu i informacji, które słowo zostało zmienione przez błąd (tą informację uzyskamy dzięki analizie bitów parzystości $p'_{i,j}$), można wyznaczyć maskę błędu $e'_{k,j}$. Zgodnie z wzorem (5.30) mamy

$$e_{k,j} = \text{InvSubBytes}\left(\text{SubBytes}\left(s_{1,j}\right) \oplus ae'_{1,j}\right) \oplus s_{1,j}. \tag{5.33}$$

Z równań (5.21), (5.24), (5.27) i (5.33) wynika, że jeśli atakujący wprowadza pojedyncze błędy zmieniające jeden bajt kolumny stanu $\mathbb{W}_j$, to możliwe jest określenie maski tego błędu, na podstawie porównania parzystości przewidywanej i rzeczywistej dla tej kolumny. □

Na podstawie przewidywanych i rzeczywistych bajtów parzystości możliwe jest określenie maski błędu, która zmieniła bajty w kolumnie wyjściowej stanu $\mathbb{S}'$. Kolejnym etapem lokalizacji jest określenie, które bajty macierzy $\mathbb{S}'$ zostały zmienione przez błąd. Poszukiwanie to wykonywane jest w oparciu o bity parzystości $p_{i,j}$ powiązane z każdym bajtem $s_{i,j}$. Podobnie jak w przypadku określania maski błędu, lokalizacja błędnych bajtów jest wykonywana oddzielnie dla każdej kolumny $\mathbb{W}_j$.

Twierdzenie 5.3 (Określenie bajtów zmienionych przez wprowadzony błąd)**.** Niech $\mathbb{W}_j$, $s_{i,j}$ i $p_{i,j}$ oznaczają odpowiednio j-tą kolumnę macierzy stanu, i-ty element tej kolumny (bajt) i jego parzystość przed transformacją, a $p'_{i,j}$ i $p^R_{i,j}$ przewidywaną i rzeczywistą parzystość bajta $s'_{i,j}$ po transformacji. Jeśli atakujący wprowadza błąd $e_{k,j}$ taki, że $w_H\left(e_{k,j}\right) = 2n + 1$ do pojedynczego, losowo wybranego bajta $s_{k,j}$ j-tej kolumny macierzy stanu, to na podstawie porównania rzeczywistej i przewidywanej parzystości po transformacji można określić, który bajt został zmieniony przez błąd.

Dowód. Dowód powyższego twierdzenia jest natychmiastowy dla transformacji AddRoundKey, ShiftRows i SubBytes. Pierwsza transformacja jest przekształceniem liniowymi a druga w ogóle nie zmienia wartości elementów macierzy stanu. W obu transformacjach błąd wprowadzony do bajta wejściowego jest w niezmienionej postaci przenoszony na bajt wyjściowy, zmieniając jego parzystość. Dla nieliniowej transformacji SubBytes wprowadzony błąd zmienia bajt wejściowy powodując zmianę jego parzystości. Zgodnie ze sposobem implementacji transformacji SubBytes zaproponowanym w niniejszej pracy, bajt wejściowy z błędną parzystością generuje poprawny bajt wyjściowy z niepoprawną parzystością.

Otrzymana po każdej z powyższych trzech transformacji rzeczywista parzystość bajta różni się od

Tabela 5.1: Wzory zmienionych bitów parzystości dla różnych masek wprowadzonego błędu i różnych miejsc wprowadzenia błędu

Maska błędu	Różnica $\Delta_{i,j} = p'_{i,j} \oplus p^R_{i,j}$	Miejsce wprowadzenia błędu			
		$s_{0,j}$	$s_{1,j}$	$s_{2,j}$	$s_{3,j}$
0xxx xxx	$p'_{0,j} \oplus p^R_{0,j}$	1	0	1	1
	$p'_{1,j} \oplus p^R_{1,j}$	1	1	0	1
	$p'_{2,j} \oplus p^R_{2,j}$	1	1	1	0
	$p'_{3,j} \oplus p^R_{3,j}$	0	1	1	1
1xxx xxx	$p'_{0,j} \oplus p^R_{0,j}$	0	1	1	1
	$p'_{1,j} \oplus p^R_{1,j}$	1	0	1	1
	$p'_{2,j} \oplus p^R_{2,j}$	1	1	0	1
	$p'_{3,j} \oplus p^R_{3,j}$	1	1	1	0

parzystości przewidzianej, wtedy i tylko wtedy, gdy do bajta wprowadzono błąd. W takim przypadku różnica `XOR` obu parzystości $p'_{i,j} \oplus p^R_{i,j}$ jest równa 1 co sygnalizuje, że błąd wprowadzono do słowa $s_{i,j}$.

Dla transformacji MixColumns sprawa jest nieco bardziej skomplikowana ze względu na rozprzestrzenianie się wprowadzonego błędu w całej kolumnie macierzy stanu. Pomimo tej trudności określenie słowa $s_{i,j}$ obarczonego błędem jest możliwe ze względu na właściwości transformacji. Zgodnie ze wzorami (4.3)

$$\begin{aligned} p_{0,j} &= p_{0,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{0,j} \oplus s^{(7)}_{1,j} \\ p_{1,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{3,j} \oplus s^{(7)}_{1,j} \oplus s^{(7)}_{2,j} \\ p_{2,j} &= p_{0,j} \oplus p_{1,j} \oplus p_{2,j} \oplus s^{(7)}_{2,j} \oplus s^{(7)}_{3,j} \\ p_{3,j} &= p_{1,j} \oplus p_{2,j} \oplus p_{3,j} \oplus s^{(7)}_{3,j} \oplus s^{(7)}_{0,j}, \end{aligned}$$

przewidywana wyjściowa parzystość bajtów po transformacji MixColumns zależy od parzystości trzech i najstarszego bitu dwóch bajtów wejściowych. Dokładna analiza tych wzorów pozwala zauważyć, że najstarsze bity każdego słowa $s^{(7)}_{i,j}$ występują w dokładnie dwóch równaniach, podczas gdy parzystość tego bajta $p_{i,j}$ występuje w dokładnie trzech równaniach. Co więcej istnieje tylko jedno równanie, w którym występuje zarówno najstarszy bit jak i parzystość tego samego bajta. W konsekwencji mamy do czynienia z dwoma przypadkami. Jeśli błąd wprowadzony do $s_{i,j}$ nie zmienia jego najstarszego bitu, to wówczas dokładnie trzy bity parzystości są zmienione (wszystkie te, w których występuje $p_{i,j}$). W sytuacji, gdy błąd zmienia najstarszy bit, to również trzy bity parzystości ulegają zmianie (2 bity, które zależą tylko od parzystości $p_{i,j}$ oraz bit parzystości, który zależy od $s^{(7)}_{i,j}$ i nie zależy od $p_{i,j}$).

W razie wprowadzenia błędu w trakcie wykonywania transformacji MixColumns, zmianie ulegają

zawsze trzy wyjściowe bity parzystości, a ich układ i informacja, czy najstarszy bit został zmieniony przez błąd (t.j. znajomość maski błędu), pozwala jednoznacznie określić miejsce wprowadzenia błędu (Tab. 5.1). □

Zgodnie z twierdzeniami 5.2 i 5.3, na podstawie znajomości przewidywanych i rzeczywistych bajtów i bitów parzystości oraz wejściowej macierzy stanu $\mathbb{S}$, możliwe jest odszukanie maski błędu po transformacji oraz miejsca wprowadzenia błędu. Możliwość taka istnieje wtedy, gdy błąd wprowadzany jest do pojedynczego słowa kolumny stanu, a jego waga Hamminga jest liczbą nieparzystą. Wyznaczana w wyniku transformacji błędna wartość stanu wyjściowego $\overline{\mathbb{S}'}$ może być wówczas skorygowana poprzez dodanie macierzy korygującej $\mathbb{C}$

$$\mathbb{S}' = \overline{\mathbb{S}'} \oplus \mathbb{C} = \left[\overline{s'_{i,j}} \oplus c_{i,j}\right]_{4\times 4}. \tag{5.34}$$

Macierz korygująca $\mathbb{C}$ może być wyznaczona dla każdej transformacji algorytmu AES.

Twierdzenie 5.4 (Wyznaczenie macierzy korygującej)**.** Niech $\Delta_j = p'_j \oplus p^R_j$ i $\Delta_{i,j} = p'_{i,j} \oplus p^R_{i,j}$ dla $j = 0, 1, 2, 3$ oznaczają odpowiednio różnicę `XOR` po transformacji rzeczywistego i przewidzianego bajta parzystości kolumny $\mathbb{W}_j$ i słowa $s_{i,j}$ tej kolumny. Na podstawie tych informacji możliwe jest wyznaczenie macierzy korygującej $\mathbb{C}$ pozwalającej skorygować pojedyncze błędy $e_{k,j}$ typu zmiana bajta wprowadzone do $\mathbb{W}_j$.

Dowód. Twierdzenie ma natychmiastowy dowód dla transformacji AddRoundKey, w której wprowadzone błędy nie ulegają rozproszeniu. W takim przypadku, zgodnie z twierdzeniami 5.2 i 5.3 dla j-tej kolumny mamy

$$\Delta_j = e_{k,j} \tag{5.35}$$

$$\Delta_{i,j} = \begin{cases} 0 & \text{dla } i \neq k \\ 1 & \text{dla } i = k \end{cases} \tag{5.36}$$

i j-ta kolumna macierzy korygującej ma postać

$$\mathbb{C}_j = [c_{i,j}]_{1\times 4} \text{ gdzie } c_{i,j} = \begin{cases} 0 & \text{dla } i \neq k \\ e_{k,j} & \text{dla } i = k \end{cases}. \tag{5.37}$$

Jeśli błędy $e_{k_1,0}$, $e_{k_2,1}$, $e_{k_3,2}$, $e_{k_4,3}$ zostały wprowadzone do wszystkich kolumn macierzy stanu, to wówczas macierz korygująca ma postać

$$\mathbb{C} = [c_{i,j}]_{4\times 4} \text{ gdzie } c_{i,j} = \begin{cases} e_{k_1,0} & \text{dla } i = k_1 \text{ i } j = 0 \\ e_{k_2,1} & \text{dla } i = k_2 \text{ i } j = 1 \\ e_{k_3,2} & \text{dla } i = k_3 \text{ i } j = 2 \\ e_{k_4,3} & \text{dla } i = k_4 \text{ i } j = 3 \\ 0 & \text{w przeciwnym przypadku} \end{cases}. \tag{5.38}$$

Podobna sytuacja występuje w transformacji SubBytes gdzie na podstawie Δ_j można wyznaczyć maskę błędu $e'_{k,j}$ zmieniającą k-te słowo j-tej kolumny macierzy stanu. Macierz korygująca ma więc postać

$$\mathbb{C} = [c_{i,j}]_{4\times 4} \text{ gdzie } c_{i,j} = \begin{cases} e'_{k_1,0} & \text{dla } i = k_1 \text{ i } j = 0 \\ e'_{k_2,1} & \text{dla } i = k_2 \text{ i } j = 1 \\ e'_{k_3,2} & \text{dla } i = k_3 \text{ i } j = 2 \\ e'_{k_4,3} & \text{dla } i = k_4 \text{ i } j = 3 \\ 0 & \text{w przeciwnym przypadku} \end{cases} . \tag{5.39}$$

Dla transformacji ShiftRows macierz $\mathbb{C}$ jest tworzona analogicznie, a następnie poddawana transformacji ShiftRows. Wynikowa macierz korygująca jest zatem zdefiniowana jako

$$\mathbb{C} = [c_{i,j}]_{4\times 4} \text{ gdzie } c_{i,j} = \begin{cases} e_{k_1,0} & \text{dla } i = k_1 \text{ i } j = 0 + k_1 \bmod 4 \\ e_{k_2,1} & \text{dla } i = k_2 \text{ i } j = 1 + k_2 \bmod 4 \\ e_{k_3,2} & \text{dla } i = k_3 \text{ i } j = 2 + k_3 \bmod 4 \\ e_{k_4,3} & \text{dla } i = k_4 \text{ i } j = 3 + k_4 \bmod 4 \\ 0 & \text{w przeciwnym przypadku} \end{cases} . \tag{5.40}$$

W MixColumns natomiast macierz $\mathbb{C}$ ma postać

$$\mathbb{C} = [\mathbb{C}_j]_{4\times 1} \text{ gdzie } \mathbb{C}_j = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot [e_{i,j}]_{1\times 4}, \quad e_{i,j} = \begin{cases} 0 & \text{dla } i \neq k \\ e_{k,j} & \text{dla } i = k \end{cases} . \tag{5.41}$$

To kończy dowód. □

5.4 Implementacja algorytmu korekcji błędów

Złożoność implementacyjna zaproponowanej korekcji błędów i narzut implementacyjny, w stosunku do algorytmu AES bez zabezpieczeń, jest silnie zależny od sposobu implementacji algorytmu. Zróżnicowanie to wynika z właściwości AES, który może być zrealizowany na wiele sposobów pozwalających uzyskać różne szybkości działania kosztem złożoności implementacyjnej [32, 61]. W rozwiązaniach wykorzystujących mało zasobów AES jest realizowany w oparciu o zależności matematyczne leżące u podstaw algorytmu — za pomocą prostych sum `XOR`, mnożeń, przesunięć i przeglądania tablicy. W szybkich implementacjach algorytm wykonuje niemal jedynie operacje przeglądania tablic i proste sumy `XOR` dodatkowo zrównoleglone tak, aby jednocześnie można było przetwarzać wszystkie elementy macierzy stanu [32]. Duża szybkość działania algorytmu okupiona jest jednak dużą złożonością implementacyjną.

Wspomniane wyżej dwa parametry — zajmowany obszar A i wprowadzane opóźnienie T, są najistotniejszymi parametrami układów cyfrowych, które pozwalają porównywać ze sobą różne implementacje. Parametry te są inaczej określane w przypadku implementacji w układach reprogramowalnych FPGA i układach dedykowanych ASIC (ang. *Application Specific Integrated Circuit*), co wynika z ich różnej struktury. W implementacji w układach dedykowanych (ASIC) istotna jest liczba i rodzaj bramek logicznych. W układach FPGA obszar zajmowany przez rozwiązanie jest najczęściej opisywany liczbą wykorzystanych tablic LUT, które wraz z otoczeniem zawierającym multipleksery, przerzutniki i bramki `XOR`, są blokiem składowym większych układów. Opóźnienie wprowadzane przez taka implementację jest natomiast szacowane za pomocą liczby poziomów tablic LUT.

Ze względu na specyfikę algorytmu AES wszystkie operacje związane z wyznaczaniem, predykcją i porównywaniem bitów i bajtów parzystości wymagają wykorzystania wielo wejściowych funkcji `XOR`, komparatorów oraz pamięci. Ponieważ pojedyncza tablica LUT pozwala zrealizować dowolną funkcję logiczną 4 zmiennych, to realizacja n-wejściowej funkcji `OR` wymaga zbudowania drzewa o głębokości

$$T_{\texttt{OR}}(n) = \lceil \log_4 n \rceil \,, \tag{5.42}$$

złożonego z nie więcej niż

$$A_{\texttt{OR}}(n) = \left\lceil \frac{n-1}{3} \right\rceil \tag{5.43}$$

tablic LUT. Ze względu na strukturę bloków układu FPGA dla niektórych funkcji logicznych można uzyskać lepsze parametry. Możliwość taka wynika z budowy bloku w skład którego, oprócz tablicy LUT, wchodzi m.in. dodatkowa bramka `XOR` i multipleksery. Jedną z takich funkcji jest `XOR`, który może być zrealizowany z wykorzystaniem tablicy LUT i bramki `XOR`. Pozwala to na realizację w jednym bloku 5-wejściowej bramki `XOR`, a implementacja n-wejściowej bramki `XOR` wymaga drzewa o głębokości

$$T_{\texttt{XOR}}(n) = \lceil \log_5 n \rceil \,, \tag{5.44}$$

złożonego z nie więcej niż

$$A_{\texttt{XOR}}(n) = \left\lceil \frac{n-1}{4} \right\rceil \tag{5.45}$$

tablic LUT.

Podobnie można określić złożoność komparatora porównującego dwa ciągi n–bitowe. Implementacja takiego komparatora wymaga implementacji n 2-wejściowych bramek `XOR` i jednej n-wejściowej bramki `OR`. Całość może być więc zorganizowana w postaci drzewa o głębokości

$$T_{\mathrm{COMP}}(n) = T_{\texttt{XOR}}(2) + T_{\texttt{OR}}(n) = 1 + \lceil \log_4 n \rceil \tag{5.46}$$

złożonego z nie więcej niż

$$A_{\mathrm{COMP}}(n) = nA_{\texttt{XOR}}(2) + A_{\texttt{OR}}(n) = n + \left\lceil \frac{n-1}{3} \right\rceil \tag{5.47}$$

tablic LUT.

W przypadku implementacji tablic konieczne jest wykorzystanie tablic LUT do realizacji bloków pamięci ROM i układu dekodera adresu. Parametry pamięci ROM o n-bitowym słowie adresowym i m-bitowym słowie wyjściowym mogą być oszacowane jako

$$a = \left\lceil \frac{2^n m}{16} \right\rceil, \tag{5.48}$$

$$T = \lceil \log_4 n \rceil . \tag{5.49}$$

Warto tu podkreślić, że powyższe oszacowania są oszacowaniami od góry i rzeczywiste implementacje mogą zajmować mniejsze obszary wykorzystując bramki i multipleksery zintegrowane z tablicami LUT, a także dedykowane pamięci ROM wbudowane we współczesne układy FPGA. Narzut implementacyjny proponowanego rozwiązania zabezpieczającego wynika z dodatkowych podukładów, których zadaniem jest:

- wyznaczenie parzystości macierzy stanu,
- przewidywanie parzystości dla poszczególnych transformacji algorytmu AES,
- porównanie parzystości przewidzianej i rzeczywistej,
- wygenerowanie macierzy korygującej i przeprowadzenie korekty macierzy stanu.

W dalszej części pracy kolejno opisano liczbę i rodzaj bramek potrzebnych do implementacji poszczególnych podukładów oraz liczbę bloków niezbędnych przy implementacji rozwiązania w układzie FPGA.

Wyznaczenie parzystości macierzy stanu

W proponowanym rozwiązaniu wykorzystywane są cztery bajty parzystości i 16 bitów parzystości. Wygenerowanie każdego z bajtów parzystości wymaga wykonania ośmiu 4-wejściowych operacji `XOR`, podczas gdy wyznaczenie każdego bitu parzystości — jednej 8-wejściowej operacji `XOR`. Wyznaczenie parzystości dla całej macierzy wymaga więc wykonania trzydziestu dwóch 4-wejściowych i szesnastu 8-wejściowych operacji `XOR`. Sumaryczny koszt wyznaczenia parzystości macierzy stanu wynosi więc

$$\begin{aligned} &T_{parzystosc} = 2, \\ &A_{parzystosc} = 32 \cdot 1 + 16 \cdot 2 = 64 \text{ [LUT]}. \end{aligned} \tag{5.50}$$

Porównanie parzystości przewidzianej i rzeczywistej

Porównanie parzystości można zrealizować poprzez wyznaczenie sumy `XOR` parzystości przewidzianych i rzeczywistych a następnie sprawdzenie czy wszystkie z wyznaczonych bitów są równe zero — sprawdzenie

to można wykonać za pomocą funkcji OR. Porównanie każdego z bajtów parzystości wymaga wyznaczenia ośmiu 2-wejściowych funkcji XOR, podczas gdy porównanie każdej pary bitów wymaga jednej 2-wejściowej funkcji XOR. W wyniku sprawdzenia uzyskujemy 48 bitów stanowiących wejście do funkcji OR.

Koszt implementacji całej operacji porównania bitów i bajtów parzystości rzeczywistej i przewidywanej wynosi więc

$$\begin{aligned} T_{\mathrm{COMP}} &= 4, \\ A_{\mathrm{COMP}} &= 48 + 16 = 64 \text{ [LUT]}. \end{aligned} \tag{5.51}$$

Przewidywanie parzystości dla poszczególnych transformacji algorytmu AES

Złożoność implementacyjna procedur przewidywania parzystości jest różna dla różnych transformacji algorytmu.

Dla transformacji **ShiftRows** przewidywane bity parzystości są dokładnie takie same jak bity parzystości przed transformacją, stąd przewidywanie to nie wymaga żadnych dodatkowych układów. Przewidywanie bajtów parzystości odbywa się natomiast w oparciu o zależność (5.12), w której wyznaczana jest suma XOR siedmiu słów 8-bitowych dla każdego bajta parzystości. Realizacja wymaga więc wykonania ośmiu 7-wejściowych operacji XOR co w układzie FPGA wymaga wykorzystania 16 tablic LUT zorganizowanych w strukturach dwupoziomowych.

W transformacji **MixColumns** predykcja bajtów parzystości nie wymaga żadnych dodatkowych operacji (5.13), podczas gdy wyznaczenie bitu parzystości polega na wyznaczeniu sumy XOR 5 bitów (5.8). Implementacja tej operacji dla wszystkich elementów macierzy stanu wymaga więc wykorzystania 16 tablic LUT zorganizowanych w strukturze jednopoziomowej.

W transformacji **AddRoundKey** predykcja bitów i bajtów parzystości wymaga wcześniejszego wyznaczenia bitów i bajtów parzystości dla macierzy klucza rundy. Operacja ta może zostać wykonana z wykorzystaniem 64 dodatkowych tablic LUT (5.50), lub za pomocą bloku wykorzystywanego do wyznaczenia bitów i bajtów parzystości macierzy stanu. Wyznaczenie przewidywanych bitów parzystości wymaga następnie wykonania szesnastu 2-wejściowych operacji XOR a wyznaczenie pojedynczego bajta parzystości — ośmiu 2-wejściowych operacji XOR. Implementacja całej predykcji dla transformacji AddRoundKey wymaga więc 48 tablic LUT.

W najbardziej złożonej transformacji **SubBytes** predykcja bitów parzystości wykonywana jest w oparciu o rozszerzoną tablicę XSBox, której wejściem i wyjściem są słowa 8-bitowe poszerzone o bit parzystości. Koszt implementacji takiej tablicy z wykorzystaniem tablic LUT wynosi

$$\begin{aligned} T_{\mathrm{SB}}^{(1)} &= \lceil \log_4 9 \rceil = 2, \\ A_{\mathrm{SB}}^{(1)} &= \left\lceil \tfrac{2^9 \cdot 9}{2^4} \right\rceil = 9 \cdot 2^5 = 288 \text{ [LUT]}. \end{aligned} \tag{5.52}$$

Zaproponowany sposób predykcji bajtów parzystości jest nieco bardziej skomplikowany i przebiega według wzoru (5.17)

$$p'_j = a \left(\left(p_j \bigoplus_{i=1}^{3} s_{i,j} \right)^{-1} \bigoplus_{i=1}^{3} s_{i,j}^{-1} \right). \tag{5.53}$$

Predykcja wymaga wykonania sześciu 8-bitowych operacji `XOR` i jednej operacji przekształcenia afinicznego, które zgodnie z wzorem (2.13) można zrealizować za pomocą ośmiu 5-wejściowych operacji `XOR`. Wszystkie te operacje wymagają łącznie $6 \cdot 2 + 8 \cdot 1 = 12 + 8 = 20$ LUT zorganizowanych w strukturach dwupoziomowych. Algorytm predykcji bajtów parzystości wymaga również wyznaczenia 4 odwrotności multiplikatywnych. Operację tą można efektywnie zaimplementować za pomocą tablic SBox wykorzystywanych w algorytmie AES i dodatkowego przekształcenia afinicznego. Koszt realizacji takiej tablicy wynosi więc

$$\begin{aligned} T_{\text{SBox}} &= \lceil \log_4 8 \rceil = 2, \\ A_{\text{SBox}} &= \left\lceil \tfrac{2^8 \cdot 8}{2^4} \right\rceil = 8 \cdot 2^4 = 128 \text{ [LUT]}. \end{aligned} \tag{5.54}$$

Realizacja dodatkowej operacji transformacji afinicznej wymaga wykorzystania ośmiu 5-wejściowych bramek `XOR`, co w realizacji FPGA, zajmuje 8 LUT zorganizowanych w strukturach jednopoziomowych. Koszt predykcji bajtów parzystości wynosi więc

$$\begin{aligned} T_{\text{SB}}^{(2)} &= 5, \\ A_{\text{SB}}^{(2)} &= 20 + 128 + 8 = 156 \text{ [LUT]}. \end{aligned} \tag{5.55}$$

Całkowity koszt predykcji bitów i bajtów parzystości dla transformacji SubBytes może być zmniejszony jeśli w obu procedurach użyjemy tej samej tablicy XSBox. W takim rozwiązaniu do przewidywania bajtów i bitów parzystości wykorzystywana jest jedna rozszerzona tablica XSBox. Tym samym koszt implementacji całej predykcji wynosi

$$\begin{aligned} T_{\text{SB}} &= 5, \\ A_{\text{SB}} &= 288 + 20 + 8 = 316 \text{ [LUT]}. \end{aligned} \tag{5.56}$$

Wygenerowanie macierzy korygującej i przeprowadzenie korekty macierzy stanu

W procedurze detekcji błędu zadaniem mechanizmu zabezpieczającego jest wyznaczenie macierzy korygującej $\mathbb{C}$. Wymaga to określenia maski wprowadzonego błędu, która, zgodnie z dowodem twierdzenia 5.2, jest wyznaczana bezpośrednio z porównania przewidywanych i rzeczywistych bajtów parzystości. Wyjątkiem jest tu transformacja SubBytes w przypadku, gdy błędy zostały wprowadzone do 2, 3 albo 4 wiersza macierzy stanu. Niezbędne jest wówczas wykonanie dwóch 8-bitowych operacji `XOR` i dwóch operacji SubBytes. Procedura ta wymaga dodatkowych 4 LUT w celu wyznaczenia funkcji `XOR`, podczas gdy operacje

Tabela 5.2: Złożoność implementacyjna procedur przewidywania parzystości dla różnych transformacji algorytmu AES

Transformacja	Liczba LUT A	Liczba poziomów T
ShiftRows	16	2
MixColumns	16	1
AddRoundKey	48	1
SubBytes	316	5

SubBytes mogą być wykonane za pomocą tablic już zaimplementowanych w układzie. Opóźnienie całej procedury jest równe $T = 3$.

Zgodnie z twierdzeniem 5.4, na podstawie znajomości maski błędów generowana jest bezpośrednio macierz korygująca. Wyjątkiem jest tu tym razem, transformacja MixColumns, w której konieczne jest wykonanie przekształcenia MixColumns na wektorze błędów wprowadzonym do kolumny (5.41). Implementacja tej operacji dla pojedynczego błędu (wprowadzonego do jednej tylko kolumny macierzy stanu) wymaga wykonania pojedynczego mnożenia przez czynnik 02 modulo 11B oraz jednej sumy `XOR` dwóch słów 8-bitowych. Ponieważ mnożenie przez czynnik 02 modulo 11B może zostać zrealizowane jako cykliczne przesunięcie z opcjonalną redukcją modulo 11B, to jego wyznaczenie wymaga wykorzystania czterech 2-wejściowych operacji `XOR`. W FPGA wyznaczenie jednej kolumny macierzy korygującej wymaga więc wykorzystania dwunastu 2-wejściowych funkcji `XOR` implementowanych za pomocą 12 LUT. Wygenerowanie całej macierzy wymaga więc $4 + 12 \cdot 4 = 52$ LUT.

Jak już wspomniano algorytm AES może być zaimplementowany na wiele sposobów pozwalających uzyskać kompromis pomiędzy złożonością i szybkością działania. Różnorodność implementacji utrudnia porównanie i ocenę proponowanego rozwiązania poprzez konieczność przyjęcia założeń dotyczących sposobu implementacji AES i procedur wchodzących w skład rozwiązania zabezpieczającego.

Implementacja zaproponowanego rozwiązanie ochronnego wymaga czterech dodatkowych bloków funkcjonalnych, z których każdy wprowadza pewną nadmiarowość tab. 5.3 i opóźnienie, które może być zmniejszone przez zrównoleglenie transformacji algorytmu i procedur predykcji parzystości.

Z powyższych powodów proponowaną modyfikację porównano z realizacją algorytmu AES z kluczem o rozmiarze 128 bitów podaną w pracy [61]. Implementacja standardowa, bez potokowania, zaprezentowana w tej pracy, wykorzystuje 3 814 tablic LUT. Sumaryczna złożoność proponowanego rozwiązania to 576 LUT, a więc narzut implementacyjny wynosi około $15,1\%$.

Tabela 5.3: Złożoność implementacyjna procedur korekcji błędów dla algorytmu AES

Operacja	Liczba LUT – A
Wyznaczenie parzystości	64
Porównanie parzystości	64
Predykcja parzystości	396
Generacja macierzy $\mathbb{C}$	52

Tabela 5.4: Skuteczność proponowanego rozwiązania ochronnego dla różnych rodzajów błędów

Rodzaj błędów	Błędów wykrytych	Błędów korygowanych
wprowadzane do pojedynczych bitów	100%	100%
wprowadzane do bajtów	100%	50%
wprowadzane do kolumn	99%	0%

5.5 Skuteczność proponowanego rozwiązania

Rozwiązanie ochronne zaproponowane dla algorytmu AES jest rozszerzeniem macierzowego kodu korekcyjnego. Ze względu na rodzaj błędów wykorzystywanych do ataków z uszkodzeniami na AES, kod ten zapewnia znacznie lepszą ochronę niż wynika z jego ogólnych właściwości (przy założeniu dowolnych błędów). W atakach z uszkodzeniami wykorzystujących błędy wprowadzane do pojedynczych bitów [14, 35] proponowane rozwiązanie pozwala wykryć i skorygować wszystkie wprowadzone błędy — wynika to bezpośrednio z właściwości kodu macierzowego, który wykrywa wszystkie błędy podwójne i koryguje wszystkie błędy pojedyncze. W przypadku modelu błędów przyjętego na początku niniejszego rozdziału (Def. 5.1), proponowane rozwiązanie pozwala wykrywać wszystkie wprowadzone błędy i korygować połowę z nich.

Twierdzenie 5.5. Niech $\mathbb{S}$, $\mathbb{W}_j$ i $s_{i,j}$ oznaczają odpowiednio stan algorytmu AES, j-tą kolumnę stanu i i-ty element tej kolumny, a p_j i $p_{i,j}$ będą bajtem parzystości j-tej kolumny i bitem parzystości i-tego elementu. W razie wystąpienia błędu losowego w pojedynczym bajcie kolumny $\mathbb{W}_j$ prawdopodobieństwo jego wykrycia wynosi 100%.

Dowód. Zgodnie z definicją bajta parzystości (Def. 5.2) i jak pokazano w dowodzie twierdzenia 5.2, każdy błąd losowy wprowadzony do pojedynczego bajta kolumny stanu powoduje, że suma `XOR` parzystości przewidywanej i rzeczywistej po transformacji jest różna od zera. Sprawdzenie wartości tej sumy daje więc informację czy błąd był wprowadzony czy nie. □

Twierdzenie 5.6. Niech $\mathbb{S}$, $\mathbb{W}_j$ i $s_{i,j}$ oznaczają odpowiednio stan algorytmu AES, j-tą kolumnę stanu

i i-ty element tej kolumny a p_j i $p_{i,j}$ będą bajtem parzystości j-tej kolumny i bitem parzystości i-tego elementu. W razie wystąpienia błędu losowego w pojedynczym bajcie kolumny $\mathbb{W}_j$ prawdopodobieństwo, że błąd ten może być skorygowany, wynosi co najmniej 50%.

Dowód. Zgodnie z twierdzeniem 5.2 dla każdego błędu losowego, wprowadzonego do pojedynczego bajta kolumny stanu, przewidziany i rzeczywisty bajt parzystości pozwalają określić jaki błąd $e_{i,j}$ został wprowadzony. Jednocześnie informacja o masce wprowadzonego błędu identyfikuje kolumnę, w której błąd był wprowadzony. Dodatkowo, zgodnie z twierdzeniem 5.3, bity parzystości $p_{i,j}$ pozwalają określić, który bajt $s_{i,j}$ kolumny $\mathbb{W}_j$ został zmieniony przez błąd $e_{i,j}$, jeśli błąd ten był błędem o nieparzystej krotności (tj. $w_H(e_j) = 2k+1$). Ze względu na założenie, że atakujący wprowadza błędy losowe z równym prawdopodobieństwem, określenie, który bajt został zmieniony przez błąd uda się w 50% przypadków. Stąd, ponieważ do korekcji błędu potrzebna jest znajomość jego maski i lokalizacji, prawdopodobieństwo korekcji błędów wynosi 50%. □

W przypadku rozszerzenia modelu błędu na błędy losowe wprowadzane do dowolnej liczby bajtów (jednego lub wielu) kolumny stanu [71], zdolności detekcyjne pogarszają się pozwalając wykrywać nieco ponad 99% wprowadzanych błędów. Zdolność korekcyjna natomiast jest znacznie gorsza i wynosi $2^{-23} \approx 0\%$. Warto jednak przypomnieć, że błędy tego typu nie są wykorzystywane w atakach z uszkodzeniami, ze względu na dużą złożoność analizy uzyskiwanych wyników.

Rozdział 6

Ochrona podpisów cyfrowych ElGamala i DSA

Analiza metod ochrony przed kryptoanalizą z uszkodzeniami (rozd. 4) pozwala zauważyć, że jedną z metod ochrony jest rozdzielenie obliczeń i rozpraszanie błędów. Metoda ta jest stosowana w celu ochrony algorytmu RSA-CRT (rozd. 4.4). Możliwość rozpraszania błędów w tym algorytmie wynika bezpośrednio z tego, że w RSA-CRT obliczenia są wykonywane oddzielnie modulo liczby pierwsze p i q, a wynik końcowy jest wyznaczany z wykorzystaniem konwersji odwrotnej. Podobną metodę ochrony można zastosować w algorytmie ElGamala i DSA.

Prezentowana w rozdziale modyfikacja schematów ElGamala i DSA wprowadza do tych algorytmów rozpraszanie błędów. W algorytmie ElGamala jest ono zrealizowane z wykorzystaniem chińskiego twierdzenia o resztach. W schemacie DSA rozpraszanie błędów jest uzyskiwane z wykorzystaniem nieliniowości operacji odwrotności multiplikatywnej. Wspólnym elementem modyfikacji obu algorytmów jest kontrola poprawności użytego klucza prywatnego a, polegająca na sprawdzeniu czy jego aktualna wartość jest taka sama jak w chwili generowania klucza publicznego y. Jeśli klucz prywatny, użyty do złożenia podpisu, został zmieniony, to sprawdzenie zwraca niezerową wartość pomocniczego czynnika T, nazywanego czynnikiem rozpraszania błędów (ang. *error difusion term*). Czynnik T jest następnie wykorzystywany do rozproszenia błędu w całym generowanym podpisie s.

6.1 Model błędów

Analiza przeprowadzona w rozdziale 3.3 pokazała, że w przypadku ataków z uszkodzeniami na schematy podpisów cyfrowych ElGamala i DSA najczęściej zakłada się, że atakujący wprowadza błąd typu zamiana pojedynczego bitu lub pojedynczego bajta klucza prywatnego a.

Definicja 6.1 (Błędy wprowadzane do algorytmu ElGamala i DSA)**.** Niech a oznacza klucz prywatny w schemacie podpisów ElGamala/DSA, p, q będą parametrami tego schematu, a g generatorem grupy Z_p^* dla podpisów ElGamala i Z_q^* dla podpisów DSA. Atakujący może wprowadzić uszkodzenie w trakcie generowania podpisu powodując zmianę pojedynczego bitu klucza prywatnego a.

Model błędu wprowadzanego do algorytmu ElGamala i DSA	
rodzaj błędu	zamiana bitu
liczba zmienionych bitów	1
miejsce wprowadzenia błędu	losowy bit klucz prywatny a
moment wprowadzenia	w trakcie wyznaczania elementu s podpisu cyfrowego
czas trwania	błędy przemijające

Zgodnie z przyjętym modelem błędów atakujący ma częściową kontrolę nad miejscem wprowadzenia błędu — błędy są wprowadzane do zmiennych przechowujących klucz prywatny a jednak atakujący nie może wybrać, który bit ma być zmieniony.

Zakładając powyższy model błędów można oszacować prawdopodobieństwo, że atakujący zdoła wprowadzić taki błąd.

Lemat 6.1. Niech p, g, a, y będą parametrami schematu podpisów ElGamala, przy czym p jest n-bitową liczbą pierwszą, $a < p$ kluczem prywatnym a $y = g^a \bmod p$ kluczem publicznym. Jeśli atakujący wprowadza losowe błędy typu zamiana pojedynczego bitu klucza a, to prawdopodobieństwo wprowadzenia błędu $e = 2^i$ dla konkretnego $i = 0, 1, \ldots, n-1$ jest równe $\frac{1}{n}$.

Dowód. Dowód lematu jest natychmiastowy, ponieważ z faktu, że p jest liczbą n-bitową wynika, że $p^{(n-1)} = 1$ i że istnieje co najmniej jeden bit $j = 1, 2, \ldots, n-2$ taki, że $p^{(j)} = 1$. W przeciwnym razie p nie jest liczbą n–bitową, albo nie jest liczbą pierwszą (jeśli $p^{(j)} = 0$ dla $j = 1, 2, \ldots, n-2$, to $p = 2^{n-1}$ i jest złożone). Oznacza to, że $p > 2^{n-1} \geq 2^i$ dla $i = 0, 1, \ldots, n-1$, a więc atakujący może wprowadzić n różnych błędów typu zamiana bitu. Zakładając losowe wprowadzanie błędów (każdy błąd jest równie prawdopodobny), prawdopodobieństwo wprowadzenia błędu e jest równe $\frac{1}{n}$. □

Powyższy model błędów jest modelem powszechnie przyjmowanym w atakach na algorytmy ElGamala i DSA, rzadko rozszerzanym na błędy o krotnościach nie większych niż 8. Taki model błędów wynika z dążenia do uproszczenia ataku z uszkodzeniami, którego złożoność obliczeniowa zależy od krotności wprowadzonego błędu. Z drugiej strony proponowane rozwiązanie zabezpieczające pozwala na ochronę schematów podpisów przed atakami wykorzystującymi również błędy wielokrotne i dowolne. Z tego

względu w dalszej części rozdziału będą analizowane zarówno błędy pojedyncze jak i dowolne. W przypadku błędów wielokrotnych atakujący może wprowadzić do rejestru przechowującego klucz prywatny, błąd $e > p$. Wówczas, przy wykonaniu pierwszej operacji arytmetycznej, błąd taki jest redukowany modulo p. Tym samym generowany podpis jest obarczony błędem $e_1 = e \bmod p$.

Lemat 6.2. Niech p, g, a, y będą parametrami schematu podpisów ElGamala, przy czym p jest n-bitową liczbą pierwszą, $a < p$ kluczem prywatnym a $y = g^a \bmod p$ kluczem publicznym. Wprowadzając losowe błędy e do rejestru przechowującego klucz prywatny, atakujący zdoła wprowadzić błąd $e_1 < p$ z prawdopodobieństwem nie większym niż $1/2^{n-1}$.

Dowód. Wystarczy zauważyć, że ponieważ p jest liczbą n-bitową to $2^{n-1} < p < 2^n$ i wprowadzając dowolny błąd do rejestru n-bitowego atakujący może wprowadzić błędy $e = q + e_1 \geq p$. W takim przypadku błąd zmieniający wartość klucza prywatnego jest równy $e \bmod p = e_1$, czyli taki sam jak w sytuacji, gdy atakujący bezpośrednio wprowadza błąd e_1. Wszystkich błędów $e \geq p$ jest $2^n - p \leq 2^n - 2^{n-1} = 2^{n-1}$ tj. nie więcej niż błędów $e < p$. Zakładając, że atakujący wprowadza błędy losowo wówczas prawdopodobieństwo wprowadzenia błędu $e_1 < p$ jest nie większe niż $\frac{2}{2^n - 1} \approx \frac{1}{2^{n-1}}$ □

Analogiczny lemat obowiązuje w przypadku schematów podpisów DSA, z tą jednak różnicą, że błędy wprowadzone do tego algorytmu są mniejsze od q. Warto tu również zauważyć, że ze względu na rozmiary liczb p i q, stosowanych w algorytmach ElGamala i DSA, prawdopodobieństwo to jest bliskie 0. Oznacza to, że w przypadku losowego wprowadzania błędów szanse na wprowadzenie konkretnego błędu są bardzo małe.

6.2 Ochrona schematu podpisów cyfrowych ElGamala

Możliwość ochrony algorytmu ElGamala przez rozpraszanie błędów wynika z tego, że w procedurze składania podpisu ElGamala (Alg. 2.7) obliczenia z użyciem tajnego klucza a są wykonywane modulo $p - 1$. Ponieważ p jest liczbą pierwszą to $p - 1$ jest liczbą złożoną, podzielną przez 2. Można ją więc przedstawić w postaci iloczynu dzielników pierwszych

$$p - 1 = \prod_{P} p_i^{e_i} \quad \text{gdzie } P = \{p_i : p_i | (p-1), p_i \in PRIME\} \quad \text{i} \quad e_i > 0. \tag{6.1}$$

Interesującym pytaniem jest rząd zbioru P dla losowo wybieranych n-bitowych liczb pierwszych p. Pytanie to jest istotne z punktu widzenia proponowanej modyfikacji algorytmu ElGamala, gdyż wymaga ona aby liczba $p - 1$ posiadała dwa duże, względnie pierwsze dzielniki w_1 i w_2.

Jeśli jedynym dzielnikiem liczby $p - 1$ jest 2, to p jest liczbą Fermata. Warunkiem koniecznym, aby liczba Fermata $p = 2^k + 1$ była liczbą pierwszą, jest k będące potęgą dwójki (nie jest to jednak warunek

wystarczający). W przypadku standardowego algorytmu ElGamala liczba pierwsza p jest wybierana losowo spośród liczb $\left(2^{767}, 2^{1024}\right)$. Każda liczba postaci $2^k + 1$ należąca do tego przedziału ma wykładniki k, które nie są potęgą dwójki. Oznacza to, że w przedziale z którego losowane są liczby pierwsze nie ma liczb pierwszych Fermata. W konsekwencji dla każdej liczby pierwszej p stosowanej w algorytmie ElGamala, liczba $p - 1$ nie jest postaci 2^k i ma co najmniej dwa różne dzielniki pierwsze.

Znacznie bardziej skomplikowane jest oszacowanie prawdopodobieństwa, że dla losowo wybranej n-bitowej liczby pierwszej p, $p-1$ ma co najmniej 3 różne dzielniki pierwsze. Odpowiedzi na to pytanie można szukać poprzez oszacowanie prawdopodobieństwa, że dla losowo wybranej n-bitowej liczby pierwszej p, $p - 1$ ma dokładnie dwa dzielniki pierwsze. Innymi słowy szukamy prawdopodobieństwa, że wylosowana liczba pierwsza p może zostać zapisana w postaci

$$q^l 2^k + 1, \tag{6.2}$$

gdzie $q \geq 3$ jest pierwsze a $l, k > 0$.

Dokładna liczba liczb pierwszych postaci (6.2), w przedziale $(1, N]$ nie jest znana. W literaturze można natomiast odnaleźć przypuszczenia pozwalające oszacować liczbę liczb pierwszych różnej postaci. Przypuszczenia te nie są jednak stricte matematycznymi wyprowadzeniami a jedynie przybliżeniami pozbawionymi formalnych dowodów. Jedno z oszacowań liczby liczb pierwszych powiązanych zależnościami wielomianowymi przedstawiono w pracy [24]. Oszacowanie to przedstawia liczb pierwsze jako wartości pewnych wielomianów $f_i(x)$ i szacuje liczbę takich $0 \leq x < N$ dla których $f_i(x)$ dla każdego i jest liczbą pierwszą. W pracy [24] pokazano, że jeśli wielomiany $f_i(x)$ nie są rozkładalne i mają wszystkie współczynniki całkowite, to liczba $0 < x < N$ dla dużych N i dla których $f_i(x)$ są pierwsze, może być oszacowana z zależności:

$$\Pi(N) = \frac{1}{d_1 d_2 \dots d_t} \prod_{r \in \texttt{PRIMES}} \frac{1 - w(r)/r}{(1 - 1/r)^t} \int_2^N \frac{dx}{\ln^t x} \approx \frac{N}{d_1 d_2 \dots d_t \ln^t N} \prod_{r \in \texttt{PRIMES}} \frac{1 - w(r)/r}{(1 - 1/r)^t}, \tag{6.3}$$

gdzie iloczyn jest brany po wszystkich liczbach pierwszych r, t jest liczbą wielomianów $f_i(x)$, d_i ich stopniem, a $w(r)$ jest liczbą rozwiązań równania

$$f_1(x) f_2(x) \dots f_t(x) \bmod r = 0 \tag{6.4}$$

dla $x = \{0, 1, 2, \dots, r - 1\}$.

W specyficznym przypadku, gdy mamy do czynienia tylko z jedną funkcją postaci $f_1(x) = x$, powyższe oszacowanie pozwala nam wyznaczyć liczbę liczb pierwszych mniejszych od N, które wynosi

$$\Pi(N) = \frac{N}{\ln N}. \tag{6.5}$$

Korzystając z tego oszacowania, można również oszacować liczbę liczb pierwszych określonego rozmiaru bitowego. Liczba n-bitowych liczb pierwszych może być bowiem oszacowana jako

$$\Pi_n = \Pi(2^n) - \Pi(2^{n-1}). \tag{6.6}$$

Oszacowania n-bitowych liczb pierwszych postaci $f(x)$ będziemy dalej oznaczali symbolem $\Pi_{n,f(x)}$.

Korzystając z tego typu oszacowań można postawić następujące przypuszczenie.

Przypuszczenie 6.1. Prawdopodobieństwo wylosowania n-bitowej liczby pierwszej p, takiej, że $p-1$ ma co najmniej trzy różne dzielniki jest nie większe niż

$$1 - \frac{1.32032 \cdot 2n}{(n-1)^2 \ln 2},$$

Dla liczb pierwszych p stosowanych w schemacie podpisów ElGamala prawdopodobieństwo to jest nie większe niż 0.995.

Uzasadnienie: Z ograniczenia rozmiaru liczby p wynika ograniczenie rozmiarów bitowych wykładników l, k i liczby pierwszej q, które muszą spełniać następującą równość

$$n = l\lceil \log_2 q \rceil + k. \tag{6.7}$$

Z równości tej wynika natomiast, że dla ustalonych wartości n, l i k liczba q musi być $m = (n-k)/l$-bitowa, tj.

$$2^{\frac{n-k}{l}-1} - 1 < q < 2^{\frac{n-k}{l}}. \tag{6.8}$$

Chcąc odszukać liczbę n-bitowych liczb pierwszych $p = q^l 2^k + 1$ takich, że $q \geq 3$ jest pierwsze i $l, k > 0$ musimy określić funkcje $f_i(x)$. Dla tak zdefiniowanych liczb pierwszych p, q mamy do czynienia z $t = 2$ funkcjami

$$f_1(x) = x \tag{6.9}$$

$$f_2(x) = x^l 2^k + 1, \tag{6.10}$$

dla których $d_1 = 1$, $d_2 = l$. Funkcja $f_1(x)$ spełnia wymagania warunkujące wykorzystanie oszacowania (6.3). Warunku nierozkładalności nie spełniają natomiast niektóre funkcje z rodziny $f_2(x) = x^l 2^k + 1$. Grupą rozkładalnych funkcji są bowiem wszystkie funkcje $f_2(x)$ dla których $l = k$ jest liczbą nieparzystą, większą od jedynki. Jeśli bowiem $l = k = 2i + 1$ dla $i = 1, 2, 3, \ldots$, to wielomian $(2x)^{2i+1} + 1$ jest rozkładalny

$$(2x)^{2i+1} + 1 = (2x+1) \sum_{j=0}^{2i} (-1)^i (2x)^i \quad \text{dla } i = 1, 2, 3, \ldots \tag{6.11}$$

Z oszacowania (6.3) można jednak skorzystać dla niektórych wartości parametrów l i k — np.: $l = 1$. W przypadkach tych liczbę n-bitowych liczb pierwszych szukanej postaci ($\Pi_{n,q^l 2^k+1}$) można oszacować jako

$$\begin{aligned}\Pi_{n,q^l 2^k+1} &= \Pi\left(2^m\right) - \Pi\left(2^{m-1}\right) = \left[\frac{2^m}{l\left(\ln 2^m\right)^2} - \frac{2^{m-1}}{l\left(\ln 2^{m-1}\right)^2}\right] \cdot \prod_{r\in\texttt{PRIMES}} \frac{1-\frac{w(r)}{r}}{\left(1-\frac{1}{r}\right)^2} \\ &= \frac{2^{m-1}}{l\ln^2 2}\left[\frac{2}{m^2} - \frac{1}{(m-1)^2}\right] \cdot \prod_{r\in\texttt{PRIMES}} \frac{1-\frac{w(r)}{r}}{\left(1-\frac{1}{r}\right)^2} \end{aligned} \tag{6.12}$$

gdzie $m = \frac{n-k}{l} >> 1$. Wyznaczenie oszacowania wymaga jeszcze wyznaczenia czynnika $w(r)$, czyli znalezienia liczby rozwiązań równania

$$f_1(x) f_2(x) \bmod r = x\left(x^l 2^k + 1\right) \bmod r = 0. \tag{6.13}$$

Jednym z rozwiązań tego równania, niezależnym od r, l oraz k, jest $x = 0$. Ponadto dla $r > 2$ kolejnymi rozwiązaniami są x spełniające zależność $\left(x^l 2^k + 1\right) \bmod r = 0$ i ponieważ $\gcd(2^k \bmod r, r) = 1$ to rozwiązania te można wyznaczyć jako

$$x^l = -2^{-k} \bmod r. \tag{6.14}$$

Kolejne rozwiązania równania (6.13) istnieją więc wtedy i tylko wtedy, gdy istnieje pierwiastek l-tego stopnia z -2^{-k} modulo r. Ponadto, jeśli pierwiastek istnieje, to dla l nieparzystych istnieje jedno rozwiązanie, a dla l parzystych — dwa.

Rozwiązanie równania (6.14) i wyznaczenie wartości $w(r)$, dla wszystkich kombinacji parametrów l i k, nie jest możliwe. Trudne jest też oszacowanie liczby tych rozwiązań. Dla $l = 1$ jednak, równanie to ma dokładnie jedno rozwiązanie dla każdej wartości k, a przez to wartość współczynnika $w(r)$ jest równa

$$w(r) = \begin{cases} 1 & \text{dla } r = 2 \\ 2 & \text{dla } r > 2 \end{cases}. \tag{6.15}$$

Nierozkładalność wielomianów $f_1(x)$ i $f_2(x)$, oraz znajomość $w(r)$ dla $l = 1$ pozwala oszacować liczbę n-bitowych liczb pierwszych p, które można zapisać w postaci $p = q2^k + 1$. Oszacowanie to jest równe

$$\Pi_{n,q2^k+1} = \Pi\left(2^{n-k}\right) - \Pi\left(2^{n-k-1}\right) = \frac{2^{n-k-1}}{l\ln^2 2}\left[\frac{2}{(n-k)^2} - \frac{1}{(n-k-1)^2}\right] \cdot \prod_{r\in\texttt{PRIMES}} \frac{1-\frac{w(r)}{r}}{\left(1-\frac{1}{r}\right)^2}. \tag{6.16}$$

Dla $w(r)$ określonego wzorem (6.15) iloczyn

$$\prod_{r\in\texttt{PRIMES}} \frac{1-\frac{w(r)}{r}}{\left(1-\frac{1}{r}\right)^2} = 2C_2, \tag{6.17}$$

gdzie C_2 jest stałą nazywaną *twin prime constant* i ma przybliżoną wartość równą 0.66016. Ostatecznie szukane oszacowanie ma postać

$$\Pi_{n,q2^k+1} = \Pi\left(2^{n-k}\right) - \Pi\left(2^{n-k-1}\right) = \frac{1.32032 \cdot 2^{n-k-1}}{l\ln^2 2}\left[\frac{2}{(n-k)^2} - \frac{1}{(n-k-1)^2}\right]. \tag{6.18}$$

Sumując oszacowania dla kolejnych wartości k otrzymamy

$$\sum_{k=1}^{n-1} \Pi_{n,q2^k+1} = \sum_{k=1}^{n-1} \Pi\left(2^{n-k}\right) - \Pi\left(2^{n-k-1}\right) \approx \Pi\left(2^{n-1}\right) = \frac{1.32032 \cdot 2^n}{(n-1)^2 \ln^2 2} \tag{6.19}$$

Ponieważ wszystkich n-bitowych liczb pierwszych, jest około

$$\Pi_n = \Pi(2^n) - \Pi(2^{n-1}) = \frac{2^n}{n \ln 2} - \frac{2^{n-1}}{(n-1)\ln 2} \approx \frac{2^{n-1}}{n \ln 2}, \tag{6.20}$$

to prawdopodobieństwo wylosowania liczby pierwszej postaci $p = q2^k + 1$ może być oszacowane jako

$$P(n, q2^k + 1) = \frac{\frac{1.32032 \cdot 2^n}{(n-1)^2 \ln^2 2}}{\frac{2^{n-1}}{n \ln 2}} = \frac{1.32032 \cdot 2n}{(n-1)^2 \ln 2}. \tag{6.21}$$

Dla liczb pierwszych p, stosowanych w algorytmie ElGamala $n = 768$ i wówczas prawdopodobieństwo wylosowania liczby pierwszej postaci $q2^k + 1$ jest w przybliżeniu równe 0.004973. □

Analogiczne oszacowanie liczby n-bitowych liczb pierwszych postaci $p = q^l 2^k + 1$ dla $l > 1$ w oparciu o tą metodę jest bardziej skomplikowane (wymaga oszacowania liczby pierwiastków l-tego stopnia modulo r), lub wręcz niemożliwe (dla niektórych wartości l i k, $f_2(x)$ jest rozkładalne). Można natomiast zauważyć, że ze wzrostem l szybko maleje wartość $m = \frac{n-k}{l}$ występująca w wykładniku oszacowania (6.12). Sugeruje to, że dla $l > 1$ liczba n-bitowych liczb pierwszych postaci $q^l 2^k + 1$, będzie znacznie mniejsza od oszacowania dla liczb postaci $q2^k + 1$.

Przybliżenie liczby liczb pierwszych postaci $p = q^l 2^k + 1$ może być również wyznaczone poprzez oszacowanie ile liczb $\frac{p-1}{2^k}$ ma dokładnie $d = 1$ dzielników pierwszych. Oszacowanie takie można znaleźć między innymi w pracy [48]. Jednym z przybliżeń liczby liczb $x < N$, mających dokładnie d dzielników pierwszych jest

$$\pi(N, d) = F\left(\frac{d}{\ln \ln N}\right) \frac{N}{\ln N} \cdot \frac{(\ln \ln N)^{d-1}}{(d-1)!}, \tag{6.22}$$

gdzie

$$F(x) = \frac{1}{\Gamma(x+1)} \cdot \prod_{r \in \texttt{PRIMES}} \left(1 + \frac{x}{r-1}\right) \left(1 - \frac{1}{r}\right)^x, \tag{6.23}$$

$N \geq 3$ i $1 \leq d \leq C \ln \ln N$ dla pewnej stałej C.

Bazując na takim oszacowaniu można wysnuć następujące przypuszczenie.

Przypuszczenie 6.2. Prawdopodobieństwo wylosowania n-bitowej liczby pierwszej p, takiej, że $p-1$ ma co najmniej trzy różne dzielniki jest nie mniejsze niż

$$1 - \frac{n}{n-1} F\left(\frac{1}{\ln \ln 2^{n-1}}\right),$$

gdzie $F(x)$ jest wyznaczane zgodnie z (6.23). Dla liczb pierwszych p stosowanych w schemacie podpisów ElGamala prawdopodobieństwo to jest nie mniejsze niż 0.82.

Uzasadnienie: Ze względu na postać szukanych liczb pierwszych p, interesują nas liczby $2^{n-k-1} < x < 2^{n-k}$, które posiadają dokładnie $d = 1$ dzielnik. Korzystając ze wzoru (6.22) ich liczbę można oszacować jako

$$\pi(2^{n-k},1) - \pi(2^{n-k-1},1) = F\left(\frac{1}{\ln\ln 2^{n-k}}\right)\frac{2^{n-k}}{\ln 2^{n-k}} - F\left(\frac{1}{\ln\ln 2^{n-k-1}}\right)\frac{2^{n-k-1}}{\ln 2^{n-k-1}}. \tag{6.24}$$

Sumując po wszystkich k otrzymamy

$$\sum_{k=1}^{n-2}\pi(2^{n-k},1) - \pi(2^{n-k-1},1) \approx \pi(2^{n-1},1) = F\left(\frac{1}{\ln\ln 2^{n-1}}\right)\frac{2^{n-1}}{\ln 2^{n-1}} \tag{6.25}$$

i zgodnie z tym oszacowaniem prawdopodobieństwo wylosowania liczby pierwszej $p = q^l 2^k + 1$ jest równe

$$P(n, q^l 2^k + 1) = \frac{F\left(\frac{1}{\ln\ln 2^{n-1}}\right)\frac{2^{n-1}}{\ln 2^{n-1}}}{\frac{2^{n-1}}{n\ln 2}} = \frac{n}{n-1}F\left(\frac{1}{\ln\ln 2^{n-1}}\right). \tag{6.26}$$

Prawdopodobieństwo to, dla $n = 768$ i przy wyznaczeniu wartości funkcji $F(x)$ dla 100 000 pierwszych liczb pierwszych r wynosi około 0.1706. □

Należy jednak zauważyć, że powyższe oszacowanie jest zawyżone. Wynika to z tego, że nie dla każdej $n-k$ bitowej liczby A, mającej jeden dzielnik pierwszy, liczba $A \cdot 2^k + 1$ jest liczbą pierwszą. Przypuszczenia powyższe zostały zweryfikowane doświadczalnie dla losowych, 768-bitowych liczb pierwszych p. Testy polegały na losowym wybieraniu p i sprawdzaniu ile dzielników pierwszych posiada liczba $p-1$. Przeprowadzone testy dla 5600 losowo wybranych liczb pierwszych pokazały, że w zaledwie około 4% przypadków liczba $p-1$ posiada tylko dwa dzielniki. Z drugiej strony, dla ponad 33% wylosowanych liczb, $p-1$ miało sześć, lub więcej, różnych dzielników. Ze względu na duże rozmiary analizowanych liczb, poszukiwanie ograniczono do dzielników nie większych niż $p_{5761482} = 100\ 000\ 493$. Pozwoliło to znacznie przyspieszyć testy, ale jednocześnie spowodowało przekłamanie uzyskiwanych wyników. Jest to wywołane tym, że taki algorytm faktoryzacji uznaje liczby niepodzielne przez $p_i \leq p_{5761482}$ za pierwsze. Ponieważ faktoryzujemy liczby rzędu 2^{768}, znacznie większe od $p_{5761482} \approx 2^{23}$, to wyznaczona liczba dzielników jest zaniżana. Oznacza to, że uzyskane wyniki zawyżają liczbę liczb posiadających dokładnie dwa dzielniki pierwsze.

Z powyższych analiz wynika, że w ogromnej większości przypadków (ponad 95%) dla losowo wybranych 786-bitowych liczb pierwszych p, $p-1$ ma co najmniej trzy dzielniki pierwsze ($|P| \geq 3$). Zbiór P wszystkich dzielników takiej liczby można więc podzielić na dwa rozłączne podzbiory P_1 i P_2 takie, że $P_1 \cap P_2 = \emptyset$, $P_1 \cup P_2 = P$ i 2 nie jest jedynym elementem żadnego z podzbiorów. Dla każdego podzbioru można następnie obliczyć iloczyn jego elementów

$$w_1 = \prod_{p_i \in P_1} p_i^{e_i}, \qquad w_2 = \prod_{p_i \in P_2} p_i^{e_i}. \tag{6.27}$$

Liczby w_1, w_2 posiadają następujące właściwości

- iloczyn $w_1 w_2 = p - 1$,
- największy wspólny dzielnik $\gcd(w_1, w_2) = 1$,
- $w_i > 2$ dla $i = 1, 2$,

Pierwsze dwie z powyższych właściwości pozwalają obliczenia modulo $p-1$ zastąpić obliczeniami modulo w_1 oraz w_2, a wynik końcowy modulo $p-1$ wyznaczyć z wykorzystaniem chińskiego twierdzenia o resztach. Ostatnia właściwość jest istotna nie dla poprawności działania proponowanego algorytmu, lecz służy zagwarantowaniu odporności na ataki z uszkodzeniami. Jak bowiem zostanie pokazane w dalszej części rozdziału, małe wartości w_i powodują małą złożoność ataków wykorzystujących przegląd zupełny.

W dalszej części pracy założymy, że liczba p będąca parametrem algorytmu ElGamala została tak dobrana, aby $p-1$ posiadało co najmniej trzy różne dzielniki pierwsze (jednym z nich jest na pewno liczba 2), a moduły w_1 i w_2 były znacznie większe od 2.

Możliwość rozdzielenia obliczeń prowadzonych modulo $p-1$ i późniejsza rekonstrukcja wyniku z wykorzystaniem chińskiego twierdzenia o resztach jest podstawą zaproponowanej modyfikacji podpisów ElGamala. Modyfikacja ta dotyczy wyłącznie algorytmu generowania podpisu (Alg. 2.7) podczas gdy, procedury generacji kluczy i weryfikacji podpisów pozostają bez zmian. Modyfikacja algorytmu generowania podpisu dotyczy tylko ostatniego kroku algorytmu, którego oryginalna postać

$$s = k^{-1} \left(h(m) - ar\right) \bmod (p-1), \tag{6.28}$$

zastąpiona jest sekwencją 4 obliczeń

$$\begin{aligned} s_1 &= k^{-1} \left(h(m) - ar\right) \bmod w_1, && (6.29)\\ s_2 &= k^{-1} \left(h(m) - ar\right) \bmod w_2, && (6.30)\\ T &= f(a, k, s_1, s_2), && (6.31)\\ s &= \mathrm{CRT}\,(T, s_1, s_2). && (6.32) \end{aligned}$$

Wyrażenie T, wyznaczane jako funkcja klucza prywatnego a, liczby losowej k i podpisów częściowych s_i (6.31), jest czynnikiem rozpraszania błędów (ang. *error diffusion term*) wykorzystywanym w trakcie wyznaczania właściwego podpisu s (6.32). W ogólnym przypadku dokładna postać funkcji $f(a, k, s_1, s_2)$ może być różna. Spełniony musi być natomiast warunek, że $T = 0$ wtedy i tylko wtedy, gdy wszystkie obliczenia zostały wykonane poprawnie. W ramach pracy doktorskiej analizowano dwie postacie funkcji f

$$T = \left(\mathrm{SHA1}\,\left(va^{-1} + r\right) - \mathrm{SHA1}\,(0)\right) \bmod (p-1), \tag{6.33}$$

Algorytm 6.1 Ogólna postać zmodyfikowanego algorytmu generacji podpisów cyfrowych ElGamala

Wejście: wiadomość m, klucz prywatny a, generator g i moduły p, w_1, w_2

Wyjście: podpis $\langle r, s \rangle$

1: wylosuj liczbę pierwszą k taką, że $1 \leq k \leq p-2$ i $\gcd(k, p-1) = 1$,

2: oblicz $r = g^k \bmod p$,

3: oblicz $k^{-1} \bmod (p-1)$,

4: oblicz $s_1 = k^{-1}\left(h(m) - ar\right) \bmod w_1$,

5: oblicz $s_2 = k^{-1}\left(h(m) - ar\right) \bmod w_2$,

6: oblicz $T = f\left(a, k, s_1, s_2\right)$,

7: oblicz podpis s zgodnie z następującym algorytmem konwersji odwrotnej

$$s = \left[s_1\left(w_2 \oplus T\right)\left(w_2^{-1} \bmod w_1\right) + s_2\left(w_1 \oplus T\right)\left(w_1^{-1} \bmod w_2\right)\right] \bmod (p-1)$$

oraz

$$T = \left(\left(y^r g^v \bmod p\right) - 1\right) \bmod (p-1), \tag{6.34}$$

gdzie $v = \mathrm{CRT}\left(s_1, s_2\right)$ a $y = g^a \bmod p$ jest kluczem publicznym schematu ElGamala. Pierwsza z nich wykorzystuje funkcję skrótu SHA1 w celu zagwarantowania rozproszenia wprowadzonych błędów i zapewnienia, że $T \neq 0$ jeśli tylko do obliczeń wprowadzono błąd. Wadą tej propozycji jest mały rozmiar bitowy czynnika T, wynikający z wykorzystanej funkcji skrótu, oraz wykorzystywanie odwrotności multiplikatywnej klucza prywatnego a^{-1}. Powyższych wad nie posiada druga z analizowanych funkcji, która do sprawdzenia poprawności wykonania algorytmu wykorzystuje klucz publiczny y. Minusem drugiego rozwiązania jest wykorzystanie operacji potęgowania co powoduje zwiększenie złożoności obliczeniowej.

W dalszej części pracy rozważana jest modyfikacja, w której czynnik T wyznaczany jest według zależności 6.34. Mimo większej złożoności obliczeniowej tego rozwiązania pozwala ono na bardziej szczegółową analizę propagacji błędów jak i potencjalnych możliwości ataku. Zmodyfikowana procedura generacji podpisów ElGamala przebiega zgodnie z algorytmem (Alg. 6.2). Analiza tego algorytmu pozwala zauważyć, że wygenerowany poprawny podpis cyfrowy jest identyczny z podpisem generowanym za pomocą standardowego algorytmu ElGamala.

Twierdzenie 6.1. Poprawny podpis ElGamala wygenerowany za pomocą algorytmu (Alg. 6.2) jest identyczny z podpisem generowanym za pomocą standardowego algorytmu ElGamala (Alg. 2.7).

Dowód. W przypadku, gdy w czasie wykonania algorytmu (Alg. 6.2) do jego przebiegu nie są wprowadzane błędy wówczas podpisy częściowe s_i obliczone w 4 i 5 kroku są poprawne, a więc v_1 i v_2 są odpowiednio równe $-ar \bmod w_1$ i $-ar \bmod w_2$. Wyznaczany w kolejnym kroku algorytmu wynik chińskiego twierdzenia

Algorytm 6.2 Zmodyfikowany algorytm generacji podpisów cyfrowych ElGamala

Wejście: wiadomość m, klucz prywatny a, w_1, w_2

Wyjście: podpis $\langle r, s\rangle$

1: wylosuj liczbę pierwszą k taką, że $1 \leq k \leq p-2$ i $\gcd(k, p-1) = 1$,

2: oblicz $r = g^k \bmod p$,

3: oblicz $k^{-1} \bmod (p-1)$,

4: oblicz $s_1 = k^{-1}\left(h(m) - ar\right) \bmod w_1$,

5: oblicz $s_2 = k^{-1}\left(h(m) - ar\right) \bmod w_2$,

6: oblicz $v_1 = s_1 k - h(m) \bmod w_1$, $v_2 = s_2 k - h(m) \bmod w_2$,

7: oblicz $v = \mathrm{CRT}\left(v_1, v_2\right)$,

8: oblicz $T = \left((y^r g^v \bmod p) - 1\right) \bmod (p-1)$,

9: oblicz podpis s zgodnie z następującym algorytmem konwersji odwrotnej

$$s = \left(s_1\left(w_2 \oplus T\right)\left(w_2^{-1} \bmod w_1\right) + s_2\left(w_1 \oplus T\right)\left(w_1^{-1} \bmod w_2\right)\right) \bmod (p-1)$$

o resztach dla v_1 i v_2 jest równy $-ar \bmod (p-1)$. Stąd czynnik rozpraszania błędów T, obliczany w 8 kroku algorytmu, jest równy

$$\begin{aligned} T &= \left((y^r g^v \bmod p) - 1\right) \bmod (p-1) = \left(\left(g^{ar} g^{-ar} \bmod p\right) - 1\right) \bmod (p-1) \\ &= \left(\left(g^0 \bmod p\right) - 1\right) \bmod (p-1) = 1 - 1 = 0. \end{aligned}$$

Dla $T = 0$ podpis s, wyznaczany w ostatnim kroku, jest rezultatem wykonania konwersji odwrotnej zgodnie z chińskim twierdzeniem o resztach

$$\begin{aligned} s &= \left(s_1\left(w_2 \oplus T\right)\left(w_2^{-1} \bmod w_1\right) + s_2\left(w_1 \oplus T\right)\left(w_1^{-1} \bmod w_2\right)\right) \bmod (p-1) \\ &= \left(s_1 w_1\left(w_1^{-1} \bmod w_2\right) + s_2 w_2\left(w_2^{-1} \bmod w_1\right)\right) \bmod (p-1) \\ &= \mathrm{CRT}\left(s_1, s_2\right) = k^{-1}\left(h(m) - ar\right) \bmod (p-1), \end{aligned}$$

i jest identyczny z podpisem generowanym przez algorytm (Alg. 2.7). □

Przeprowadzając atak na zaproponowany algorytm generacji podpisów ElGamala z wykorzystaniem losowych błędów typu zamiana bitu (Def. 6.1), atakujący ma dwie możliwości wprowadzenia błędu:

- wprowadzenie losowego błędu do jednego z obliczeń wykonywanych w krokach 4 i 5,
- wprowadzenie losowych błędów do obu obliczeń wykonywanych w krokach 4 i 5.

W pierwszym przypadku, bez utraty ogólności możemy założyć, że atakujący wprowadza losowy błąd typu zamiana bitu do obliczeń podpisu częściowego s_1, podczas gdy podpis s_2 jest poprawny:

$$\begin{aligned}\overline{s_1} &= k^{-1}\left(h(m) - (a \oplus 2^i)r\right) \bmod w_1 = k^{-1}\left(h(m) - (a \pm 2^i)r\right) \bmod w_1 \\ &= k^{-1}\left(h(m) - a\ r\right) \bmod w_1 - k^{-1}\left(\pm 2^i r\right) \bmod w_1 = s_1 \mp 2^i k^{-1} r \bmod w_1\end{aligned}$$

Błędna wartość $\overline{s_1}$ rożni się od wartości poprawnej o czynnik $\mp 2^i k^{-1} r$, który może przystawać do 0 modulo w_1. W takim przypadku wprowadzony błąd nie zmieni wartości s_1 i wykonywane w dalszej części obliczenia jak i wygenerowany podpis ElGamala będą poprawne.

W przypadku gdy $\mp 2^i k^{-1} r \bmod w_1 \neq 0$ w kolejnych krokach algorytmu następuje propagacja błędu powodująca obliczenie błędnej wartości $\overline{v}$ równej

$$\begin{aligned}\overline{v_1} &= \overline{s_1}k - h(m) \bmod w_1 = \left(s_1 \mp 2^i k^{-1} r\right) k - h(m) \bmod w_1 \\ &= -ar \mp 2^i r \bmod w_1,\end{aligned}$$

która następnie służy do obliczenia wartości chińskiego twierdzenia o resztach (krok 7 algorytmu). Zgodnie z definicją 2.2, CRT jest funkcją różnowartościową argumentów $v_1 \bmod w_1$ i $v_2 \bmod w_2$. Oznacza to, że wartość

$$\mathrm{CRT}\left(\overline{v_1}, v_2\right) = -ar + e \bmod (p-1) \neq \mathrm{CRT}\left(v_1, v_2\right) = -ar \bmod (p-1), \tag{6.35}$$

a w konsekwencji w kroku 8 algorytmu nie następuje redukcja $y^r g^v \neq g^0 \bmod p$ i czynnik $T \neq 0$. Niezerowa wartość czynnika rozpraszania błędów T powoduje, że obliczenie wykonywane w ostatnim kroku algorytmu nie przedstawia chińskiego twierdzenia o resztach a uzyskiwany wynik jest istotnie różny od wyniku poprawnego. Odpowiedni dobór parametrów proponowanego rozwiązania zapewnia, że analiza wprowadzonego błędu i jego propagacji w kolejnych obliczeniach algorytmu, nie pozwala na przeprowadzenie ataku. Jeśli bowiem atakujący wprowadzi błąd do jednego z obliczeń, wykonywanych modulo w_1 albo w_2, to niepoprawna wartość $\overline{v}$ rożni się od wartości poprawnej v, odpowiednio o wielokrotność modułu w_2 albo w_1. Właściwość ta wynika bezpośrednio z chińskiego twierdzenia o resztach, które dla poprawnego v_2 i niepoprawnej wartości $\overline{v_1}$ jest równe

$$\begin{aligned}\overline{v} &= \mathrm{CRT}\left(\overline{v_1}, v_2\right) = \left[\overline{v_1} w_2 (w_2^{-1} \bmod w_1) + v_2 w_1 (w_1^{-1} \bmod w_2)\right] \bmod (p-1) \\ &= \left[(v_1 + e) w_2 (w_2^{-1} \bmod w_1) + v_2 w_1 (w_1^{-1} \bmod w_2)\right] \bmod (p-1) \\ &= \left[v + e w_2 (w_2^{-1} \bmod w_1)\right] \bmod (p-1).\end{aligned} \tag{6.36}$$

Oznacza to, że

$$\overline{v} - v = t w_2 \tag{6.37}$$

dla pewnego $t = 1, 2, 3, \ldots, w_1 - 1$. Jeśli moduł w_1 jest mały, to wówczas atakujący może wyznaczyć wszystkie możliwe wartości czynnika rozpraszania błędów T, równego:

$$T = \left[\left(y^r g^{v+tw_2}\right) \bmod p - 1\right] \bmod (p-1) = \left(g^{tw_2} \bmod p - 1\right) \bmod (p-1). \tag{6.38}$$

Znając $\overline{s}, w_1, w_2, p$ i możliwe wartości T atakujący, może próbować odszukać wartości $\overline{s_1}$ i s_2 spełniające zależność wynikającą z chińskiego twierdzenia o resztach

$$\overline{s} = \left[\overline{s_1}\,(w_2 \oplus T)\left(w_2^{-1} \bmod w_1\right) + s_2\,(w_1 \oplus T)\left(w_1^{-1} \bmod w_2\right)\right] \bmod (p-1). \tag{6.39}$$

Choć wyznaczenie $\overline{s}_1$ i s_2 nie jest możliwe, to w przypadku gdy jeden z modułów w_1 albo w_2 jest niewielki, atakujący może odgadnąć wartość jednej z niewiadomych i obliczyć wartość drugiej z nich. Znając $\overline{s}_1, s_2$ może następnie wyznaczyć $\tilde{s} = \mathrm{CRT}\,(\overline{s}_1, s_2)$ ($\tilde{s}$ różni się od $\overline{s}$ tym, że przy jego wyznaczaniu nie uwzględnia się czynnika T) i próbować przeprowadzić analogiczną analizę jak w przypadku ataków [4, 36]. Zabezpieczeniem przed tego typu atakiem jest taki dobór liczby p przy którym w_1 i w_2 będą duże. Utrudni to odgadnięcie nieznanych wartości, wydłuży czas potrzebny na przeprowadzenie ataku i zwiększy jego złożoność — atakujący nie ma bowiem możliwości zweryfikowania poprawności odgadniętych wartości, a przez to musi prowadzić analizę dla wszystkich możliwych wartości.

Przeprowadzenie ataku z uszkodzeniami jest również niemożliwe w przypadku, gdy atakujący wprowadza błędy do obu obliczeń podpisów częściowych s_1, s_2. W takiej sytuacji z prawdopodobieństwem $1 - 1/n$ błędy e_1 i e_1 wprowadzone odpowiednio do s_1 i s_2 są różne. Ponadto z prawdopodobieństwami $1 - 1/w_1$ i $1 - 1/w_2$ błędne wartości $\overline{s_1}$ i $\overline{s_2}$ różnią się od wartości poprawnych s_1, s_2. Warto zauważyć, że jakkolwiek możliwe jest, że jedna z błędnych wartości jest równa wartości poprawnej (tj. wprowadzony błąd jest wielokrotnością jednego z modułów w_i) to niemożliwe jest, aby

$$\begin{aligned} \overline{s_1} &= s_1 \\ \overline{s_2} &= s_2. \end{aligned} \tag{6.40}$$

Gdyby powyższe równania były spełnione to wprowadzony błąd $\mp 2^i k^{-1} r$ przystawałby do 0 modulo w_1 i w_2, a to oznacza również, że przystawałby do 0 modulo $p-1 = w_1 w_2$. Oznacza to, że błąd wprowadzony do tajnego klucza a byłby wielokrotnością $p-1$, wygenerowany podpis byłby poprawny, a przeprowadzenie ataku niemożliwe. Jeśli natomiast błąd przystaje do 0 modulo jeden z modułów w_i, to wówczas mamy do czynienia z sytuacją taką samą, jak przy wprowadzeniu błędu do jednego z obliczeń wykonywanych w kroku 4 albo 5 proponowanego algorytmu.

Gdy oba podpisy częściowe są obarczone błędem, wówczas wyznaczone w kroku 6 wartości $\overline{v_1}$ i $\overline{v_2}$ różnią się od wartości poprawnych powodując wyznaczenie niepoprawnej wartości

$$\overline{v} = \mathrm{CRT}\,(\overline{v_1}, \overline{v_2}) = -ar + e \bmod (p-1) \neq \mathrm{CRT}\,(v_1, v_2) = -ar \bmod (p-1). \tag{6.41}$$

Konsekwencją niepoprawnej wartości $\overline{v}$ jest $T \neq 0$ powodujące, podobnie jak w przypadku wcześniejszym, że obliczenie wykonywane w ostatnim kroku algorytmu nie przedstawia chińskiego twierdzenia o resztach. W rezultacie powstałych błędów wygenerowany podpis $\overline{s}$ jest istotnie różny od podpisu poprawnego.

Wprowadzając błędy do obu obliczeń może się również zdarzyć, że atakujący wprowadzi do klucza prywatnego, dwukrotnie taki sam błąd e. W takim przypadku niepoprawna wartość $\overline{v}$ będzie również obarczona błędem e. Jeśli atakujący potrafi wprowadzać błędy określonego rodzaju (lub wręcz konkretne błędy określając precyzyjnie ich lokalizacje, miejsce i postać), to wówczas łatwo może ograniczyć prawdopodobne wartości czynnika T (w skrajnym przypadku mógłby nawet znać jego wartość zakładając, że potrafi wprowadzać żądane błędy) i szukać wartości $\overline{s}_1$ i $\overline{s}_2$. Prawdopodobieństwo przeprowadzenia tego typu ataku jest jednak niewielkie. Po pierwsze dlatego, że prawdopodobieństwo wprowadzenia dwukrotnie tego samego błędu wynosi $1/n$ i dla liczb pierwszych stosowanych w algorytmie ElGamala jest bliskie zero. Po drugie przy atakach z uszkodzeniami nigdy nie zakłada się, że atakujący ma pełną kontrolę nad wprowadzanym błędem. Z tego względu atakujący ma informację jedynie o prawdopodobnych wartościach czynnika T i musi odgadnąć jego rzeczywistą wartość.

Zaproponowana modyfikacja algorytmu ElGamala pozwala zwiększyć odporność na ataki z uszkodzeniami dzięki zastosowaniu rozpraszania błędów i uwikłaniu analizy błędnych podpisów. Skuteczność zaproponowanych zabezpieczeń wymaga jednak nałożenia na parametry schematu ElGamala dodatkowych ograniczeń:

- $p - 1$ musi posiadać co najmniej trzy różne dzielniki pierwsze,
- moduły w_1 i w_2 powinny być mniej więcej tych samych rozmiarów.

Niespełnienie powyższych wymagań spowoduje uproszczenie analizy wprowadzanych błędów a w konsekwencji może negatywnie wpłynąć na gwarantowany poziom ochrony przed atakami z uszkodzeniami. Duże wartości modułów w_1 i w_2 zwiększają złożoność obliczeniową i pamięciową ataku, przez co korzystnie wpływają na bezpieczeństwo całego algorytmu.

6.3 Ochrona schematu podpisów cyfrowych DSA

W przypadku schematu podpisów DSA zastosowanie rozdzielania obliczeń i wykorzystanie chińskiego twierdzenia o resztach jest niemożliwe. Ograniczenie to wynika z tego, że podpis w algorytmie DSA jest generowany w podgrupie rzędu q grupy Z_p^*, gdzie q jest liczbą pierwszą. W praktyce oznacza to, że podpis s wyznaczany jest jako

$$s = k^{-1} \left(H(m) + ar\right) \bmod q. \tag{6.42}$$

Ponieważ q jest pierwsze to operacji tej nie można rozdzielić, tak jak to było możliwe w przypadku podpisów ElGamala.

Z tego względu proponowana modyfikacja zwiększająca odporność algorytmu DSA na kryptoanalizę z uszkodzeniami wykorzystuje nieliniowość operacji odwrotności multiplikatywnej $k^{-1} \bmod q$, jako mechanizm rozpraszania błędów wprowadzonych do obliczeń. Zaletą wykorzystania odwrotności jest to, że nawet w przypadku wprowadzenia tego samego błędu e do różnych wartości k (wybieranych losowo przy każdym wykonaniu algorytmu) wyznaczane wartości odwrotności obarczone są różnymi błędami E zależącymi nieliniowo od k i e. W praktyce oznacza to, że atakujący nie potrafi wprowadzić błędów w taki sposób, aby błąd E był określonego typu — np.: typu zamiana pojedynczego bitu.

Powyższa obserwacja stała się podstawą modyfikacji algorytmu DSA (Alg. 6.3).

Algorytm 6.3 Generacja podpisu DSA

Wejście: wiadomość m, klucz prywatny a

Wyjście: podpis $\langle r, s \rangle$

1: wylosuj liczbę pierwszą k taką, że $1 \leq k \leq q-1$,

2: oblicz $r = g^k \bmod p \mod q$,

3: oblicz $v = k + ar \bmod q$,

4: oblicz $T = (y^{-r} g^v) \bmod p - r \bmod q$,

5: oblicz $k' = (k \oplus T)^{-1} \bmod q$,

6: oblicz $s = k' (H(m) + v) - 1 \bmod q$.

Twierdzenie 6.2. Poprawny podpis DSA wygenerowany za pomocą algorytmu (Alg. 6.3) jest identyczny z podpisem generowanym za pomocą standardowego algorytmu DSA (Alg. 2.10).

Dowód. W przypadku poprawnego wykonania algorytmu (Alg. 6.3) czynnik T przyjmuje wartość

$$\begin{aligned} T &= \left(y^{-r} g^v\right) \bmod p - r \bmod q = \left(g^{-ar} g^{k+ar}\right) \bmod p - r \bmod q \\ &= g^k \bmod p - r \bmod q = r - r = 0. \end{aligned} \tag{6.43}$$

W konsekwencji $k' = k^{-1} \bmod q$ i podpis wyznaczany w ostatnim kroku algorytmu jest równy

$$\begin{aligned} s &= k' (H(m) + v) - 1 \bmod q = k^{-1} (H(m) + k + ar) - 1 \bmod q \\ &= k^{-1} (H(m) + ar) + k^{-1} k - 1 \bmod q = k^{-1} (H(m) + ar) \bmod q, \end{aligned} \tag{6.44}$$

a tym samym identyczny jak podpis generowany w ostatnim kroku standardowego algorytmu DSA. □

Tym samym, w przypadku gdy atakujący nie wprowadza błędów, podpis generowany przez zaproponowaną modyfikację jest identyczny z podpisem generowanym przez oryginalny algorytm DSA. Z tego

względu podpis ten może być więc weryfikowany za pomocą standardowego algorytmu weryfikacji podpisów.

Bezpieczeństwo zaproponowanej modyfikacji algorytmu DSA polega na zagwarantowaniu propagacji i rozproszenia każdego błędu wprowadzonego do klucza prywatnego a. Propagacja i rozproszenie powoduje, że czynnik T ma pseudolosową wartość różną od zera. Podobnie jak w przypadku modyfikacji algorytmu ElGamala, klucz publiczny y jest wykorzystywany w procedurze wyznaczania czynnika T. Pozwala to na sprawdzenie czy klucz prywatny a, użyty do generacji podpisu, odpowiada kluczowi publicznemu. Jeśli atakujący nie wprowadził błędów do a, to w czasie obliczania T redukują się elementy zależne od klucza prywatnego. W przeciwnym przypadku redukcja nie następuje, a wprowadzony błąd powoduje wyznaczenie niezerowej wartość czynnika T.

W przypadku wprowadzenia błędu e do klucza prywatnego ($\overline{a} = a + e$), w trakcie generacji podpisu cyfrowego, obliczenia prowadzone w kolejnych krokach są obarczone błędem:

$$\overline{v} = k + \overline{a}r \bmod q = k + (a+e)\, r \bmod q = k + ar + er \bmod q \tag{6.45}$$

$$T = \left(g^{-ar} g^{k+ar+er}\right) \bmod p \; - r \bmod q = g^{er} \bmod p \bmod q. \tag{6.46}$$

Konsekwencją wprowadzenia błędu jest wyznaczenie wartości czynnika $T = g^{er} \bmod p \bmod q$, która jest równa zero wtedy i tylko wtedy, gdy er jest wielokrotnością q (ponieważ g jest generatorem grupy rzędu q). Ponieważ obliczane w drugim kroku algorytmu r jest mniejsze od q i błąd e nie jest wielokrotnością q to iloczyn er także nie jest wielokrotnością q. Proponowane rozwiązanie zabezpieczające gwarantuje więc, że dla każdego błędu e wprowadzonego do klucza prywatnego schematu podpisów DSA, czynnik T jest różny od 0.

Niezerowa wartość T powoduje obliczenie niepoprawnej wartości odwrotności multiplikatywnej

$$\overline{k'} = (k \oplus T)^{-1} \bmod q = k^{-1} + E \bmod q, \tag{6.47}$$

i w konsekwencji wygenerowanie błędnego podpisu

$$\begin{aligned}
\overline{s} &= \overline{k'}\,(H(m) + \overline{v}) - 1 \bmod q = \left(k^{-1} + E\right)(H(m) + v + er) - 1 \bmod q \\
&= k^{-1}\left[(H(m) + v + er) + kE\,[H(m) + v + er]\right] - 1 \bmod q \\
&= k^{-1}\left[(H(m) + v) + er + kE\,[H(m) + v + er]\right] - 1 \bmod q \\
&= k^{-1}\left[(H(m) + ar) + er + kE\,[H(m) + k + ar + er]\right] \bmod q \\
&= k^{-1}\left[s + er + kE\,[H(m) + k + (a+e)\, r]\right] \bmod q.
\end{aligned} \tag{6.48}$$

Błędny podpis wygenerowany za pomocą zaproponowanego algorytmu rożni się od podpisu poprawnego o czynnik $k^{-1}er$ i $E\,(H(m) + k + (a+e)\, r)$. Pierwszy z nich występuje również w przypadku standardowego algorytmu DSA i umożliwia przeprowadzenie ataku, jeśli tylko atakujący zna postać błędu e (np:

wie, że są to błędy pojedyncze). W przypadku zaproponowanej modyfikacji występuje dodatkowo drugi czynnik, którego wartość zależy zarówno od wprowadzonego błędu e jak i losowej wartości k. Ponadto zależność od k jest dwojaka: czynnik ten występuje bezpośrednio w wyrażeniu, a także, poprzez algorytm wyznaczania T i odwrotności multiplikatywnej, wpływa na wartość błędu E. Powoduje to, że metoda analizy błędnego podpisu stosowana w przypadku standardowego algorytmu DSA jest nieskuteczna. Poszukuje ona bowiem wartości wprowadzonego błędu dla którego spełniona będzie równość

$$A\left(g^{H(m)}y^{r}g^{er}\right)^{\overline{s}^{-1}} \bmod p \bmod q = r. \tag{6.49}$$

W powyższym wzorze czynnika A wynika z wprowadzonych, w proponowanym algorytmie, propagacji i rozpraszania uszkodzeń. Jeśli atakujący, dysponując niepoprawnym podpisem i parametrami zmodyfikowanego schematu DSA, będzie potrafił wyznaczyć czynnik A, to będzie mógł również poszukiwać wprowadzonego błędu e.

Upraszczając równanie (6.49) otrzymujemy

$$A\ \left(g^{H(m)+(a+e)r}\right)^{\overline{s}^{-1}} \bmod p \bmod q = A\ \left(g^{s+er}\right)^{\frac{k}{s+er+kE(H(m)+k+(a+e)r) \bmod q}} \bmod p \bmod q. \tag{6.50}$$

Powyższe wyrażenie będzie równe r wtedy i tylko wtedy jeśli

$$A\ = \left(g^{kE(H(m)+k+(a+e)r)}\right)^{\overline{s}^{-1}} \bmod p. \tag{6.51}$$

Obliczenie A wymaga jednak znajomości liczby losowej k. Znajomość $r = g^k \bmod p \bmod q$ jest niewystarczająca, podobnie jak znajomość $r' = g^k \bmod p$ (zakładając, że atakujący będzie dysponował wiedzą pozwalającą wyznaczyć r' na podstawie parametrów schematu i podpisu). Jeśli bowiem atakujący zna wartość r', to informacja ta pozwala mu uprościć jedynie część obliczeń

$$A\ = r'^{\overline{s}^{-1}E(H(m)+k+(a+e)r)} \bmod p. \tag{6.52}$$

Zgodnie z powyższym wzorem wyznaczenie A i możliwość przeprowadzenia ataku wymaga znajomości liczby losowej k.

Powyższe analizy pozwalają zauważyć, że każdy błąd $0 < e < q$ wprowadzony do klucza prywatnego a, w czasie generacji podpisu, spowoduje wartość czynnika $T \neq 0$, a w efekcie wyznaczenie błędnej wartości odwrotności multiplikatywnej. Ze względu na nieliniowość tej operacji błąd E wprowadzany do wartości k' zależy zarówno od wartości k jak i e. Znajomość wartości E nie daje atakującemu jeszcze żadnej przewagi, gdyż nie dostarcza ani informacji o a ani o k. Zauważmy bowiem, że na podstawie (6.46,6.47) i znając e oraz E, atakujący może próbować odszukać k spełniające zależność

$$(k \oplus g^{er})^{-1} \bmod q = k^{-1} + E \bmod q. \tag{6.53}$$

Znalezienie k spełniającego powyższe równanie wymaga jednak obliczenia odwrotności multiplikatywnej dla wszystkich możliwych k. Z tego powodu atak taki ma taką samą złożoność jak przegląd zupełny w poszukiwaniu logarytmu dyskretnego z y.

Atak jest również niemożliwy w przypadku, gdy atakujący wprowadza błędy do innych elementów schematu podpisów. Jednym z typowych schematów ataku na wcześniejsze modyfikacje schematu DSA była próba wprowadzenia błędu dopiero w ostatnim kroku zaproponowanego algorytmu i odszukiwanie użytego klucza prywatnego poprzez rozwiązanie problemu HNP (Def. 3.6). W ten sposób wszystkie wcześniejsze obliczenia są poprawne powodując, że czynnik $T = 0$ i rozpraszanie błędów nie występuje. Jednak, po wprowadzeniu błędu do wartości v w ostatnim kroku algorytmu, urządzenie generuje błędny podpis $\langle r, \overline{s}\rangle$, w którym $\overline{s} = k^{-1}\left(h(m) + v + e\right) - 1 \bmod q$. Korzystając z tego podpisu atakujący przeprowadza atak zgodnie z typową procedurą, tj. oblicza:

$$\left(g^{h(m)} y^r g^e\right)^{\overline{s}^{-1}} \bmod p \bmod q, \tag{6.54}$$

dla różnych wartości możliwych błędów e (np.: błędy typu zamiana bitu) szukając wartości e dla której zachodzi równość

$$\left(g^{h(m)} y^r g^e\right)^{\overline{s}^{-1}} \bmod p \bmod q = r. \tag{6.55}$$

Po odszukaniu błędu e spełniającego to równanie atakujący zna błąd wprowadzony do v w ostatnim kroku schematu podpisów. Jeśli atakujący potrafi wprowadzać błędy szczególnej postaci (np.: zamiana bitów), to wówczas otrzymuje częściową informację na temat v. Zbierając wiele podpisów $\langle r_i, \overline{s_i}\rangle$ i gromadząc informacje o v_i atakujący może próbować wyznaczyć klucz a rozwiązując problem HNP dla równania z trzeciego kroku algorytmu

$$v_i = k_i + a r_i \bmod q. \tag{6.56}$$

Na szczęście działanie takie jest możliwe jedynie w przypadku, gdy atakujący zna l najmniej znaczących bitów v_i oraz, co najważniejsze, liczby k_i są stałe. Oznacza to, że w proponowanej modyfikacji algorytmu DSA tego typu atak nie może być wykorzystany. Przeszkodą dla atakującego jest bowiem:

- zależność liczby znanych bitów od rodzaju wprowadzonego błędu — oznacza to, że atakujący, który potrafi wprowadzać tylko błędy typu zamiana pojedynczego bitu posiada zbyt mało informacji, aby móc odtworzyć klucz prywatny a (powtarzanie procedury podpisywania, nawet przy założeniu wprowadzenia tego samego błędu, też nie pozwala na zebranie informacji o wymaganej liczbie bitów, ponieważ w każdym podpisie używana jest inna liczba losowa k),
- zmiana wartości liczb k_i, które są losowo wybierane podczas każdej iteracji algorytmu.

Z powyższych rozważań wynika, że przeprowadzenie ataku bazującego na sprowadzeniu poszukiwania klucza prywatnego a do rozwiązania problemu HNP jest niemożliwe.

6.4 Złożoność implementacyjna proponowanych rozwiązań

Przedstawiona w rozdziale 5.4 złożoność implementacyjna algorytmu AES została oszacowana przy założeniu, że algorytm jest implementowany w układzie FPGA. Przeprowadzenie takiego oszacowania było możliwe ze względu na rodzaj wykorzystywanych operacji, oraz to, że posiadają one prostą implementację sprzętową. W przypadku modyfikacji schematów ElGamala i DSA sprawa jest bardziej skomplikowana. Schematy te, podobnie jak proponowane modyfikacje, bazują bowiem na arytmetyce modularnej dużych liczb i wykorzystują w swoim działaniu operacje dodawania, mnożenia i potęgowania modulo oraz obliczanie odwrotności multiplikatywnej. Ze względu na złożoność tych operacji są one zazwyczaj implementowane w postaci sekwencji innych operacji — mnożenie jest implementowane jako sekwencja dodawań, potęgowanie jako sekwencja mnożeń itd.). Taki sposób implementacji powoduje, że w implementacji sprzętowej konstruowana jest jedna jednostka arytmetyczna realizująca wszystkie potrzebne operacje. Porównywanie złożoności implementacyjnej na poziomie pojedynczych bramek czy tablic LUT traci więc sens.

Z tego powodu oszacowania złożoności proponowanych modyfikacji schematów podpisów cyfrowych, podobnie jak większości algorytmów asymetrycznych, wykonywane jest poprzez oszacowanie liczby elementarnych operacji potrzebnych do ich realizacji. Złożoność operacji określa się poprzez podanie oszacowania liczby operacji elementarnych niezbędnych do jej zrealizowania, lub uznaje się ją za pomijalnie małą, w porównaniu do pojedynczej operacji elementarnej. W oszacowaniach złożoności algorytmów wykorzystujących arytmetyką modularną, zazwyczaj zakłada się, że operacją elementarną jest pojedyncze mnożenie modulo. Założenie takie wynika z tego, że złożoność operacji dodawania modulo jest bardzo mała (wymaga wykonania jednego dodawania w liczbach rzeczywistych i ewentualnego odjęcia modułu) oraz, że mnożenie modulo jest elementarną operacją pozostałych operacji. Złożoność dodawania modulo, w takim przypadku, uznaje się za pomijalnie małą i pomija w oszacowaniach złożoności całego algorytmu.

Oszacowanie złożoności implementacyjnej podane w niniejszym rozdziale zostało wykonane na podstawie liczby i rodzajów operacji modulo, których wykonania wymagają proponowane rozwiązania zabezpieczające. Złożoność ta została również porównana ze złożonością oryginalnych schematów generowania podpisów. Dodatkowo porównano ją ze złożonością procedur weryfikacji podpisów, które są najprostszym sposobem w jaki można próbować ochronić algorytmy ElGamala i DSA przed kryptoanalizą z uszkodzeniami.

Złożoność implementacyjna proponowanego rozwiązania dla schematu ElGamala

W porównaniu do standardowego schematu podpisów ElGamala (Alg. 2.7), zaproponowany algorytm wykonuje dodatkowe operacje mające zagwarantować, że każdy błąd wprowadzony do tajnego klucza a

zostanie rozpropagowany powodując istotną zmianę wygenerowanego podpisu s.

W oryginalnym schemacie wygenerowanie podpisu s polega na wykonaniu pojedynczego obliczenia modulo

$$s = k^{-1}\left(h(m) - ar\right) \bmod (p-1), \tag{6.57}$$

gdzie $k^{-1} \bmod (p-1)$, $h(m)$ i r zostały wyznaczone wcześniej. Obliczenie to wymaga więc wykonania dwóch mnożeń i jednego dodawania modulo $p-1$.

W przypadku zmodyfikowanego schematu ElGamala obliczenie podpisu s wykonywane jest poprzez wyznaczenie podpisów częściowych s_i modulo w_i a następnie wyznaczenie właściwego podpisu z chińskiego twierdzenia o resztach

$$\begin{aligned} s_1 &= k^{-1}\left(h(m) - ar\right) \bmod w_1 \\ s_2 &= k^{-1}\left(h(m) - ar\right) \bmod w_2 \\ s &= \left(s_1\left(w_2 \oplus T\right)\left(w_2^{-1} \bmod w_1\right) + s_2\left(w_1 \oplus T\right)\left(w_1^{-1} \bmod w_2\right)\right) \bmod (p-1) \end{aligned} . \tag{6.58}$$

Ze względu na przyjęte założenie, że wartości modułów w_1 i w_2 są stałe, odwrotności $w_1^{-1} \bmod w_2$ i $w_2^{-1} \bmod w_1$ mogą być wyznaczone jednokrotnie i zapisane w urządzeniu. Zakładając dalej, że w_1 i w_2 mają mniej więcej ten sam rozmiar bitowy (są o około połowę krótsze niż moduł p), można przyjąć, że złożoność powyższych trzech operacji jest porównywalna ze złożonością obliczenia standardowego podpisu s.

Dodatkowe operacje, zwiększające złożoność zaproponowanego algorytmu obejmują wyznaczenie czynnika rozpraszania błędów T, które wykonywane jest w czterech krokach:

$$\begin{aligned} v_1 &= s_1 k - h(m) \bmod w_1 \\ v_2 &= s_2 k - h(m) \bmod w_2 \\ v &= \mathrm{CRT}\left(v_1, v_2\right) \\ T &= \left(\left(y^r g^v \bmod p\right) - 1\right) \bmod (p-1) \end{aligned} . \tag{6.59}$$

Najbardziej złożoną operacją tego obliczenia jest wykonanie dwukrotnego potęgowania i mnożenia modulo p. Złożoność pozostałych operacji wykonywanych modulo w_1 i w_2 oraz CRT może być pominięta ze względu na mniejsze rozmiary modułów w_1 i w_2. Dodatkowo, możliwe jest przyspieszenie obliczania CRT jeśli zostanie ono zaimplementowane zgodnie z algorytmem Garnera [62].

Ze względu na dodatkowe operacje potęgowania modulo p i mnożenia modulo w_1, w_2, złożoność całego, zmodyfikowanego algorytmu ElGamala, jest nieco większa od złożoności algorytmu weryfikacji podpisu (Alg. 2.8), gdzie wykonywane są trzy potęgowania i dwa mnożenia modulo p.

Zaproponowana modyfikacja posiada również zwiększoną złożoność pamięciową w porównaniu do algorytmu oryginalnego. Zwiększenie to wynika z konieczności zapamiętania w urządzeniu modułów w_1 i w_2, ich odwrotności oraz klucza publicznego y. Zakładając, że rozmiary bitowe modułów w_1, w_2 są o około

połowę mniejsze niż rozmiar bitowy liczby p to przechowanie tych dodatkowych danych wymaga $O(3n)$ bitów. W porównaniu do standardowej implementacji algorytmu ElGamala, wymagającej przechowania liczb p, g i a, oznacza to dwukrotne zwiększenie złożoności pamięciowej. W rzeczywistości jednak złożoność ta jest zwiększona o około 50%, ponieważ urządzenia do składania podpisów cyfrowych przechowują najczęściej również i informacje potrzebne do ich weryfikacji — a więc klucz publiczny y. Rozwiązanie takie jest powszechnie stosowane w kartach μP dzięki czemu każdy ma możliwość szybkiego dostępu do klucza publicznego i weryfikacji podpisów.

Nowym wymaganiem, w stosunku do oryginalnego schematu ElGamala, jest konieczność odpowiedniego doboru liczby p. Liczba ta musi być tak dobrana, aby wielkość modułów w_1 i w_2 uniemożliwiała atak z wykorzystaniem przeglądu zupełnego. Wymusza to na użytkowniku, ustalającym parametry schematu, faktoryzację liczby $p-1$ i sprawdzenie czy jej dzielniki p_i można tak pogrupować, aby zagwarantowały odpowiednie wartości w_1, w_2. Z przeprowadzonych testów wynika, że dla 768-bitowych liczb p wybieranych losowo, prawdopodobieństwo, że $w_1, w_2 > 2^{11}$ jest nie mniejsze niż 33%. Prawdopodobieństwo to wynika z obserwacji, że dla nieco ponad 33% losowo wybieranych liczb p, $p-1$ ma 6 lub więcej dzielników pierwszych. Oznacza to, że minimalna wartość mniejszego z modułów wynosi $\prod_{i=1}^{5} p_i = 2310 > 2^{11}$. Wartość modułów $w_1, w_2 > 2^{11}$ nie jest wystarczająca do ochrony przed atakiem jednak należy pamiętać, że jest to oszacowanie najgorszego przypadku, kiedy liczba $p-1$ ma jeden duży i resztę małych dzielników. Przeprowadzone testy pokazały, że dla około 24.9%, 10.6% i około 1% losowo wybranych liczb pierwszych moduły w_1 i w_2 są większe od odpowiednio 2^{48}, 2^{64} i 2^{128}. Warto zaznaczyć, że wyniki te są zaniżone ze względu na przyjęty algorytm faktoryzacji dużych liczb, który poszukiwał dzielników mniejszych od 100 000 493.

Złożoność implementacyjna proponowanego rozwiązania dla schematu DSA

Porównując standardowy algorytm generacji podpisów DSA (Alg. 2.10) z zaproponowaną modyfikacją (Alg. 6.3) można łatwo zauważyć, że rozdziela ona operacje wyznaczania podpisu

$$s = k^{-1} \left(H(m) + ar \right) \bmod q \tag{6.60}$$

na dwa etapy

$$v = k + ar \bmod q \tag{6.61}$$

$$s = k' \left(H(m) + v \right) - 1 \bmod q. \tag{6.62}$$

Dodatkowe modyfikacje, dodanie k w pierwszym równaniu i odjęcie 1 w drugim, mają za zadanie zabezpieczyć zmodyfikowany algorytm przed atakami wykorzystującym HNP oraz zagwarantować zgodność

ze standardowym schematem DSA. W proponowanym rozwiązaniu zmodyfikowano również obliczenie odwrotności multiplikatywnej poprzez dodanie dodatkowej operacji XOR liczby losowej k i czynnika rozpraszania błędów T. Obie modyfikacje wprowadzają konieczność wykonania dwóch dodatkowych dodawań modulo q i jednej operacji XOR.

Dodatkową operacją wykonywaną w proponowanym algorytmie jest wyznaczenie czynnika

$$T = \left(y^{-r} g^{v}\right) \bmod p \ - r \bmod q. \tag{6.63}$$

Powyższe obliczenie wymaga wykonania dwóch potęgowań i mnożenia modulo p oraz jednego dodawania modulo q. Ze względu na to, że rozmiar liczby q jest znacznie mniejszy niż rozmiar p, złożoność dodatkowych operacji dodawania modulo q i operacji XOR może być pominięta. Złożoność proponowanej modyfikacji zależy zatem od dwóch operacji potęgowania modulo p. Tym samym jest ona mniejsza niż złożoność algorytmu weryfikacji podpisu DSA, w czasie którego wykonywane są trzy potęgowania i dwa mnożenia modulo p.

Zaproponowana modyfikacja posiada również zwiększoną złożoność pamięciową. Wynika to z konieczności przechowania klucza publicznego y wykorzystywanego do wyznaczania czynnika T. Podobnie jednak jak w przypadku algorytmu ElGamala klucz ten jest bardzo często przechowywany przez urządzenia do składania podpisu. W takim przypadku proponowana modyfikacja nie zwiększa złożoności pamięciowej.

6.5 Skuteczność proponowanego rozwiązania

Idea rozwiązania zabezpieczającego zaproponowanego dla schematów podpisów cyfrowych ElGamala i DSA jest rozszerzeniem rozwiązań zaproponowanych dla algorytmu RSA. Podstawą zapewnienia bezpieczeństwa jest propagacja błędów powodująca wygenerowanie błędnego wyniku, nieprzydatnego do kryptoanalizy. W rozdziale 4.4 przedstawiono trzy cechy idealnego rozwiązania zabezpieczającego tego typu:

- zapewniać odporność na błędy przemijające i trwałe wprowadzane do różnych elementów algorytmu kryptograficznego,
- w przypadku wystąpienia błędów generować wyniki utrudniające bądź uniemożliwiające przeanalizowanie i odtworzenie zależności błędu od wykorzystywanego klucza kryptograficznego czy przetwarzanej wiadomości,
- być silnie zintegrowanym z algorytmem kryptograficznym.

Dwa pierwsze wymagania są oczywiste a ich spełnienie niezbędne do zagwarantowania bezpieczeństwa. Trzeci warunek ma na celu zapewnić, że rozwiązanie zabezpieczające nie może zostać wyłączone bez kon-

sekwencji dla poprawnego działania całego algorytmu. Trudnością przy projektowaniu rozwiązań zabezpieczających poprzez rozpraszanie błędów jest spełnienie wszystkich tych trzech warunków jednocześnie.

Dobrym przykładem jest tu rozwiązanie zaproponowane w pracy [85], którego celem była ochrona algorytmu RSA (Alg. 4.1). W zaproponowanym algorytmie rozwiązanie zabezpieczające jest silnie związane z algorytmem RSA powodując, że wyłączenie czy zmiana wyznaczanych wartości spowoduje błędny wynik algorytmu, który w ogóle nie będzie wynikiem szyfrowania RSA. Rozwiązanie to zapewnia silną integrację, nie gwarantując jednak odporności na atak z uszkodzeniami ze względu na nie spełnienie pierwszego warunku (rozdział 4.4). Podobnym przykładem może być modyfikacja algorytmu RSA zaproponowana przez Blömera [15], która również silnie integruje algorytmy zabezpieczające z RSA pozostając nadal podatna na niektóre ataki z uszkodzeniami [16, 78].

Sytuacja wygląda inaczej w przypadku rozwiązania zabezpieczającego zaproponowanego przez Kima i innych [56]. Rozwiązanie to jest jak dotąd uważane za rozwiązanie gwarantujące odporność na ataki z uszkodzeniami wykorzystującymi błędy losowe. Tym samym, dla ustalonego modelu błędu, proponowany algorytm zapewnia odporność na atak i spełnia dwa pierwsze warunki. Z drugiej strony, już pobieżna analiza algorytmu pozwala zauważyć, że rozwiązanie zabezpieczające nie jest silnie zintegrowane z samym algorytmem. Fakt ten daje atakującemu możliwość wyłączenia mechanizmów ochronnych i przeprowadzenia ataku. W przypadku rozwiązania [56] możliwość wprowadzenia błędów ustawiających wartość wybranych bajtów, pozwala atakującemu wyeliminować wpływ czynnika rozpraszania błędów na przebieg algorytmu a tym samym wyłączyć rozpraszanie błędów.

Podobna sytuacja zachodzi w przypadku zaproponowanej modyfikacji schematów podpisów cyfrowych ElGamala i DSA. Modyfikacje te spełniają dwa pierwsze warunki idealnego rozwiązania zabezpieczającego gwarantując, że każdy błąd zostanie wykryty, powodując jednocześnie wygenerowanie wyników błędnych, nieprzydatnych z punktu widzenia kryptoanalizy z uszkodzeniami. Rozwiązania te nie zapewniają natomiast silnej integracji mechanizmów zabezpieczających ze schematami podpisów. Oznacza to, że jeśli atakujący potrafi wyłączyć procedury wyznaczania czynnika T albo spowodować wyzerowanie jego wartości to cały algorytm będzie nadal działał poprawnie pozostając podatny na ataki. Zagrożenie to nie jest jednak poważne, gdyż przyjmując model błędów (Def. 6.1) prawdopodobieństwo, że atakujący wprowadzi błąd zerujący wartość czynnika T jest bardzo małe.

Modyfikacje schematów podpisów cyfrowych ElGamala i DSA mają złożoność obliczeniową zbliżoną do złożoności procedur weryfikacji tych podpisów. Oznacza to, że podobną złożonością cechują się oczywiste rozwiązania zabezpieczające bazujące na weryfikacji podpisu cyfrowego. Zabezpieczenie takie posiada te same cechy co proponowana modyfikacja — pozwala wykryć wszystkie wprowadzone błędy, pozostając słabo zintegrowana z algorytmem podpisów. Wadą takiej procedury zabezpieczającej, oprócz możliwości

wyłączenia weryfikacji, jest jednak to, że wykrycie błędu polega na sprawdzeniu czy zachodzi równość

$$\left(g^{h(m)}y^r\right)^{s^{-1}} \bmod p \bmod q = r. \tag{6.64}$$

W rzeczywistych implementacjach weryfikacja taka polega na odjęciu obu stron równości od siebie i ustawieniu flagi informującej czy otrzymany wynik był równy zero czy nie (a tym samym czy obie wartości były takie same czy nie). Następnie, na podstawie wartości tej flagi, wygenerowany podpis jest podawany na wyjście urządzenia albo kasowany [85]. Procedura taka daje atakującemu możliwość wprowadzenia błędu modyfikującego ustawienie flagi i spowodowanie, że błędny podpis zostanie wygenerowany przez urządzenie i podany na wyjście urządzenia [84, 85].

Brak procedury decyzyjnej w zaproponowanych modyfikacjach podpisów jest zaletą minimalizującą możliwość przeprowadzenia tego typu ataków.

Rozdział 7

Podsumowanie i wnioski

W niniejszej rozprawie poruszono problem ataków z uszkodzeniami, które od mniej więcej dziesięciu lat stanowią jedno z najbardziej poważnych zagrożeń dla współcześnie projektowanych i stosowanych algorytmów kryptograficznych. Ogromne możliwości przeprowadzenia ataku oraz ich duża skuteczność w połączeniu z trudnością ochrony przed nimi powodują, że zagadnienie ochrony algorytmów kryptograficznym jest problem złożonym i ciągle nierozwiązanym. Projektowanie rozwiązań zabezpieczających wymaga połączenia i praktycznego zastosowania wiedzy z różnych gałęzi nauki — kryptografii, techniki cyfrowej i technologii wykonywania układów kryptograficznych.

W rozprawie przedstawiono metody ataku na urządzenia kryptograficzne wykorzystujące różnego rodzaju techniki wprowadzania uszkodzeń. Metody te pogrupowano ze względu na rodzaj błędów powodowanych w urządzeniach. W podobny sposób sklasyfikowano metody ochrony układów kryptograficznych. Ta część pracy pozwala zauważyć, że pierwszą linią obrony przed atakami jest zapobieganie wprowadzaniu błędów do urządzeń kryptograficznych. W sytuacji, gdy jest to niemożliwe, drugą linią obrony jest wykrywanie błędów i przeciwdziałanie ich skutkom. Może się to odbywać poprzez powtórzenie obliczeń, które dały niepoprawny rezultat, albo poprzez przerwanie wykonywania algorytmu czy korekcję błędów. Ostatnim sposobem ochrony jest rozpraszanie błędów, którego celem jest zapewnienie, że błędne wyniki, generowane przez urządzenie kryptograficzne, będą nieprzydatne do przeprowadzenia ataku.

W ramach pracy doktorskiej przeanalizowano możliwości ochrony symetrycznego algorytmu szyfrowania AES oraz schematów podpisów cyfrowych ElGamala i DSA. Dla algorytmów tych zaproponowano rozwiązania pozwalające zwiększyć ich odporność na kryptoanalizę z uszkodzeniami. W przypadku algorytmu AES, zaproponowane rozwiązanie wykorzystuje kody korekcyjne w celu ochrony elementarnych operacji algorytmu i zagwarantowania, że szyfrogram wygenerowany przez urządzenie będzie poprawny. W przypadku schematów podpisów cyfrowych, ze względu na trudność ochrony operacji modulo za pomocą kodów korekcyjnych, zaproponowano rozwiązanie wykorzystujące rozpraszanie uszkodzeń.

W przypadku algorytmu AES rozbudowano propozycję wprowadzoną po raz pierwszy przez Bertoniego [7, 8], które pozwalało jedynie na detekcję błędów. Zaproponowane rozszerzenie tego rozwiązania daje możliwość wykrywania i korygowania wprowadzanych błędów, poprawiając właściwości dotychczasowych rozwiązań. Zaproponowana modyfikacja pozwala wykryć wszystkie błędy stosowane w atakach z uszkodzeniami na AES i skorygować ponad 50% z nich (100% błędów pojedynczych i połowę błędów wprowadzanych do bajtów). Zaproponowane rozwiązanie wykorzystuje algebraiczne właściwości algorytmu AES powodujące, iż implementacja wykorzystywanych w nim operacji może zostać uproszczona i sprowadzona do dodawania modulo 2. Stąd, w przypadku trzech z czterech elementarnych transformacji AES, wprowadzone modyfikacje cechują się małym narzutem implementacyjnym. Na ich tle gorzej wypada nieliniowa transformacja SubBytes, dla której procedury korekcji cechują się dużą złożonością implementacyjną. Złożoność ta wynika zarówno z nieliniowego charakteru transformacji jak i z tego, że proponowane rozwiązanie wykorzystuje kontrolę parzystości, która jest funkcją liniową. Dalsze badania powinny pozwolić na usprawnienie i uproszczenie procedur korekcji, obniżając złożoność implementacyjną całego rozwiązania. Pewne rozwiązania zmierzające w tym kierunku są możliwe do uzyskania drogą odpowiedniej interpretacji oraz realizacji transformacji SubBytes. Standardowa interpretacja tej transformacji, przekształcenie afiniczne elementów ciała $\mathrm{GF}(2^8)$, może być bowiem zastąpiona interpretacją dla ciała $\mathrm{GF}\left(\left(2^4\right)^2\right)$ [77]. Ciało to jest izomorficzne w stosunku do $\mathrm{GF}(2^8)$, a ponadto pozwala na bardziej efektywną realizację operacji w układach cyfrowych. W szczególności, odwrotność multiplikatywna, która w ciele $\mathrm{GF}(2^8)$ jest transformacją nieliniową, w ciele $\mathrm{GF}\left(\left(2^4\right)^2\right)$ da się przedstawić za pomocą operacji liniowych. Obserwacja ta pozwala przypuszczać, że taka interpretacja transformacji SubBytes pozwoli na podwyższenie efektywności procedur predykcji bitów i bajtów parzystości oraz wyznaczania macierzy korygującej. Modyfikacja taka wydaje się być bardzo obiecująca mimo, że wymaga dodatkowych układów realizujących przejścia pomiędzy oboma ciałami. Wstępne analizy pozwalają przypuszczać, że ten dodatkowy narzut implementacyjny zostanie zrekompensowany przez zmniejszoną złożoność układu predykcji i generowania macierzy korygującej.

W przypadku schematów podpisów cyfrowych ElGamala i DSA zaproponowano całkowicie odmienne rozwiązanie zabezpieczające. Wprowadzone modyfikacje nie mają bowiem na celu zapewnienia poprawności generowanego wyniku, niezależnie od tego czy błąd został wprowadzony czy nie, lecz mają za zadanie uniemożliwić wykorzystanie błędnego wyniku do kryptoanalizy. W obu schematach właściwość taka została zrealizowana poprzez wprowadzenie mechanizmów propagacji i rozpraszania błędów. Zadaniem tych mechanizmów jest zapewnienie, że wprowadzenie błędu dowolnego typu, do procedury składania podpisu spowoduje jego propagację i taką zmianę generowanego podpisu, że będzie on nieprzydatny dla atakującego. Ogólna idea rozpraszania błędu, w przypadku obu schematów, polega na weryfikacji klucza

prywatnego, wykorzystywanego do złożenia podpisu, z kluczem, który został użyty do wygenerowania klucza publicznego. Jeśli klucz jest poprawny, to wynikiem weryfikacji jest $T = 0$, które powoduje, że kolejne operacje są wykonywane zgodnie ze schematem generując poprawny podpis. W przeciwnym razie, $T \neq 0$ a generowany wynik jest błędny i nieprzydatny atakującemu. Zaletą tak realizowanego sprawdzenia poprawności klucza prywatnego, jest brak bezpośredniego porównania jakichkolwiek danych. Zaproponowane modyfikacje są więc pozbawione procedury decyzyjnej, która jest słabym punktem dotychczasowych zabezpieczeń. Różnica w ochronie obu schematów polega jedynie na różnej metodzie rozprzestrzeniania błędu. Dla podpisów ElGamala rozproszenie to bazuje na chińskim twierdzeniu o resztach i jest przeniesieniem na grunt podpisów ElGamala podobnej idei zaproponowanej wcześniej dla algorytmu RSA [15, 56, 80, 85]. Rozwiązanie to różni się jednak od podobnych rozwiązań stosowanych w przypadku RSA. Najistotniejsza różnica polega na tym, że w algorytmie RSA chronione są czynniki p, q modułu N podczas, gdy w schemacie ElGamala czynniki w_1, w_2 mogą być ujawnione nie powodując bezpośredniego zagrożenia dla klucza prywatnego.

Ponieważ rozpraszanie błędów z wykorzystaniem chińskiego twierdzenia o resztach nie może być wykorzystanie w przypadku schematu podpisów DSA, dlatego w tym przypadku zaproponowano rozwiązanie wykorzystujące nieliniowość operacji wyznaczania odwrotności multiplikatywnej w ciele $\mathbb{Z}_q^*$. Mechanizm ochronny schematu DSA wykorzystuje w tym przypadku to, że niepoprawna wartość odwrotności multiplikatywnej zależy zarówno od nieznanej, losowej liczby k jak i wprowadzonego błędu. Ogranicza to wiedzę atakującego na temat postaci błędu przekłamującego generowany podpis i uniemożliwia prowadzenie kryptoanalizy. Zaletą rozwiązania zaproponowanego dla schematu DSA jest jego uniwersalność, która pozwala zastosować je rownież w przypadku schematu podpisów ElGamala.

Złożoność implementacyjna obu proponowanych rozwiązań jest zbliżona do złożoności oczywistego rozwiązania polegającego na sprawdzeniu poprawności podpisu poprzez jego weryfikację. Zaproponowane algorytmy nie posiadają jednak dość poważnych wad takiego mechanizmu, które stwarzają realne zagrożenie dla bezpieczeństwa [52, 84]. Trudnością w stosowaniu rozwiązania zaproponowanego dla schematu ElGamala jest konieczność generacji liczb pierwszych p, dla których $p-1$ ma dwa duże dzielniki względnie pierwsze w_1, w_2. Ponieważ liczby pierwsze stosowane w algorytmie ElGamala mają rozmiar nie mniejszy niż 768 bitów, to ich wylosowanie i weryfikacja są czasochłonne. Jak dotąd jest to jednak jedyny, skuteczny sposób dobierania liczb pierwszych. Może on być przyspieszony poprzez poszukiwanie dzielników pierwszych, mniejszych niż ustalony próg. Pozwala to znacznie przyspieszyć dobór liczby p zaniżając jednocześnie liczbę czynników i powodując, zmniejszenie minimalnego rozmiaru modułów w_1, w_2. Z tego powodu, usprawnienie procesu doboru parametrów zaproponowanego schematu może być interesującym kierunkiem dalszych badań.

Zaletą proponowanych rozwiązań zabezpieczających, w porównaniu do bezpośredniej weryfikacji podpisu, jest gwarancja, że każdy wprowadzony błąd zostanie wykryty i rozpropagowany w sposób pseudolosowy. Zagrożeniem jest jedynie sytuacja, w której atakujący będzie w stanie wyzerować wybrany rejestr układu kryptograficznego i tym samym wykasować informację o błędzie. Mając jednak na uwadze analizę metod wprowadzania błędów przedstawioną w rozdziale 3 można powiedzieć, że prawdopodobieństwo wprowadzenia takiego błędu przemijającego jest bardzo małe. Zagrożenie ze strony błędów trwałych można natomiast wyeliminować poprzez odpowiednią implementację i wykorzystanie tego samego rejestru do przechowywania wielu różnych wyników pośrednich. W ten sposób, wprowadzenie błędu trwałego spowoduje niepoprawne wartości znacznej części zmiennych wykorzystywanych w algorytmie i całkowitą zmianę generowanego wyniku.

Analiza metod ochrony przed kryptoanalizą z uszkodzeniami pozwoliła zauważyć, że oprócz zapobiegania wprowadzaniu błędów do układów kryptograficznych, nie ma innych uniwersalnych rozwiązań chroniących przed kryptoanalizą z uszkodzeniami. W praktyce oznacza to konieczność projektowania, implementacji i dostosowywania rozwiązań do konkretnych algorytmów. Widać to bardzo dobrze na przykładzie schematów podpisów ElGamala i DSA. Algorytmy te należą do jednej rodziny, bazują na tym samym trudnym problemie logarytmu dyskretnego, lecz mimo to możliwości ich zabezpieczenia są różne. Co więcej, schemat podpisów Schnorra, należący do tej samej grupy podpisów cyfrowych, nie może być zabezpieczony za pomocą żadnego z rozwiązań zaproponowanych dla ElGamala czy DSA. Ponadto wszystkie rozwiązania zabezpieczające (zarówno sprzętowe jak i algorytmiczne) mogą stać się obiektem ataku i zostać wyłączone. Nie gwarantują one tym samym pełnego bezpieczeństwa, a jedynie obniżają skuteczność ataków i zawężają możliwości atakującego. Analiza przeprowadzona w ramach pracy doktorskiej pozwala wnioskować, że spośród sprzętowych i algorytmicznych rozwiązań zabezpieczających lepsze są te drugie. Dla poprawnego działania muszą one jednak spełniać szereg wymagań, wśród których jednym z najistotniejszych jest silna integracja rozwiązania z algorytmem. Integracja taka pozwala bowiem zagwarantować to, że w przypadku wyłączenia mechanizmów ochronnych, algorytm przestaje realizować swoją funkcję i generuje całkowicie niepoprawne wyniki. Jednakże, zagwarantowanie takiego powiązania jest trudne, gdyż w znacznym stopniu zależy ono od samego algorytmu kryptograficznego, który zazwyczaj nie daje zbyt wielu możliwości wprowadzenia modyfikacji.

Jak pokazano w pracy, algorytmy kryptograficzne mogą być chronione poprzez korygowanie i rozpraszanie błędów. Przeprowadzone prace pozwalają przypuszczać, że rozwiązania tego typu mogą być efektywnie implementowane również w przypadku innych algorytmów. Jest to tym istotniejsze, że znaczna część wykorzystywanych współcześnie algorytmów kryptograficznych jest podatna na atak z uszkodzeniami, a mimo to, do dnia dzisiejszego, nie zaproponowano dla nich skutecznych metod ochrony.

Bibliografia

[1] Onur Aciiçmez, Çetin Kaya Koç, Jean-Pierre Seifert, *On the power of simple branch prediction analysis*, ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, Computer, Communications security, s.312–320, 2007

[2] Ross J. Anderson, Markus G. Kuhn, *Tamper Resistance - a Cautionary Note*, The Second USENIX Workshop on Electronic Commerce Proceedings, s.18–21, 1996

[3] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, Jean-Pierre Seifert, *Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures*, LNCS, vol.2523, s.81–95, 2003

[4] Feng Bao, Robert H. Deng, Yongfei Han, Albert B. Jeng, A. Desai Narasimhalu, Teow-Hin Ngair, *Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults*, Proceedings of the 5th International Workshop on Security Protocols, LNCS, vol.1361, s.115-124, 1997

[5] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, Claire Whelan, *The Sorcerer's Apprentice Guide to Fault Attacks*, Proccedings of the IEEE, vol.94, s.370–382, 2006

[6] Daniel J. Bernstein, *Cache-Timing Attacks on AES*, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2005

[7] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, Vincenzo Piuri, *Error analysis and detection procedures for a hardware implementation of the advanced encryption standard*, IEEE Transactions on Computers, vol.52(4), s.492–505, 2003

[8] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, Vincenzo Piuri, *Detecting and locating faults in VLSI implementations of the Advanced Encryption Standard*, Proc. of the 18th IEEE International Symposium on Defect, Fault Tolerance in VLSI Systems, s.105–113, 2003

[9] Janusz Biernat, Maciej Nikodem, *Fault Cryptanalysis of ElGamal Signature Scheme*, LNCS, vol.3643, s.327–336, Springer-Verlag, 2005

[10] Janusz Biernat, Mariusz Czapski, Maciej Nikodem, *Error Detection Procedures for Advanced Encryption Standard*, Discrete-Event System Design 2006 (DESDes'06), s.307–312, 2006

[11] Eli Biham, Adi Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, LNCS, vol.1294, s.513–525, 1997

[12] Eli Biham, Louis Granboulan, Phong Q. Nguyen, *Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4*, LNCS, vol.3537, s.359–367, 2005

[13] Daniel Bleichenbacher, *Generating ElGamal Signatures without Knowing the Secret Key*, LNCS, vol.1070, s.10–18, 1996

[14] Johannes Blömer, Jean-Pierre Seifert, *Fault based cryptanalysis of the Advanced Encryption Standard (AES)*, LNCS, vol.2742, s.162–181, 2003

[15] Johanne Blömer, Martin Otto, Jean-Pierre Seifert, *A New CRT-RSA Algorithm Secure Against Bellcore Attacks*, Proceedings of Conference on Computer, Communications Security, s. 311-320, 2003

[16] Johannes Blömer, Martin Otto, *Wagner's Attack on a Secure CRT-RSA Algorithm Reconsidered*, Workshop on Fault Diagnosis, Tolerance in Cryptography (FDTC 2006), LNCS, vol.4236, s.13–23, 2006

[17] Dan Boneh, Richard A. DeMillo, Richard J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Advances in Cryptology, EUROCRYPT'97, LNCS, vol.1233, s.37–51, 1996

[18] Dan Boneh, Richard A. DeMillo, Richard J. Lipton, *On the Importance of Eliminating Errors in Cryptographic Computations*, Journal of Cryptology: the journal of the International Association for Cryptologic Research, vol.14, no.2, s.101–119, 2001

[19] Dan Boneh, Ramarathnam Venkatesan, *Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes*, In Proceedings of Crypto'96, LNCS, vol.1109, s. 129–142, 1996

[20] Dan Boneh, Ramarathnam Venkatesan, *Rounding in Lattices and Its Cryptographic Applications*, SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical, Experimental Analysis of Discrete Algorithms), s.675–681, 1997

[21] Luca Breveglieri, Israel Koren, Paolo Maistri, M. Ravasio, *Incorporating Error Detection in an RSA Architecture*, LNCS, vol.4236, s.71–79, 2006

[22] Eric Brier, Benoît Chevallier-Mames, Mathieu Ciet, Christophe Clavier, *Why One Should Also Secure RSA Public Key Elements*, Cryptographic Hardware, Embedded Systems (CHES 2006), LNCS, vol.4249, s.321–338, 2006

[23] David Brumley, Dan Boneh, *Remote Timing Attacks are Practical*, Proceedings of the 12$^{\text{th}}$ Usenix Security Symposium, s.1–14, 2003

[24] Chris K. Caldwell, *An Amazing Prime Heuristic*, 2000 http://www.utm.edu/ caldwell/preprints/Heuristics.pdf

[25] Chien-Ning Chen, Sung-Ming Yen, *Differential Fault Analysis on AES Key Schedule and Some Countermeasures*, Proc. of the ACISP 2003, LNCS, vol.2727, s.118–129, 2003

[26] Mariusz Czapski, Maciej Nikodem, *Error Correction Procedures for Advanced Encryption Standard*, Int. Workshop on Coding and Cryptography (WCC 2007), s. 89–98, INRIA, April 16-20, 2007

[27] Mariusz Czapski, Maciej Nikodem, *Error Detection and Error Correction Procedures for the Advanced Encryption Standard*, zaakceptowany do publikacji w czasopiśmie Designs Codes and Cryptography, 2008

[28] Debaleena Das, Nur A. Touba, Markus Seuring, Michael Gössel, *Low Cost Concurrent Error Detection Based on Modulo Weight-Based Codes*, Proceedings of the 6th IEEE International On-Line Testing Workshop, IEEE Computer Society, s.171–176, 2000

[29] Whitfield Diffie, Martin Hellman, *New directions in cryptography*, IEEE Trans. on Information Theory, vol.22, s.644-654, November 1976

[30] Emmanuelle Dottax, *Fault Attacks on NESSIE Signature and Identification Schemes*, New European Schemes for Signatures, Integrity and Encryption (NESSIE), Report NES/DOC/ENS/WP5/031/1, 2002

[31] Pierre Dusart, Gilles Letourneux, Olivier Vivolo, *Differential Fault Analysis on AES*, LNCS, vol.2846, s.293–306, 2003

[32] A. J. Elbirt, W. Yip, B. Chetwynd, Christof Paar, *An FPGA Implementation, Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists*, AES Candidate Conference, s.13–27, 2000

[33] Taher ElGamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, no.31, s.469–472, 1985

[34] Santiago Fernández-Gómez, J. J. Rodríguez-Andina, E. Mandado, *Concurrent Error Detection in Block Ciphers*, ITC '00: Proceedings of the 2000 IEEE International Test Conference, IEEE Computer Society, s.979–984, 2000

[35] Christophe Giraud, *DFA on AES*, Proc. of the AES 2004, LNCS, vol.3373, s.27–41, 2005

[36] Christophe Giraud, Erik W. Knudsen, *Fault Attacks on Signature Schemes*, LNCS, vol.3108, s.478–491, 2004

[37] Christophe Giraud, *An RSA Implementation Resistant to Fault Attacks, to Simple Power Analysis*, IEEE Transaction on Computers, vol.55(9), s.1116–1120, 2006

[38] Marcin Gomułkiewicz, Mirosław Kutyłowski, Heinrich Theodor Vierhaus, Paweł Wlaź, *Synchronization Fault Cryptanalysis for Breaking A5/1*, LNCS, vol.3503, s.415–427, 2005

[39] Marcin Gomułkiewicz, Maciej Nikodem, Tadeusz Tomczak, *Low-cost and universal secure scan: a design-for-test architecture for crypto chips*, Proceedings of the International Conference on Dependability of Computer Systems, DepCoS - RELCOMEX 2006, IEEE Computer Society, s.282–288, 2006

[40] Sudhakar Govindavajhala, Andrew W. Appel, *Using Memory Errors to Attack a Virtual Machine*, IEEE Symposium on Security, Privacy, s.154–165, 2003

[41] Shay Gueron, Jean-Pierre Seifert, *Is It Wise to Publish Your Public RSA Keys*, Fault Diagnosis and Tolerance in Cryptography, LNCS, vol.4236, s.1–12, 2006

[42] Gunnar Gaubatz, Berk Sunar, Mark G. Karpovsky, *Non-linear Residue Codes for Robust Public-Key Arithmetic*, Workshop on Fault Diagnosis, Tolerance in Cryptography (FDTC 2006) ,s. 173–184, 2006

[43] Helena Handschuh, Howard M. Heys, *A Timing Attack on RC5*, Selected Areas in Cryptography 1998, LNCS, vol.1556, s.306–318, 1999

[44] Arash Hariri, Arash Reyhani-Masoleh, *Fault Detection Structures for the Montgomery Multiplication over Binary Extension Fields*, Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), s.37–46, 2007

[45] Martin Hellman, *The Mathematics of Public Key Cryptography*, Scientific American, vol.241, s.130-139, August 1979

[46] David Hely, Frederic Bancel, Marie-Lise Flottes, Bruno Rouzeyre, *Secure scan techniques: a comparison*, IOLTS '06: Proceedings of the 12th IEEE International Symposium on On-Line Testing, s.119–124, 2006

[47] David Hely, Marie-Lise Flottes, Frederic Bancel, Bruno Rouzeyre, Nicolas Berard, Michel Renovell, *Scan design and secure chip*, IOLTS '04: Proceedings of the 10th IEEE International On-Line Testing Symposium, s.219–224, 2004

[48] Adolf Hildebrand, Gérald Tenenbaum, *On the number of prime factors of an integer*, Duke Mathematical Journal, vol.56, No.3, s.471-501, 1988

[49] Jonathan J. Hoch, Adi Shamir, *Fault Analysis of Stream Ciphers*, Cryptographic Hardware, Embedded Systems - CHES 2004, LNCS, vol.3156, s.240–253, 2004

[50] Nikhil Joshi, Kaijie Wu, Ramesh Karri, *Concurrent Error Detection Schemes for Involution Ciphers*, Cryptographic Hardware and Embedded Systems - CHES 2004, LNCS, vol.3156, s.400–412, 2004

[51] Nikhil Joshi, Kaijie Wu, Jayachandran Sundararajan, Ramesh Karri, *Concurrent error detection for involutional functions with applications in fault-tolerant cryptographic hardware design*, IEEE Transactions on Computer-Aided Design of Integrated Circuits, Systems, vol.25(6), s.1163–1169, 2006

[52] Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen, Moti Yung, *Observability Analysis - Detecting When Improved Cryptosystems Fail*, Topics in Cryptology - CT-RSA 2002, LNCS, vol.2271, s.17–29, 2002

[53] Kaijie Wu, Ramesh Karri, *Idle Cycles Based Concurrent Error Detection of RC6 Encryption*, DFT '01: Proceedings of the IEEE International Symposium on Defect, Fault Tolerance in VLSI Systems (DFT'01), IEEE Computer Society, s.200–205, 2001

[54] Mark G. Karpovsky, Konrad J. Kulikowski, Alexander Taubin, *Differential Fault Analysis Attack Resistant Architectures for the Advanced Encryption Standard*, 6th International Conference on Smart Card Research, Advanced Applications (CARDIS 2004), s.177–192, 2004

[55] Ramesh Karri, Kaijie Wu, Piyush Mishra, Yongkook Kim, *Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers*, IEEETransaction on Computer-Aided Design of Integrated Circuits, Systems, vol.21(12), s.1509–1517, 2002

[56] Chang-Kyun Kim, Jae-Cheol Ha, Sang-Jae Moon, Sung-Ming Yen, Sung-Hyun Kim, *A CRT-Based RSA Countermeasure Against Physical Cryptanalysis*, Proceedings of the High Performance Computing and Communications Conference (HPCC 2005), LNCS, vol.3726, s.549-–554, Springer-Verlag, 2005

[57] Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems*, LNCS, vol.1109, s.104–113, 1996

[58] Paul Kocher, Joshua Jaffe, Benjamin Jun, *Differential Power Analysis*, LNCS, vol.1666, s.388–397, 1999

[59] Oliver Kömmerling, Markus G. Kuhn, *Design Principles for Tamper-Resistant Smartcard Processors*, USENIX Workshop on Smartcard Technology - Smartcard 99, USENIX Association, s.9–20, 1999

[60] Konrad J. Kulikowski, Mark G. Karpovsky, Alexander Taubin, *Fault Attack Resistant Cryptographic Hardware with Uniform Error Detection*, Proceedings of the FDTC 2006, LNCS, vol.4236, s.185–195, 2006

[61] Anna Labbé, Annie Pérez, *AES Implementation on FPGA: Time - Flexibility Tradeoff*, FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications, LNCS, vol.2438, s.651–670, 2002,

[62] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, www.cacr.math.uwaterloo.ca/hac, 1996

[63] P.L. Montgomery, *Modular Multiplication without Trial Division*, Mathematics of Computation, vol.44, no.177, s.519–521, 1985

[64] D. Mukhopadhyay, S. Banerjee, D. RoyChowdhury, B. B. Bhattacharya, *CryptoScan: A Secured Scan Chain Architecture*, Proceedings of the 14th Asian Test Symposium on Asian Test Symposium, IEEE Computer Society, s.348–353, 2005

[65] David Naccache, Phong Q. Nguyen, Michael Tunstall, Claire Whelan, *Experimenting with Faults, Lattices and the DSA*, LNCS, vol.3386, s.16–28, 2005

[66] Michael Neve, Eric Peeters, David Samyde, Jean-Jacques Quisquater, *Memories: a survey of their secure uses in smart cards*, IEEE Security in Storage Workshop 2003, s.62–72, 2003

[67] Phong Q. Nguyen, Igor E. Shparlinski, *The Insecurity of the Digital Signature Algorithm with Partially Known Nonces*, Journal of Cryptology, vol.15(3), s.151–176, June 2002

[68] Maciej Nikodem, *Error Prevention, Detection and Diffusion Algorithms for Cryptographic Hardware*, Proceedings of the International Conference on Dependability of Computer Systems, DepCoS - RELCOMEX 2007, IEEE Computer Society, s.127–134, 2007

[69] Eugene Normand, *Single Event Upset at Ground Level*, IEEE Transactions on Nuclear Science, vol.43, no.6, pp2742–2750, 1996

[70] V. Ocheretnij, G. Kouznetsov, M. Gossel, Ramesh Karri, *On-line error detection and BIST for the AES encryption algorithm with different S-box implementations*, 11th IEEE On-Line Testing Symposium (IOLTS 2005), s.141–146, 2005

[71] D. Peacham, B. Thomas, *A DFA Attack Against the AES Key Schedule*, SiVenture, 2006, http://www.siventure.com/pdfs/AES_KeySchedule_DFA_whitepaper.pdf

[72] Jean-Jacques Quisquater, David Samyde, *Eddy current for magnetic analysis with active sensor*, Proceedings of E-Smart 2002, September 2002.

[73] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM 21, no.2, s.120–126, 1978

[74] Luís Santos, Mário Zenha Rela, *Constraints on the Use of Boundary-Scan for Fault Injection*, LNCS, vol.2847, s.39–55, 2003

[75] Simon Singh, *The Code Book - The Science of Secrecy from Acient Egypt to Quantum Cryptography*, Anchor Books, A Division of Random House, Inc., New York, ISBN: 0-385-49532-3, 2000

[76] Sergei P. Skorobogatov, Ross J. Anderson, *Optical Fault Induction Attacks*, Procc. of CHES 2002, LNCS, vol.2523, s.2–12, 2003

[77] Elena Trichina, Lesya Korkishko, *Secure, Efficient AES Software Implementation for Smart Cards*, Information Security Applications, LNCS, vol.3325, s.425–439, 2005

[78] David Wagner, *Cryptanalysis of a provably secure CRT-RSA algorithm*, CCS '04, Proc. of the 11th ACM Conference on Computer and Communications Security, Washington DC USA, s. 92–97, ACM Press, 2004

[79] Kaijie Wu, Ramesh Karri, Grigori Kuznetsov, Michael Goessel, *Low cost concurrent error detection for the advanced encryption standard*, Proceedings of the International Test Conference 2004, s.1242–1248, 2004

[80] Yonghong Yang Abid, Z.Wei Wang, *CRT-Based Three-Prime RSA with Immunity Against Hardware Fault Attack*, Proceedings of the 4th IEEE International Workshop on system-on-Chip for Real-Time Applications (IWSOC'04), s. 73-76, 2004

[81] Bo Yang, Kaijie Wu, Ramesh Karri, *Scan based side channel attack on dedicated hardware implementations of Data Encryption Standard*, Proceedings. Test Conference 2004, s.339–344, 2004

[82] Bo Yang, Kaijie Wu, Ramesh Karri, *Scan Based Side Channel Attack on Data Encryption Standard*, Cryptology ePrint Archive, Report 2004/083, 2004

[83] Bo Yang, Kaijie Wu, Ramesh Karri, *Secure scan: A design for test Architecture for Crypto Chips*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.25(10), s.2287–2293, 2006

[84] Sung-Ming Yen, Marc Joye, *Checking before Output May Not Be Enough against Fault-Based Cryptanalysis*, IEEE Transactions on Computers, vol.49(9), s.967–970, 2000

[85] Sung–Ming Yen, Seungjoo Kim, Seongan Lim, Sangjae Moonn, *RSA Speedup with Chinese Remainder Theorem Immune Against Hardware Fault Cryptanalysis*, IEEE Transaction on Computers, vol.52(4), s.461–472, 2003

[86] Sung-Ming Yen, Dongryeol Kim, Sang-Jae Moon, *Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection*, Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2006), LNCS, vol.4236, s.53–61, 2006

[87] Chih-Hsu Yen, Bing-Fei Wu, *Simple Error Detection Methods for Hardware Implementation of Advanced Encryption Standard*, IEEE Transactions on Computers, vol.55(6), s.720–731, 2006

[88] Boeing Radiation Effects Laboratory (BREL), http://www.boeing.com/assocproducts/radiationlab/