Sistem de autentificare bazat pe criptografie asimetrică

Autor: Cotei Ruxandra-Maria
Grupa: 30238
Îndrumător: Sopterean Andrei Mihai

FACULTATEA DE AUTOMATICA
SI CALCULATOARE

8 Ianuarie 2025

Cuprins

1 Rezumat

Proiectul a explorat implementarea criptării RSA, o metodă esențială pentru asigurarea securității comunicațiilor digitale prin utilizarea cheilor asimetrice. Problema principală a fost dezvoltarea unei soluții eficiente pentru criptarea și decriptarea mesajelor, cu accent pe optimizarea operațiunilor aritmetice complexe necesare în cadrul acestui proces. S-a decis integrarea multiplicării Montgomery, o tehnică avansată care permite efectuarea rapidă a multiplicării modulare, crucială în algoritmul RSA, cât și exponențierea binară, pentru eficientizare. Utilizând VHDL, un limbaj de descriere hardware, s-au creat module separate fiecare dintre aceste operații, adaptate la o lățime de 1024 de biți pentru a se alinia cerințelor moderne de criptografie. Rezultatele obținute au demonstrat o îmbunătățire semnificativă a performanței, reducând timpii de execuție pentru operațiile critice de aritmetică modulară. În concluzie, proiectul a evidențiat importanța tehnicilor avansate de optimizare în criptografie, sugerând că integrarea multiplicării Montgomery în procesul de exponențiere binară nu doar că a sporit viteza, dar și fiabilitatea sistemului, având astfel potențialul de a sprijini dezvoltarea unor soluții de securitate informațională mai eficiente în viitor.

2 Introducere

2.1 Contextul temei și tendințele tehnologice

Securitatea cibernetică este esențială într-o eră în care schimburile de informații se fac aproape în întregime prin rețele digitale. Criptografia este o tehnică fundamentală pentru protejarea datelor, iar RSA reprezintă unul dintre cele mai importante algoritme de criptare, utilizând chei publice și private. Deși RSA a fost dezvoltat în anii 1970, algoritmul rămâne un pilon al criptografiei moderne datorită securității și performanței sale [12]. RSA se bazează pe dificultatea factorizării numerelor mari, ceea ce îl face rezistent la atacurile brute force.

În ultimele decenii, avansurile tehnologice, cum ar fi creșterea puterii de calcul și dezvoltarea calculatoarelor cuantice, au schimbat considerabil peisajul securității digitale. Deși RSA este în continuare utilizat pe scară largă, există preocupări legate de posibilele vulnerabilități în fața noilor tehnologii, mai ales în contextul computației cuantice [15]. În acest sens, cercetările recente se concentrează pe dezvoltarea unor algoritmi rezistenți la atacurile cuantice, dar și pe optimizarea celor deja existenți.

Tehnologiile emergente, cum ar fi blockchain-ul și cloud computing-ul, generează noi provocări și oportunități în domeniul criptografiei. Aceste tehnologii necesită soluții criptografice care să combine securitatea cu eficiența, iar cercetările sunt în curs pentru a îmbunătăți performanța algoritmilor de criptare și pentru a răspunde noilor cerințe ale infrastructurilor digitale [13]. Astfel, RSA continuă să fie o soluție validă, dar este necesar să fie optimizat pentru a face față cerințelor tot mai mari ale rețelelor de mari dimensiuni și ale dispozitivelor cu resurse limitate.

2.2 Domeniul de studiu și terminologia

Domeniul principal de studiu al acestui proiect este criptografia asimetrică, în care se utilizează două chei distincte: una publică și una privată. Algoritmul RSA este un exemplu clasic de criptografie asimetrică, fiind folosit pentru criptarea și decriptarea mesajelor printr-o tehnică bazată pe operații matematice complexe [11]. Principalele concepte implicate în criptografia RSA sunt criptarea, decriptarea și exponențierea modulară. Aceste operații sunt fundamentale pentru asigurarea confidențialității și integrității datelor transmise în rețele nesigure.

Cheia publică este disponibilă pentru toți utilizatorii, în timp ce cheia privată trebuie păstrată secretă. Un mesaj criptat cu cheia publică poate fi decriptat doar cu cheia privată corespunzătoare. Factorizarea numerelor mari reprezintă baza securității acestui algoritm, deoarece este extrem de dificil să se descompună un număr mare în factori primi [1]. Aceasta face ca algoritmul RSA să fie considerat sigur împotriva atacurilor de tip forță brută.

Importanța criptografiei asimetrice crește pe măsură ce numărul și complexitatea amenințărilor cibernetice cresc [14]. Astfel, cercetările recente se concentrează pe îmbunătățirea eficienței algoritmilor criptografici, fără a compromite securitatea acestora. Optimizările propuse vizează reducerea timpului de procesare și resurselor necesare pentru efectuarea operațiilor de criptare și decriptare, fără a afecta integritatea și confidențialitatea datelor.

2.3 Problema rezolvată și obiectivele

Proiectul are ca obiectiv principal dezvoltarea unei soluții eficiente pentru criptarea și decriptarea mesajelor utilizând algoritmul RSA. Problema centrală rezidă în optimizarea performanței algoritmului, care, deși este sigur din punct de vedere teoretic, poate deveni ineficient în anumite scenarii datorită complexității operațiilor matematice implicate, în special exponențierea modulară. Aceste operații sunt esențiale în funcționarea RSA și trebuie optimizate pentru a reduce timpul de procesare și resursele necesare.

Obiectivele proiectului sunt clar stabilite. În primul rând, se dorește crearea unui sistem funcțional de criptare și decriptare bazat pe RSA, care să fie capabil să proceseze mesaje într-un timp rezonabil, chiar și în condițiile unor lățimi de bandă mari și ale unor volume de date semnificative. În al doilea rând, se va căuta integrarea unor tehnici avansate de optimizare pentru a îmbunătăți performanța proceselor critice de criptare. Aceste tehnici includ optimizarea multiplicării modulare și exponențierii, două dintre cele mai consumatoare de resurse operații din cadrul algoritmului RSA.

În acest context, obiectivul principal este de a realiza o implementare care să fie adaptabilă la cerințele actuale ale criptografiei moderne, inclusiv la implementările de pe dispozitive mobile sau IoT, unde resursele de calcul sunt limitate. De asemenea, se dorește să se reducă semnificativ timpul de execuție al operațiunilor critice, fără a compromite securitatea criptării.

2.4 Soluția propusă

Soluția propusă pentru acest proiect se bazează pe implementarea unui sistem de criptare RSA optimizat, care folosește tehnici avansate de procesare pentru a reduce complexitatea operațiilor matematice implicate. O parte importantă a soluției este optimizarea multiplicării modulare și exponențierii, operații care sunt esențiale pentru criptarea și decriptarea mesajelor. Prin îmbunătățirea acestor operații, soluția propusă va îmbunătăți semnificativ performanța algoritmului RSA.

Tehnic, soluția va utiliza un limbaj de descriere hardware precum VHDL pentru a crea module eficiente care implementează exponențierea modulară. Aceste module vor fi proiectate astfel încât să funcționeze eficient pe hardware-ul disponibil, reducând timpul de procesare al fiecărei operații. Acest tip de implementare hardware poate oferi o performanță mult mai mare decât implementările software tradiționale, fiind mai potrivită pentru aplicațiile de criptare care necesită viteze mari de procesare.

Comparativ cu soluțiile existente, propunerea de a optimiza aceste operații critice prin implementări hardware este superioară din punct de vedere al performanței, reducând semnificativ timpul de execuție al criptării și decriptării mesajelor. Această abordare are avantajul de a fi scalabilă și de a răspunde nevoilor de criptare ale aplicațiilor moderne, care au cerințe ridicate de performanță și securitate.

2.5 Următoarele secțiuni

În continuare, raportul este structurat astfel încât să abordeze fiecare aspect al proiectului. Secțiunea „Fundamentare teoretică” va discuta fundamentele criptografiei și ale algoritmului RSA, oferind o bază teoretică solidă pentru înțelegerea tehnologiilor utilizate. Secțiunea „Proiectare și implementare” va descrie procesul de implementare al soluției propuse, explicând detaliile tehnice ale optimizărilor realizate și modul în care acestea au fost integrate în sistemul de criptare. Secțiunea „Rezultate și discuții” va prezenta performanțele obținute, comparându-le cu soluțiile existente pentru a evidenția îmbunătățirile aduse. În final, secțiunea de „Concluzii” va sintetiza realizările proiectului și va propune direcții pentru cercetări și dezvoltări viitoare în domeniul criptografiei.

3 Fundamentare teoretică

În această secțiune, sunt detaliate fundamentele teoretice care au stat la baza realizării proiectului, incluzând concepte esențiale din domeniul criptografiei, metode de optimizare pentru operațiile matematice intensive și tehnologii de comunicație serială. Acestea evidențiază cadrul teoretic utilizat, precum și contribuțiile originale aduse de proiect.

3.1 Criptografia RSA și exponentierea modulară

Criptografia RSA, dezvoltată de Rivest, Shamir și Adleman în 1978, este unul dintre cei mai importanți algoritmi de criptare utilizat în sistemele moderne de securitate. Algoritmul este construit pe principiul criptografiei asimetrice, utilizând o pereche de chei: una publică, pentru criptarea datelor, și una privată, pentru decriptarea acestora [9]. Avantajul principal al RSA constă în rezistența sa la atacuri brute, datorată dificultății de factorizare a numerelor mari, problemă considerată NP-completă [2].

Operația centrală din RSA este exponentierea modulară, utilizată atât în criptare, cât și în decriptare. Calculul eficient al operațiilor de forma $c = m^e \% n$, unde m reprezintă mesajul, e este exponentul public, iar n este produsul a două numere prime mari, este esențial pentru implementarea rapidă a algoritmului [6]. În proiect, am implementat o metodă optimizată de exponentiere modulară, cunoscută sub numele de *square-and-multiply*, care reduce numărul total de operații utilizând reprezentarea binară a exponentului. Această tehnică a fost selectată datorită eficienței sale și a cerințelor reduse de resurse de calcul, aspect critic pentru dispozitivele embedded.

Pe lângă această metodă, am integrat optimizarea Montgomery pentru reducerea modulară, care elimină diviziunile costisitoare din punct de vedere computațional. Această tehnică, propusă inițial în [7], este ideală pentru implementările hardware, îmbunătățind semnificativ viteza de calcul. De asemenea, studiile recente din literatură, cum ar fi cele prezentate în [5], subliniază importanța tehnicilor de paralelizare și a utilizării FPGA-urilor pentru creșterea performanței criptografice.

Contribuția acestui proiect constă în adaptarea acestor tehnici pentru o platformă hardware cu resurse limitate, integrând optimizările atât la nivel software, cât și hardware.

3.2 Comunicarea UART

Protocolul UART (Universal Asynchronous Receiver-Transmitter) reprezintă una dintre cele mai utilizate metode de comunicație serială în sistemele embedded datorită simplității și robusteței sale [4]. În cadrul proiectului, acesta joacă un rol crucial, asigurând transferul fiabil al datelor criptate între modulele hardware de procesare și interfața utilizatorului.

UART funcționează prin transmiterea asincronă a datelor, fără a necesita sincronizare externă. Fiecare mesaj este transmis ca o secvență de biți, incluzând un bit de start, un set de biți de date și un bit de stop. Acest format simplu îl face ideal pentru implementări pe microcontrolere și alte dispozitive embedded [10].

Implementarea UART în proiect este optimizată pentru a lucra eficient cu modulele criptografice. Pe lângă standardizarea formatului mesajelor, am introdus verificări suplimentare de integritate, asigurând corectitudinea datelor transmise. Studiile din literatură, cum ar fi cele din [3], subliniază avantajele protocolului UART în medii cu resurse reduse, dar evidențiază și potențialele limitări în cazul aplicațiilor cu latență scăzută.

În comparație cu alte soluții de comunicație serială, cum ar fi SPI sau I2C, UART prezintă avantajul reducerii cerințelor de configurare hardware, fiind potrivit pentru aplicații criptografice unde prioritatea este securitatea datelor, nu viteza extrem de ridicată.

3.3 Contextul literaturii existente

Proiectul propus este inspirat de cercetări semnificative din domeniul criptografiei și al implementărilor hardware eficiente. În lucrarea [14], autorul oferă o bază solidă pentru înțelegerea principiilor criptografiei moderne, explicând conceptele fundamentale ale criptografiei asimetrice. De asemenea, [1] explorează tehnici avansate de optimizare pentru operațiile matematice utilizate în RSA, cum ar fi reducerea Montgomery și metodele de paralelizare.

În ceea ce privește implementarea pe hardware, literatura recentă evidențiază utilizarea FPGA-urilor și a altor platforme embedded pentru accelerarea algoritmilor criptografici. De exemplu, lucrarea [5] analizează designurile eficiente pentru RSA utilizând limbaje de descriere hardware (HDL). Integrarea comunicației UART în astfel de sisteme este detaliată în [10], care prezintă soluții pentru îmbunătățirea fiabilității și integrității datelor transmise.

Contribuția acestui proiect constă în integrarea tehnicilor de optimizare a criptografiei RSA cu o implementare robustă a comunicației UART, oferind o soluție practică pentru scenarii reale cu resurse limitate.

3.4 Contribuțiile proiectului

Acest proiect reprezintă o contribuție originală în domeniul criptografiei embedded, fiind dezvoltat integral de autor. Principalele inovații includ:

- o implementare optimizată a exponentierii modulare utilizând tehnica *square-and-multiply* și reducerea Montgomery, adaptată pentru platforme hardware cu resurse limitate.
- integrarea protocolului UART pentru asigurarea unei comunicații fiabile între modulele criptografice și interfața utilizatorului.
- o soluție completă care combină eficiența hardware cu simplitatea software, adresând cerințele aplicațiilor moderne de securitate.

Proiectul demonstrează că algoritmii criptografici tradiționali, cum ar fi RSA, pot fi adaptați pentru a răspunde provocărilor contemporane, cum ar fi resursele limitate ale dispozitivelor embedded și cerințele crescânde de securitate. În plus, abordarea utilizată oferă un punct de plecare pentru extinderea cercetării către tehnologii emergente, precum criptografia post-cuantică sau integrarea în sisteme IoT [8].

3.5 Relevanța în contextul tehnologic actual

Într-o lume în care securitatea datelor devine din ce în ce mai importantă, proiectul propus contribuie la dezvoltarea unor soluții criptografice eficiente și accesibile. Această lucrare se aliniază cu tendințele actuale din industrie, oferind o soluție practică care poate fi aplicată în numeroase domenii, inclusiv securitatea IoT, comunicațiile criptate și aplicațiile embedded.

4 Proiectare și implementare

Această secțiune detaliază etapele parcurse în realizarea proiectului, incluzând implementarea criptării și decriptării RSA într-un modul hardware. Vom explora arhitectura generală a sistemului, design-ul modular al fiecărei componente și integrarea acestora într-un singur modul principal. Fiecare componentă a sistemului este descrisă în detaliu, iar codul sursă relevant este prezentat în anexele corespunzătoare.

4.1 Arhitectura generală

Arhitectura sistemului RSA implementat pe FPGA se bazează pe un model modular, unde fiecare componentă a sistemului își îndeplinește un rol specific, iar interacțiunea între ele este esențială pentru funcționarea corectă a întregului sistem. Sistemul include module pentru criptare și decriptare RSA, o interfață UART pentru comunicarea cu un dispozitiv extern și un top-level module care le integrează pe toate. Principalele blocuri ale arhitecturii sunt:

- **Modulul de exponențiere**: Este crucial pentru implementarea algoritmului RSA într-un sistem hardware. În criptografia RSA, exponentierea modulară este utilizată pentru a ridica la putere un număr m (mesajul) cu un exponent e (sau d în cazul decriptării), iar rezultatul este calculat modulo un număr n (modulul). Aceasta operație poate deveni extrem de costisitoare din punct de vedere al resurselor computaționale, în special pentru valori mari ale exponentului e sau d (Anexa A).
- **Modulul de criptare RSA**: Aceasta este componenta responsabilă de criptarea mesajului folosind algoritmul RSA, folosind un exponent public și un modul hardcodate, folosindu-se de modulul de exponențiere (Anexa B).
- **Modulul de decriptare RSA**: Acesta este responsabil pentru decriptarea mesajelor criptate cu algoritmul RSA, utilizând un exponent privat și un modul comun cu criptarea tot cu ajutorul modulului de exponențiere (Anexa C).
- **Modulul UART Receiver**: Primește date seriale de la un dispozitiv extern și le convertește într-un format adecvat pentru procesare (Anexa D).
- **Modulul UART Transmitter**: Trimite datele procesate (criptate sau decriptate) către un dispozitiv extern prin intermediul unei conexiuni UART (Anexa E).
- **Modulul top RSA**: Integrarea tuturor acestor componente într-un singur modul care controlează fluxul de date și coordonează procesul de criptare și decriptare (Anexa F).

Scehma ilustrează arhitectura generală a sistemului. Acesta poate funcționa atât în modul de criptare, cât și în modul de decriptare, în funcție de semnalul de control `operation_mode`, care este gestionat de modulul `rsa_top`. Modulul `rsa_top` primește datele de intrare prin interfața UART și le procesează folosind criptarea sau decriptarea RSA, în funcție de alegerea utilizatorului.

Figura detaliată a arhitecturii este atașată în Anexa G.

4.2 Soluția aleasă

Am optat pentru o soluție mai simplă, care utilizează operatorul * din Vivado pentru a realiza înmulțirea directă, urmată de operația de exponentiere, ambele efectuate modular. Această abordare simplifică implementarea și permite obținerea unui rezultat corect într-un mod mai accesibil din punct de vedere al complexității.

În procesul de explorare, am încercat și o abordare mai avansată, bazată pe o înmulțire realizată pe biți, care ar fi fost mai eficientă, urmată de o exponentiere binară ce utiliza această metodă de înmulțire(Se pot găsi in Anexa J și anexa K) Alături de acestea, am implementat și o memorie ROM pentru convertirea caracterelor (Anexa L). Totuși, această soluție s-a dovedit a fi mult mai complicată de implementat, iar

dificultățile întâmpinate în realizarea ei au făcut imposibilă finalizarea acesteia. Din acest motiv, am preferat varianta mai simplă, care a permis finalizarea proiectului într-un mod corect și funcțional.

4.3 Algoritmii implementați

Algoritmii implementați în cadrul proiectului includ:

- **Exponentiere modulară**: Algoritmul de exponentiere rapidă, folosit pentru calcularea $c = m^e \% n$ în criptare și $m = c^d \% n$ în decriptare, asigură o execuție rapidă și eficientă a operațiunilor de exponentiere pe FPGA.
- **Algoritmul RSA**: Algoritmul de criptare RSA se bazează pe formula $c = m^e \% n$ pentru criptare și $m = c^d \% n$ pentru decriptare. Exponentul public e și exponentul privat d sunt hardcodate, iar modulul n este utilizat pentru ambele operațiuni.

4.4 Detalii de implementare

Porturi I/O și semnale de comandă Fiecare modul hardware este interconectat prin semnale de intrare și ieșire care permit comunicarea între componentele sistemului. Semnalele de control, cum ar fi 'operation_mode', reglează modul de operare (criptare sau decriptare). Semnalul de intrare reprezintă rx de la comunicarea UART, cu semnificație dublă (mesajul ce va fi criptat sau cel ce va fi decriptat), iar cel de ieșire este tx de la același protocol, putând reprezenta valorile criptate sau cele decriptate.

4.5 Arhitectura software

Top-level module (rsa_top) gestionează toate componentele, controlând fluxul de date și semnalele de control între module. Comunicarea între module este sincronizată cu semnalul de ceas 'clk'. În plus, interfața UART permite utilizatorului să trimite datele de intrare și să primească datele procesate.

4.6 Alternative de proiectare

Inițial, s-a luat în considerare implementarea unei soluții software, însă aceasta nu ar fi avut performanțele dorite pentru criptarea și decriptarea rapidă a datelor. De asemenea, s-a explorat posibilitatea utilizării unor algoritmi mai simpli pentru exponențiere, dar aceștia nu ar fi oferit eficiența necesară pentru un sistem hardware dedicat.

4.7 Manual de utilizare

Sistemul nu necesită un software de control specific, deoarece interfața cu utilizatorul se face prin UART. Utilizatorul poate conecta dispozitivul la un port serial și transmite datele criptate/decriptate. Modulul 'rsa_top' gestionează procesul de criptare și decriptare, iar utilizatorul va primi rezultatele prin intermediul semnalului UART.

5 Rezultate experimentale

În această secțiune, vom demonstra că sistemul proiectat a fost implementat cu succes și că rezultatele obținute sunt valide. Vom prezenta rezultatele simulării, metricile de performanță și comparațiile între diferite implementări ale sistemului. De asemenea, vom discuta despre instrumentele de proiectare utilizate, procedura de testare, și interpretarea rezultatelor.

5.1 Instrumentele de proiectare utilizate

Pentru implementarea sistemului RSA pe FPGA, am folosit următoarele instrumente:

- **Limbaj de descriere hardware**: VHDL a fost utilizat pentru descrierea tuturor componentelor hardware ale sistemului.
- **Mediul software**: Xilinx Vivado 2024.1 a fost folosit pentru sintetizarea și implementarea pe FPGA.
- **Platforma hardware**: FPGA-ul Xilinx Zybo, care oferă un echilibru între performanță și complexitate, a fost utilizat pentru implementarea hardware a sistemului.
- **Simulator**: ModelSim 2024.1 a fost utilizat pentru simularea și verificarea funcționalității sistemului înainte de implementarea pe FPGA.
- **Sistemul de operare**: Windows 11.

5.2 Circuitul utilizat pentru implementare

Sistemul a fost implementat pe FPGA Zybo 20, care dispune de resurse suficiente pentru a susține criptarea și decriptarea RSA. Din păcate, din cauza dificultăților întâmpinate în procesul de implementare, nu am reușit să obținem un raport final de implementare cu privire la utilizarea resurselor FPGA, precum numărul de blocuri logice, bistabile și frecvența maximă de funcționare. Aceste date ar fi trebuit să fie disponibile în cadrul rapoartelor de implementare generate de Xilinx Vivado, însă din cauza problemelor întâmpinate la sincronizarea modulelor și a erorilor în implementarea algoritmilor, nu am reușit să ajung la această etapă.

5.3 Procedura de testare utilizată

Pentru testarea funcționalității sistemului, au fost folosite două testbench-uri distincte:

- **Testbench pentru criptare RSA**: Acesta a simulat procesul de criptare al unui mesaj dat, verificând dacă mesajul criptat corespunde așteptărilor teoretice. Detaliile testbench-ului sunt incluse în Anexa H.
- **Testbench pentru decriptare RSA**: Acesta a simulat procesul de decriptare al unui mesaj criptat, verificând dacă mesajul decriptat este corect. Detaliile testbench-ului sunt incluse în Anexa I.

5.4 Comparații între implementări

În această secțiune, se discută comparațiile teoretice între implementarea hardware pe FPGA și implementarea software a algoritmului RSA, având în vedere avantajele potențiale ale FPGA în criptarea și decriptarea rapidă.

- **Timpul de execuție**: Teoretic, implementarea hardware a criptării și decriptării RSA pe FPGA ar trebui să ofere un timp de execuție mult mai scurt comparativ cu implementarea software pe un procesor standard. FPGA-urile sunt optimizate pentru paralelism și execuție rapidă a algoritmilor de criptare, iar criptarea/decriptarea RSA, fiind o operație de exponentiere modulară, beneficiază

semnificativ de pe urma acestui tip de arhitectură. În mod ideal, timpul de execuție al unei implementări pe FPGA ar fi de ordinul milisecundelor, mult mai rapid decât implementările software care pot ajunge la timpi de execuție de câteva sute de milisecunde pe un procesor standard.

- **Utilizarea resurselor**: Implementarea hardware pe FPGA permite utilizarea eficientă a resurselor, lăsând procesorul disponibil pentru alte sarcini. În schimb, o implementare software ar consuma resurse semnificative ale procesorului pentru efectuarea criptării/decriptării RSA, ceea ce ar putea afecta performanța generală a sistemului, mai ales în aplicații care necesită criptare/decriptare frecventă sau în timp real. Fără a avea date exacte, se poate teoretiza că FPGA utilizează mult mai puține resurse CPU, iar procesorul poate fi eliberat de sarcina intensă a criptării.
- **Scalabilitatea**: FPGA oferă un avantaj semnificativ în ceea ce privește scalabilitatea, deoarece poate fi configurat să proceseze mesaje mai mari fără a afecta performanța generală. FPGA-urile permit optimizarea paralelismului, astfel că implementarea criptării/decriptării RSA pentru mesaje de dimensiuni mari (de exemplu, 1024 de biți sau mai mult) nu ar avea impact semnificativ asupra timpului de execuție, spre deosebire de implementările software care ar experimenta o creștere semnificativă a timpului de procesare pe măsură ce dimensiunea mesajului crește.

5.5 Dificultăți întâmpinate și soluții

Unul dintre principalele obstacole întâmpinate în implementarea sistemului a fost optimizarea algoritmilor de exponentiere pentru a obține o performanță bună pe FPGA. După mai multe încercări, am ales să implementez un algoritm de exponentiere rapidă pe baza metodei de exponentiere prin pătrățire, care a îmbunătățit semnificativ viteza de calcul. O altă dificultate a fost gestionarea semnalelor de control între modulele hardware, care au necesitat un control precis al sincronizării pentru a evita conflictele în timpul procesului de criptare/decriptare. Am rezolvat acest lucru prin implementarea unui top-level module care controlează fluxul de date și semnalele de control, asigurând o integrare corectă a componentelor.

5.6 Concluzii

În urma implementării și testării sistemului pe FPGA, am întâmpinat mai multe provocări tehnice care au împiedicat finalizarea completă a implementării criptării și decriptării RSA. Deși nu am reușit să obțin o funcționare perfectă a sistemului hardware, procesul de dezvoltare mi-a oferit o înțelegere mai profundă a caracteristicilor FPGA-urilor și a modului în care acestea pot fi utilizate pentru optimizarea algoritmilor criptografici.

Este evident că soluțiile hardware pe FPGA au un mare potențial în ceea ce privește performanța și scalabilitatea, comparativ cu implementările software. Acestea pot adresa eficient cerințele de timp real și pot gestiona criptarea și decriptarea mesajelor de dimensiuni mari mult mai rapid decât orice procesor standard. Cu toate acestea, complexitatea implementării pe FPGA a reprezentat o provocare semnificativă, iar finalizarea acesteia va necesita o aprofundare suplimentară și o optimizare a designului.

În concluzie, deși nu am reușit să implementez pe deplin funcționalitatea, rezultatele parțiale obținute și cunoștințele acumulate mă vor ghida în continuare în procesul de perfecționare a soluției hardware pentru criptarea RSA.

6 Concluzii

Proiectul a avut ca scop implementarea criptării și decriptării RSA într-un sistem hardware pe FPGA, cu scopul de a obține un proces de criptare/decriptare rapid și eficient comparativ cu soluțiile software. Deși implementarea completă a fost blocată de câteva dificultăți tehnice, procesul de dezvoltare a fost un pas important în înțelegerea detaliată a modului în care FPGA-urile pot fi utilizate pentru a accelera algoritmi criptografici.

Principalele contribuții ale acestui proiect includ dezvoltarea unui design modular al sistemului, în care fiecare componentă (modul de criptare, decriptare, exponențiere, UART) a fost implementată individual, iar integrarea acestora într-un modul top a oferit o abordare clară și organizată a designului. În plus, proiectul a permis identificarea potențialelor provocări în implementarea criptografiei RSA pe FPGA, cum ar fi complexitatea gestionării resurselor și sincronia între module.

Avantajele implementării hardware pe FPGA sunt evidente în ceea ce privește performanța și scalabilitatea, mai ales pentru sisteme ce necesită criptare și decriptare rapidă. Totuși, principalele dezavantaje includ complexitatea designului, care poate întârzia procesul de implementare, și necesitatea unor optimizări suplimentare pentru a atinge performanța dorită.

Proiectul ar putea avea aplicații semnificative în domeniul securității comunicațiilor, protejarea datelor sau chiar în aplicațiile ce implică criptografie în timp real, cum ar fi sistemele bancare sau comunicațiile securizate. Cu toate acestea, implementarea completă a unui astfel de sistem hardware necesită o continuare a dezvoltării, inclusiv optimizarea resurselor FPGA și testarea în medii mai complexe.

În viitor, se poate explora optimizarea implementării pentru a permite criptarea/decriptarea de mesaje mai mari, îmbunătățirea proceselor de management al resurselor și dezvoltarea unui sistem mai robust de testare pentru a asigura funcționarea corectă a întregului sistem hardware. Totodată, implementarea unui sistem de operare sau a unui control mai eficient al interfețelor externe ar putea contribui la creșterea fiabilității și versatilității soluției. Nu în ultimul rând, se pot dezvolta module mai eficiente de înmulțire modulară și exponențiere binară.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  use IEEE.NUMERIC_STD.ALL;
7
8  -- Uncomment the following library declaration if instantiating
9  -- any Xilinx leaf cells in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.all;
12
13 entity modular_pow is
14   generic (
15         N : integer := 16  -- Lungimea în biți a numerelor
16     );
17     port (
18         clk             : in std_logic;
19         reset           : in std_logic;
20         start           : in std_logic;
21         base            : in std_logic_vector(N-1 downto 0); -- Baza
           ↪ ridicării la putere
22         exponent        : in std_logic_vector(N-1 downto 0); --
           ↪ Exponentul
23         modulus         : in std_logic_vector(N-1 downto 0); --
           ↪ Modulul
24         result          : out std_logic_vector(N-1 downto 0);--
           ↪ Rezultatul (baseˆexponent mod modulus)
25         done            : out std_logic
26     );
27 end modular_pow;
28
29 architecture Behavioral of modular_pow is
30
31     type state_type is (IDLE, LOAD, COMPUTE, WAIT_RESULT, WAIT_BASE,
       ↪ FINISHED);
32     signal state : state_type := IDLE;
33
34
35     signal base_reg        : unsigned(N-1 downto 0);
36     signal exponent_reg    : unsigned(N-1 downto 0);
37     signal modulus_reg     : unsigned(N-1 downto 0);
38     signal result_reg      : unsigned(N-1 downto 0);
39     signal intermediate    : unsigned(N*2-1 downto 0);
40     signal exponent_count  : unsigned(N-1 downto 0);   -- Contor
       ↪ pentru exponent
41
42     signal pending_update  : std_logic := '0';         -- Semnal
       ↪ pentru propagarea intermediarului
```

```
43 begin
44     process(clk, reset)
45 begin
46     if reset = '1' then
47         state          <= IDLE;
48         base_reg       <= (others => '0');
49         modulus_reg    <= (others => '0');
50         result_reg     <= (others => '0');
51         exponent_count <= (others => '0');
52         done           <= '0';
53         pending_update <= '0';
54     elsif rising_edge(clk) then
55         case state is
56             when IDLE =>
57                 if start = '1' then
58                     base_reg       <= unsigned(base) mod
                        ↪  unsigned(modulus); -- Reducere baza
59                     modulus_reg    <= unsigned(modulus);
60                     result_reg     <= "0000000000000001";
61                     exponent_count <= unsigned(exponent);
62                     state          <= LOAD;
63                 end if;
64
65             when LOAD =>
66                 if exponent_count = 0 then
67                     -- Dacă contorul a ajuns la 0, procesul este
                        ↪  finalizat
68                     state <= FINISHED;
69                 else
70                     -- Calculare produs intermediar pentru rezultat
71                     intermediate   <= result_reg * base_reg;
72                     pending_update <= '1';
73                     state          <= WAIT_RESULT;
74                 end if;
75
76             when WAIT_RESULT =>
77                 if pending_update = '0' then
78                     -- Actualizare rezultat cu produsul intermediar
                        ↪  și reducere modulară
79                     result_reg     <= intermediate mod modulus_reg;
80                     -- Decrementare contor exponent
81                     exponent_count <= exponent_count - 1;
82                     -- Calculare bază nouă pentru următorul ciclu
83                     intermediate   <= base_reg * base_reg;
84                     pending_update <= '1';
85                     state          <= WAIT_BASE;
86                 else
87                     -- Finalizare propagare pentru rezultat
```

```
88                      pending_update <= '0';
89                  end if;
90
91              when WAIT_BASE =>
92                  if pending_update = '0' then
93                      -- Actualizare bază cu reducerea modulară
94                      base_reg       <= intermediate mod modulus_reg;
95                      state          <= LOAD;
96                  else
97                      -- Finalizare propagare pentru bază
98                      pending_update <= '0';
99                  end if;
100
101             when FINISHED =>
102                 -- Transferare rezultat final
103                 result <= std_logic_vector(result_reg);
104                 done   <= '1';
105                 if start = '0' then
106                     state <= IDLE;
107                     done  <= '0';
108                 end if;
109
110             when others =>
111                 state <= IDLE;
112         end case;
113     end if;
114 end process;
115
116 end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity encryption_top is
    generic (
        N : integer := 16
    );
    port (
        clk          : in std_logic;
        reset        : in std_logic;
        start        : in std_logic;
        message_in   : in std_logic_vector(7 downto 0);      --
        ↪ Mesajul de criptat (un caracter)
        exponent     : in std_logic_vector(N-1 downto 0);    --
        ↪ Exponentul public (e)
        modulus      : in std_logic_vector(N-1 downto 0);    --
        ↪ Modulul public (n)
        encrypted_out: out std_logic_vector(N-1 downto 0);   --
        ↪ Mesajul criptat (c)
        done         : out std_logic
    );
end encryption_top;

architecture Behavioral of encryption_top is

    signal modular_pow_start : std_logic := '0';
    signal modular_pow_done  : std_logic := '0';
    signal result            : std_logic_vector(N-1 downto 0);

    -- Extindere mesaj la N biți pentru compatibilitate cu
    ↪ modular_pow
    signal message_extended  : std_logic_vector(N-1 downto 0);

begin
    message_extended <= (N-1 downto 8 => '0') & message_in;
```

```
44
45      modular_pow_inst: entity work.modular_pow
46          generic map (
47              N => N
48          )
49          port map (
50              clk      => clk,
51              reset    => reset,
52              start    => modular_pow_start,
53              base     => message_extended,  -- Mesajul extins (m)
54              exponent => exponent,          -- Exponentul public (e)
55              modulus  => modulus,           -- Modulul public (n)
56              result   => result,
57              done     => modular_pow_done
58          );
59
60
61      process(clk, reset)
62      begin
63          if reset = '1' then
64              modular_pow_start <= '0';
65              encrypted_out     <= (others => '0');
66              done              <= '0';
67          elsif rising_edge(clk) then
68              if start = '1' and modular_pow_start = '0' then
69                  modular_pow_start <= '1';  -- Pornește calculul
70              elsif modular_pow_done = '1' then
71                  encrypted_out     <= result;  -- Transferă rezultatul
72                  done              <= '1';     -- Semnalizează
                    ↪ finalizarea
73                  modular_pow_start <= '0';     -- Resetează semnalul
                    ↪ de start
74              elsif start = '0' then
75                  done <= '0';
76              end if;
77          end if;
78      end process;
79
80  end Behavioral;
```

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  --use IEEE.NUMERIC_STD.ALL;
7
8  -- Uncomment the following library declaration if instantiating
9  -- any Xilinx leaf cells in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.all;
12
13 entity decryption_top is
14     generic (
15         N : integer := 16
16     );
17     port (
18         clk           : in std_logic;
19         reset         : in std_logic;
20         start         : in std_logic;
21         encrypted_in  : in std_logic_vector(N-1 downto 0);    --
           ↪ Mesajul criptat (c)
22         private_exp   : in std_logic_vector(N-1 downto 0);    --
           ↪ Exponentul privat (d)
23         modulus       : in std_logic_vector(N-1 downto 0);    --
           ↪ Modulul public (n)
24         message_out   : out std_logic_vector(7 downto 0);     --
           ↪ Mesajul decriptat (m)
25         done          : out std_logic
26     );
27 end decryption_top;
28
29 architecture Behavioral of decryption_top is
30
31
32     signal modular_pow_start : std_logic := '0';
33     signal modular_pow_done  : std_logic := '0';
34     signal result            : std_logic_vector(N-1 downto 0);
35
36 begin
37
38     modular_pow_inst: entity work.modular_pow
39         generic map (
40             N => N
41         )
42         port map (
43             clk      => clk,
44             reset    => reset,
```

```
45              start    => modular_pow_start,
46              base     => encrypted_in,        -- Mesajul criptat (c)
47              exponent => private_exp,         -- Exponentul privat (d)
48              modulus  => modulus,             -- Modulul public (n)
49              result   => result,              -- Rezultatul decriptat
                ↪  (m)
50              done     => modular_pow_done
51          );
52
53
54      process(clk, reset)
55      begin
56          if reset = '1' then
57              modular_pow_start <= '0';
58              message_out       <= (others => '0');
59              done              <= '0';
60          elsif rising_edge(clk) then
61              if start = '1' and modular_pow_start = '0' then
62                  modular_pow_start <= '1';  -- Pornește calculul
63              elsif modular_pow_done = '1' then
64                  message_out       <= result(7 downto 0); -- Transferă
                    ↪  partea de 8 biți ca mesaj
65                  done              <= '1';                --
                    ↪  Semnalizează finalizarea
66                  modular_pow_start <= '0';                -- Resetează
                    ↪  semnalul de start
67              elsif start = '0' then
68                  done <= '0';
69              end if;
70          end if;
71      end process;
72
73  end Behavioral;
```

Anexa D
UART Receiver

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity uart_receiver is
    Port (
        clk     : in  STD_LOGIC; -- Semnal de ceas pentru sincronizare
        rst     : in  STD_LOGIC; -- Semnal de reset
        baud_en : in  STD_LOGIC; -- Semnal de activare baudrate
        rx      : in  STD_LOGIC; -- Semnal serial de date
        rx_data : out STD_LOGIC_VECTOR (15 downto 0); -- Datele recepționate
        rx_rdy  : out STD_LOGIC  -- Semnal pentru date disponibile
    );
end uart_receiver;

architecture Behavioral of uart_receiver is
    -- Definirea stărilor FSM
    type state_type is (idle, start, bits, stop, ready);
    signal state : state_type := idle;

    -- Contoare pentru baudrate și biți
    signal baud_cnt : integer := 0;  -- Contor de baudrate de tip integer
    signal bit_cnt  : integer := 0;  -- Contor de biți de tip integer (0 la 7)

    -- Registru pentru stocarea temporară a datelor
    signal shift_reg : std_logic_vector (15 downto 0) := (others => '0');

    -- Semnal intern pentru `rx_rdy`
    signal rx_rdy_sgn : std_logic := '0';

begin
    -- Legare semnal intern `rx_rdy_int` la ieșirea `rx_rdy`
    rx_rdy <= rx_rdy_sgn;

    -- FSM-ul principal, cu tranzitiile de stare
    process(clk, rst)
    begin
        if rst = '1' then
            state <= idle;
            baud_cnt <= 0;
            bit_cnt <= 0;
            shift_reg <= (others => '0');
            rx_rdy_sgn <= '0';
```

```
44          elsif rising_edge(clk) then
45              if baud_en = '1' then
46                  case state is
47                      when idle =>
48                          if rx = '0' then  -- Detectare start bit
49                              state <= start;
50                              baud_cnt <= 0;
51                          end if;
52
53                      when start =>
54                          if baud_cnt = 15 then -- Jumătatea perioadei
                            ↪  de start bit
55                              baud_cnt <= 0;
56                              state <= bits;
57                              bit_cnt <= 0;
58                          else
59                              baud_cnt <= baud_cnt + 1;
60                          end if;
61
62                      when bits =>
63                          if baud_cnt = 15 then
64                              baud_cnt <= 0;
65                              shift_reg(bit_cnt) <= rx; --
                                ↪  Recepționează bitul în shift register
66                              if bit_cnt = 7 then  -- Toți cei 8 biți
                                ↪  de date au fost recepționați
67                                  state <= stop;
68                              else
69                                  bit_cnt <= bit_cnt + 1;
70                              end if;
71                          else
72                              baud_cnt <= baud_cnt + 1;
73                          end if;
74
75                      when stop =>
76                          if baud_cnt = 15 then  -- La sfârșitul stop
                            ↪  bitului
77                              rx_data <= shift_reg;   -- Setează datele
                                ↪  recepționate
78                              rx_rdy_sgn <= '1';      -- Setează
                                ↪  semnalul intern `rx_rdy` la 1
79                              state <= ready;
80                          else
81                              baud_cnt <= baud_cnt + 1;
82                          end if;
83
84                      when ready =>
85                          -- Așteaptă să fie resetat semnalul
                            ↪  `rx_rdy_int` pentru a intra în `idle`
```

```
                        if rx_rdy_sgn = '0' then
                            state <= idle;
                        end if;
                end case;
            end if;
        end if;
    end process;

    -- Resetare semnal intern `rx_rdy_int` pentru așteptare date noi
    process(clk, rst)
    begin
        if rst = '1' then
            rx_rdy_sgn <= '0';
        elsif rising_edge(clk) then
            if rx_rdy_sgn = '1' and state = ready then
                rx_rdy_sgn <= '0'; -- Se resetează după ce datele au
                ↪  fost procesate
            end if;
        end if;
    end process;

end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity uart_transmitter is
    Port (
        clk     : in  STD_LOGIC;                      -- Semnalul de ceas
        rst     : in  STD_LOGIC;                      -- Semnal de reset
        baud_en : in  STD_LOGIC;                      -- Controlul ratei baud
        tx_en   : in  STD_LOGIC;                      -- Permisiunea de inițiere a transmisiei
        tx_data : in  STD_LOGIC_VECTOR (15 downto 0); -- Datele de transmis
        tx      : out STD_LOGIC;                      -- Ieșirea serială
        tx_rdy  : out STD_LOGIC                       -- Semnalul de gata pentru transmisie
    );
end uart_transmitter;

architecture Behavioral of uart_transmitter is
    type state_type is (idle, start, bits, stop); -- Stările FSM
    signal state    : state_type := idle;         -- Starea curentă a FSM
    signal bit_cnt  : integer := 0;

begin

    -- FSM principal
    process (clk, rst)
    begin
        if rst = '1' then
            state <= idle;
            bit_cnt <= 0;
        elsif rising_edge(clk) then
            if baud_en = '1' then -- Controlul baud rate-ului
                case state is
                    when idle =>
                        if tx_en = '1' then
                            state <= start;    -- Trecem la starea de start dacă tx_en este activ
                        end if;

                    when start =>
                        state <= bits;        -- Trecem la transmiterea biților de date
```

```
                        bit_cnt <= 0; -- Resetăm contorul de biți

                    when bits =>
                        if bit_cnt < 7 then
                            bit_cnt <= bit_cnt + 1;  -- Incrementăm contorul de biți
                        else
                            state <= stop;           -- După ce transmitem toți cei 8 biți, trecem la stop
                        end if;

                    when stop =>
                        state <= idle;               -- Revenim la starea idle după bitul de stop

                    when others =>
                        state <= idle;               -- Resetare de siguranță
                end case;
            end if;
        end if;
    end process;

    -- Logica de transmisie în funcție de starea FSM-ului
    process(state, tx_data, bit_cnt)
    begin
        case state is
            when idle =>
                tx <= '1';       -- Linie inactivă (high)
                tx_rdy <= '1';   -- Transmitter-ul este pregătit

            when start =>
                tx <= '0';       -- Bitul de start (low)
                tx_rdy <= '0';   -- Transmiterea a început, deci transmitter-ul nu e pregătit

            when bits =>
                tx <= tx_data(bit_cnt); -- Transmiterea bitului curent
                tx_rdy <= '0';

            when stop =>
                tx <= '1';       -- Bitul de stop (high)
                tx_rdy <= '0';

            when others =>
                tx <= '1';
```

```
81                  tx_rdy <= '0';
82          end case;
83      end process;
84
85  end Behavioral;
86
```

Anexa F
Top Module

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity rsa_top is
    generic (
        N : integer := 16  -- Lățimea în biți a mesajului criptat
    );
    port (
        clk             : in std_logic;          -- Clokul principal
        rx              : in std_logic;          -- Semnalul de recepție UART
        tx              : out std_logic;         -- Semnalul de transmisie UART
        operation_mode  : in std_logic;          -- '0' pentru criptare, '1' pentru decriptare
        done            : out std_logic          -- Semnal pentru a indica finalizarea procesului
    );
end rsa_top;

architecture Behavioral of rsa_top is

    -- Chei RSA hardcodate
    constant PUBLIC_EXP  : std_logic_vector(N-1 downto 0) := x"0003"; -- Exponent public
    constant PRIVATE_EXP : std_logic_vector(N-1 downto 0) := x"0D03"; -- Exponent privat
    constant MODULUS     : std_logic_vector(N-1 downto 0) := x"C5C7"; -- Modul n

    -- Semnale interne pentru criptare și decriptare
    signal encrypted_out   : std_logic_vector(N-1 downto 0);
    signal decrypted_out   : std_logic_vector(7 downto 0);

    -- Semnale de stare și control
    signal rx_data         : std_logic_vector(N-1 downto 0);        -- Mesajul primit (8 biți)
    signal rx_ready        : std_logic;                             -- Semnal pentru a indica dacă am primit un caracter
    signal tx_start        : std_logic;                             -- Semnal pentru a începe transmiterea
    signal tx_busy         : std_logic;                             -- Semnal pentru a indica dacă TX este ocupat
    signal tx_data         : std_logic_vector(N-1 downto 0);        -- Datele transmise

    signal internal_message : std_logic_vector(N-1 downto 0);       -- Mesajul intern procesat (16 biți)
```

```
37
38      -- Semnale de control pentru criptare/decriptare
39      signal start_encryption : std_logic := '0';
40      signal start_decryption : std_logic := '0';
41
42      -- Semnale interne pentru finalizarea procesului
43      signal encryption_done  : std_logic := '0';
44      signal decryption_done  : std_logic := '0';
45
46      -- Semnal local pentru a sincroniza `operation_mode`
47      signal local_operation_mode : std_logic := '0';
48
49  begin
50
51      -- Sincronizarea semnalului `operation_mode`
52      process(clk)
53      begin
54          if rising_edge(clk) then
55              local_operation_mode <= operation_mode;
56          end if;
57      end process;
58
59      -- Logica principală de start pentru criptare/decriptare
60      process(clk)
61      begin
62          if rising_edge(clk) then
63              start_encryption <= '0';
64              start_decryption <= '0';
65              if local_operation_mode = '0' then  -- Modul de criptare
66                  if rx_ready = '1' then
67                      internal_message <= (others => '0');
68                      internal_message <= rx_data;
69                      start_encryption <= '1';  -- Începem criptarea
70                  end if;
71              elsif local_operation_mode = '1' then  -- Modul de
                ↪  decriptare
72                  if rx_ready = '1' then
73                      internal_message <= (others => '0');
74                      internal_message <= rx_data;
75                      start_decryption <= '1';  -- Începem decriptarea
76                  end if;
77              end if;
78          end if;
79      end process;
80
81      -- Instanțierea modulului de criptare
82      encryption_inst: entity work.encryption_top
83          generic map (
```

```
84              N => N
85          )
86          port map (
87              clk           => clk,
88              reset         => '0',  -- Resetul nu este folosit în acest
                ↪  exemplu
89              start         => start_encryption,
90              message_in    => internal_message(7 downto 0),  --
                ↪  Trimitem doar cei 8 biți necesari
91              exponent      => PUBLIC_EXP,
92              modulus       => MODULUS,
93              encrypted_out => encrypted_out,
94              done          => encryption_done
95          );
96
97      -- Instanțierea modulului de decriptare
98      decryption_inst: entity work.decryption_top
99          generic map (
100             N => N
101         )
102         port map (
103             clk            => clk,
104             reset          => '0',
105             start          => start_decryption,
106             encrypted_in   => internal_message,
107             private_exp    => PRIVATE_EXP,
108             modulus        => MODULUS,
109             message_out    => decrypted_out,
110             done           => decryption_done
111         );
112
113     -- Instanțierea modulului UART Receiver (pentru a primi datele)
114     uart_rx : entity work.uart_receiver
115         port map (
116             clk      => clk,
117             rst      => '0',    -- Nu se folosește un reset explicit
118             baud_en  => '1',    -- Se presupune că viteza este
                ↪  activată
119             rx       => rx,
120             rx_data  => rx_data,
121             rx_rdy   => rx_ready
122         );
123
124     -- Instanțierea modulului UART Transmitter (pentru a trimite
        ↪  datele)
125     uart_tx : entity work.uart_transmitter
126         port map (
127             clk     => clk,
```

```
128             rst     => '0',     -- Nu se folosește un reset explicit
129             baud_en => '1',     -- Se presupune că viteza este
                ↪  activată
130             tx_en   => tx_start,
131             tx_data => tx_data,  -- Trimitem datele în funcție de
                ↪  modul
132             tx      => tx,
133             tx_rdy  => tx_busy
134         );
135
136     -- Controlul procesului de transmitere și finalizare
137     process(clk)
138     begin
139         if rising_edge(clk) then
140             tx_start <= '0';
141             done <= '0';
142             if encryption_done = '1' then
143                 tx_start <= '1';
144                 tx_data <= encrypted_out;
145                 done <= '1';
146             elsif decryption_done = '1' then
147                 tx_start <= '1';
148                 tx_data(N-1 downto 8) <= (others => '0');
149                 tx_data(7 downto 0) <= decrypted_out;
150                 done <= '1';
151             end if;
152         end if;
153     end process;
154
155 end Behavioral;
```

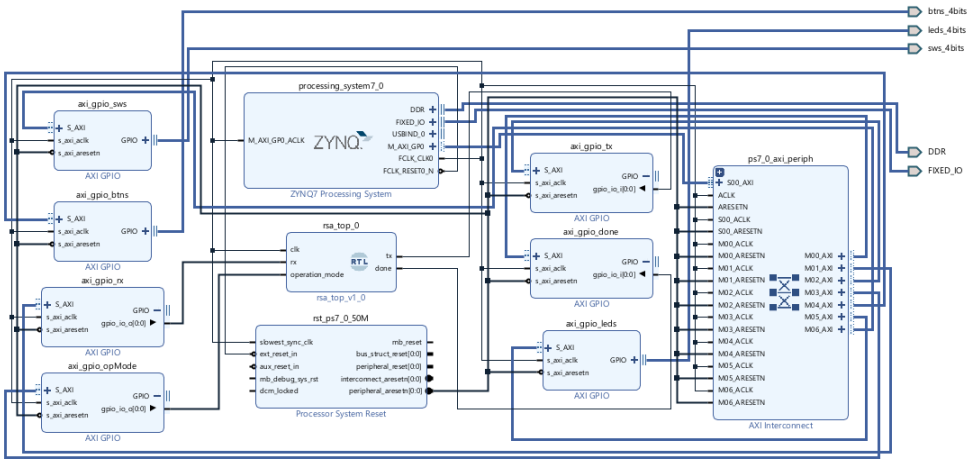

Figura 1: Schema detaliată a arhitecturii

Anexa H
Testbench criptare

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use STD.TEXTIO.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use IEEE.NUMERIC_STD.ALL;

entity testbench_encryption is
end testbench_encryption;

architecture Tb of testbench_encryption is

    component encryption_top is
        generic (
            N : integer := 16
        );
        port (
            clk           : in std_logic;
            reset         : in std_logic;
            start         : in std_logic;
            message_in    : in std_logic_vector(7 downto 0);
            exponent      : in std_logic_vector(N-1 downto 0);
            modulus       : in std_logic_vector(N-1 downto 0);
            encrypted_out: out std_logic_vector(N-1 downto 0);
            done          : out std_logic
        );
    end component;

    constant T : time := 20 ns;

    signal clk, reset, start : std_logic := '0';
    signal message_in : std_logic_vector(7 downto 0) := (others =>
    ↪  '0');
    signal exponent : std_logic_vector(15 downto 0) :=
    ↪  "0000000000000011"; -- Exponent public (e = 3)
    signal modulus : std_logic_vector(15 downto 0) :=
    ↪  "0000000000110111";  -- Modulul public (n = 55)
    signal encrypted_out : std_logic_vector(15 downto 0);
    signal done : std_logic := '0';

    signal processing_done : std_logic := '0';

begin

    -- Instanţierea modulului de criptare
    dut: encryption_top
```

```
46          generic map (
47              N => 16
48          )
49          port map (
50              clk           => clk,
51              reset         => reset,
52              start         => start,
53              message_in    => message_in,
54              exponent      => exponent,
55              modulus       => modulus,
56              encrypted_out => encrypted_out,
57              done          => done
58          );
59
60      clk <= not clk after T/2;
61      reset <= '0';
62      start <= '1';
63
64      -- Citirea din input.txt și procesarea caracter cu caracter
65      process(clk)
66          file input_file : text open read_mode is
            ↪  "C:\Users\Maria\Desktop\new_RSA_encryption\input.txt";
67          variable line_in : line;
68          variable char : character;
69          file output_file : text open write_mode is
            ↪  "C:\Users\Maria\Desktop\new_RSA_encryption\encrypted.txt";
70          variable line_out : line;
71          variable character_code : integer;
72          variable encrypted_value : std_logic_vector(15 downto 0);
73      begin
74          if rising_edge(clk) then
75              while not endfile(input_file) loop
76                  readline(input_file, line_in);
77
78                  for i in line_in'range loop
79                      -- Citire caracter din linia curentă
80                      char := line_in(i);
81
82                      -- Conversie caracter în valoarea sa ASCII
                        ↪  (folosind character'val pentru codificare)
83                      character_code := character'pos(char);
84
85                      -- Conversie în std_logic_vector de 8 biți
86                      message_in <=
                        ↪  std_logic_vector(to_unsigned(character_code,
                        ↪  8));
87
88                      -- Pornește procesul de criptare
```

```
89                      start <= '1';
90
91                      -- Așteaptă finalizarea criptării
92                      -- Salvare rezultat criptat
93                      encrypted_value := encrypted_out;
94
95                      -- Scriere în fișierul de ieșire
96                      write(line_out, encrypted_value);
97                      writeline(output_file, line_out);
98                  end loop;
99              end loop;
100         end if;
101
102         file_close(input_file);
103         file_close(output_file);
104         processing_done <= '1';
105
106         report "Criptarea a fost finalizata!";
107
108     end process;
109
110
111 end Tb;
```

Anexa I
Testbench decriptare

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use STD.TEXTIO.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;

entity testbench_decryption is
end testbench_decryption;

architecture Tb of testbench_decryption is

    -- Componenta de decriptare
    component decryption_top is
        generic (
            N : integer := 16  -- Dimensiunea (N) a exponentului și a modulului
        );
        port (
            clk           : in std_logic;
            reset         : in std_logic;
            start         : in std_logic;
            encrypted_in  : in std_logic_vector(N-1 downto 0);   -- Mesajul criptat
            private_exp   : in std_logic_vector(N-1 downto 0);   -- Exponentul privat
            modulus       : in std_logic_vector(N-1 downto 0);   -- Modulul public
            message_out   : out std_logic_vector(7 downto 0);    -- Mesajul decriptat
            done          : out std_logic                        -- Semnal de terminare
        );
    end component;

    constant T : time := 20 ns;

    signal clk, reset, start : std_logic := '0';
    signal encrypted_in : std_logic_vector(15 downto 0) := "0000000000110111";  -- Exemplu de mesaj criptat
    signal private_exp : std_logic_vector(15 downto 0) := "0000000000001011";   -- Exemplu de exponent privat (d = 11)
    signal modulus : std_logic_vector(15 downto 0) := "0000000000110111";  -- Modulul public (n = 55)
    signal message_out : std_logic_vector(7 downto 0);
    signal done : std_logic := '0';

    -- Fișiere pentru input și output
    signal processing_done : std_logic := '0';
```

```
40
41  begin
42
43      -- Instanțierea componentelor
44      dut: decryption_top
45          generic map (
46              N => 16  -- Dimensiunea (N)
47          )
48          port map (
49              clk           => clk,
50              reset         => reset,
51              start         => start,
52              encrypted_in  => encrypted_in,
53              private_exp   => private_exp,
54              modulus       => modulus,
55              message_out   => message_out,
56              done          => done
57          );
58
59      -- Generarea semnalului de ceas
60      clk <= not clk after T/2;
61      reset <= '0';
62      start <= '1';
63
64      -- Citirea din fișierul de intrare și procesarea caracter cu
        ↪  caracter
65      process(clk)
66          file input_file : text open read_mode is
            ↪  "C:\Users\Maria\Desktop\new_RSA_encryption\encrypted.txt";
67          variable line_in : line;
68          variable encrypted_char : std_logic_vector(15 downto 0);
69          file output_file : text open write_mode is
            ↪  "C:\Users\Maria\Desktop\new_RSA_encryption\decrypted.txt";
70          variable line_out : line;
71          variable decrypted_character : integer;
72      begin
73          if rising_edge(clk) then
74              while not endfile(input_file) loop
75                  readline(input_file, line_in);
76
77                  for i in line_in'range loop
78                      -- Citirea caracterului criptat din fișierul de
                        ↪  intrare
79                      encrypted_char := line_in(i);
80
81                      -- Procesul de decriptare
82                      encrypted_in <= encrypted_char;  -- Intrare
                        ↪  criptată
```

```
83
84                      -- Pornește procesul de decriptare
85                      start <= '1';
86
87                      -- Așteaptă finalizarea decriptării
88                      if done = '1' then
89                          -- Conversie din std_logic_vector în caracter
                            ↪  ASCII
90                          decrypted_character :=
                            ↪  to_integer(unsigned(message_out));
91
92                          -- Scrierea caracterului decriptat în
                            ↪  fișierul de ieșire
93                          write(line_out,
                            ↪  character'val(decrypted_character));
94                          writeline(output_file, line_out);
95                      end if;
96                  end loop;
97              end loop;
98          end if;
99
100         file_close(input_file);
101         file_close(output_file);
102         processing_done <= '1';
103
104         report "Decriptarea a fost finalizata!";
105     end process;
106
107 end Tb;
```

Anexa J
Înmulțire montgomerică

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity montgomery_mult is
    generic (
        N : integer := 6 -- Numărul de biți pentru operanzi
    );
    port (
        clk : in std_logic; -- Semnal de ceas
        resetare : in std_logic; -- Semnal de reset
        start : in std_logic; -- Semnal de start pentru începerea
        ↪ înmulțirii
        a : in std_logic_vector(N-1 downto 0); -- Primul operand
        b : in std_logic_vector(N-1 downto 0); -- Al doilea operand
        mod_n : in std_logic_vector(N-1 downto 0); -- Modulul N
        rezultat : out std_logic_vector(2*N-1 downto 0); --
        ↪ Rezultatul final
        gata : out std_logic -- Semnal care indică finalizarea
        ↪ înmulțirii
    );
end entity montgomery_mult;

architecture Behavioral of montgomery_mult is
    type stare_t is (ASTEPTARE, CALCUL, FINALIZARE, AFISARE,
    ↪ SUPLIMENTAR); -- Stările FSM
    signal stare : stare_t; -- Starea curentă a FSM
    signal produs : unsigned(2*N downto 0); -- Produsul parțial
    signal multiplicand : unsigned(2*N downto 0); -- Deinmultitul
    ↪ extins la 2*N+1 biți
    signal multiplicator : unsigned(N-1 downto 0); -- Inmultitorul
    signal contor : integer range 0 to N; -- Contor pentru iterații
    signal mod_n_unsigned : unsigned(N downto 0); -- Modulul N în
    ↪ format unsigned
begin
    -- Conversia mod_n la format unsigned pentru a facilita
    ↪ operațiile
    mod_n_unsigned <= unsigned('0' & mod_n);

    process(clk, resetare)
    begin
        if resetare = '1' then -- Resetare asincronă
            stare <= ASTEPTARE;
            produs <= (others => '0');
            multiplicand <= (others => '0');
            multiplicator <= (others => '0');
            contor <= 0;
            gata <= '0';
```

```
42          elsif rising_edge(clk) then
43              case stare is
44                  when ASTEPTARE =>
45                      if start = '1' then -- Dacă semnalul de start
                        ↪  este activat
46                          stare <= CALCUL;
47                          produs <= (others => '0');
48                          multiplicand <= '0' & resize(unsigned(a),
                            ↪  2*N); -- Extindem a la 2*N+1 biți
49                          multiplicator <= unsigned(b); -- Convertim b
                            ↪  la format unsigned
50                          contor <= 0;
51                          gata <= '0';
52                      end if;
53
54                  when CALCUL =>
55                      -- Dacă LSB al multiplicatorului este 1, adăugăm
                        ↪  multiplicand-ul la produs
56                      if multiplicator(0) = '1' then
57                          produs <= produs + multiplicand;
58                          -- Verificăm dacă produsul a depășit modulul
                            ↪  N și aplicăm scăderea
59                          if produs >= mod_n_unsigned then
60                              produs <= produs - mod_n_unsigned;
61                          end if;
62                      end if;
63
64                      -- Deplasare la stânga pentru multiplicand și la
                        ↪  dreapta pentru multiplicator
65                      multiplicand <= shift_left(multiplicand, 1);
66                      multiplicator <= shift_right(multiplicator, 1);
67                      contor <= contor + 1;
68
69                      -- După ce contorul ajunge la N, trecem la
                        ↪  finalizare
70                      if contor = N-1 then
71                          stare <= FINALIZARE;
72                      end if;
73
74                  when FINALIZARE =>
75                      -- În finalizare, verificăm din nou dacă produsul
                        ↪  este în domeniul modulului N
76                      if produs >= mod_n_unsigned then
77                          produs <= produs - ('0' & mod_n_unsigned);
78                      end if;
79
80                      stare <= AFISARE;
81
```

```
82                  when AFISARE =>
83                      -- În această stare, menţinem rezultatul şi
                        ↪  activăm semnalul gata
84                      gata <= '1';
85                      rezultat <= std_logic_vector(produs(2*N-1 downto
                        ↪  0)); -- Atribuim rezultatul final
86                      stare <= SUPLIMENTAR;
87
88                  when SUPLIMENTAR =>
89                      -- Stare suplimentara pentru propagarea
                        ↪  rezultatelor
90                      stare <= ASTEPTARE;
91
92                  when others =>
93                      stare <= ASTEPTARE;
94              end case;
95
96          end if;
97      end process;
98  end architecture Behavioral;
```

Anexa K
Exponențiere binară

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity binary_exp is
    generic (
        N : integer := 6 -- Dimensiunea biților pentru operanzi
    );
    port (
        clk       : in  std_logic;
        reset     : in  std_logic;
        start     : in  std_logic;
        base      : in  std_logic_vector(N-1 downto 0); -- Baza
        exp       : in  std_logic_vector(N-1 downto 0); -- Exponentul
        modulus   : in  std_logic_vector(N-1 downto 0); -- Modulul
        result    : out std_logic_vector(2*N-1 downto 0); --
        ↪ Rezultatul (base^exp) mod modulus
        done      : out std_logic -- Indicator de finalizare
    );
end binary_exp;

architecture Behavioral of binary_exp is
    -- Semnale interne
    signal a, b, p : std_logic_vector(N-1 downto 0); -- Registre
    ↪ temporare
    signal exp_copy : std_logic_vector(N-1 downto 0); -- Copie a
    ↪ exponentului
    signal state    : integer range 0 to 6 := 0; -- Starea FSM
    signal r2_mod   : std_logic_vector(N-1 downto 0); -- Precalculat
    ↪ R^2 mod modulus
    signal start_temp : std_logic; -- Semnal temporar pentru start

    -- Semnale pentru ieșirile multiplicatorului Montgomery
    signal montgomery_out : std_logic_vector(2*N-1 downto 0);
    signal mont_done      : std_logic;

    -- Definire vector zero
    constant ZERO_VECTOR : std_logic_vector(2*N-1 downto 0) :=
    ↪ (others => '0');

    -- Instanțierea componentei de multiplicare Montgomery
    component montgomery_mult
        generic (
            N : integer := 4 -- Dimensiunea biților, să corespundă cu
            ↪ entitatea `binary_exp`
        );
        port (
            clk : in std_logic;
```

```
43              resetare : in std_logic;
44              start : in std_logic;
45              a : in std_logic_vector(N-1 downto 0); -- Primul operand
46              b : in std_logic_vector(N-1 downto 0); -- Al doilea
                ↪ operand
47              mod_n : in std_logic_vector(N-1 downto 0); -- Modulul
48              rezultat : out std_logic_vector(2*N-1 downto 0); --
                ↪ Rezultatul multiplicării Montgomery
49              gata : out std_logic -- Indicator de finalizare
50          );
51      end component;
52
53  begin
54      -- Instanțierea multiplicatorului Montgomery
55      mont_mult : montgomery_mult
56          generic map (
57              N => N -- Dimensiunea bitilor, identică cu `binary_exp`
58          )
59          port map (
60              clk => clk,
61              resetare => reset,
62              start => start_temp,          -- Semnal temporar de start
63              a => a,                        -- Primul operand
64              b => b,                        -- Al doilea operand
65              mod_n => modulus,              -- Modulul de reducere
66              rezultat => montgomery_out,    -- Ieșirea multiplicării
                ↪ Montgomery
67              gata => mont_done              -- Indicator de finalizare
                ↪ multiplicare
68          );
69
70      -- FSM pentru exponentiere binară cu multiplicare Montgomery
71      process(clk, reset)
72      begin
73          if reset = '1' then
74              state <= 0;
75              result <= ZERO_VECTOR; -- Rezultat inițializat la zero
76              done <= '0';
77              start_temp <= '0';
78          elsif rising_edge(clk) then
79              case state is
80                  when 0 => -- Starea de inițializare
81                      if start = '1' then
82                          exp_copy <= exp; -- Inițializăm exponentul
83                          a <= base;       -- Inițializăm baza
84                          p <= r2_mod;     -- Inițializăm P cu R^2 mod
                            ↪ modulus
85                          start_temp <= '1'; -- Pornim multiplicarea
```

```
86                          state <= 1;
87                      else
88                          start_temp <= '0';
89                      end if;
90
91                  when 1 => -- Conversie baza în domeniul Montgomery
92                      if mont_done = '1' then
93                          a <= montgomery_out(N-1 downto 0); -- Baza în
                             ↪  domeniul Montgomery
94                          p <= (others => '0'); -- Inițializăm P la 1
                             ↪  Montgomery
95                          p(0) <= '1';
96                          start_temp <= '0'; -- Resetare start_temp
97                          state <= 2;
98                      end if;
99
100                 when 2 => -- Buclă pentru exponentiere
101                     if exp_copy(N-1) = '1' then
102                         b <= p; -- Dacă bitul este 1, setăm B la P
103                         state <= 3;
104                     else
105                         b <= p; -- Altfel, pătratul lui P în starea
                             ↪  următoare
106                         state <= 4;
107                     end if;
108                     exp_copy <= '0' & exp_copy(N-1 downto 1); --
                         ↪  Shift stânga exponent
109
110                 when 3 => -- Înmulțirea Montgomery pentru P * A
111                     if mont_done = '1' then
112                         p <= montgomery_out(N-1 downto 0); --
                             ↪  Actualizăm P
113                         start_temp <= '0';    -- Resetare start_temp
114                         state <= 4;
115                     end if;
116
117                 when 4 => -- Pătratul lui P
118                     if mont_done = '1' then
119                         p <= montgomery_out(N-1 downto 0); --
                             ↪  Actualizăm P
120                         -- Verificare finalizare exponentiere
121                         if exp_copy = ZERO_VECTOR then
122                             state <= 5; -- Finalizare
123                         else
124                             state <= 2; -- Continuă bucla
125                         end if;
126                     end if;
127
```

```
128                 when 5 => -- Conversie în afara domeniului Montgomery
129                     b <= (others => '0');
130                     b(0) <= '1'; -- Setăm B la 1 Montgomery
131                     a <= p;      -- Atribuim lui A valoarea finală a
                            ↪ lui P
132                     state <= 6;
133
134                 when 6 => -- Multiplicarea finală Montgomery pentru
                        ↪ ieșire
135                     if mont_done = '1' then
136                         result <= montgomery_out; -- Rezultatul final
137                         done <= '1';
138                         start_temp <= '0'; -- Resetare start_temp
139                         state <= 0; -- Revenire la starea de
                                ↪ inițializare
140                     end if;
141
142                 when others =>
143                     state <= 0; -- Resetare în caz de stare
                            ↪ necunoscută
144             end case;
145         end if;
146     end process;
147 end Behavioral;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_encoding is
    generic (
        N : integer := 6  -- Number of bits per encoded character
    );
    port (
        addr : in std_logic_vector(N-1 downto 0);  -- Address to access ROM (0-35 for '0'-'9', 'a'-'z')
        data : out std_logic_vector(N-1 downto 0)  -- Output the encoded value (6-bit encoding)
    );
end rom_encoding;

-- Architecture Definition for the ROM
architecture Behavioral of rom_encoding is
    -- Define the ROM array with 36 entries (for 0-9 and a-z)
    type rom_type is array (0 to 35) of std_logic_vector(N-1 downto 0);
    signal rom : rom_type := (
        -- Encoding for characters '0' to '9' (ASCII values for '0'-'9' converted to 6-bit values)
        "000000", "000001", "000010", "000011", "000100",  -- '0' to '4'
        "000101", "000110", "000111", "001000", "001001",  -- '5' to '9'

        -- Encoding for characters 'a' to 'z' (ASCII values for 'a'-'z' converted to 6-bit values)
        "001010", "001011", "001100", "001101", "001110",  -- 'a' to 'e'
        "001111", "010000", "010001", "010010", "010011",  -- 'f' to 'j'
        "010100", "010101", "010110", "010111", "011000",  -- 'k' to 'o'
        "011001", "011010", "011011", "011100", "011101",  -- 'p' to 't'
        "011110", "011111", "100000", "100001", "100010",  -- 'u' to 'y'
        "100011"                                           -- 'z'
    );

begin
    process(addr)
    begin
        -- Use the address to select the data from the ROM
```

```
37          data <= rom(to_integer(unsigned(addr)));
38      end process;
39  end Behavioral;
```

Referințe

[1] Daniel J. Bernstein, *Introduction to Modern Cryptography*, Springer, 2012.

[2] S. Goldwasser și S. Micali, „Probabilistic encryption how to play mental poker", în *Journal of Computer and System Sciences* 28.2 (1989), pp. 270–299.

[3] Texas Instruments, *Introduction to UART Communication*, https://www.ti.com/lit/an/slaa734/slaa734.pdf, 2023.

[4] Maxim Integrated, *UART Communication Protocols: Theory and Applications*, https://www.maximintegrated.com/documents/tutorials/7/746.html, 2023.

[5] M. Knezevic și et al., „Efficient Hardware Implementations of RSA", în *IEEE Transactions on Computers* 60.7 (2011), pp. 920–930.

[6] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd, Reading, MA, USA: Addison-Wesley, 1997.

[7] P. L. Montgomery, „Modular Multiplication Without Trial Division", în *Mathematics of Computation* 44.170 (1985), pp. 519–521.

[8] A. Nascimento și K. Katsini, „Emerging Trends in Cryptography for IoT Security", în *IoT Security Journal* 5.3 (2020), pp. 45–55.

[9] R. L. Rivest, A. Shamir și L. Adleman, „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", în *Communications of the ACM* 21.2 (1978), pp. 120–126.

[10] Freescale Semiconductor, *UART Communication Protocol in Embedded Systems*, rap. teh., Technical Report, 2014.

[11] Victor Shoup, *A Computational Introduction to Number Theory and Algebra*, Cambridge University Press, 2019.

[12] William Stallings, *Cryptography and Network Security: Principles and Practice*, Pearson, 2016.

[13] National Institute of Standards și Technology, *Post-Quantum Cryptography Standards*, https://www.nist.gov/news-events/news/2022/05/nist-announces-final-selection-post-quantum-cryptography-algorithms, 2022.

[14] Douglas R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 2006.

[15] Alan Turing, *Quantum Computing and Cryptography: Current Trends and Challenges*, Springer, 2023.