Protocolul de autentificare zero-knowledge Feige-Fiat-Shamir (in Zn)

1st Șerban Tiurbe
Automatică și Informatică Aplicată
Universitatea Politehnica Timișoara
Timișoara, România
serban.tiurbe@student.upt.ro

***Abstract*—Acest proiect investighează implementarea și evaluarea performanței protocolului de autentificare zero-cunoștințe Feige-Fiat-Shamir (FFS), concentrându-se pe eficiența și securitatea acestuia în comparație cu algoritmi criptografici tradiționali, precum RSA și DSA. Folosind o simulare bazată pe Java, am implementat meticulos și testat protocolul FFS în diverse configurații computaționale, măsurând timpul de generare a cheilor, timpul total de execuție al protocolului și timpul mediu pe rundă de autentificare pentru multiple iterații (10, 100 și 1000). Similar, am implementat RSA și DSA pentru a compara performanța lor în fazele de generare a cheilor, semnare și verificare. Rezultatele dezvăluie că, în timp ce RSA și DSA sunt robuste pentru comunicații securizate și semnături digitale, FFS oferă o alternativă computațional eficientă pentru scenarii de autentificare, evidențiată în special de timpul său redus mediu pe rundă și timpul minim de configurare a cheilor. Aceste rezultate subliniază potențialul FFS ca protocol criptografic viabil în medii unde ciclurile rapide de autentificare sunt esențiale. Studiul nu doar demonstrează aplicabilitatea practică a FFS în sistemele criptografice moderne, dar oferă și o perspectivă comparativă prin care să evaluăm compromisurile între diferite tehnici criptografice în termeni de viteză și suprasarcină computațională.**

***Cuvinte Cheie*—Protocolul Feige-Fiat-Shamir, Autentificare zero-knowledge, Criptografie, RSA, DSA**

I. Introduction

A. Context general

Securitatea informațională a devenit un pilon central în arhitectura sistemelor digitale moderne, motiv pentru care algoritmii criptografici joacă un rol crucial în protecția datelor și a comunicațiilor. Printre acești algoritmi, RSA (Rivest-Shamir-Adleman) și DSA (Digital Signature Algorithm) sunt două dintre cele mai utilizate metode pentru asigurarea integrității și autenticității datelor. RSA, un algoritm de criptare cu cheie publică, este larg răspândit pentru securizarea schimburilor de informații, permițând atât criptarea cât și decriptarea datelor. DSA, pe de altă parte, este folosit predominant pentru crearea de semnături digitale, garantând că mesajele nu au fost modificate de la momentul semnării lor.

B. Protocolul Feige-Fiat-Shamir

Protocolul Feige-Fiat-Shamir, propus inițial de Uri Feige, Amos Fiat și Adi Shamir în 1986, reprezintă o metodologie inovatoare în domeniul criptografiei, fiind clasificat ca un protocol de autentificare zero-cunoștințe. Acest protocol permite unui utilizator (prover) să dovedească altui utilizator (verificator) că deține o anumită cunoaștere secretă, fără a fi necesar să divulge acea cunoaștere în sine. În cadrul acestui proces, securitatea nu se bazează pe păstrarea secretului unui mesaj, ci pe capacitatea de a genera o serie de angajamente și răspunsuri criptografice care să demonstreze posesia cunoștinței fără a o expune [1].

Un exemplu esențial al aplicării acestui protocol este în sistemele de autentificare, unde confidențialitatea și integritatea informațiilor sunt critice. În plus, datorită naturii sale non-interactice și eficienței de calcul, protocolul este deosebit de adecvat pentru medii unde resursele de procesare sunt limitate și unde este necesară minimizarea schimbului de date.

Aceste proprietăți ilustrează de ce Feige-Fiat-Shamir este considerat unul dintre cele mai eficiente și sigure protocoale de autentificare în contextul actual al necesităților crescute de securitate cibernetică. Prin compararea acestuia cu algoritmi stabiliți precum RSA și DSA, proiectul urmărește să evidențieze adaptabilitatea și superioritatea potențială a FFS în anumite scenarii de utilizare criptografică.

C. Scopul și Obiectivele Proiectului

Această lucrare explorează atât fundamentele teoretice cât și aplicabilitatea practică a acestor algoritmi. Prin compararea directă a performanțelor și eficienței FFS cu cele ale RSA și DSA, dorim să oferim o perspectivă clară asupra potențialului fiecărui algoritm în cadrul aplicațiilor criptografice contemporane, evaluând capacitatea fiecărui algoritm de a răspunde nevoilor actuale de securitate cibernetică.

II. Fundamentele Criptografice ale Protocolului Feige-Fiat-Shamir

A. Conceptul de Zero-Knowledge

Protocolul Feige-Fiat-Shamir este un exemplu de dovadă zero-knowledge. Acesta permite unui prover (Peggy) să demonstreze unui verificator (Victor) că deține o informație secretă fără a dezvălui acea informație în sine. Acest principiu este fundamental în criptografia modernă, oferind un nivel înalt de securitate și confidențialitate [1].

B. Implementarea Matematică

Protocolul utilizează aritmetică modulară pentru a gestiona acest proces de verificare. În setup-ul său, Peggy și Victor

convenesc asupra unui număr compus n care este produsul a două numere prime mari. Peggy deține un set de numere secrete care, când sunt ridicate la pătrat și reduse modulo n, oferă valorile pe care Victor le poate verifica, dar fără să poată deduce valorile secrete ale lui Peggy [1].

C. Procedura de Verificare

Procesul implică mai mulți pași unde Peggy alege un număr întreg aleator și calculează un angajament care este trimis lui Victor. Victor apoi alege o serie de valori pe care le trimite înapoi lui Peggy, care calculează o valoare de răspuns bazată pe aceste alegeri. Victor verifică apoi dacă răspunsul lui Peggy corespunde angajamentului inițial, fără a învăța despre numerele secrete ale lui Peggy [1].

D. Securitatea Protocolului

Securitatea Feige-Fiat-Shamir se bazează pe dificultatea calculării rădăcinilor pătrate modulo un număr compus necunoscut. Orice eavesdropper sau terță parte nu poate obține informații utile despre numerele secrete ale lui Peggy, chiar dacă interceptează comunicarea dintre Peggy și Victor [1].

III. Arhitectura și Designul Sistemului de Autentificare Bazat pe Feige-Fiat-Shamir

Arhitectura unui sistem criptografic este vitală pentru funcționarea și securitatea acestuia. În acest capitol, vom explora structura simplă dar eficientă a sistemului nostru care implementează protocolul Feige-Fiat-Shamir, un protocol recunoscut pentru metoda sa de autentificare fără a dezvălui informații secrete. Vom vedea cum diferitele componente ale sistemului colaborează pentru a asigura o autentificare sigură și rapidă, punând bazele pentru o înțelegere clară a operațiunilor criptografice ce stau la fundația securității digitale.

A. Descrierea Componentelor

1) Clasa FeigeFiatShamirProtocol:

- Coordonare: Clasa FeigeFiatShamirProtocol coordonează pasii necesari pentru rularea protocolului, implicând inițializarea părților terțe de încredere (TrustedThirdParty), a proverului (Prover) și a verificatorului (Verifier).
- Inițierea Protocolului: În metoda main, aceasta inițiază protocolul cu parametri specifici, cum ar fi lungimea bitului și numărul de iterații, stabilind cadru de lucru pentru testarea și validarea autentificării.
- Gestionarea Iterațiilor: Clasa gestionează un număr de iterații în care proverul și verificatorul își execută rolurile, permițând evaluarea repetitivă și consistentă a funcționalității protocolului.
- Generarea Valorilor Aleatoare: Implementează funcții pentru generarea de valori aleatorii necesare în cadrul protocolului, cum ar fi generateRandomValue pentru crearea unui număr aleator r și generateRandomChallenge pentru crearea provocărilor randomizate trimise de verificator către prover.
- Verificarea Rezultatelor: Evaluați și înregistrați rezultatele fiecărei runde de autentificare, determinând dacă procesul de verificare a fost convingător sau nu.

2) Clasa KeyGenerator: Rolul principal al clasei KeyGenerator este de a asigura generarea securizată și eficientă a cheilor criptografice necesare în cadrul protocolului Feige-Fiat-Shamir. Cheile secrete sunt păstrate sigure și folosite de către Prover pentru a genera angajamente și răspunsuri în cadrul procesului de autentificare, în timp ce cheile publice sunt distribuite Verifier-ului pentru a valida aceste răspunsuri. Astfel, clasa KeyGenerator stă la baza integrității și securității întregului proces de autentificare.

- Inițializare: Constructorul clasei KeyGenerator primește doi parametri, bitLength și k. bitLength indică lungimea bitilor pentru numerele prime utilizate în calculul cheii, în timp ce k este un parametru de securitate ce indică numărul de chei secrete și publice care vor fi generate.
- Generarea Cheilor: Metoda generateKeys() este responsabilă pentru generarea cheilor necesare. Procesul include:
  - Generarea Numerelor Prime: Folosește clasa PrimeGenerator pentru a genera două numere prime, p și q, folosind lungimea specificată de bitLength.
  - Calculul lui n: Calculează n ca fiind produsul p și q, care servește ca modul pentru restul calculelor.
  - Generarea Cheilor Secrete și Publice: Generează un array de chei secrete s și un array corespondent de chei publice v. Fiecare cheie secretă din s este generată aleatoriu și apoi este folosită pentru a calcula corespondentul său în v, v[i], care este

$$s[i]^2 \mod n \tag{1}$$

- Getteri: Clasa include metode getter (getSecretKeys() și getPublicKeys()) care permit accesul la cheile secrete și publice generate, facilitând utilizarea acestora în alte părți ale sistemului de autentificare.

3) Clasa PrimeGenerator: Rolul în sistem:

- Asigurarea Securității: Generarea sigură a numerelor prime este critică pentru securitatea oricărui sistem criptografic care se bazează pe matematica numerelor prime, cum ar fi RSA sau DSA. Numerele prime trebuie să fie alese astfel încât să fie dificil de factorizat, ceea ce este esențial pentru menținerea confidențialității și integrității datelor criptate.
- Utilizare în Diverse Algoritmi: Deși în contextul acestui proiect este utilizată pentru generarea componentelor cheilor în protocolul Feige-Fiat-Shamir, clasa PrimeGenerator poate fi, de asemenea, utilizată în orice alt context criptografic care necesită numere prime, demonstrând versatilitatea și importanța sa.

Descriere și Funcționalități:

- Constructorul Clasei: Clasa PrimeGenerator folosește un constructor anotat cu @RequiredArgsConstructor de la Lombok, care automatizează crearea unui constructor ce acceptă un singur argument, bitLength. Acesta specifică lungimea în biți a numerelor prime dorite, care este folosită ulterior în metoda de generare a numerelor prime.
- Generarea Numerelor Prime:

- Metoda generatePrime() este inima acestei clase, responsabilă pentru crearea numerelor prime.
- Folosește SecureRandom, o clasă din biblioteca standard Java care furnizează un generator de numere aleatorii criptografic sigur, pentru a asigura aleatorietatea și siguranța numerelor prime generate.
- Apelul BigInteger.probablePrime(bitLength / 2, rand) generează un număr prim probabil cu o lungime specificată, garantând că numărul este suficient de robust pentru utilizare în scopuri criptografice. Divizarea lungimii bitului prin 2 sugerează că această metodă este posibil să fie utilizată pentru a genera componente ale unui număr compus, cum ar fi în cazul generării cheilor RSA, unde

$$n = p \cdot q \tag{2}$$

și p și q sunt numere prime.

4) Clasa Prover: Rolul clasei Prover este de a asigura că partea care deține cunoașterea secretă (proverul) poate să genereze dovezi convingătoare ale acestei cunoașteri fără a o dezvălui. Aceasta este esențială pentru securitatea protocolului Feige-Fiat-Shamir, permițând autentificări sigure și eficiente în medii unde confidențialitatea informației este de importanță capitală.

Descriere și Funcționalități:

- Constructor și Atribute: Clasa este anotată cu @Getter, @Setter, și @NoArgsConstructor de la Lombok, ceea ce înseamnă că are constructori automat generați pentru inițializare simplă și metode getter și setter pentru toate atributele. Atributele BigInteger n și BigInteger[] s reprezintă modulul și cheile secrete utilizate în calculele criptografice.
- Generarea Angajamentului:
  - Metoda generateCommitment(BigInteger r) calculează și returnează un angajament bazat pe numărul aleator r. Aceasta folosește exponențierea modulară (r.modPow(BigInteger.TWO, n)) pentru a genera angajamentul, ceea ce este o parte critică a protocolului de verificare zero-cunoștințe.
- Generarea Răspunsului:
  - Metoda generateResponse(BigInteger r, byte[] c, boolean simulateError) este folosită pentru a calcula răspunsul la provocările primite de la verificator. Răspunsul, y, este calculat prin înmulțirea valorii angajamentului r cu fiecare cheie secretă corespunzătoare valorilor '1' din vectorul de provocări c. Acest proces reflectă aplicarea cunoștințelor secrete într-o formă care poate fi verificată fără a dezvălui cheile însele.
  - Opțional, metoda poate simula un răspuns eronat prin adăugarea valorii 1 la răspuns (y = y.add(BigInteger.ONE)), ceea ce permite testarea comportamentului sistemului în condiții de răspuns incorect.

5) Clasa Verifier: Rolul clasei Verifier este de a asigura integritatea și corectitudinea procesului de autentificare, evaluând acuratețea răspunsurilor primite din partea proverului. Aceasta funcționează ca un mecanism de control care confirmă că proverul deține într-adevăr cunoștințele secrete presupuse fără a le expune direct. Verificarea corectă și eficientă este crucială pentru securitatea întregului proces de autentificare, asigurând că sistemul este rezistent la încercările de falsificare sau de eludare a autentificării.

Descriere și Funcționalități:

- Atribute: Clasa este echipată cu atribute pentru stocarea modulului n și a vectorului de chei publice v, utilizate în procesul de verificare.
- Metoda de Verificare:
  - verify(final BigInteger x, final BigInteger y, final byte[] c): Aceasta este metoda centrală a clasei, unde x este angajamentul primit de la prover, y este răspunsul la provocările c trimise de verificator.
  - Verificarea se face prin compararea a două valori calculate.
  - Valoarea left: Se calculează ca

$$y^2 \mod n \tag{3}$$

, reprezentând răspunsul proverului ridicat la pătrat modulo n.
  - Valoarea right: Inițial este setată la x, și este modificată iterativ prin înmulțirea cu fiecare cheie publică corespunzătoare elementelor din c unde valoarea este 1, totul modulo n.
  - Rezultatul metodei este un boolean care indică dacă cele două valori calculate, left și right, sunt egale, validând astfel răspunsul proverului.

6) Clasa TrustedThirdParty: Clasa TrustedThirdParty este esențială pentru menținerea integrității și securității procesului de autentificare. Prin furnizarea unui mecanism sigur de distribuție a cheilor, aceasta ajută la prevenirea atacurilor care ar putea compromite protocolul prin expunerea cheilor secrete sau prin manipularea neautorizată a cheilor publice. De asemenea, rolul său în asigurarea coerenței și uniformității datelor criptografice între Prover și Verifier este crucial pentru succesul autentificărilor reciproc verificabile.

Descriere și Funcționalități:

- Generarea și Distribuția Cheilor:
  - Constructorul clasei TrustedThirdParty inițiază generarea cheilor criptografice necesare pentru protocol. Acesta instanțiază clasa KeyGenerator cu parametri specifici (bitLength și k), care determină complexitatea și securitatea cheilor generate.
  - Metoda generateKeys() din KeyGenerator este apelată pentru a produce cheile secrete și publice, precum și modulul n, care este utilizat în calculele criptografice.
- Interacțiunea cu Prover și Verifier:
  - Odată ce cheile sunt generate, metoda sendKeys() este responsabilă pentru distribuirea acestora către Prover și Verifier. Acest lucru asigură că ambele părți au acces la parametrii necesari pentru a participa în cadrul protocolului de autentificare.
  - Prover primește setul de chei secrete (s) și modulul (n), necesare pentru a genera răspunsuri valide la provocările venite din partea Verifier.

- Verifier primește setul de chei publice (v) și modulul (n), folosite pentru a verifica corectitudinea răspunsurilor primite de la Prover.

B. Fluxul de Date

1) Inițierea Protocolului:

- Clasa FeigeFiatShamirProtocol inițiază procesul și setează parametrii necesari, cum ar fi lungimea bitului și numărul de iterații.

2) Generarea Cheilor:

- TrustedThirdParty inițializează KeyGenerator.
- KeyGenerator utilizează PrimeGenerator pentru a crea numere prime, apoi calculează cheile secrete (s) și cheile publice (v) folosind aceste numere.
- TrustedThirdParty distribuie cheile publice și secrete către Verifier și respectiv Prover.

3) Procesul de Autentificare: În fiecare iterație:

- Prover generează un angajament folosind o valoare aleatoare r și cheile secrete.
- Verifier generează o provocare aleatoare (c).
- Prover calculează un răspuns bazat pe provocarea c și cheile secrete.
- Verifier verifică dacă răspunsul corespunde angajamentului și datelor din provocare.

4) Verificarea Finală:

- FeigeFiatShamirProtocol înregistrează și afișează rezultatul fiecărei iterații, determinând dacă autentificarea a fost convingătoare.

C. Diagrama UML

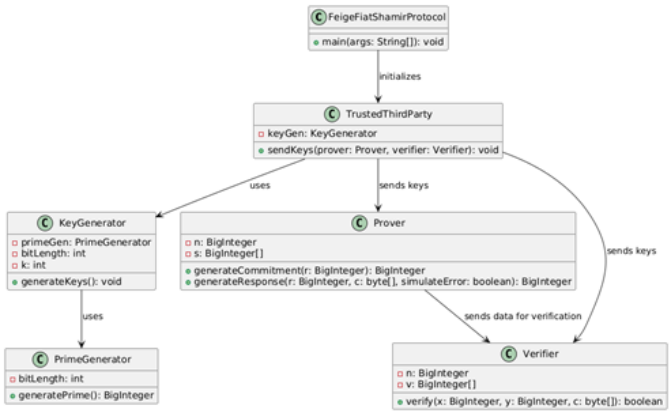

Fig. 1. Diagrama UML a sistemului.

IV. Experimente

A. Configurația și Metodologia

Am desfășurat o serie de teste pentru a evalua performanța protocolului Feige-Fiat-Shamir, comparativ cu algoritmi tradiționali precum RSA și DSA. Experimentele au fost efectuate pe un sistem cu specificațiile următoare: [detaliază hardware-ul, sistemul de operare, versiunea Java etc.]. Am testat protocolul pentru seturi de date variind în funcție de numărul de iterații și lungimea cheilor.

B. Rezultatele Experimentelor

Rezultatele obținute sunt prezentate în tabelul următor:

Număr de Iterații	Timp de generare a cheilor (ns)	Timp total de execuție (ns)	Timp mediu per rundă (ns)
10	83710600	93400700	166080
100	97861400	119591600	86555
1000	97547800	182654200	53354

TABLE I
REZULTATELE MĂSURĂTORILOR PENTRU PROTOCOLUL FEIGE-FIAT-SHAMIR

C. Compararea cu Alți Algoritmi

Pentru a oferi o comparație relevantă, am măsurat și performanța algoritmilor RSA și DSA sub aceleași condiții. Rezultatele sunt sumarizate în tabelele de mai jos:

Număr de Iterații	Timp de generare a cheilor (ns)	Timp de semnare (ns)	Timp mediu de verificare (ns)
10	496184400	3277100	307050
100	242638200	3983600	151024
1000	230735400	3967800	81944

TABLE II
REZULTATELE MĂSURĂTORILOR PENTRU RSA

Număr de Iterații	Timp de generare a cheilor (ns)	Timp mediu de semnare (ns)	Timp mediu de verificare (ns)
10	86015200	3863070	3119660
100	63256800	1299235	995800
1000	68530500	575856	853858

TABLE III
REZULTATELE MĂSURĂTORILOR PENTRU DSA

D. Analiza Datelor

Rezultatele indică faptul că protocolul Feige-Fiat-Shamir are un timp mediu pe rundă semnificativ mai scăzut comparativ cu celelalte două algoritme, demonstrând eficiența sa în cicluri rapide de autentificare. Deși RSA și DSA oferă o securitate robustă, timpul lor de răspuns este mai mare, ceea ce poate fi un dezavantaj în aplicații care necesită rapiditate.

V. Concluzii

A. Sumarul Descoperirilor

Studiul realizat a evidențiat mai multe puncte forte ale protocolului Feige-Fiat-Shamir, incluzând eficiența sa comparativă în termeni de timp de execuție și consum redus de resurse în timpul autentificărilor. Acesta oferă un nivel înalt de securitate fără a necesita schimbul de informații secrete între părți, ceea ce îl face ideal pentru aplicații unde confidențialitatea datelor este critică. Cu toate acestea, complexitatea implementării și necesitatea unei înțelegeri aprofundate a principiilor matematice implicite pot limita adoptarea sa rapidă în anumite medii industriale.

B. Implicații Practice

Implementarea protocolului Feige-Fiat-Shamir în sisteme reale oferă beneficii semnificative în ceea ce privește securitatea datelor. Este deosebit de adecvat pentru medii unde autentificarea rapidă și sigură este esențială, cum ar fi în sistemele de plăți electronice, comunicații securizate între dispozitive IoT și în sistemele de control acces. Pentru a maximiza beneficiile acestui protocol, este esențial ca dezvoltatorii să colaboreze strâns cu experții în criptografie pentru a asigura o implementare corectă și eficientă.

C. Direcții Viitoare de Cercetare

Având în vedere rezultatele promițătoare obținute, cercetările viitoare ar putea explora integrarea acestui protocol cu alte forme de tehnologii de securitate, cum ar fi blockchain-ul și tehnologiile de autentificare biometrică. De asemenea, ar fi benefică dezvoltarea unor algoritmi optimizați pentru a reduce și mai mult timpul de răspuns al protocolului, făcându-l practicabil pentru aplicații cu cerințe de timp real. Un alt aspect important ar fi evaluarea robusteții protocolului în fața unor atacuri criptografice noi și emergente, asigurându-se că rămâne un instrument viabil în peisajul securității informatice în continuă evoluție.

References

[1] Wikipedia contributors, "Feige–Fiat–Shamir Identification Scheme," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Feige%E2%80%93Fiat%E2%80%93Shamir_identification_scheme. [Accessed: Jan. 7, 2025].