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

Paul Horvath Serban
Stundent masterand SISC

Tehnici de cripatre moderne (TCM)
Timis, Timisoara
paul.horvath@upt.ro

***Rezumat*—Demonstrațiile de tip zero-knowledge sunt protocoale criptografice care permit unei părți (Prover) să convingă o altă parte (Verifier) de validitatea unei afirmații fără a dezvălui informația secretă aferentă. Aceste demonstrații asigură atât confidențialitatea, cât și verificabilitatea, fiind esențiale pentru mecanismele de autentificare securizată.**

Protocolul de autentificare Feige–Fiat–Shamir (FFS), introdus în 1988 de Uriel Feige, Amos Fiat și Adi Shamir, reprezintă un exemplu de sistem de demonstrație zero-knowledge paralel. Acesta permite Prover-ului să demonstreze că deține informații secrete către Verifier fără a dezvălui secretul propriu-zis. Schema FFS utilizează aritmetică modulară și un proces de verificare paralel, minimizând astfel comunicarea dintre părți și menținând integritatea demonstrației.

Cuvinte cheie: FFS, zero-knowledge, Prover și Verifier.

I. Introducere

A. Contextul Teoretic al Protocoalelor Zero-Knowledge

În criptografie, o demonstrație zero-knowledge este un protocol în care o parte (Prover – denumit și Peggy în exemple triviale) poate convinge o altă parte (Verifier – denumit și Victor în exemple triviale) că o anumită afirmație dată este adevărată, fără a transmite verificatorului nicio informație dincolo de simplul fapt al adevărului acelei afirmații. Cu alte cuvinte, P convinge V de faptul că acesta deține o informație secretă, validă și de interes pentru V [1].

Un exemplu clasic utilizat pentru a ilustra conceptele de demonstrație zero-knowledge este analogia cu peștera lui Ali Baba: o peșteră circulară fictivă, în formă de inel, cu o intrare principală și două tuneluri (A și B) care converg la o ușă magică în partea opusă. Această ușă poate fi deschisă doar dacă cineva cunoaște cuvântul secret. În acest scenariu, o persoană (**P – Prover**) dorește să demonstreze unei alte persoane (**V - Verifier**) că știe cuvântul secret pentru a deschide ușa, fără a dezvălui cuvântul în sine. Procesul funcționează astfel:

Pasul 1: V rămâne la intrarea în peșteră, iar P intră prin unul dintre cele două tuneluri (aleator, fie A, fie B).

Pasul 2: V alege un tunel și îi cere lui P să iasă afară prin acel tunel.

Pasul 3: Dacă P într-adevăr știe cuvântul secret, poate deschide ușa și ieși prin tunelul cerut, indiferent de alegerea lui V. Dacă nu știe cuvântul, are doar o șansă de 50% să iasă prin tunelul corect, la o singură rundă de verificare.

Dacă să repetă acest experiment de mai multe ori, probabilitatea ca P să nu spună de fapt adevărul, scade exponențial, după formula:

$$p(n) = \frac{1}{2^n}$$

unde n este numărul maxim de runde sau iterări. După un număr suficient de iterări, V poate fi convins că P știe cuvântul secret fără ca acesta să fie dezvăluit [2].

B. Scopul și Obiectivele Proiectului

Scopul acestui proiect îl reprezintă implementarea practică a protocolului de autentificare Feige–Fiat–Shamir (FFS), un protocol de demonstrație zero-knowledge, pentru a ilustra cum poate fi utilizat în sisteme de autentificare securizată. Proiectul este gândit în așa fel încât să poată fi ușor integrat în sisteme software complexe care necesită autentificarea unei entități.

II. Teoria protocolului Feige–Fiat–Shamir (FFS)

A. Descrierea Generală

Algoritmul FFS funcționează prin utilizarea aritmeticii modulare (Z_n), un element matematic care asigură complexitatea calculului și rezistența la atacuri. În cadrul acestei scheme, autentificarea este realizată printr-o serie de pași iterativi între cele două entități principale:

Prover (P) – cel care deține informația secretă și dorește demonstrarea existenței acesteia.

Verifier (V) – cel care evaluează validitatea demonstrației de existență a secretului, fără a dobândi acces la secretul propriu-zis.

Procesul de autentificare implică următoarele faze principale:

1. **Inițializare și înregistrare** - se stabilesc parametrii fundamentali ai algoritmului, precum cheia publică (n), cheia publică derivata (v) și secretul (s – generat aleator în cazul proiectului prezent):

$$n = p \cdot q$$

unde p și q sunt două numere prime

$$v \equiv s^2 \bmod n$$

2. **Faza de angajament** - P generează o valoare intermediară (x) utilizând o valoare aleatoare (r) și o

trimite lui V. Acest pas are rolul de reducere al riscului ca secretul (s) să fie expus în timpul procesului.

$$x \equiv r^2 \ mod \ n$$

3. **Faza de Provocare -** V generează o valoare aleatoare (e), care poate lua valori între 0 si 1, și o trimite lui P. La nivel de analiza, în cazul în care (e) este ales 0, runda curentă nu trebuie calculată folosind secretul s. Dacă (e) este ales ca fiind 1, secretul s trebuie inclus în calcul iar orice încercare de convingere frauduloasă din partea P se va detecta în această rundă. Această etapă introduce variabilitate în procesul de verificare, prevenind astfel orice posibilă simulare frauduloasă din partea unui adversar ilegitim.

$$e \in \{0,1\}$$

4. **Faza de Răspuns -** P calculează un răspuns (y) bazat pe r, e și s. Acest pas conectează angajamentul (r) cu secretul s și asigură că răspunsul este valid doar dacă P cunoaște cu adevărat secretul.

$$y \equiv r \cdot s^e \ mod \ n$$

5. **Verificarea -** V compară răspunsul (y) cu valoarea derivată din angajament și cheia publică. Rolul acestui pas este important, deoarece determină dacă demonstrația este validă. Dacă relația matematică stabilită este satisfăcută, V este convins că P cunoaște secretul fără a-l dezvălui, iar această rundă de verificare are un rezultat pozitiv.

$$y^2 \ mod \ n = x \cdot v^e \ mod \ n$$

Protocolul FFS utilizează iterații repetate ale acestor pași descriși anterior pentru a crește probabilitatea că demonstrația este validă.

B. *Analiza procesului*

Protocolul FFS utilizează aritmetica modulară deoarece asigură o bază matematică care conferă rezistență la factorizare, ce contribuie la aspectul confidențial al lui s. De asemenea, aritmetica modulară conferă eficiență la nivel computațional, aspect important deoarece valorile sunt mari, chiar și la nivel software. Astfel, operațiile care implică astfel de numere sunt realizate rapid și nu blochează procesul sistemului.

Un aspect suplimentar de siguranță este aplicat asupra algoritmului prin rezistența la atacuri pasive: Atacatorii care observă valorile x, e, y modificate, nu pot deduce secretul s datorita operațiilor modulare si valorii aleatoare r.

Atacurile active sunt și ele neutralizate, deoarece introducerea valorilor aleatoare în algoritm face imposibilă repetarea aceleiași demonstrații valide în două sau mai multe runde diferite.

Procesul iterativ de verificare pentru n runde este esențială, deoarece așa cum s-a menționat anterior, scade exponențial șansa unui atacator de a manipula V în a crede ca acesta deține un secret s valid.

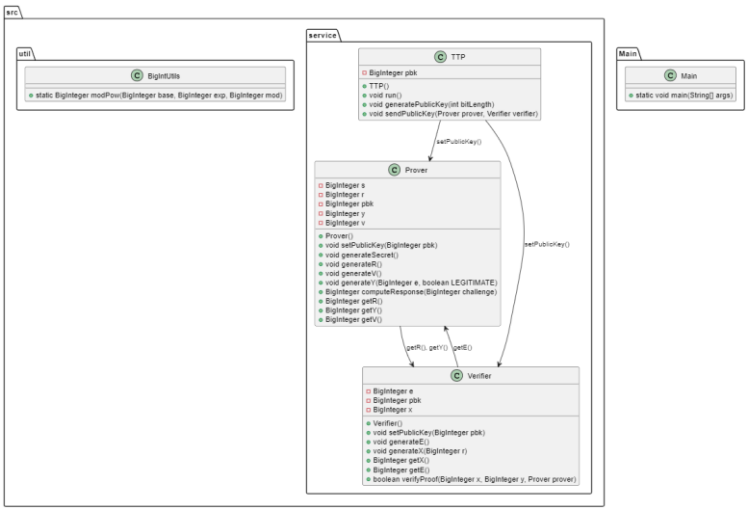
III. Arhitectura Sistemului

A. *Componente principale*

Arhitectura implementării protocolului Feige–Fiat–Shamir este construită pe baza a trei componente principale: Terța Parte de Încredere (Trusted Third Party - TTP), Proverul (P) și Verifierul (V). Aceste componente colaborează pentru a realiza procesul de autentificare în conformitate cu algoritmul descris. Fiecare componentă are un rol specific și bine definit în schema de autentificare zero-knowledge.

TTP – este responsabil pentru inițializarea procesului prin generarea și distribuirea cheii publice (n) către P și V. Prin utilizarea TTP în cadrul algoritmului, se elimină riscul ca P sau V să manipuleze într-un fel inițializarea cheii.
Prover (P) - Proverul este entitatea care deține secretul s și care trebuie să demonstreze lui V că îl cunoaște, fără a-l divulga. Proverul efectuează generarea angajamentului, calcularea răspunsului în funcție de e primit de la V.
Verifier (V) - Verifierul este entitatea care validează demonstrația oferită de Prover. Acest participant are rolul de a genera provocarea e și validarea răspunsului prin verificarea egalității de la pasul 5 al algoritmului.

B. *Diagrama de Arhitectura*



IV. Implementarea Tehnică

A. *Structura codului*

Implementarea schemei Feige–Fiat–Shamir este organizată într-o arhitectură modulară, ceea ce permite o separare clară a responsabilităților și facilitează extinderea și întreținerea codului. Proiectul este structurat în mai multe pachete și clase, fiecare având un rol specific în implementarea protocolului:
src.service: Conține componentele principale ale protocolului (TTP, Prover, Verifier), care implementează logica procesului Feige–Fiat–Shamir.
src.util: Include utilitare necesare pentru calcule matematice, cum ar fi operațiile pe numere mari (BigInteger).

Main: Conține punctul de intrare al aplicației, unde sunt implementate relațiile dintre actori.

Fluxul general al codului urmează procesul descris în algoritmul FFS:

- TTP generează cheia publică (n) și o distribuie către Prover și Verifier.
- P calculează x și îl trimite lui V.
- V generează provocarea e.
- P calculează y pe baza e, r și s.
- V validează răspunsul y pentru fiecare rundă

B. Detalii ale Implementării

Implementarea schemei Feige–Fiat–Shamir în Java este realizată prin utilizarea unor clase dedicate pentru fiecare componentă a procesului. Codul este construit pe baza aritmeticii modulare, folosind biblioteca standard *java.math.BigInteger*, care oferă suport pentru manipularea numerelor mari.

- **Clasa *TTP:***
  - → Metoda *generatePublicKey(int bitLength):* generează cheia publică *n* folosind metoda *probabilePrime()* al clasei *BigInteger* pentru a asigura securitatea acesteia. Dimensiunea cheii publice n este configurabilă (ex.: 512 biți), oferind flexibilitate pentru cerințe de securitate diferite.
  - → Metoda *sendPublicKey(Prover prover, Verifier verifier)* transmite cheia publică calculată anterior către P și V, stabilind o bază comună de încredere.

- **Clasa *Prover:***
  - → Metoda *generateR()* generează un număr aleatoriu r utilizat în calcularea angajementului. Se utilizează clasa *SecureRandom* pentru asigura aspectul aleatoriu.
  - → Metoda *generateV()* calculează cheia publica v, utilizată de Verifier în procesul de validare.
  - → Metoda *generateY(BigInteger e, boolean LEGITIMATE)* calculează valaorea y și leagă angajamentul r de secretul și provocarea e. În cazul în care parametrul *LEGITIMATE* este setat pe *false,* această metoda simulează un P fraudulos, care folosește un secret prestabilit și invalid, care încearcă să manipuleze convingerea lui V.

- **Clasa *Verifier*:**
  - → Metoda *generateE()* creează o provocare aleatoare e care poate lua ori valoarea 0 ori 1. Acesta poate fi considerat ca un flag care indică dacă Verifier solicită includerea secretului s în calculul lui y sau nu.
  - → Metoda *generateX()* calculează valoarea angajamentului x.
  - → Metoda *verifyProof(BigInteger x, BigInteger y, Prover prover)* verifică egalitatea din pasul 5 al algoritmului FFS, și emite verdictul prin care V poate fi convins sau nu de autenticitatea secretului.

- **Clasa *Main:***
  - → Metoda *main(String[] args)* coordonează întregul proces, incluzând multiple iterări ale protocolului pentru a reduce probabilitatea de fraudă.

C. Testare

Proiectul a fost testat atât cu un P legitim, cât și cu unul ilegitim. În cazul în care numărul rundelor este n=10, probabilitatea de succes fraudulos este: 1/2^10 =0.1%. În cazul n=1000, probabilitatea este practic neglijabilă. Rata de succes rezultată în urma rulării programului este ilustrată în tabelul de mai jos.

P	Rata de succes
Legitim	========== ======STATS====== ========== Success rate: 1.0 Iterations: 100 Rounds per iteration: 1000 Max round reached while illegitimate: 1000 Chance of this happening: 1.0 Happened on iteration: 999/1000
Ilegitim	========== ======STATS====== ========== Success rate: 0.0 Iterations: 100 Rounds per iteration: 1000 Max round reached while illegitimate: 6 Chance of this happening: 0.006 Happened on iteration: 5/1000

Este important de menționat faptul că sistemul este gândit să oprească încercarea de comunicare dintre V și P atunci cand V primește o valoarea la care nu se așteaptă. De asemenea, câmpul *max round reached while illegitimate* reprezintă numărul maxim de runde consecutive cu rezultat pozitiv, în timp ce P era ilegitim. Codul care se ocupă cu afișarea statisticilor se află pe branchul *test* pe GitHub.

V. Concluzii

Această lucrare a prezentat o implementare practică a schemei Feige–Fiat–Shamir (FFS), care reprezintă baza protocoalelor zero-knowledge bazate pe aritmetica modulară. Comparativ cu alți algoritmi, precum zk-SNARKs, FFS este mai ușor de implementat și este potrivit pentru aplicațiile criptografice de baza.

Bibliografie

[1] J.-J. Quisquater, L. C. Guillou and T. A. Berson, "How to Explain Zero-Knowledge Protocols to Your Children", Advances in Cryptology — CRYPTO' 89 Proceedings, vol. 435, 1990, p. 628–631.

[2] I. (. M. V. Aad, A. Mermoud, V. Lenders and B. (. Tellenbach, "Zero-Knowledge Proof", Trends in Data Protection and Encryption Technologies, Springer Nature Switzerland, p. 25–30.