Practical Escrow-free Identity-based Mutual Authentication and Key Management without Pairings

Project PN-II-PT-PCCA-2013-4-1651

Phase III: Solution Consolidation

December 5, 2016

Partea I

Descriere ştiinţifică şi tehnică

Capitolul 1

Protocoale Criptografice Utilizate

1.1 Protocolul Diffie-Hellman

Protocolul Diffie-Hellman [6] este unul dintre cele mai cunoscute protocoale pentru stabilirea cheii. Acest protocol poate fi implementat în orice grup ciclic, dar cele mai multe implementări folosesc grupul multiplicativ $\mathbf{Z}_p^*$, unde p este un număr prim mare. În acest caz, există un element $\alpha \in \mathbf{Z}_p^*$ astfel încât $\mathbf{Z}_p^* = \{\alpha^0 \ mod \ p, \alpha^1 \ mod \ p, \alpha^2 \ mod \ p, \ldots, \alpha^{p-2} \ mod \ p\}$. Elementul α este numit *generator* al grupului respectiv, sau *rădăcină primitivă* modulo p. Pentru detalii despre modul de generare a rădăcinilor primitive modulo un număr prim, cititorul poate consulta [9].

Protocolul Diffie-Hellman derulat între doi utilizatori, A şi B este descris în continuare:

- (Setup general) Se generează p, un număr prim mare şi α, o rădăcină primitivă modulo p
- A generează aleator $x \in \mathbf{Z}_{p-1}$ şi trimite $\tilde{x} = \alpha^x \ mod \ p$ lui B.
- B generează aleator $y \in \mathbf{Z}_{p-1}$ şi trimite $\tilde{y} = \alpha^y \ mod \ p$ lui A.
- A şi B pot calcula fiecare cheia $k = \alpha^{xy} \ mod \ p$, utilizatorul A calculând $\tilde{y}^x \ mod \ p$ iar utilizatorul B calculând $\tilde{x}^y \ mod \ p$.

1.2 Protocolul RPM folosind schema $Cocks$

Protocolul Diffie-Hellman, descris in Secţiunea 1.1 poate fi folosit pentru crearea unei chei iniţiale între doi utilizatori, cheie ce poate fi utilizată pentru autentificare şi asigurarea confidenţialităţii în comunicarea ulterioară. Din motive de securitate, cheia respectivă trebuie reînnoită periodic.

Cum s-ar putea realiza acest lucru? O soluţie naivă ar fi să se ruleze protocolul Diffie-Hellman de fiecare dată când se doreşte acest lucru, dar această abordare este ineficientă atât din punct de vedere computaţional cât şi din punct de vedere al dimensiunii mesajelor transmise. Ca o alternativă eficientă şi flexibilă, am propus în [7] o combinare a protocolului RPM [19] cu schema $Cocks$ [4, 23]. Ideea de bază este de a folosi anumite fragmente din mesajele din cadrul RPM pentru a transmite informaţii necesare schemei $Cocks$.

Pentru o mai bună înţelegere a ideei menţionate mai sus, vom prezenta mai întâi protocolul *PDAF network configuration* din cadrul RPM, urmând ca apoi să evidenţiem elementele noi adăugate în urma combinării cu schema Cocks.

Protocolul *PDAF network configuration* din cadrul RPM presupune că cei doi utilizatori împărtăsesc o cheie iniţială, EK (aceasta poate fi generată, de exemplu, prin protocolul Diffie-Hellman). Protocolul este descris în Figura 1.1 (pentru mai multe detalii privind acest protocol, cititorul poate consulta [7]).

A (EK)	B (EK)
1. generează aleator ID_1, ID_2, S	
2. calculează $K = f(ID_2, S)$	
3. calculează $ORID_1 = ID_1 \oplus_r EK$	
4. calculează $ORID_2 = ID_2 \oplus_r EK$	
5. calculează $ORS = ID_1 \oplus_r S$	
$\xrightarrow{\{m\}_K, ORID_1, ORID_2, ORS}$	
	7. recuperează ID_1 din $ORID_1$
	8. recuperează S din ORS
	9. recuperează ID_2 din $ORID_2$
	10. calculează $K = f(ID_2, S)$
	11. recuperează m din $\{m\}_K$

Figura 1.1: Protocolul *PDAF network configuration*, în care A îi trimite lui B un mesaj m

Protocolul rezultat în urma combinării protocolului precedent cu schema Cocks este descris în Figura 1.2 (pentru mai multe detalii privind acest protocol, cititorul poate consulta de asemenea [7]). Se presupune că cei doi utilizatori împărtăsesc o cheie iniţială, EK (aceasta poate fi generată, de exemplu, prin protocolul Diffie-Hellman) iar această cheie poate fi ulterior modificată prin folosirea biţilor transmişi suplimentar în timpul rulării noului protocol).

Protocolul descris în Figura 1.2 satisface următoarele proprietăţi:

1. Pentru a actualiza cheia (prin utilizarea biţilor transmişi b în mod adiţional), nu este nevoie de nici un mesaj suplimentar şi deci, astfel, de nici un pas de comunicare suplimentar (fiecare nou bit suplimentar este "înfăşurat" în pachetul obişnuit de date);

2. Un atacator nu are nici un indiciu despre faptul că părţile implicate în comunicare au de gând să-şi actualizeze cheia (prin utilizarea biţilor transmişi b în mod adiţional) deoarece mesajele transmise în cadrul noului protocol sunt identice cu cele din protocolul iniţial;

3. Părţile implicate în comunicare îşi pot actualiza cheia partajată ori de câte ori doresc, fără a perturba comunicarea obişnuită;

n, modul public şi h, funcţie hash	
A (EK)	B $(EK, ID(B), a = h(ID(B)), u)$
1.1. calculează cheia publică a lui B, $a = h(ID(B))$	
1.2. generează aleator S	
1.3. generează aleator t_1 a.î. $(t_1\|n) = (-1)^b$	
1.4. generează aleator t_2 a.î. $(t_2\|n) = (-1)^b$	
1.5. calculează $ID_1 = (t_1 + at_1^{-1})\ mod\ n$	
1.6. calculează $ID_2 = (t_2 - at_2^{-1})\ mod\ n$	
2. calculează $K = f(ID_2, S)$	
3. calculează $ORID_1 = ID_1 \oplus_r EK$	
4. calculează $ORID_2 = ID_2 \oplus_r EK$	
5. calculează $ORS = ID_1 \oplus_r S$	
$\xrightarrow{\{m\}_K, ORID_1, ORID_2, ORS}$	
	8. recuperează ID_1 din $ORID_1$
	9. recuperează S din ORS
	10. recuperează ID_2 din $ORID_2$
	11. calculează $K = f(ID_2, S)$
	12. recuperează m din $\{m\}_K$
	13. recuperează b din ID_1 şi ID_2

Figura 1.2: Protocolul *PDAF network configuration* combinat cu schema $Cocks$, în care A îi trimite lui B un mesaj m şi un bit suplimentar b

4. Paşii de procesare necesari în Paşii 1.3-1.6 şi 13 din noul protocol (cel din Figura 1.2) nu afectează semnificativ viteza de comunicare, deoarece:

   - Paşii menţionaţi sunt executaţi numai atunci când părţile implicate în comunicare au de gând să-şi actualizeze cheia; altfel, aceşti paşi pot fi săriţi, obţinându-se protocolul iniţial (cel din Figura 1.1);
   - Toate calculele suplimentare menţionate cumulate, nu depăşesc timpul necesar unei singure exponenţieri modulare, ceea ce implică că această metodă este, din punct de vedere computaţional, mult mai eficientă comparativ cu (re)invocarea protocolului Diffie-Hellman.

Securitatea noi scheme este garantată de securitatea schemei $Cocks$, ţinând cont şi de faptul că randomitatea elementelor ID_1 şi ID_2 este menţinută.

Capitolul 2

Generatori de Numere Aleatoare

2.1 Noțiuni introductive

Numerele aleatoare sunt utilizate în domeniul securității informațiilor, reprezentând un pilon de bază al criptografiei. Aceste numere aleatoare sunt necesare în diverse protocoale criptografice, precum protocolul Diffie-Hellman (a se vedea Secţiunea 1.1), protocolul *RPM* (a se vedea Secţiunea 1.2) sau protocoalele de autentificare a entităţii din clasa *provocare-răspuns* (challenge-response).

O secvență de biți aleatori poate fi interpretată ca rezultatul aruncării cu o monedă pe a cărei fețe sunt înscrise numerele "1" și "0", fiecare față având probabilitatea de apariție de 0.5. [14]

Una dintre proprietățile principale ale generatorilor de numere aleatoare este proprietatea de *"independență"* între biții generați, astfel încât un bit generat să nu influențeze în niciun fel generările ulterioare.

O altă proprietate de baza a generatorilor de numere aleatoare este *impredictibilitatea* care se referă la faptul că oricâte generări anterioare am cunoaște (folosind același generator), nu putem în niciun fel să presupunem valoarea următorului bit generat.

Generatorii de numere aleatoare, din punct de vedere constructiv, se împart în următoarele două categorii:

1. **Generatori de Numere Pseudo-Aleatoare - TRNG** - bazați pe funcții matematice; au o caracteristică deterministă (elementul i este strict dependent de elementele anterioare *i-1*);
2. **Generatori de Numere Pur Aleatoare - PRNG** - bazat pe un proces fizic nedeterminist (zgomotul termic, radiații cosmice, jitterul unui oscilator etc.). Totodată, acest generator are totodată și o caracteristica impredictibilă.

Schema de bază a unui Generator de Numere Pur Aleatoare este descrisă în Fig. 2.1 și este compusă din următoarele elemente:

1. **Generatorul de zgomot** - pilonul de bază al unui generator de numere aleatoare, având ca obiectiv oferirea de secvențe aleatoare bazate pe un proces fizic nedeterminist;
2. **Decorelatorul (Extractorul)** - funcție matematică care are rolul de a "modela" imperfecțiunile generatorului de zgomot;
3. **Testerul de aleatorism** - baterie de teste de aleatorism menite să determine dacă secvențele sunt cu adevărat aleatoare din punct de vedere statistic și că nu există corelări între ele.

Prin introducerea elementelor componente aferente blocurilor 2 (Decorelatorul) și 3 (Testerul de aleatorism), Generatorii de Numere Pur Aleatoare obțin viteze de generare mai mici decât Generatorii de Numere Pseudoaleatoare.

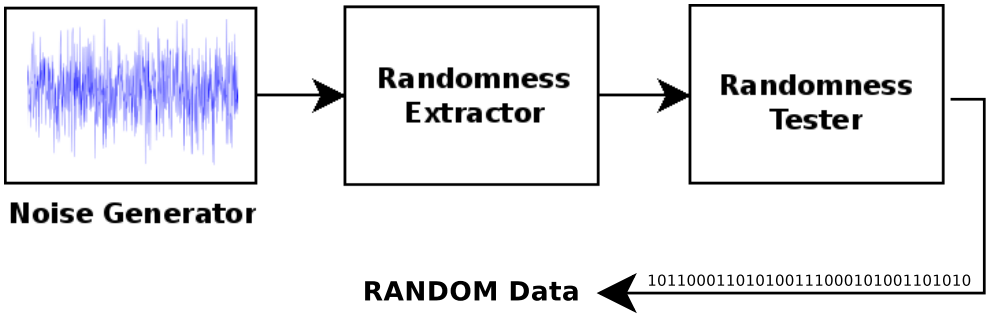


Figura 2.1: Schema de bază a unui TRNG.

2.1.1 Decorelatori

Un Generator de Numere Pur Aleatoare trebuie să aibă proprietatea că probabilitatea apariției bitului "1" să fie egală cu probabilitatea apariției bitului "0". În realitate, acest fenomen nu are loc iar, de regulă, generatorii de numere aleatoare tind să încline spre un anumit rezultat cu o probabilitate ce tinde spre 0.5 dar fiind diferită de această valoare.

Decorelatorii (sau Extractoarele) sunt funcții matematice menite să uniformizeze distribuția de biți a rezultatului ieșirii unui generator de zgomot, astfel încât probabilitatea apariției biților "0" și "1" să fie cât mai apropiate.

John von Neumann a presupus un joc de "cap sau pajură", folosind o monedă imperfectă (în care una dintre fețe iese de mai multe ori decât cealaltă) și a demonstrat că probabilitatea apariției la o primă aruncare a capului și la a doua aruncare a pajurii este egală cu probabilitatea apariției la prima aruncare a pajurii și la a doua aruncare a capului. Modelând matematic acest lucru [28], decorelatorul Von Neumann lucrează cu perechi de câte doi biți, ne luând în calcul perechile care au cei doi biți identici și rezultând primul bit al perechilor care au biții diferiți. Tabela de adevăr a decorelatorului Von Neumann este prezentată în Tab. 2.1. O altă abordare pentru atingerea

input1	input2	output
0	0	DROP
0	1	0
1	0	1
1	1	DROP

Tabela 2.1: Decorelatorul Von Neumann

obiectivelor unui decorelator este utilizarea funcțiilor HASH. Problema acestei abordări este că necesită putere computațională suplimentară iar în situațiile în care generatorul este limitat din punct de vedere al resurselor hardware, această soluție poate constrânge performanțele acestuia.

2.1.2 Testare statistică

Termenul de *numere aleatoare* se referă defapt la un *șir de numere aleatoare*. Atributul de **aleator** este datorat unui context statistic în care secvența a fost testată[1].

În criptografie, este esențial ca un număr aleator să fie impredictibil. Foarte multe atacuri criptanalitice se bazează exclusiv pe imperfecțiunile generatorilor slabi de numere aleatoare, care stau la baza generării cheilor criptografice sau la baza diverselor protocoale.

Cele mai utilizate baterii de teste de aleatorism în domeniul generării de numere aleatoare sunt: Diehard[5], Dieharder și NIST STS[14].

Bateria de teste Diehard Mai jos, sunt descrise testele din pachetul de teste statistice Diehard:

- **Birthday spacings** - Numele testului se bazează pe paradoxul zilei de naștere. Scopul testului este de a selecta aleator câteva puncte la o distanță suficient de mare. Distanțele dintre ele trebuie să aibă o distribuție exponențială asimptotică;
- **Overlapping permutations** - Acest test analizează secvențe a câte 5 numere consecutive ținând cont că după primele 120 de apariții numerele trebuie să apară cu aceeași probabilitate;
- **Ranks of matrices** - Acest test calculează ordinul matricilor disjuncte care pot fi create cu ajutorul secvențelor date;
- **Monkey tests** - Testul utilizează blocuri aleatoare pentru care calculează numărul de apariții;
- **Count the 1s** - Testul numără aparițiile bitului "1" dintr-un octet, transformându-l într-o "literă", numărând ulterior aparițiile cuvintelor de câte 5 astfel de litere consecutive;
- **Parking lot test** - Se consideră o matrice de 100 x 100 (considerată parcare). Se plasează o "mașină" într-un mod aleatoriu. În cazul în care locul este ocupat, se consideră eșec și se reia procedura curentă. După 12000 de încercări, numărul de mașini parcate trebuie să respecte o anumită distribuție;
- **Minimum distance test** - Testul implică plasarea a 8000 de puncte într-o matrice de 10000 x 1000, folosind ulterior distanța dintre perechi. Bineînțeles, această distanță trebuie să respecte o anumită distribuție;
- **Random spheres test** - Se aleg în mod aleator 400 de puncte într-un cub cu latura de 1000. Fiecare punct va crea o sferă a cărei rază este definită de distanța minimă dintre acesta și cel mai apropiat vecin. Volumul sferei cu cea mai mică rază trebuie să aibă o valoare care să respecte o anumită distribuție;
- **The squeeze test** - Se înmulțește numărul 2^{31} cu numere aleatoare aparținând intervalului (0,1) până când se ajunge la valoarea 1. Numărul de astfel de iterații trebuie să respecte o anumită distribuție;
- **Overlapping sums test** - Testul implică adunarea a 100 de numere aleatoare aparținând intervalului (0,1). Suma acestor numere trebuie să respecte o anumită distribuție;
- **Runs test** - Testul generează o secvență foarte mare de numere de tip "float" numărând valorile crescătoare și pe cele descrescătoare;
- **The craps test** - Testul supune secvența supusă testării la 200 000 de jocuri de Craps (zaruri) calculând numărul de câștiguri. [20][5];

Bateria de teste NIST STS National Institute of Standards and Technology din Statele Unite ale Americii este o organizație de standardizare, responsabilă cu elaborarea de standarde FIPS, în special pentru guvernul federal al SUA. Așadar, NIST a definit un standard care implementează un pachet de 15 teste statistice de aleatorism menite să testeze secvențe de biți de dimensiuni arbitrare, indiferent de sursa cu care acestea au fost generate. Teste verifică următoarele aspecte:

- **Frequency** - Acest test calculează numărul de apariții ale biților "1" și "0" și compară valorile. În cazul in care diferența dintre aceste valori este foarte mare (nu se află în limita impusă de standard), acest test va emite un rezultat negativ;
- **Block Frequency** - Acest test folosește același principiu ca primul, singura diferență fiind că lucrează cu grupuri de biți, de diferite dimensiuni. Frecvențele apariției grupurilor trebuie să fie cât mai apropiate.
- **Cumulative sums** - Acest test calculează sumele subsecvențelor șirului dat spre analiză;
- **Runs** - Acest test încearcă să identifice secvențele de biți care apar de mai multe ori în cadrul șirului analizat, calculând numărul de apariții;
- **Longest Run** - Acest test se folosește de datele primite de la cel anterior, calculând lungimea celui mai lung șir;
- **Rank** - Acest test calculează ordinul matricilor disjuncte care pot fi create cu ajutorul secvențelor date;
- **FFT** - Acest test calculează și interpretează valorile maxime ale Transformatei Fourier Rapide (Fast Fourier Transform);
- **NonOverlappingTemplates** - Acest test implică utilizarea unor șabloane pe care le caută de-a lungul șirului și le calculează numărul aparițiilor;
- **OverlappingTemplate** - Acest test funcționează la fel ca cel precedent, diferența constând în algoritmii de căutare;
- **Universal** - Acest test încearcă să aplice algoritmi de compresie asupra secvențelor, ținând cont de proprietatea Secvențelor Aleatoare de a nu putea fi arhivate în mod eficient;
- **Aproximate Entropy** - Acest test compară frecvențele blocurilor de n biți și pe cele ale blocurilor de $n+1$ biți;
- **Serials** - Acest test caută șabloane de dimensiuni fixe și le calculează numărul de apariții;
- **LinearComplexity** - Pe baza oricărei secvențe de date aleatoare se poate crea un LFSR (Linear Feedback Shift Register) care să regenereze această secvență. Acest test calculează lungimea acestui LFSR. [14][1]

2.1.3 Generatori de zgomot bazați pe oscilatoare în inel

Oscilatoarele în inel (RO) sunt circuite electronice bazate pe un număr impar de porți inversoare interconectate precum în Figura 2.2, fiind practic generatori de semnale de ceas.

Datorită unor factori naturali precum diferențele mici de fabricație dintre componentele electronice, temperaturile la care sunt supuse acestea în timpul funcționării sau alți factori externi,

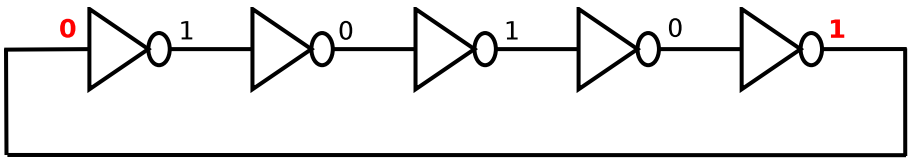


Figura 2.2: Schema funcțională a unui oscilator în inel

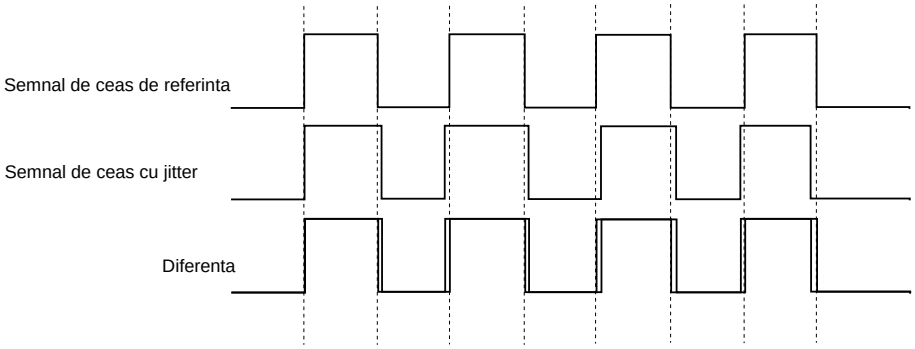


Figura 2.3: Evidențierea jitter-ului unui semnal de ceas

apare fenomenul de **jitter**, care poate fi observat în Figura 2.3. Jitterul reprezintă practic deviația periodicității unui semnal periodic care se presupune a fi perfect.

Pentru a ne putea folosi de acest fenomen impredictibil și complet necontrolabil în scopul construirii un generator de numere aleatoare trebuie să putem *evidenția* elementul aleatoriu din cadrul acestuia. Pentru acest lucru, putem să utilizăm ***două metode (construcții) de achiziție a aleatorismului***: *metoda desincronizării* și *metoda numărătorului*, descrise în cadrul lucrării [25].

2.2 Generatorul propus

2.2.1 Principiul de funcționare

Schema propusă pornește de la principiul definit de Sunar [22] ce implică utilizarea unui număr mare de oscilatoare în inel pentru construirea unui Generator de Numere Pur-Aleatoare. Datorită faptului că schema propusă de Sunar implică utilizarea unui număr mare de resurse hardware, schema propusă se dorește a fi un compromis resurse utilizate/securitate/viteză încercând să optimizeze prima schemă în acest sens [26].

Așadar, după optimizările aplicate conform articolelor [26] și [24], schema generatorului este prezentă în cadrul Figurii 2.4 și implică utilizarea a 32 de oscilatoare în inel, formate din 3, 5, 7, 11 porți inversoare. Această schemă funcționează după cum urmează, iar rezultatele testelor statistice aplicate asupra datelor generate sunt prezentate în cadrul lucrărilor [26] și [24].

Generatorul funcționează pe baza următorului principiu:

- Fiecare oscilator în inel este conectat la un decorelator von Neumann(descris în cadrul subcapitolului 2.1.1);

- Decorelatorul von Neumann funcționează pe baza unui semnal de ceas, care în cazul acesta este chiar ceasul intern al FPGA-ului. Acesta semnalizează la ieșire când are un bit valid;
- Arbitrul trece pe la fiecare bloc von Neuman în parte, colectând biții valizi ai acestora;
- După ce arbitrul a strâns 32 de biți valizi, îi adună modulo 2, rezultând un bit de ieșire al generatorului precum și un semnal de tip "data valid" care să semnalizeze când anume generatorul are un bit valid.
- Bitul rezultat este adunat modulo 2 cu valorile oscilatoarelor în inel de la momentul respectiv rezultând bitul de ieșire al întregului generator;

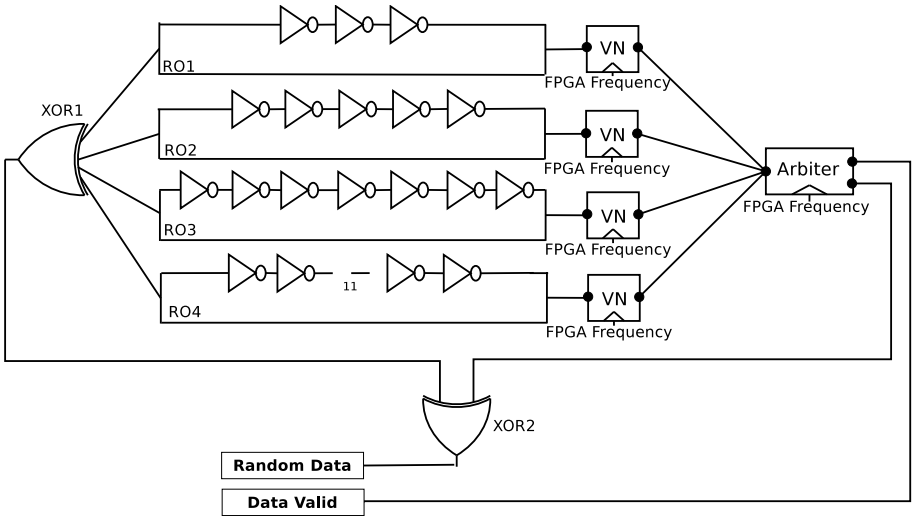


Figura 2.4: Schema Generatorului de Numere Pur-Aleatoare Propus - finalizată/optimă

2.2.2 Testarea calității generatorilor de numere

Un generator (pseudo)-aleator (GPA) de secvențe binare este un (algoritm)proces fizic care generează secvențe de biți de 0 și 1. În domeniul criptografiei, aceste structuri (matematice sau fizice după caz) constituie unul dintre elementele structurale de securitate ale primitivelor criptografice.

Un criteriu general utilizat și acceptat de către comunitatea academică în evaluarea de securitate a primitivelor criptografice (generatoare, algoritmi sau protocoale criptografice) este prin intermediul testelor statistice de verificare a caracterului aleator.

Evaluarea comportamentului uniform al generatoarelor (pseudo)aleatoare utilizate în criptografie este o condiție necesară dar nu și suficientă în "demonstrarea" securității acestora.

National Institute of Standards and Technologies (NIST), principalul organism de standardizare din SUA, a propus în cadrul documentului NIST SP 800-22 [14] o suită de 15 teste statistice pentru verificarea caracterului aleator al secventelor binare utilizate în aplicațiile criptografice. Subiectul este abordat și în literatura academică atât din punct de vedere matematic (Knuth [10]) cât și din punct de vedere al resurselor necesare implementării (Marsaglia [12], L'Ecuyer [11]).

Dacă eșantioanele, rezultate ca urmare a rulării GPA, trec testele statistice atunci nu rezultă implicit faptul că generatorul care a stat la baza generării eșantionului este "bun" pentru aplicații criptografice. În schimb, dacă se constată abateri de la comportamentul aleator atunci se poate trage concluzia că generatorul este "slab" din punct de vedere criptografic. Termenul de "slab" trebuie perceput prin faptul că există un "mecanism/metodă" prin care să poata fi prezis comportamentul generatorului.

Reținem faptul că aceste teste au fost principalul element de analiză a algoritmilor criptografici care au fost transmiși spre evaluare în cadrul competiției organizată de către NIST pentru selectarea algoritmului care va deveni standardul AES (Soto [21]).

În cadrul procesului de testare statistică avem două ipoteze statistice și anume:

- ipoteza nulă H_0 (secvența testată prezintă caracteristici de uniformitate);
- ipoteza alternativă HA secvența testată prezintă abateri de uniformitate Rezultatele testării statistice sunt supuse erorilor de ordinul 1 și 2: o eroare de ordinul 1 (notată cu α) este probabilitatea de a accepta o ipoteza falsă; o eroare de ordinul 2 (notată cu β) este probabilitatea de a respinge o ipoteză adevărata.

Testele NIST de evaluare a uniformității (algoritmilor)proceselor fizice care generează secvențe de biți au eroarea de ordinul 1 setată implicit la $\alpha = 0.01$ dar nu oferă nicio estimare privind eroarea de ordinul 2 (Murphy [13]). Aspecte similare fiind constate și în categoriile de aplicații software agreate la nivel internațional (academic și industrial).

Obiectivele asumate în cadrul proiectului au fost centrate pe analiza testelor statistice specificate în NIST SP 800-22, după cum urmează:

- parametrizarea ipotezelor statistice astfel încât acestea să răspundă la întrebarea privind apartenența eșantionului la o distribuție Bernoulli cu valoarea p diferită de 0.5;
- evaluarea gradului de corelație al testelor statistice;
- parametrizarea testelor astfel încât eroarea de ordinul 1 să fie variabilă;
- posibilitatea determinării erorii de ordinul 2 pentru fiecare test statistic;
- determinarea dimensiunii optime a volumului eșantionului astfel încât frecvența relativă să aproximeze probabilitatea teoretică cu o eroarea de cel mult ϵ;

O parte a rezultatelor obținute au fost publicate în [27][16][2].

Capitolul 3

Aspecte privind Implementarea şi Testarea

3.1 Tehnologii și Librării Utilizate

Proiectul a fost creat astfel încât să poată fi independent de distribuția de *Linux* utilizată în cadrul procesului de dezvoltare, iar limbajul de programare utilizat este C/C++.

Următoarele tehnologii și librării au fost utilizate în cadrul implementării funcțiilor criptografice:

- Librăriile **GMP** și **NTL** - GNU Multiple Precision Arithmetic Library [8] și Number Theory Library [15].

  Criptografia cu chei publice se bazează pe operații matematice cu numere foarte mari (1024-2048 de biți). Ținând cont de faptul că tipurile de date utilizate în cadrul limbajului de programare C/C++ nu oferă capabilitatea lucrului cu astfel de numere, acestea fiind limitate la maxim 64 de biți (vezi Tabela 3.1), s-a utilizat o librărie externă capabilă să efectueze operațiile necesare.

- **CMAKE** ([3]) - Pentru ușurarea procesului de programare s-a folosit utilitarul CMAKE care ne ajută la fabricarea proiectelor (structura de fișiere, fișiere de tip Makefile, etc) pentru diverse medii de dezvoltare (Eclipse, Netbeans, CodeBlocks etc.);

- Pachetul **openssl** - conține majoritatea funcțiilor criptografice sub formă de librării;

- **PJSIP** [17] - PJSIP este o bibliotecă multimedia de tip Open Source, scrisă în limbajul C, implementând protocoale de bază precum SIP, SDP, RTP, STUN, TURN și ICE;

- Utilitarul **SIPp** [18] - aplicație ce permite testarea unei comunicații de tip VoIP, permițând generarea de trafic și testarea acestuia. Acest utilitar pune la dispoziție și mecanisme de testare a infrastructurii (TLS, SCTP, SIP authentication, UDP retransmission, etc);

Toate librăriile și utilitarele utilizate sunt de tip Open Source.

3.2 Condiții minime de lucru

Pentru a putea începe dezvoltarea funcțiilor criptografice sunt necesare următoarele premise:

- Instalarea unei distribuții de Linux oarecare (în cadrul exemplului a fost utilizată Fedora 24) (a se vedea şi Secţiunea 3.4);

- Crearea următoarei structuri de fișiere:

Tipul de dată	Dimensiune (bytes)	Plajă de valori
Character	1	-128 - 127
Unsigned Char	1	0 - 255
Signed Char	1	-128 - 127
Integer	4	-2,147,483,648 - 2,147,483,647
Signed Int	4	0 - 4,294,967,295
Unsigned Int	4	-2,147,483,648 - 2,147,483,647
Short Int	2	-32,768 - 32,767
Signed short Int	2	0 - 65,535
Unsigned Short Int	2	-32,768 - 32,767
Long Int	4	-2,147,483,648 - 2,147,483,647
Signed Long Int	8	0 - 4,294,967,295
Unsigned Long Int	8	-2,147,483,648 - 2,147,483,647
Float	4	1.8E-38 - 3.4E+38
Double	8	2.2E-308 - 1.8E+308
Long Double	8	2.2E-308 - 1.8E+308
Bool	1	True/False - True/False

Tabela 3.1: Tipurile de date din limbajul de programare C/C++

```
->home
 ->user
  ->workspace
   ->ibmake
   ->pjsip
   ->ibmake
  ->workspace-build
   ->ibmake
    ->pjproject-2.4.5
    ->pjproject-2.5.5
    ->ibmake
```

- Următoarele pachete trebuie instalate (sau echivalentul acestora):

```
sudo dnf install -y \
 openssl-devel \
 SDL2-devel.x86_64 \
 opus-devel.x86_64 \
 libyuv-devel.x86_64 \
 alsa-lib-devel \
 gcc-c++ \
 libstdc++-devel \
 ffmpeg-libs \
 ncurses-devel \
 lzip
```

- Instalarea GMP:

```
cd /tmp
```

```
wget https://gmplib.org/download/ \
 gmp/gmp-6.1.1.tar.lz
mkdir -p ~/workspace/ibmake/ibmake/ \
 lib/gmp-6.1.1
tar --lzip -xf gmp-6.1.1.tar.lz \
 -C ~/workspace/ibmake/ibmake/lib
cd ~/workspace/ibmake/ibmake/lib/ \
 gmp-6.1.1
./configure --prefix=${HOME} \
 workspace-build/lib/gmp/6.1.1
make && make install
```

- Instalarea NTL:

```
cd /tmp
wget http://www.shoup.net/ntl/ntl-10.1.0.tar.gz
tar xf ntl-10.1.0.tar.gz -C ~/workspace/ \
 ibmake/ibmake/lib
rm -f /tmp/ntl-10.1.0.tar.gz
mkdir -p ${HOME}/workspace-build/ \
 lib/ntl/10.1.0
cd ~/workspace/ibmake/ibmake/lib/ \
 ntl-10.1.0/src
./configure \
PREFIX=${HOME}/workspace-build/ \
 lib/ntl/10.1.0 \
GMP_PREFIX=${HOME}/workspace-build/ \
 lib/gmp/6.1.1
make && make install
```

- Instalarea pjproject:

```
cd ~/Downloads
wget http://www.pjsip.org/release/ \
 2.5.5/pjproject-2.5.5.tar.bz2
mkdir -p ${HOME}/workspace/ibmake/ \
 pjproject/pjproject-2.5.5
tar -xjvf ~/Downloads/ \
 pjproject-2.5.5.tar.bz2 -C ${HOME}/workspace/ibmake/pjproject
mkdir -p ${HOME}/workspace-build/ \
 pjproject-2.5.5

cd ${HOME}/workspace/ibmake/pjproject/ \
 pjproject-2.5.5
./configure \
--prefix=${HOME}/workspace-build/ \
 pjproject-2.5.5 \
--enable-shared \
--disable-floating-point \
```

```
--disable-sound \
--disable-small-filter \
--disable-large-filter \
--disable-g711-plc \
--disable-speex-aec \
--disable-g711-codec \
--disable-l16-codec \
--disable-gsm-codec \
--disable-speex-codec \
--disable-ilbc-codec \
--disable-tls
make dep
make clean
make
make install
```

3.3 Testarea funcțiilor criptografice

Pentru testarea funcțiilor criptografice au fost create 3 fișiere sursă:

- `CryptoGen.hpp` - Clasa Criptografică implementată;
- `CryptoGen.cpp` - Implementarea funcțiilor și metodelor clasei `CryptoGen.hpp`;
- `gen_params.cpp` - fișierul sursă ce conține funcția "main", în cadrul căreia se testează funcționalitatea;

Pentru compilarea fișierelor s-a creat o sursă CMAKE ce conține următoarele:

```
cmake_minimum_required(VERSION 2.8)
project(gen_params CXX)

#SET(GMP_INCLUDE_DIR  CACHE PATH "GMP include dir")
#SET(GMP_LIBRARY_DIR  CACHE PATH "GMP library dir")
INCLUDE_DIRECTORIES(${GMP_INCLUDE_DIR})
link_directories(${GMP_LIBRARY_DIR})

#SET(IBMAKE_LIB_DIR   CACHE PATH "IBMAKE libs dir")
#SET(NTL_INCLUDE_DIR     CACHE PATH "NTL include dir")
#SET(NTL_LIBRARY_DIR     CACHE PATH "NTL library dir")
INCLUDE_DIRECTORIES(${NTL_INCLUDE_DIR})
link_directories(${NTL_LIBRARY_DIR})

MESSAGE (STATUS "SRC dir: ${CMAKE_SOURCE_DIR}")

include_directories()

SET(SOURCE_LIST
gen_params.cpp
CryptoGen.cpp
)

ADD_EXECUTABLE(${PROJECT_NAME} ${SOURCE_LIST})

TARGET_LINK_LIBRARIES(${PROJECT_NAME}
```

```
gmp
 ntl
)
```

Această sursă se va apela conform:

```
#se invoca din ${HOME}/workspace-build/ibmake

cmake \
-G"Eclipse CDT4 - Unix Makefiles" \
-DCMAKE_BUILD_TYPE=Debug \
-DGMP_INCLUDE_DIR="${HOME}/workspace-build/lib/gmp/6.1.1/include" \
-DGMP_LIBRARY_DIR="${HOME}/workspace-build/lib/gmp/6.1.1/lib" \
-DNTL_INCLUDE_DIR="${HOME}/workspace-build/lib/ntl/10.1.0/include" \
-DNTL_LIBRARY_DIR="${HOME}/workspace-build/lib/ntl/10.1.0/lib" \
${HOME}/workspace/ibmake/ibmake/ibmake
```

3.4 Crearea unei distribuții de Linux

3.4.1 Yocto Project

Yocto reprezintă o colecție de unelte ce facilitează realizarea unui sistem de operare destinat rulării pe un dispozitiv embedded. Aceste unelte permit unui dezvoltator să construiască un sistem de operare pornind de la o configuratie minimală(inițială), la care să adauge diferite aplicații. Proiectul Yocto a fost inceput în cadrul Linux Foundation și inițiat de Richard Purdie, care contribuie în continuare la acest proiect. Yocto este bazat pe niveluri și rețete. Fiecare nivel, sau layer reprezinta descrierea unei anumite funcționalități(kernel, bootloader, aplicație, etc.), iar rețeta defineste modul de funcționare si caracteristicile unei anumite aplicații rezidente in nivelul aferent. Acest proiect nu limiteaza în niciun fel dezvoltatorul, ci dimpotrivă, îi ofera posibilitatea folosirii unei structuri minimale ca punct de plecare, la care acesta sa poata adăuga un număr nelimitat de rețete și niveluri. În acest fel, se poate așadar dezvolta un sistem de operare Linux modificat în funcție de o nevoie specifică.

3.4.2 Configurare

În vederea configurării mediului Yocto, trebuie avute in vedere mai multe fișiere, și anume:

- *./build/conf/local.conf*: în acest fișier sunt stabilite opțiunile globale, valabile pentru orice *layer*. Opțiuni disponibile: numărul de threaduri pe care să se faca build-ul, tipul managerului de pachere(*package manager*), tipul formatului în care să se salveze sistemul de operare(ex.: *tar, ext4, sdcard*). Tot în cadrul acestui fișier sunt stabilite pachetele de bază care se vor instala în orice imagine generata pe viitor.
- *./build/conf/bblayers.conf*: în acest fișier sunt stabilite nivelurile care urmează să fie parsate.

3.4.3 Instalare

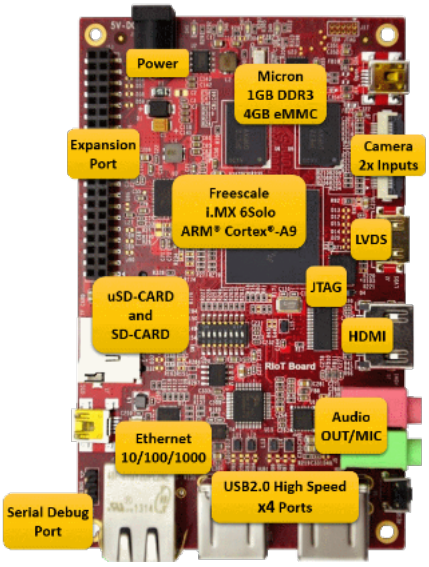
Pentru a construi o imagine funcțională de Linux, se vor folosi niveluri care vor descrie:

- *Bootloader-ul* care va fi responsabil cu recunoasterea si configurarea inițială a componentelor hardware pe care va rula sistemul de operare Linux. Acest bootloader va fi reprezentat de U-Boot, si anume **Universal Bootloader**.

- *Kernel-ul*
- Aplicația VoIP rezidentă în sistemul de fisiere

Construcția propriu-zisă a sistemului de operare se face folosind utilitarul furnizat de sistemul de build *poky*, și anume *bitbake*.

Instalarea propriu-zisă va avea loc pe placa de dezvoltare Riotboard.



Această placă de dezvoltare dispunde de un procesor ARM Cortex-A9, Gigabit Ethernet, microSD, codec audio si facilități de conectare la dispozitive video prin HDMI sau LVDS. La această placa se adaugă un display de tip touchscreen care va facilita utilizarea aplicației VoIp. Pentru o mai bună înțelegere a facilităților de care dispune această placă de dezvoltare, se poate urmări schema bloc a acestei plăci, disponibilă pe website-ul plăcii Riotboard.

Partea II

Realizări

Realizările majore ale acestei etape, descrise în paginile anterioare, se concretizează prin implementări asupra protocoalelor de securitate la nivel VoIP, fundamentate prin publicaţiile ştiinţifice:

1. **Andrei Marghescu** and Paul Svasta. Pushing the optimization limits of ring oscillator-based true random number generators. In *International Conference for Information Technology and Communications*, pages 209–224. Springer, 2016.

2. **Andrei Marghescu**, Paul Svasta, and **Emil Simion**. Randomness extraction techniques for jittery oscillators. In *2015 38th International Spring Seminar on Electronics Technology (ISSE)*, pages 161–166. IEEE, 2015.

3. **Andrei Marghescu**, Paul Svasta, and **Emil Simion**. Optimising ring oscillator-based true random number generators concept on FPGA. In *Electronics Technology (ISSE), 2016 39th International Spring Seminar on*, pages 149–153. IEEE, 2016.

4. **Emil Simion**. The relevance of statistical tests in cryptography. *IEEE Security & Privacy*, 13(1):66–70, 2015.

5. **F.L. Ţiplea** and **Anca Maria Nica**. Continuous mutual authentication and data security, 2016. (work in progress).

6. **Ferucio Laurentiu Ţiplea**, **Sorin Iftene**, **George Teşeleanu**, and **Anca Maria Nica**. Security of identity-based encryption from quadratic residuosity. In *International Conference for Information Technology and Communications*, pages 63–77. Springer, 2016.

Bibliografie

[1] Adrian Atanasiu. *Prelegerea 3, Cursul de Criptografie - Generatori de numere pseudo-aleatoare*. Universitatea din Bucuresti, Facultatea de Matematica.

[2] Mihai Barbulescu, Adrian Stratulat, Vlad Traista-Popescu, and Emil Simion. RSA weak public keys available on the Internet.

[3] CMAKE. Cmake. `https://cmake.org/`.

[4] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363. Springer-Verlag, 2001.

[5] Diehard. Diehard. `http://stat.fsu.edu/pub/diehard/`.

[6] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[7] **F.L. Ţiplea** and **Anca Maria Nica**. Continuous mutual authentication and data security, 2016. (work in progress).

[8] GMP. Gmp library. `https://gmplib.org/`.

[9] S. Iftene. Some connections between primitive roots and quadratic non-residues modulo a prime. Cryptology ePrint Archive, Report 2012/470, 2012. `http://eprint.iacr.org/2012/470`.

[10] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[11] Pierre L'Ecuyer and Richard Simard. Testu01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):22, 2007.

[12] George Marsaglia. The Marsaglia random number CDROM including the Diehard battery of tests of randomness, 1995. *URL http://www. stat. fsu. edu/pub/diehard*, 2008.

[13] Sean Murphy. The power of NIST's statistical testing of aes candidates. *Preprint. January*, 17, 2000.

[14] NIST. Nist - sts. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[15] NTL. Ntl library. `http://www.shoup.net/ntl/`.

[16] Ioana-Cristina Panait, Cristian Pop, Alexandru Sirbu, Adelina Vidovici, and Emil Simion. TOR-didactic pluggable transport.

[17] PJSIP. Pjsip. `http://www.pjsip.org/`.

[18] PJSIP. Pjsip. `http://sipp.sourceforge.net/`.

[19] Relevant Security Corp., Denver, Colorado, USA. *Real Privacy Management (RPM). Cryptographic Description Version 3.2*, 2010. (available at `www.therpmlab.com/papers.htm`).

[20] Juan Soto. Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, volume 10, page 12. NIST Gaithersburg, MD, 1999.

[21] Juan Soto Jr. *Randomness testing of the advanced encryption standard candidate algorithms*. PhD thesis, National Institute of Standards and Technology, 1999.

[22] Berk Sunar, William J Martin, and Douglas R Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on computers*, 56(1):109–119, 2007.

[23] **Ferucio Laurentiu Ţiplea**, **Sorin Iftene**, **George Teşeleanu**, and **Anca Maria Nica**. Security of identity-based encryption from quadratic residuosity. In *International Conference for Information Technology and Communications*, pages 63–77. Springer, 2016.

[24] **Andrei Marghescu** and Paul Svasta. Pushing the optimization limits of ring oscillator-based true random number generators. In *International Conference for Information Technology and Communications*, pages 209–224. Springer, 2016.

[25] **Andrei Marghescu**, Paul Svasta, and Emil Simion. Randomness extraction techniques for jittery oscillators. In *2015 38th International Spring Seminar on Electronics Technology (ISSE)*, pages 161–166. IEEE, 2015.

[26] **Andrei Marghescu**, Paul Svasta, and Emil Simion. Optimising ring oscillator-based true random number generators concept on FPGA. In *Electronics Technology (ISSE), 2016 39th International Spring Seminar on*, pages 149–153. IEEE, 2016.

[27] **Emil Simion**. The relevance of statistical tests in cryptography. *IEEE Security & Privacy*, 13(1):66–70, 2015.

[28] John Von Neumann. Various techniques used in connection with random digits. *Applied Math Series, Vol 12*, 1951.