

Universitatea Alexandru Ioan Cuza Iași
Facultatea de Informatică



Lucrare de Licență

RSA - variante, criptanaliză și aplicabilitate

Propusă de:

Cernescu Ștefan-Bogdan

Sesiunea iulie,2017

Coordonator științific

Lect.dr. Iftene Sorin

Universitatea Alexandru Ioan Cuza Iași
Facultatea de Informatică

RSA - variante, criptanaliză și aplicabilitate

Sesiunea iulie,2017

Cernescu Ștefan-Bogdan

Coordonator științific

Lect.dr. Iftene Sorin

Declarație privind originalitatea originalitatea și respectarea drepturilor de autor

Prin prezenta declar că Lucrarea de licență cu titlul "RSA - variante, criptanaliză și aplicabilitate" este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie în altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, iulie 2017

Absolvent
Cernescu Ștefan-Bogdan

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul "RSA - Variante și criptanaliză", codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru Ioan Cuza" Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, iulie 2017

Absolvent

Cernescu Ștefan-Bogdan

Cuprins

| | | |
|----------|---|-----------|
| 1 | Introducere în criptografie | 7 |
| 1.1 | Criptografia ca și domeniu [1] | 7 |
| 1.2 | Criptografia cu chei simetrice | 7 |
| 1.3 | Criptografia cu chei asimetrice | 8 |
| 1.4 | RSA Clasic [5] | 8 |
| 2 | Variante și Optimizări RSA | 12 |
| 2.1 | Bazele matematice | 12 |
| 2.2 | RSA Clasic | 15 |
| 2.3 | RSA Rebalansat | 23 |
| 2.4 | Multi-Prime RSA | 25 |
| 2.5 | Multi-Power RSA | 28 |
| 2.6 | RSA pe grupuri | 33 |
| 2.7 | RSA varianta distribuită (Schema lui Shoup) | 36 |
| 3 | Criptanaliză RSA | 44 |
| 3.1 | Criptanaliză pe RSA clasic [5] | 44 |
| 3.2 | Criptanaliză pe variante RSA | 57 |
| 4 | Aplicabilitate RSA - KeePass Plugin | 67 |
| 4.1 | Introducere | 67 |
| 4.2 | Tehnologii Utilizate | 68 |
| 4.3 | Arhitectura aplicației | 69 |
| 4.4 | Utilizarea aplicației | 73 |

Obiectivele lucrării

Lucrarea de față reprezintă un studiu amănunțit asupra criptosistemului cu asimetrice RSA (Rives, Shamir, Adleman) cât și o aplicație care folosește RSA ca și algoritm pentru securizarea datelor. În partea teoretică, voi prezenta un background referitor la tehnicile folosite de acest algoritm, voi descrie o serie de tehnici folosite în implementare, o serie de derivări ale acestui algoritm precum și o serie de atacuri care se pot efectua în anumite situații.

Pentru partea practică, voi prezenta o aplicație ce permite utilizatorului să-și exporte, folosind un cont parolele ținute în KeePass precum și importarea lor într-o altă bază de date, dacă acesta deține credențialele.

Motivație

Lucrarea își propune să exploreze și să aducă în atenție posibilitățile oferite în domeniul criptografiei, de unul dintre cei mai puternici algoritmi de criptare care există în momentul actual. Chiar dacă acesta este un algoritm destul de vechi, gradul lui de securitate este încă foarte ridicat datorită teoremei matematice care stă la baza acestuia (problema factorizării unui număr).

Pentru partea de aplicație, utilizatorul va beneficia de un plugin pentru aplicația KeePass, și acesta va putea oricând să-și țină datele sale într-un mediu securizat oferit de serviciul Vault, și să poată oricând să-și importe datele indiferent dacă acesta nu are acces la calculatorul unde aceste parole au fost salvate inițial.

Contribuții

Lucrarea reprezintă culegerea unor informații destul de importante dintr-o serie de articole științifice, și îndetalierea unor aspecte, pentru ca acest algoritm să poată fi înțeles mai ușor de

persoanele care nu sunt familiarizate cu acest domeniu, dar și oferirea unei implementări unice în ceea ce privește pluginul de KeePass, deoarece nu mai există vreo altă implementare făcută public în ceea ce privește comunicarea dintre KeePass și serviciile Vault.

Mai exact, contribuțiile mele la această lucrare sunt:

- Structurarea unor informații culese din articole științifice
- Demonstrații făcute asupra unor teoreme
- Calculul în detaliat pentru a se putea înțelege de unde provin anumite informații
- Scrierea unor algoritmi în format pseudocod pentru ușurință în implementare
- Rezolvarea unor modele folosind algoritmii în pseudocod pentru clarificare
- Prezentarea unor riscuri ce pot apărea în implementări eronate
- Oferirea unui plugin pentru unul dintre cele mai bune storage-uri de parole
- Autentificarea userului în KeePass
- Modalitatea de a exporta parolele într-un mediu securizat
- Modalitatea de a importa datele din Vault înapoi către KeePass

Structura lucrării

În capitolul I, am efectuat o scurtă introducere în ceea ce înseamnă criptografie, diferența dintre criptografia cu chei simetrice și asimetrice, precum și introducerea în criptosistemul RSA.

În capitolul al II-lea am expus o serie de derivări ale algoritmului RSA, prezentând și o serie de algoritmi care ajută la implementare, cât și la optimizarea acestuia (Euclid extins, teorema Chineză a resturilor, invers modularul, algoritmul lui Garner, lema lui Hensel).

În capitolul al III-lea am prezentat o parte dintre atacurile care s-au efectuat de-a lungul timpului asupra acestui algoritm cât și asupra derivatelor acestuia. În această secțiune vor fi detaliate modalitățile prin care se poate ataca criptosistemul, din ce cauză se pot efectua aceste atacuri dar și modalități prin care putem să le evităm.

În capitolul al IV-lea voi prezenta un plugin pentru aplicația KeePass ce-i va permite utilizatorului să-și exporte parolele salvate într-o bază de date locală într-un mediu securizat dar și opțiunea de importare a acestora. În această secțiune vor fi detaliate modalitățile de implementare ale acestui KeePass, precum și tehnologiile utilizate.

Capitolul 1

Introducere în criptografie

1.1 Criptografia ca și domeniu [1]

Criptografia este o ramură a matematicii care se ocupă cu securizarea informației prin modificarea conținutului acesteia. Termenul de criptografie este compus din cuvintele de origine greacă *kryptos*(ascuns) și *grafein*(a scrie).

Criptologia este considerată ca fiind cu adevărat o știință de foarte puțin timp. Această curpinde atât criptografie (scriere secretizată) cât și criptanaliza. De asemenea, criptologia reprezintă nu numai o artă veche, ci și o știință nouă: veche pentru că Iulius Cezar a utilizat-o, dar nouă pentru că a devenit o temă de cercetare academico-științifică abia începând cu anii 1970. Această disciplină este legată de multe altele, de exemplu teoria numerelor, algebră, teoria complexității, informatică.

1.2 Criptografia cu chei simetrice

Criptografia cu chei simetrice [29] se referă la metode de criptare în care atât trimițătorul cât și receptorul folosesc aceeași cheie.

$$S = (G, \varepsilon, D)$$

G algoritm probabilist de generare de chei, $k = G(\lambda)$

ε algoritm probabilist de criptare, $c = \varepsilon(k, m)$

D algoritm determinist de decriptare, $m = D(k, c)$

1.3 Criptografia cu chei asimetrice

Criptografia asimetrică [34] este un tip de criptografie care utilizează o pereche de chei: o cheie publică și una privată. Așadar, există o entitate, să-i spunem Bob care dorește să primească mesaje de la o serie de persoane: Alice, Tim, Michael. Astfel, Bob își va publica cheia publică, persoanele din lista sa își vor cripta mesajul folosind cheia publicată de Bob, iar singura entitate care va putea decripta acele mesaje va fi chiar Bob.

Matematic, cele două chei sunt legate, însă cheia privată nu poate fi obținută din cheia publică. În caz contrar, oricine ar putea decripta mesajele destinate unui alt utilizator, fiindcă oricine are acces la cheia publică a acestuia.

O astfel de legătură fiind folosită și în criptosistemul RSA. În continuarea acestei lucrări, vom prezenta tot feluri de cazuri unde expunem mereu un număr N care va fi cheia noastră publică, iar legătura dintre acesta și cheia privată este factorizarea acestuia, o componentă a cheii noastre private fiind factorii primi ai lui N .

Criptografia asimetrică mai poartă și denumirea de criptografie cu chei publice.

1.4 RSA Clasic [5]

1.4.1 Descriere

RSA este un algoritm criptografic cu chei publice, primul algoritm utilizat atât pentru criptare, cât și pentru semnătura electronică. Algoritmul a fost dezvoltat în 1977 și publicat în 1978 de Ron Rivest, Adi Shamir și Leonard Adleman la MIT, iar numele algoritmului provine din inițialele lor.

Securitatea sa se bazează pe dificultatea problemei factorizării numerelor întregi, problemă la care se reduce criptanaliza RSA și pentru care toți algoritmi de rezolvare cunoscuți au complexitate exponențială. Există câteva metode de criptanaliză care ocolesc factorizarea efectivă, exploatând maniere de implementare efectivă a schemei de criptare.

1.4.2 Funcționare

RSA este un algoritm de criptare pe blocuri. Acesta înseamnă că atât textul clar cât și cel criptat sunt numere între 0 și $N - 1$, cu un N ales. Un mesaj este împărțit în segmente de lungime corespunzătoare, numit blocuri, care sunt cifrate rând pe rând. De asemenea, ca algoritm criptografic cu

chei publice, funcționează pe baza unei perechi de chei legate matematic între ele: o cheie publică, cunoscută de toată lumea, și una secretă, cunoscută doar de deținătorul acesteia.

1.4.3 Generarea cheilor

Perechea de chei se generează după următorii pași:

1. Se generează două numere prime, de preferat mari, P și Q
2. Se calculează $N = PQ$ și $\phi(N) = (P - 1)(Q - 1)$
3. Se generează un număr întreg aleator e , $1 < e < \phi(N)$, astfel încât $\gcd(e, \phi(N)) = 1$, perechea $\langle N, e \rangle$ este cheia publică.
4. Folosind algoritmul lui Euclid extins, se calculează întregul d , element din $\mathbb{Z}_{\phi(N)}^*$, cu proprietatea că $de \equiv 1 \pmod{\phi(N)}$

1.4.3.1 Funcția lui Euler

Mai sus, s-a putut observa acea notație ϕ care este de fapt funcția lui Euler. Această funcție la primirea unui parametru N va returna numărul de elemente care sunt mai mici decât N și care sunt coprime cu acesta. Funcția lui Euler are următoarele proprietăți:

- $\phi(1) = 1$
- $\phi(P) = P - 1, \forall P$ prim
- $\phi(ab) = \phi(a)\phi(b) \forall a, b \geq 1$ și coprime între ele
- $\phi(P^e) = P^e - P^{e-1}, \forall P$ nr prim și $e > 0$
- $\phi(N) = (P_1^{e_1} - P_1^{e_1-1})(P_2^{e_2} - P_2^{e_2-1}) \dots (P_k^{e_k} - P_k^{e_k-1})$ unde $N = P_1^{e_1} \cdot P_2^{e_2} \cdot \dots \cdot P_k^{e_k}$ este factorizarea lui N

1.4.4 Criptare și Decriptare

Presupunând că mesajul clar este sub forma unui număr m , mai mic decât N , atunci mesajul criptat, notat cu c este:

$$c = m^e \pmod{N}$$

unde e este cheia publică a destinatarului mesajului. Pentru a decripta mesajul, destinatarul își folosește cheia sa secretă d , care are proprietatea foarte importantă că:

$$de \equiv 1 \pmod{\phi(N)}$$

Astfel, mesajul clar este recuperat calculând:

$$m = c^d \pmod{N}$$

Oricine poate decripta mesaje cu cheia publică a destinatarului, dar numai acesta din urmă poate decripta, deoarece trebuie să folosească cheia sa secretă.

1.4.5 Implementări eficiente

În general, deoarece se bazează pe o operație destul de costisitoare din punct de vedere al timpului de calcul și al resurselor folosite, și anume exponențierea modulo n , viteza RSA este mult mai mică decât a algoritmilor de criptare cu cheie secretă. În jurul anilor 90, se estima că o implementare RSA hardware este de 1000 de ori mai lentă decât o implementare DES, iar în software, RSA este de 100 de ori mai lent.

Există anumite modificări care pot aduce performanțe sporite, precum alegerea unui exponent de criptare mic (de exemplu $e = 17$), care astfel reduce calculele necesare criptării, rezolvând în același timp și unele probleme de securitate. De asemenea, operațiile cu cheia secretă pot fi accelerate pe baza teoremei chineze a resturilor, dacă se stochează P, Q și unele rezultate intermediare, folosite des. Viteza decriptării va crește de circa 4 ori. De aceea, în sistemele de comunicație în timp real, în care viteza de criptare și decriptare este esențială (cum ar fi de exemplu aplicațiile de streaming video sau audio securizate), RSA se folosește doar la începutul comunicației, pentru a transmite cheia secretă de comunicație, care ulterior este folosită într-un algoritm cu cheie secretă, cum ar fi 3DES sau AES.

1.4.6 Securitatea

Problema decriptării unui mesaj criptat cu RSA este denumită problema RSA. Acesta constă în obținerea rădăcinii de ordin e modulo N , unde e și N au proprietatea că N este produsul a două numere prime mari P și Q , iar e este relativ prim cu produsul dintre $P - 1$ și $Q - 1$. În acest moment, cea mai eficientă de a realiza aceasta este descompunerea în factori primi ai lui n , și obținerea astfel

a cheii secrete d pe baza lui e . Astfel, este demonstrat că dificultatea spargerii unui mesaj criptat cu RSA nu este mai dificilă decât problema factorizării. Nu a fost descoperită încă o soluție generală a problemei RSA, dar nici nu s-a demonstrat matematic că nu există o altă soluție.

Factorizarea întregilor prin metode comune ajută la găsirea soluțiilor în timp util doar pentru numere mici. Pentru numere mari, algoritmi de factorizare, cu complexitatea exponențială, dau soluția după foarte mult timp. Cea mai rapidă metodă de factorizare a întregilor, algoritmul "Number Field Siev", are o complexitate de $O(e^{c((\log n)^{\frac{1}{3}}(\log \log n)^{\frac{2}{3}})})$. Aici, c este un număr ce ia valori în jur de 1,9 pentru numere de tipul lui N , adică numere cu 2 factori primi. Cel mai mare număr factorizat vreodată prin acest algoritm, rulat în anul 2005, de către specialiștii de la Agenția Federală Germană pentru Securitatea Tehnologiei Informației, are 200 de cifre zecimale, iar reprezentarea binară a factorilor primi obținuți ocupă 663 de biți. Cheile de criptare RSA cele mai sigure au lungimi de peste 1024 de biți.

Atacul RSA forței brute, adică încercarea fiecărei chei secrete posibile, consumă chiar mai mult timp decât factorizarea.

Capitolul 2

Variante și Optimizări RSA

2.1 Bazele matematice

2.1.1 Comportamentul asimptotic

2.1.1.1 Analiza unui algoritm

Cel mai bun parametru de securitate o reprezintă lungimea unei chei. O cheie mai mare ne va asigura un nivel de securitate ridicat, dar operațiile vor necesita un timp mai mare și vor consuma mai multă memorie. În momentul actual, lungimea unei chei RSA este de 1024, chiar și 2048 de biți.

2.1.2 Aritmetica numerelor întregi

2.1.2.1 Operațiile de bază

Avem o serie de operații de bază la îndemâna noastră, cum ar fi operațiile logice, evaluarea condițiilor sau saltul în program. Aceste lucruri ne ajută pentru a calcula operații mai complexe cum ar fi înmulțirea a doua numere pe n biți. Înmulțirea sau împărțirea acestor numere se numesc operații cu multiplă precizie.

2.1.2.2 Reprezentarea într-o anumită bază

Numerele întregi pozitive pot fi reprezentate în multiple moduri [7]. Cea mai folosită reprezentare a acestora este baza 10, un astfel de număr este reprezentat prin serii de 0 și 9. Spre exemplu

$a = 123$ înseamnă $a = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$. Prin extensie, este posibil să reprezentăm un întreg utilizând o bază b : $a = a_k \cdot b^k + a_{k-1} \cdot b^{k-1} + \dots + a_0 \cdot b^0$. Pentru un număr a și o bază b există mereu această reprezentare și este unică. Deoarece calculatoarele lucrează cu 0 și 1, baza 2 este mereu folosită. Așadar, vom folosi următoarea notație

$$a = (111011)_2 = 2^5 + 2^4 + 2^3 + 0 \cdot 2^2 + 2^1 + 2^0$$

Algorithm 1 Transformarea unui număr din baza 10 în baza b

Input: $a, b \in \mathbb{Z}$

Output: a_b

$a_b = []$

while $a \neq 0$ **do**

$a_b.push(a \bmod b)$

$a = a \div b$

end while

return a_b

2.1.3 Aritmetica modulară

2.1.3.1 Reducerea modulară

Majoritatea criptosistemelor sunt bazate pe grupuri finite; spre exemplu \mathbb{Z}_N . Fiecare element x din acest grup poate fi notat ca și:

$$x = r \bmod N \Leftrightarrow \exists q \in \mathbb{Z}, x = r + q \cdot N$$

2.1.3.2 Inversul modular

Inversul modular [30] poate fi calculat folosind algoritmului Euclid Extins, care nu calculează doar cel mai mare divizor comun a două numere întregi x și y , dar găsește și două numere întregi α și β astfel încât $\text{cmmdc}(x, y) = \alpha \cdot x + \beta \cdot y$

Dacă dorim să aflăm inversul lui $x \bmod N$ (presupunând faptul că numerele sunt coprime, adică $\text{cmmdc}(x, N) = 1$) apoi calculăm $X\text{cmmdc}(x, N)$ (variante extinsă a algoritmului lui Euclid) și

obținem $\alpha \cdot x + \beta \cdot N = 1$. Modulo N , această expresie devine $\alpha \cdot x = 1 \bmod N$; în alte cuinte $\alpha = x^{-1} \bmod N$

Pentru ușurința înțelegerii, vom prezenta algoritmul extins al lui Euclid într-un pseudocod, precum și algoritmul pentru calculul inversului modular.

Algorithm 2 Euclid extins

Input: $a, b \in \mathbb{Z}$

Output: (a, b) și $V(\alpha, \beta)$ astfel încât $(a, b) = \alpha a + \beta b$

$V_0 \leftarrow (1, 0)$

$V_1 \leftarrow (0, 1)$

while $b \neq 0$ **do**

$q \leftarrow a \text{ div } b$

$r \leftarrow a \bmod b$

$a \leftarrow b$

$b \leftarrow r$

$V \leftarrow V_0$

$V_0 \leftarrow V_1$

$V_1 \leftarrow V - qV_1$

end while

$(a, b) \leftarrow |a|$

$V \leftarrow V_0$

return V

Algorithm 3 Inversul modular

Input: $m \geq 1$ și $a \in \mathbb{Z}_m$

Output: $a^{-1} \in \mathbb{Z}_m$, dacă $(a, m) = 1$

 calculăm $\text{cmmdc}(a, m) = \alpha a + \beta b$

if $\text{cmmdc}(a, m) = 1$ **then**

$a^{-1} = \alpha \bmod m$

else

 Ecuția nu are soluții întregi

end if

Pentru un exemplu concret, vom încerca să calculăm $7^{-1} \in \mathbb{Z}_{26}$. Deoarece $7 < 26$, vom calcula $Xcmmdc(26, 7) = \beta b + \alpha a$, deci la final vom inversa coordonatele obținute.

$$V_{26} = (1, 0)$$

$$V_7 = (0, 1)$$

$$26 : 5 = 3 \text{ rest } 5 \Rightarrow$$

$$5 = 26 - 7 \cdot 3 \Rightarrow$$

$$V_5 = (1, 0) - 3 \cdot (0, 1) = (1, -3) \Rightarrow$$

$$7 : 5 = 1 \text{ rest } 2 \Rightarrow$$

$$2 = 7 - 5 \Rightarrow$$

$$V_2 = (0, 1) - (1, -3) = (-1, 4) \Rightarrow$$

$$5 : 2 = 2 \text{ rest } 1 \Rightarrow$$

$$V_1 = V_5 - 2 \cdot V_2 = (1, -3) - 2 \cdot (-1, 4) = (3, -11)$$

Știm că 1 este ultimul rest $\neq 0$ deci V_1 este combinația liniară pe care o căutam. $V_1 = (3, -11)$ dar, deoarece am făcut aceea, vom efectua o interschimbare la primul pas, V_1 este de fapt $(-11, 3) \Rightarrow \alpha = -11, \beta = 3$

Putem face și verificarea $-11 * 7 + 3 * 26 = -77 + 78 = 1$. Cu α cunoscut, putem afla $7^{-1} \in \mathbb{Z}_{26}$ 7 și 26 sunt coprime (cel mai mare divizor comun al lor este 1) deci, am reușit să respectăm condiția inițială a algoritmului. -11 în \mathbb{Z}_{26} este 15 iar $15 \bmod 26 = 15 \Rightarrow 7^{-1} = 15$. Verificând această afirmație, ajungem la $7 \cdot 15 \equiv 1 \pmod{26}$, deoarece $7 * 15 : 26 = 105 : 26 = 4 \text{ rest } 1$

2.2 RSA Clasic

2.2.1 Exponențiere Rapidă

2.2.1.1 Descrierea metodei binare (varianta de la dreapta la stânga)

Această metodă de exponențiere este bazată pe reprezentarea binară a exponentului și ne permite să calculăm un exponent cu un număr mic de înmulțiri și ridicări la pătrat [7]. Luăm baza M , exponentul e și modulul N ca și input, și rezultatul exponențierii $C = M^e \bmod N$ ca și output.

Pe lângă faptul că avem o reprezentare arbitrară a lui $e = \sum_{i=0}^{k-1} e_i \cdot 2^i$, unde e_i este 0 sau 1. Putem transforma exponentul în următoarea expresie:

$$\begin{aligned} M^e &= M^{\sum_{i=0}^{k-1} e_i \cdot 2^i} \bmod N \\ &= \prod_{i=0}^{k-1} M^{e_i \cdot 2^i} \bmod N \\ &= \prod_{\substack{i=0 \\ e_i=1}}^{k-1} M^{2^i} \bmod N \end{aligned}$$

Folosind relația, vom putea calcula recursiv rezultatul: prima dată îl vom inițializa cu 1, apoi vom parcurge reprezentarea binară a exponentului. Dacă bitul de la poziția i este 1, atunci o vom multiplica cu rezultatul M^{2^i} . Dar acest lucru se poate face mai rapid: dacă inițializăm rezultatul cu 1 și mergem prin reprezentarea binară a exponentului de la stânga către dreapta. La fiecare pas, ridicăm la pătrat rezultatul și în înmulțim cu M dacă bitul de pe acea poziție este 1. La final, rezultatul va fi multiplicarea acestuia cu M^{2^i} .

Algorithm 4 Algoritmul exponențierii de la dreapta la stânga

Input: $M, e = (e_{k-1} \dots e_0)_2, N$

Output: $C = M^e \bmod N$

```

C ← 1
for  $i$  from  $(k-1)$  down to 0 do
    C ← C · C mod N
    if  $e_i = 1$  then
        C ← C · M mod N
    end if
end for

```

2.2.1.2 Corectitudinea algoritmului de la dreapta la stânga

Vom dovedi prin inducție că acest algoritm funcționează. Să spunem că la un anumit pas i avem $C_i = \prod_{j=i}^{k-1} M^{e_j \cdot 2^{j-i}} \bmod N$. La următorul pas, vom calcula ridicarea la pătrat și o înmulțire dacă $e_{i-1} = 1$. Prin urmare:

$$\begin{aligned}
C_{i-1} &= M^{e_{i-1}} * C_i^2 \bmod N \\
&= M^{e_{i-1}} \cdot \left(\prod_{j=i}^{k-1} M^{e_j * 2^{j-i}} \right)^2 \bmod N \\
&= M^{e_{i-1}} \cdot \prod_{j=i}^{k-1} M^{e_j * 2^{j-i+1}} \bmod N \\
&= \prod_{j=i-1}^{k-1} M^{e_j * 2^{j-(i-1)}} \bmod N
\end{aligned}$$

La începutul buclei, la pasul $k-1$ afirmația este adevărată deoarece $C = M^{e_{k-1}} \bmod N$. Prin inducție, afirmația este adevărată cât timp $i = 0$. Iar pentru $i = 0$ avem:

$$C = \prod_{j=0}^{k-1} M^{e_j * 2^j} \bmod N$$

2.2.2 Teorema Chineză a resturilor

2.2.2.1 Descrierea

Teorema Chineză a resturilor [30] ne permite să reducem timpul computațional pentru o decryptare clasică RSA. Mai general, teorema formează o bijecție între \mathbb{Z}_m și produsul cartezian

$$\prod_{i=0}^k \mathbb{Z}_{m_i} = \mathbb{Z}_{m_0} \times \mathbb{Z}_{m_1} \times \dots \times \mathbb{Z}_{m_k}$$

Asumând faptul că $m = \prod_{i=0}^k m_i$ și că m_i au divizor comun doar pe 1. Spre exemplu, există un izomorfism între \mathbb{Z}_N și $\mathbb{Z}_P \times \mathbb{Z}_Q$ în loc de \mathbb{Z}_N , P și Q reprezentând două numere prime astfel încât $N = PQ$. Spunem că lungimea în biți a lui P și Q este jumătate din lungimea în biți ai lui N , costul înmulțirii este teoretic divizat prin 2. Este ușor a calcula imaginea unui anumit a , dar dacă $a = a_i \bmod m_i$ este dat, atunci reciproca nu este trivială. Obiectul teoremei este să dea soluția unică modulo m a unui sistem de forma:

$$\begin{cases} a \equiv a_1 \bmod m_1 \\ \vdots \\ a \equiv a_k \bmod m_k \end{cases}$$

Algorithm 5 Teorema Chineză a resturilor

Input: Un sistem de k ecuații de forma $x \equiv b \mod m$ **Output:** Soluția unică a sistemului**for** i from 1 to k **do**

$$c_i = \prod_{\substack{j=1 \\ j \neq i}}^k m_j$$

end for**for** i from 1 to k **do**calculăm soluția ecuației $c_i x \equiv b_i \mod m_i$ folosind Euclid Extins**end for****return** $(c_1 x_1 + \dots c_k x_k) \mod (m_1 \dots m_k)$

Algorithm 6 Rezolvarea unei ecuații liniare congruente

Input: $m \geq 1$ și $a, b \in \mathbb{Z}$ **Output:** toate soluțiile modulo m ale ecuației $ax \equiv b \mod m$ calculăm $\text{cmmdc}(a, m) = \alpha a + \beta m$ **if** $\text{cmmdc}(a, m) | b$ **then**listă_soluții $\leftarrow []$

$$b' \leftarrow \frac{b}{\text{cmmdc}(a, m)}$$

$$x_0 \leftarrow \alpha b'$$

for i from 0 to $\text{cmmdc}(a, m) - 1$ **do**

$$\text{listă_soluții.push}\left(\left(x_0 + \frac{im}{\text{cmmdc}(a, m)}\right) \mod m\right)$$

end for**else**

Nu există soluții întregi

end if

Pentru o înțelegere mai clară, vă vom prezenta următorul sistem de ecuații, și dorim să determinăm soluția sa unică folosind teorema Chineză a resturilor:

$$\begin{cases} x \equiv 2 \mod 3 \\ x \equiv 3 \mod 4 \\ x \equiv 6 \mod 7 \end{cases}$$

$$c_1 = 28, c_2 = 21, c_3 = 12$$

Soluțiile primelor 2 ecuații

$$28x \equiv 2 \pmod{3}$$

$$21x \equiv 2 \pmod{4}$$

se pot deduce foarte ușor, deoarece valorile lui m_i sunt destul de mici, deci $x_1 = 2$ și $x_2 = 3$. Pentru ecuația

$$21x \equiv 3 \pmod{4}$$

vom aplica algoritmul extins al lui Euclid.

$$V_{12} = (1, 0)$$

$$V_7 = (0, 1)$$

$$12 : 7 = 1 \text{ rest } 5 \Rightarrow$$

$$5 = 12 - 7 \Rightarrow$$

$$V_5 = V_{12} - V_7 = (1, 0) - (0, 1) = (1, -1)$$

$$7 : 5 = 1 \text{ rest } 2 \Rightarrow$$

$$2 = 7 - 5 \Rightarrow$$

$$V_2 = V_7 - V_5 = (0, 1) - (1, -1) = (-1, 2) \Rightarrow$$

$$5 : 2 = 2 \text{ rest } 1 \Rightarrow$$

$$1 = 5 - 2 \cdot 2 \Rightarrow$$

$$V_1 = V_5 - V_2 = (1, -1) - (-2, 4) = (3, -5)$$

Avem $\alpha = 3$ și $\beta = -5$, iar $3 \cdot 12 - 5 \cdot 7 = 1$ deci V_1 este corect.

Acum vom aplica algoritmul de rezolvare a unei ecuații liniare congruente. Deoarece $\text{cmmdc}(a, m) = 1$ înseamnă că ecuația va avea o singură soluție.

$$b' = \frac{b}{\text{cmmdc}(a, m)} = 6x_0 = \alpha b = 3 \cdot 6 = 18x_3 = x_0 \pmod{7} = 4$$

$$\begin{aligned}
x &= (c_1x_1 + c_2x_2 + c_3x_3) \bmod (3 \cdot 4 \cdot 7) \\
&= (56 + 63 + 48) \bmod 84 \\
&= 167 \bmod 84 \\
&= 83
\end{aligned}$$

Așadar, soluția finală este $x = 83$. Putem face și verificările:

$83 : 3 = 27 \text{ rest } 2$, prima ecuație este adevărată

$83 : 4 = 20 \text{ rest } 3$, a doua ecuație este și ea adevărată

$83 : 7 = 11 \text{ rest } 6$, ultima ecuație este și ea adevărată, deci rezultatul final $x = 83$ este corect.

2.2.2.2 Demonstrația teoremei

Căutăm să găsim acel a , astfel încât $a \equiv a_i \bmod m_i$ pentru i de la 1 până la k . Să dovedim că $a = \sum_{i=1}^k a_i \cdot M_i \cdot y_i$ este o soluție a sistemului.

$M_i = \prod_{j=1, j \neq i}^k m_j$ și $M_j = 0 \bmod m_i$ dacă $j \neq i$, de aceea:

$$a = a_i \cdot M_i \cdot y_i \bmod m_i$$

Dar am ales y_i astfel încât $y_i = M_i^{-1} \bmod m_i$; cu alte cuvinte $y_i \cdot M_i = 1 \bmod m_i$. În final, obținem pentru i de la 1 până la k :

$$a = a_i \bmod m_i$$

2.2.3 Algoritmul lui Garner

2.2.3.1 Descrierea algoritmului

Utilizarea clasică a teoremei Chineze a resturilor necesită o reducere modulară modulo N , unde N este un produs de două numere prime P și Q . Este posibil să estimăm calculul a celor doi inverși datorită algoritmului Garner. Totuși, costul din punct de vedere al memoriei este mai mare. Ca și în algoritmul precedent, trebuie să găsim M astfel încât $M = M_P \bmod P$ și $M = M_Q \bmod Q$. Mai avem un parametru $P_{inv_Q} = P^{-1} \bmod Q$. O descriere mai generală se poate găsi în [7]

Algorithm 7 Algoritmul lui Garner**Input:** $M_P, M_Q, P, Q, P_{inv_Q}, N$ **Output:** M

$$V \leftarrow M_Q - M_P \bmod Q$$

$$V \leftarrow V * (P_{inv_Q}) \bmod Q$$

$$M \leftarrow V * P \bmod N$$

$$M \leftarrow M + M_P \bmod N$$

return(M)

Pentru acest algoritm, vom încerca să rezolvăm din noul sistemul de ecuații

$$\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 3 \bmod 4 \\ x \equiv 6 \bmod 7 \end{cases}$$

O problemă o reprezintă faptul că acest algoritm este conceput pentru un sistem cu 2 ecuații. Totuși, putem rezolva 2 dintre aceste ecuații, iar cu rezultatul obținut, o putem rezolva și pe ultima. Pentru această metodă, am ales pentru început rezolvarea ultimelor 2 ecuații.

$$\begin{cases} x \equiv 3 \bmod 4 \\ x \equiv 6 \bmod 7 \end{cases}$$

Vom începe prin a face notațiile $M_P = 3, M_Q = 6, P = 4, Q = 7, P_{inv_Q} = 2$ (ultimul termen fiind aflat folosind algoritmul extins al lui Euclid) după care putem trece la executarea algoritmului:

$$V = 6 - 3 \bmod 7 = 3$$

$$V = 3 \cdot 2 \bmod 7 = 6$$

$$M = 6 \cdot 4 \bmod 28 = 24$$

$$M = 27$$

Acum trebuie să aplicăm din nou acest algoritm pe sistemul

$$\begin{cases} x \equiv 2 \bmod 3 \\ x \equiv 27 \bmod 28 \end{cases}$$

Efectuăm din nou notațiile $M_P = 2, M_Q = 27, P = 3, Q = 28, P_{inv_Q} = 19$ și executăm algoritmul:

$$V = 27 - 2 \bmod 28 = 25$$

$$V = 25 \cdot 19 \bmod 28 = 27$$

$$M = 27 \cdot 3 \bmod 84 = 81$$

$$M = 81 + 2 \bmod 84 = 83$$

După cum observăm, am obținut același rezultat ca atunci când am aplicat teorema Chineză a resturilor.

2.2.3.2 Corectitudinea algoritmului

Vrem să calculăm M astfel încât $M = M_P \bmod P$ și $M = M_Q \bmod Q$ și $P_{inv_Q} = P^{-1} \bmod Q$. Să arătăm că:

$$M = M_P + V * P \bmod N$$

unde $V = P_{inv_Q} \cdot (M_Q - M_P) \bmod Q$

$M = M_P \bmod P$ și :

$$\begin{aligned} P * V &= P * P_{inv_Q} \cdot (M_Q - M_P) \bmod Q \\ &= M_Q - M_P \bmod Q \end{aligned}$$

În final:

$$\begin{aligned} M &= M_P + M_Q \bmod Q \\ &= M_Q \bmod Q \end{aligned}$$

2.2.4 Implementări rapide RSA

Am prezentat o optimizare a algoritmului RSA folosind teorema Chineză a resturilor și o exponențiere mai rapidă. Cei doi inverși sunt de obicei necesari pentru teorema chineză iar numărul de inverși va fi redus la unul singur. Prin acest mod, viteza de decriptare descrește de aproximativ 4 ori.

2.2.4.1 Generarea cheilor, criptare și decriptare

1. Generarea cheilor:

Generarea cheilor este aceeași ca și în RSA standard: avem ca input parametrul de securitate n , alegem două numere prime cu lungimea în biți fiind jumătatea lui n după care le înmulțim. Apoi alegem în număr întreg e astfel încât $\phi(N)$ și e sunt relativ prime. În final calculăm d astfel încât $ed = 1 \bmod \phi(N)$ și după să-l reducem modulo $(P-1)$ și $(Q-1)$. Apoi calculăm $P_{inv_Q} = P^{-1} \bmod Q$: toate aceste valori sunt fixate și sunt necesare pentru decriptare. Cele două numere $\langle N, e \rangle$ reprezintă cheile publice și $\langle P, Q, d_P, d_Q, P_{inv_Q} \rangle$ sunt cheile secrete.

2. Criptarea:

Mesajul criptat, va trebui întâi convertit la un număr întreg. Sunt anumite tipuri de algoritmi care fac această conversie (o modalitate ar fi să folosim codificarea ascii). Având plaintextul M , este ușor să calculăm $C = M^e \bmod N$

3. Decriptarea:

Pentru decriptare nu vom face direct calculul $M = C^d \bmod N$, ci vom folosi una din schemele eficiente prezentate mai sus. $M_P = C_P^{d_P} \bmod P$ și $M_Q = C_Q^{d_Q} \bmod Q$ unde $C_P = C \bmod P$, $c_{mmdc} = d \bmod (P-1)$ și $C_Q = C \bmod Q$, $d_Q = d \bmod (Q-1)$. Apoi este posibil să reveni la plaintextul M datorită teoremei Chineze a resturilor. Această metodă este mai rapidă pentru că vom avea 2 exponențieri cu numere pe $n/2$ biți în loc de o exponențiere pe n biți.

2.3 RSA Rebalansat

De obicei, o decriptare rapidă este dorită. De obicei server SSL (care efectuează decriptare RSA) sunt supraîncărcate. Așadar, vom avea nevoie de un algoritm foarte bun în ceea ce privește decriptarea, chiar dacă va trebui să creștem puțin timpul de criptare. Acest lucru este ideea pe care se bazează RSA rebalansat, unde munca mai costisitoare (criptarea) va fi efectuată de un client (de obicei un browser web) în timp ce partea de decriptare (cea făcută de server) va dura mai puțin. Pentru o astfel de implementare, nu vom alege un exponent d de decriptare foarte mic, deoarece acest lucru este foarte nesecurizat (vom vedea mai târziu și o serie de atacuri care au succes atunci când exponentul de decriptare este foarte mic), conform [20] și [21] care spun că d este nesecurizat, atunci când $d < N^{0.292}$, astfel, vom alege d astfel încât $d \bmod (P-1)$ și $d \bmod (Q-1)$ sunt mici.

A se vedea [20] pentru mai multe detalii.

2.3.1 Generarea cheilor, criptare și decriptare

2.3.1.1 Generarea cheilor

Algoritmul primește 2 parametri de securitate ca și input: n (de obicei 1024 biți) și k (de obicei 160 biți) unde $k < n/2$. Apoi se aleg două numere prime P și Q verificând că $\text{cmmdc}(P-1, Q-1) = 2$ și care au lungimea $n/2$. Modulul N este $N = PQ$. Apoi generăm aleator două valori pe k biți d_P și d_Q astfel încât $\text{cmmdc}(d_P, P-1) = 1, \text{cmmdc}(d_Q, Q-1) = 1$ și $(d_P = d_Q \bmod 2)$. Exponentul secret d trebuie să verifice: $d = d_P \bmod (P-1)$ și $d = d_Q \bmod (Q-1)$. Nu putem calcula direct d folosind teorema chineză a resturilor pentru că $P-1$ și $Q-1$ nu sunt relativ prime, dar noi le alegem astfel încât $\text{cmmdc}(P-1, Q-1) = 2$ prin urmare:

$$\text{cmmdc}\left(\frac{P-1}{Q-1}, \frac{Q-1}{2}\right) = 1$$

De asemenea știm că $d_P = d_Q \bmod 2$. Fie $a = d_P \bmod 2$. Datorită teoremei chineze a resturilor, putem calcula un d' care verifică:

$$d' = \frac{d_P - a}{2} \bmod \left(\frac{P-1}{2}\right)$$

$$d' = \frac{d_Q - a}{2} \bmod \left(\frac{Q-1}{2}\right)$$

În alte cuvinte, există două numere întregi k_1 și k_2 astfel încât:

$$d' = \frac{d_P - a}{2} + k_1 \cdot \left(\frac{P-1}{2}\right)$$

$$d' = \frac{d_Q - a}{2} + k_2 \cdot \left(\frac{Q-1}{2}\right)$$

Fie $d = 2d' + a$ care verifică:

$$d = (d_P - a) + k_1 \cdot (P-1) + a$$

$$d = (d_Q - a) + k_2 \cdot (Q-1) + a$$

Modulo $P-1$ și $Q-1$ obținem:

$$d = d_P \bmod (P-1)$$

$$d = d_Q \bmod (Q-1)$$

Pentru a calcula exponentul public e , trebuie doar să calculăm inversul lui d modulo $\phi(N) = (P - 1)(Q - 1)$. Acest lucru este permis deoarece $\text{cmmdc}(d_P, P - 1) = \text{cmmdc}(d_Q, Q - 1) = 1$. Prin urmare, $\text{cmmdc}(d, P - 1) = \text{cmmdc}(d, Q - 1) = 1$. În final $\text{cmmdc}(d, \phi(N)) = 1$. Nu avem control asupra lui e care este de ordinul lui N . Criptarea nu va fi la fel de rapidă ca și criptarea clasică RSA dar vom putea măări viteza decriptării.

2.3.1.2 Criptarea

Aceasta este exact ca și în RSA standard, doar că e este mult mai mare iar d mult mai mic. Cheia publică este $\langle N, e \rangle$.

2.3.1.3 Decriptarea

Putem decripta criptotextul C prin calculul $M_P = C^{d_P} \bmod P$ și $M_Q = C^{d_Q} \bmod Q$. Folosind teorema Chineză a resturilor putem recupera plaintextul inițial M prin rezolvarea sistemului:

$$\begin{cases} x \equiv M_P \bmod P \\ x \equiv M_Q \bmod Q \end{cases}$$

2.4 Multi-Prime RSA

Datorită teoremei Chineze a resturilor, în loc să avem doar un modul de forma $N = PQ$, pute extinde acest algoritm și la un modul de forma $N = PQR$ [17]. Totuși, 3 numere prime pentru N pe 1024 este suficient. Creșterea numărului de numere prime, va prezenta un incident de securitate deoarece numerele maxim 256 de biți pot fi foarte ușor factorizate folosind ECM [19]

2.4.1 Algoritmul Garner cand $N = PQR$

2.4.1.1 Descrierea algoritmului

Algoritmul Garner este un algoritm pentru rezolvarea de sisteme de ecuații modulo m , ce ne permite să îmbunătățim viteza de decriptare, dar cu un consum mai mare al memoriei. Mai sus am prezentat varianta pentru două numere prime, dar în acest caz vom avea 3 numere prime. Algoritmul are ca input M_P, M_Q, M_R, P, Q, R și N , unde $N = PQR$. De asemenea, trebuie să calculăm și PQ_{inv_R} și P_{inv_Q} , verificând că $PQ_{inv_R} = (PQ)^{-1} \bmod R$ și $P_{inv_Q} = P^{-1} \bmod Q$. Calculăm M astfel

încât:

$$\begin{cases} M \equiv M_P \text{ mod } P \\ M \equiv M_Q \text{ mod } Q \\ M \equiv M_R \text{ mod } R \end{cases}$$

2.4.1.2 Corectitudinea algoritmului

Dorim să calculăm $M = M_P \text{ mod } P$, $M = M_Q \text{ mod } Q$ și $M = M_R \text{ mod } R$. La fel dorim să avem calculat și $P_{inv_Q} = P^{-1} \text{ mod } Q$ și $PQ_{inv_R} = (PQ)^{-1} \text{ mod } R$. M verifică:

$$M = M_{PQ} + P * Q(M_R - M_{PQ}) * (PQ_{inv_R}) \text{ mod } N$$

Știind faptul că $P * Q * (PQ_{inv_R}) = 1 \text{ mod } R$ obținem:

$$M = M_{PQ} + (P * Q * (PQ_{inv_R})) * (M_R - M_{PQ}) \text{ mod } R \Rightarrow$$

$$M = M_{PQ} + M_R - M_{PQ} \text{ mod } R$$

în final obținem $M = M_R \text{ mod } R$. La fel vom proceda pentru această ecuație $\text{mod } Q$ și $\text{mod } P$ și vom obține:

$$M \equiv M_Q \text{ mod } Q$$

$$M \equiv M_P \text{ mod } P$$

2.4.2 Multi-Prime RSA (modulo $N = PQR$)

2.4.2.1 Descrierea criptosistemului

1. Generarea cheilor:

Generarea cheilor este aceeași ca și la RSA clasic, diferă doar faptul că sunt trei numere prime în loc de două. Avem ca input, parametrul de securitate n , alegem 3 numere prime P , Q și R care au lungimea în biți $n/3$ și le înmulțim pentru a obține modulul $N = PQR$. Apoi alegem un număr întreg e astfel încât $\phi(N) = (P-1)(Q-1)(R-1)$ și e să fie relativ prime. În final, calculăm d astfel încât $ed \equiv 1 \text{ mod } \phi(N)$ și $d_P = d \text{ mod } (P-1)$, $d_Q = d \text{ mod } (Q-1)$ și $d_R = d \text{ mod } (R-1)$. De asemenea calculăm și $PQ_{inv_R} = (PQ)^{-1} \text{ mod } R$ și $P_{inv_Q} = P^{-1} \text{ mod } Q$. Cele două numere întregi $\langle N, e \rangle$ sunt cheile publice și $\langle P, Q, R, d_P, d_Q, d_R, PQ_{inv_R}, P_{inv_Q} \rangle$ sunt cheile private.

2. Criptarea:

Partea de criptare este la fel ca și la RSA clasic.

3. Decriptarea:

Principiul este cam același: Se folosește teorema chineză a resturilor doar că în loc de 2 numere prime vom avea 3. Această decriptare este mai rapidă, deoarece lungimea în biți ale numerelor întregi este mai scurtă. În loc de o singură exponențiere vom avea 3 exponențieri de lungime de 3 ori mai mică. Teoretic, viteza acestui algoritm este de 9 ori mai rapidă (când N este pe 2048 de biți) decât RSA multi-prime care nu folosește teorema chineză a resturilor.

2.4.3 Generalizarea algoritmului Garner

Deși, utilizarea a mai mult de 3 numere prime reprezintă un incident de securitate, lungimea unei chei RSA ar putea deveni mai mare în viitor, și ne va permite să folosim și o formă mai generală având b numere prime, în acest caz N va fi : $N = \prod_{i=1}^b P_i$.

2.4.3.1 Descrierea algoritmului în forma generală**1. Generarea cheilor:**

Generarea cheilor este aceeași ca și până acum doar că vom avea un număr b de numere prime în loc de 2 sau 3: avem ca input un parametru de securitate n , alegem b numere prime care au lungimea n/b biți și le înmulțim pentru a obține N . Alegem e astfel încât $\phi(N) = \prod_{i=1}^b (P_i - 1)$ și e sunt relativ prime. În final calculăm d astfel încât $ed = 1 \bmod \phi(N)$ și $d_i = d \bmod (P_i - 1)$. La fel vom calcula și acei inversi (de data aceasta îi vom nota cu A_2, \dots, A_b). Vom avea parametri publici $\langle N, e \rangle$ și cei secreți $\langle P_1, \dots, P_b, d_1, \dots, d_b, A_2, \dots, A_b \rangle$.

2. Criptarea:

Criptarea este la fel ca și cea clasică

3. Decriptarea:

Principiul este același, se va folosi teorema Chineză a resturilor, unde $N = \prod_{i=1}^b P_i$. Va trebui să facem b exponențieri de lungime n/b . Este posibil să avem o viteză de b^2 ori mai rapidă decât RSA fără teorema Chineză a resturilor.

2.5 Multi-Power RSA

Această variantă cu factor multiplu RSA, folosește un modul $N = P^b Q$ [25]. Dacă lungimea în biți a lui N este de 1024 de biți, atunci vom folosi $b = 2$, deci $N = P^2 Q$

2.5.1 Hensel lifting

2.5.1.1 Ideea de baza la Hensel lifting

Folosind un modul $N = P^2 Q$ folosind în mod clasic teorema Chineză a resturilor, făcând o decriptare RSA ar fi mai încheată decât dacă am folosi un modul $N = PQ$. Când calculăm $\mathbb{Z}/P^2\mathbb{Z}$, ajungem să avem lungimi de $2n/3$ biți, pe când în \mathbb{Z}_{P^2} avem lungimi de $n/3$. Așadar, va trebui să-l scriem pe P^2 sub o altă formă.

2.5.1.2 Justificarea matematică

Vom încerca să scriem reprezentarea plaintext M_P modulo P^2

$$M_P = (K_0 + P * K_1) \bmod P^2$$

Dacă vom calcula $C_P = M_P^e \bmod P^2$ obținem:

$$\begin{aligned} C_P &= (K_0 + P * K_1)^e \bmod P^2 \\ &= \sum_{i=0}^e \binom{e}{i} \cdot K_0^i \cdot P^{e-i} \cdot K_1^{e-i} \bmod P^2 \end{aligned}$$

Expresia de mai sus se va reduce la :

$$C_P = K_0^e + e * P * K_0^{e-1} \cdot K_1 \bmod P^2$$

Putem calcula foarte ușor pe K_0 :

$$K_0 = C_P^{d_P} \bmod P$$

Fie $E = (C_P - K_0^e) \bmod P^2$ de unde rezultă:

$$K_1 = \frac{E}{P} (e * K_0^{e-1} \bmod P)^{-1} \bmod P$$

Acum putem recupera mesajul $M_P = (K_0 + P \cdot K_1) \bmod P^2$

Algorithm 8 Lema Hensel**Input:** C_P, d_P, P **Output:** M_P

$$P^2 \leftarrow P \cdot P$$

$$K_0 \leftarrow C_P^{d_P} \bmod P$$

$$A \leftarrow K_0^e \bmod P^2$$

$$A \leftarrow A + C \bmod P^2$$

$$A \leftarrow \frac{A}{P}$$

$$K_1 \leftarrow K_0^{e-1} \bmod P$$

$$K_1 \leftarrow (K_1)^{-1} \bmod P$$

$$K_1 \leftarrow (K_1 \cdot A) \bmod P$$

$$M_P \leftarrow P \cdot K_1 \bmod P^2$$

$$M_P \leftarrow M_P + K_0 \bmod P^2$$

return M_P **2.5.2 Multi-Power RSA modulo $N = P^2Q$** **2.5.2.1 Algoritmul****1. Generarea cheilor:**

Avem ca input parametru de securitate n , alegem două numere prime P și Q , care au lungimea $n/3$ și calculăm modulul $N = P^2Q$. Apoi alegem e ca și în partea standard RSA și calculăm d astfel încât $ed = 1 \bmod (P^2 - P)(Q - 1)$. În final, calculăm $d_P = d \bmod (P - 1)$ și $d_Q = d \bmod (Q - 1)$. De asemenea calculăm și $P_{inv_Q}^2$ și e_{inv_P} . Cele două numere întregi $< N, e >$ vor reprezenta cheia publică, iar $< d_P, d_Q, P, Q, P_{inv_Q}^2, e_{inv_P} >$ reprezintă cheia secretă.

2. Criptarea:

Criptare clasică RSA

3. Decriptarea:

La decriptare vom calcula $M_P = C_P^{d_P} \bmod P^2$, unde $C_P = C' \bmod P^2$ și $M_P = M \bmod P^2$ folosind algoritmul Hensel. Ne permitem să calculăm exponentul unui număr de lungime $2n/3$ biți la costul unui număr de lungime $n/3$ biți. În final putem recupera plaintextul inițial folosind teorema chineză a resturilor. Această metodă are o viteză de circa 13.5 ori mare.

2.5.3 Multi-Power RSA modulo $N = P^b Q$

2.5.3.1 Algoritmul

2.5.4 Generalizarea ideii

În acest moment suntem limitați să avem $b = 2$ unde $N = P^b Q$, deoarece lungimea maximă în acest moment este de 2048 de biți.

2.5.4.1 Ideea de bază

Având $M \bmod P$, putem recupera plaintextul inițial $M \bmod P^2$ datorită algoritmului Hensel. Totuși, dacă executăm acest algoritm într-o buclă vom putea recupera mesajul chiar și cu o putere mai mare decât 2. După fiecare buclă, vom obține $M \bmod P^i$, început cu i de la 1 până la b . La final, obținem $M \bmod P^b$. Vom avea $M_{P,i}$, plaintextul modulo P^i . Vom arăta ca se poate recupera plaintextul folosind modulo P^{i+1} .

$$M_{P,i+1} = (K_0 + K_1 \cdot P^i) \bmod P^{i+1}$$

Putem scrie criptotextul $C_{P,i+1}$ modulo P^{i+1} în următorul fel:

$$\begin{aligned} C_{P,i+1} &= M_{P,i+1}^e \bmod P^{i+1} \\ &= (K_0 + K_1 \cdot P^i)^e \bmod P^{i+1} \end{aligned}$$

După ce vom desface acea paranteză aplicând Binomul lui Newton, ecuația se va reduce la:

$$C_{P,i+1} = K_0^e + e * K_0^{e-1} * P^i * K_1 \bmod P^{i+1}$$

Mai departe, vom înlocui pe K_0 cu $M_{P,i}$, și vom aduce primul termen $M_{P,i}^e$ în partea stângă. Vom obține:

$$C_{P,i+1}^e - M_{P,i}^e = e * M_{P,i}^{e-1} \cdot P^i \cdot K_1 + k * P^{i+1}$$

Ca și în varianta precedentă unde aveam $b = 2$, vom înlocui pe $C_{P,i+1}^e - M_{P,i}^e$ cu E . Prin urmare, K_1 va fi egal cu:

$$K_1 = \frac{E}{P^i} \cdot (e * M_{P,i}^{e-1})^{-1} \bmod P$$

În final se obține:

$$M_{P,i+1} = (M_{P,i} + K_1 \cdot P^i) \bmod P^{i+1}$$

2.5.4.2 Descrierea algoritmului în forma de bază

1. Generarea cheilor:

Vom avea ca parametru de securitate n , vom alege cele două numere prime P și Q , care au lungimea de $n/(b+1)$ biți și calculăm $N = P^b Q$. Apoi îl alegem pe e în modul standard RSA, după care îl calculăm pe d astfel încât $ed = 1 \bmod (P^2 - P)(Q - 1)$. În final, vom calcula $d_P = d \bmod (P - 1)$ și $d = q \bmod (Q - 1)$. De asemenea vom calcula $P_{inv_Q}^2$ și e_{inv_P} . Cele două numere întregi $\langle N, e \rangle$ sunt cheile publice, iar $d_P, d_Q, P, Q, (P_{inv_Q}^2, e_{inv_P})$ sunt cheile secrete.

2. Criptarea: Criptarea se face în modul clasic RSA

3. Decriptarea:

Vom calcula $M_P = C_P^d \bmod P^b$ unde $C_P = C \bmod P^b$ și $M_P = M \bmod P^b$ folosind algoritmul Hensel. În final, vom recupera plaintextul inițial folosind teorema chineză a resturilor. De asemenea calculăm $M_Q = C_Q^d \bmod Q$ și obținem plaintextul folosind algoritmul Garner.

2.5.5 Schema lui Takagi

Schema lui Takagi este o variantă de multi-power RSA [31]. Această variantă folosește on modul $N = P^b - 1Q$, pentru $b \geq 3$ și definește exponentul public și cel privat ca fiind invers modularul lui $\lambda'(N) = \text{cmmdc}(P - 1, Q - 1)$. Deoarece $\lambda'(N)$ nu este un multiplu a lui $\lambda(N)$ unde ($ed = 1 + k\lambda(N)$), înseamnă că decriptarea standard nu va funcționa, deci $m^{ed} \not\equiv m \bmod N$. Pentru a putea decripta un criptotext, trebuie să decriptăm parțial modulo P^{b-1} și modulo Q apoi el trebuie combinate folosind teorema Chineză a resturilor. Totuși, pentru a putea calcula decriptarea parțială $m_{P^{b-1}}$ trebuie să folosim și lema lui Hensel.

Pentru numere aleator balansate P și Q , modulul schemei Takagi este dat de $N = P^{b-1}Q$, pentru $b \geq 3$. Un exponent public mic e este ales astfel încât $\text{cmmdc}(e, P) = 1$ și exponentul privat este definit ca și inversul modular modulo $\lambda'(N)$. Așadar, exponentul privat d este ales astfel încât $ed \equiv 1 + k\lambda'(N)$. Odată ce d este determinat, exponenții teoremei Chineze

$$d_P = d \bmod (P - 1)$$

$$d_Q = d \bmod (Q - 1)$$

sunt calculați. Cheia publică este $\langle N, e \rangle$ și cheia privată este $\langle d_P, d_Q, P, Q, b \rangle$.

Criptarea în schema lui Takagi este aceeași ca și pentru RSA. Fiind dat un plaintext $m \in \mathbb{Z}_N$, criptotextul

$c = m^d \bmod N$. Menționat ca și mai sus, decriptia standar nu poate fi folosită deoarece $\lambda'(N)$ nu este un multiplu a lui $\lambda(N)$. Decriptarea în schema lui Takagi constă în decriptarea parțială

$$m_P = c^{d_P} \bmod P = m \bmod P$$

$$m_Q = c^{d_Q} \bmod Q = m \bmod Q$$

Atunci când $N = PQ$ aceste decriptări parțiale pot fi combinate folosind doar teorema Chineză a resturilor pentru a recupera plaintextul modulo N . În schema lui Takagi, decriptarea parțială m_P are nevoie de lema lui Hensel pentru a putea decripta modulo P^{b-1} . Folosind $b - 2$ iterații a lemei lui Hensel, decriptarea parțială modulo P poate fi transformată într-o decriptare parțială modulo P^{b-1} . Decriptarea parțială modulo Q și modulo P^{b-1} sunt combinate folosind teorema Chineză a resturilor pentru a recupera un plaintext modulo $N = P^{b-1}Q$.

2.5.5.1 Eficiența schemei lui Takagi

Schema lui Takagi este mai eficientă decât RSA atât la generarea cheilor cât și la decriptare. Pentru o anumită dimensiune, în schema lui Takagi este necesară doar generarea a două numere prime de dimensiune $N^{1/b}$ în loc de $N^{1/2}$ ca la RSA clasic. Chiar dacă generarea numerelor prime are timp liniar, dimensiunea redusă a acestora va îmbunătăți cu siguranță viteza de generare. Condiția extra pentru exponentul public $\text{cmmdc}(e, P) = 1$ nu reprezintă o problemă dacă e este suficient de mic. În particular, folosind un exponent public mult mai mic decât $N^{1/b}$ ne va asigura de fiecare dată faptul că $\text{cmmdc}(e, P) = 1$.

Am precizat mai sus că decriptarea parțială modulo P poate fi transformată într-o decriptare modulo P^{b-1} . O versiune completă și optimizată a algoritmului de decriptare a lui Takagi poate fi găsit în [31]. Algoritmul de decriptare va calcula decriptările parțiale m_P și m_Q , folosește algoritmul de decriptare a lui Takagi pentru a calcula $m_{P^{b-1}}$ și îl combină cu m_Q și folosește algoritmul lui Garner pentru rezolvarea sistemului.

Complexitatea algoritmului următor va depinde de exponentul public

Algorithm 9 Lema Hensel în schema lui Takagi

Input: $\langle c, e, d_P, p, b \rangle$
Output: Decriptarea parțială a lui m_{pb-1}

$$m_{p'} \leftarrow c^{d_P-1} \bmod P$$

$$m_P \leftarrow c^{d_P} \bmod P$$

for i from 1 to $b-1$ **do**

$$x \leftarrow (c - m_e^{p^i}) \bmod P^{i+1}$$

$$y \leftarrow x m_{p'} (e^{-1} \bmod P) \bmod P^{i+1}$$

$$m_{pi+1} \leftarrow m_{pi} + y$$

end for

2.6 RSA pe grupuri

Ideea este destul de schimbată pentru această variantă. Aceasta constă în gruparea unor cripto-texte, sub anumite condiții [16].

2.6.1 Ideea de bază

2.6.1.1 Notăție

De data aceasta, vom folosi echivalentul notației $M = C^d \bmod N$ care este $M = C^{e^{-1}} \bmod \phi(N)$ deoarece e este inversul modularul lui d modulo $\phi(N)$.

2.6.1.2 Formule de bază pentru decriptare

Spunem că avem k numărul de plaintexte, respectiv numărul de exponenți e_i care același modul N . Ne vom folosi de următoarele:

$$e = \prod_{i=1}^k e_i$$

$$A = \left(\left(\prod_{i=1}^k C_i^{e/e_i} \right) \bmod N \right)^{1/e} \bmod N$$

Fiecare plaintext M_i va fi aflat cu ajutorul următoarei formule:

$$M_i = \frac{A^{a_i}}{C_i^{(a_i-1)/e_i} \cdot \prod_{j=1, j \neq i}^p C_j^{a_i/e_j}} \bmod N$$

Unde a_i are următoarea proprietate:

$$a_i \equiv 1 \pmod{e}, a_i \equiv 0 \pmod{e_j}$$

2.6.1.3 Decriptarea a două texte

Vom avea două mesaje M_1 și M_2 , care au cheile publice $\langle N, 3 \rangle$ și $\langle N, 5 \rangle$. Așadar, vom avea criptotextele $C_1 = M_1^3 \pmod{N}$ și $C_2 = M_2^5 \pmod{N}$. Pentru aflarea plaintextelor, folosind formula prezentată anterior, vom obține:

$$\begin{aligned} A &= (C_1^5 \cdot C_2^3)^{1/15} \\ \frac{A^{10}}{C_1^3 \cdot C_2^2} &= \frac{C_1^{10/3} \cdot C_2^2}{C_1^3 \cdot C_2^2} = M_1 \\ \frac{A^6}{C_1^2 \cdot C_2} &= \frac{C_1^2 \cdot C_2^{6/5}}{C_1^2 \cdot C_2} = M_2 \end{aligned}$$

Costul acestor două decriptări fiind aproape la fel ca și o decriptare clasică RSA [15]

2.6.2 Optimizări

2.6.2.1 Teorema Chineză a resturilor

Ca și în alte variante de RSA, putem folosi teorema Chineză a resturilor pentru a reduce timpul.

2.6.2.2 Calculul unui singur invers modular

Când îl calculăm pe M_i , de asemenea noi vom calcula și o serie de inverși. În loc să calculăm p inverși independent, cea mai bună soluție ar fi să aducem acele frații la același numitor, și să calculăm un singur invers.

2.6.3 Îmbunătățirea batch RSA: Tehnica lui Montgomery

2.6.3.1 Descrierea algoritmului

Acest algoritm va avea ca și input numerele întregi și modulul N , și calculează produsul întregilor inversați:

Algorithm 10 Tehnica lui Montgomery

Input: A_1, \dots, A_p, N **Output:** $Z_1 = A_1^{-1}, \dots, Z_p = A_p^{-1} \bmod N$ $X_1 \leftarrow A_1$ **for** i from 2 to p **do** $X_i \leftarrow X_{i-1} * A_i \bmod N$ **end for** $Z_1 \leftarrow X_p^{-1} \bmod N$ **for** i from p down to 2 **do** $Z_i \leftarrow X_{i-1} \cdot Z_1 \bmod N$ $Z_1 \leftarrow Z_1 \cdot A_i \bmod N$ **end for****return** (Z_1, \dots, Z_p)

2.6.3.2 Corectitudinea algoritmului

Valorile X_i calculate la primul for sunt produsul valorilor A_j de la 1 la i :

$$X_i = \prod_{j=1}^i A_j \bmod N$$

După care Z_1 va reprezenta produsul inversilor, adică:

$$Z_1 = \prod_{j=1}^i A_j^{-1} \bmod N$$

Fiecare Z_i va fi înmulțit cu $X_{i-1} \cdot Z_1$ iar Z_1 va fi înmulțit cu A_i la fiecare pas i . În final vom obține:

$$Z_i = A_i^{-1} \bmod N$$

2.6.4 Îmbunătățirea RSA folosind tehnica lui Shamir**2.6.4.1 Descrierea algoritmului**

În acest algoritm vom primi ca și input M_1, M_2, \dots, M_p , p exponenți e_1, \dots, e_p și modulul N . Reprezentarea binară a fiecăruia este cunoscută: $e_i = (e_{i,k-1}, \dots, e_{i,0})_2$. Calculăm $C = M_1^{e_1} \cdot \dots \cdot M_p^{e_p}$. Acest calcul este mult mai rapid decât dacă am face exponențieri consecutive.

2.6.4.2 Explicație

La primul pas, vom calcula toate produsele posibile pentru M_i . Vom forma o bijecție b , definită prin:

$$b((i_{p-1} \dots i_0)_2) = \prod_{j=0}^{p-1} M_j^{i_j} \bmod N$$

unde $i_j \in \{0, 1\}$ La pasul de evaluare, vom proceda la fel ca și în algoritmul left-to-right (metoda de exponențiere), dar în loc să înmulțim cu o singură bază M , va trebui să alegem o bază corectă, această putând fi diferită de M . Va trebui să calculăm:

$$C = \prod_{i=1}^k M_i^{e_i} \bmod N$$

Putem folosi reprezentarea binară a lui e_i :

$$\begin{aligned} C &= \prod_{i=1}^k M_i^{\sum_{j=0}^{p-1} e_{i,j} \cdot 2^j} \bmod N \Rightarrow \\ C &= \prod_{i=1}^k \prod_{j=0}^{p-1} M_i^{e_{i,j} 2^j} \bmod N \end{aligned}$$

Putem să interschimbăm cele două produse și să introducem primul termen într-o paranteză (acest termen din paranteză va reprezenta baza):

$$C = \prod_{j=0}^{p-1} \left(\prod_{i=1}^k M_i^{e_{i,j}} \right)^{2^j} \bmod N$$

Putem face și notația $E = (e_{1,j}, \dots, e_{k,j})$ și vom obține folosind bijecția b :

$$C = \prod_{j=0}^{p-1} b(E)^{2^j} \bmod N$$

2.7 RSA varianta distribuită (Schema lui Shoup)

2.7.1 Introducere

Varianta partajată a RSA [14](schemă prin semnături) prezintă câteva aspecte foarte interesante. În primul rând, este securizată și robustă deoarece problema criptanalizei este foarte grea. Apoi, partajarea, generarea semnăturii și verificarea sunt non interactive și în final, dimensiunea unei semnături partajate este legată de o constantă înmulțită cu dimensiunea modulului RSA.

Când un mesaj trebuie să fie semnat de măcar $t + 1$ server unde $2t + 1 \leq l$, un server special numit combinator, direcționează mesajul m sau x , unde $x = H(x)$ către toate serverele, apoi fiecare server calculează semnătura cu o dovadă de corectitudine, apoi combinatorul selectează un subgrup de $t + 1$ servere și le combină.

Folosind distributivitatea aditivă, $d = \sum_{i=1}^l d_i \bmod \phi(N)$, combinatorul poate alcătui ușor semnăturile S , $s_i = x^{d_i} \bmod N$, unde $x = H(m)$, este mesajul semnat folosind formula:

$$S = \prod_{i=1}^l S_i = \prod_{i=1}^l x^{d_i} = x^{\sum_{i=1}^l d_i} = x^d \bmod N$$

Principalul dezavantaj al acestor tehnici îl reprezintă mărimea cheilor, din cauza necesității diferitelor partajări și a tehnicilor de reconstruire a semnăturilor. Ulterior, Shoup a propus prima schemă bazată pe partajarea polinomială (schema Desmedt și Frankel [2]). Protocolul trebuie să revină de t ori pentru a putea înlătura serverele rele, semnătura depinzând de un subgrup de $t + 1$ servere.

Fie $\delta = l!$. Partajarea lui d este în așa fel încât $\delta | d_i$ și $d_i = f(i)$ unde f este un polinom de grad t și cu termenul constant d .

Fie S mulțimea a $t + 1$ servere, atunci coeficientul Lagrange va fi:

$$\lambda'_{i,j} = \prod_{j' \in S \setminus j} \frac{i - j'}{j - j'}$$

De unde rezultă:

$$d = \sum_{i \in S} \lambda'_{0,i} d_i \bmod \phi(N)$$

O problemă majoră o reprezintă faptul că $\lambda'_{i,j}$ nu poate fi calculat în $\mathbb{Z}_{\phi(N)}$ deoarece $\phi(N)$ poate fi par. Apoi combinatorul trebuie să calculeze:

$$\begin{aligned} s &= \prod_{i \in S} S_i^{\lambda'_{0,i} d_i} \\ &= S^{\sum_{i \in S} \lambda'_{0,i} d_i} \\ &= S^d \bmod N \end{aligned}$$

. Cum numerele din mulțimea $\lambda'_{i,j}$ nu sunt numere întregi, combinatorul nu poate calcula rădăcina modulo a unui număr compus. Ideea esențială este să spunem că $\delta \times \lambda'_{i,j}$ sunt numere întregi. Atunci putem spune că

$$l_{0,i}^S = \delta \times \lambda'_{i,j}^S, \quad i \in \mathbb{Z}$$

Și:

$$S' = x^{l_{0,i}^S \times d_i} \bmod N$$

Combinatorul va calcula:

$$\begin{aligned} S &= \prod_{i \in S} S'_i \\ &= \prod_{i \in S} x^{l_{0,i}^S \times d_i} \\ &= \prod_{i \in S} x^{\delta \times \lambda'_{i,j} \times d_i} \\ &= \prod_{i \in S} x^{\lambda'_{0,j} d_i} \\ &= x^d \bmod N \end{aligned}$$

Dacă semnătura nu este bună, atunci combinatorul înlătură serverele rele. După t iterații, toate serverele rele sunt înlăturate din setul S iar semnăturile rămase vor fi corecte. Totuși, nici această redefinire a subgrupului nu părea să fie foarte bună, și Shoup a propus o nouă tehnică.

Problema a fost rezolvată folosind o leamă cunoscută pentru a extrage e -rădăcina lui w modulo un număr compus dintr-o e -rădăcină a unei puteri w cunoscute fără a folosi secrete. Soluția a fost să multiplicăm coeficienții Lagrange cu δ ca aceștia să devină întregi.

$$\lambda'_{i,j} = \delta \times \lambda_{i,j}^S$$

Și:

$$\delta d = \sum_{i \in S} \lambda_{0,i}^S d_i$$

Fie $s_i = x^{d_i}$, combinatorul care calculează semnătura și schimbarea grupului (folosind noua formulă Lagrange pentru cele $t + 1$ servere). Se calculează ecuația folosind $\lambda_{i,j}^S$ și se obține:

$$S^\delta = \prod_{i \in S} S_i^{\lambda_{0,i}^S = x^{\delta d} \bmod N}$$

În final, combinatorul calculează rădăcina lui x^d .

2.7.2 Dovada robusteții și folosirea numerelor prime sigure

Robustețea înseamnă faptul că serverele corupte nu le pot împiedica pe cele necorupte să-și realizeze semnătura.

În dovada corectitudinii serverele trebuie să dovedească că ridică x la puterea d_i . Serverul i deține cheia de verificare

$$v_i = \log_x S_i = d_i \bmod \phi(N)$$

Problemele sunt: \mathbb{Z}_N^* nu este un grup ciclic, generatorul V nu există și elementele de ordin maximal sunt greu de găsit. Dacă vom folosi numere prime sigure, atunci \mathbb{Z}_N^* este un grup ciclic, iar generatoarele sunt ușor de găsit.

Pentru a nu crea neclarități, este bine să descriem totuși ce este acela un grup, mai precis un grup ciclic:

2.7.2.1 Grupuri [23]

Un grup este o structură algebrică ce constă dintr-o mulțime pe care este definită o lege de compoziție internă (acesta combină două elemente ale unei mulțimi și formează un al treilea element tot din aceeași mulțime). Pentru a fi un grup, mulțimea și operația trebuie să satisfacă o serie de axiome:

1. Axioma închiderii:

$$\forall x, y \in G, \text{ și rezultatul operației } x \circ y \in G$$

2. Axioma asociativității:

$$\forall x, y, z \in G, (x \circ y) \circ z = x \circ (y \circ z)$$

3. Axioma elementului neutru:

$$\exists e \in G, \text{ astfel încât } e \circ x = x \circ e = x, \forall x \in G$$

4. Axioma elementelor simetrice:

$$\forall x \in G, \exists y \in G \text{ cu proprietatea că } x \circ y = y \circ x = e$$

5. Axioma comutativității: $\forall x, y \in G, x \circ y = y \circ x$

atunci grupul (G, \circ) se numește grup comutativ sau abelian

Un grup ciclic este un grup ale cărui elemente sunt puteri (când operația de grup este considerată a fi de natură aditivă, se preferă termenul de multipli) ai unui element a .

2.7.3 Problema

Numerele prime sigure sunt folosite în general de chei pentru a dovedi faptul că schema de partajare a secretelor (Shamir [18]) este sigură în inelul \mathbb{Z}_M , și nu într-un corp finit. Robustetea ne garantează faptul că dacă un jucător malițios trimite semnături false, schema tot va genera corect S .

Spre exemplu: combinatorul primește de la servere și trebuie să genereze semnăturile corecte. O cale de a face acest lucru este să alegem aleator $t + 1$ semnături, să generăm posibilele semnături S' și să testăm dacă S' este valid. Dacă S' trece de protocolul de verificare, atunci semnătura corectă a fost găsită, dacă nu, combinatorul trebuie să testeze alte $t + 1$ semnături. Combinatorul nu știe care sunt semnăturile false iar din această cauză, acesta poate ajunge la un număr exponențial de încercări. Așadar, este necesar să divizăm într-un mod eficient.

2.7.4 Modele de securitate

2.7.4.1 Rețeaua

Grupul de l servere conectate la un mediu prin broadcast, iar mesajele trimise către canalul de comunicare ajung la fiecare entitate.

2.7.4.2 Adversarul

Adversarii pot corupe servere în orice moment prin verificarea memoriei serverelor corupte(adversar pasiv) și/sau pot modifica comportamentul acestora(adversari activi). Adversarii decid pe cine să corupă la startul protocolului (adversari statici). Putem asuma și ca adversarii nu vor corupe mai mult de t servere din l , unde $l \geq 2t + 1$.

2.7.4.3 Definiții formale

O schemă RSA de semnături folosește următoarele componente:

1. Un algoritm de generare de chei care primește ca input parametrii de securitate N , numărul k de elemente care trebuie să genereze Q_N , numărul de l servere care trebuie să semneze, parametrul t și un string random w . Are ca și output o cheie publică $\langle N, e \rangle$, cu cheile private d_1, \dots, d_l cunoscute doar de servere corecte pentru fiecare $u \in [1, k]$ o listă $v_u^{d_1}, \dots, v_{u,l}^{d_l} = v_u^{d_l} \bmod N$ verificări de chei.

2. Un algoritm de partajare de semnături primește ca input o cheie publică $\langle N, e \rangle$, un index $1 \leq i \leq l$, cheia privată d_i , și un mesaj m ; are ca output o semnătură $s_i = x^{d_i} \bmod N$ unde $x = H(m)$ și $H()$ este o funcție de hash și padare, și dovada validității este:

$$\text{validity proof}_i \text{ (for all } u \in [1, k], \log_{v_u} v_{u,i} = \log_x S_i)$$

3. Un algoritm combinat de verificare primește ca input o cheie publică $\langle N, e \rangle$, un mesaj m , o listă s_1, \dots, s_l de semnături partajate, pentru fiecare $u \in [1, k]$ lista $v_u, v_{u,1}, \dots, v_{u,l}$ de chei de verificare și o listă $\text{proof}_1, \dots, \text{proof}_l$ de validări, și ca output o semnătură sau fail.
4. Un algoritm de verificare primește ca input o cheie publică $\langle N, e \rangle$, un mesaj m , o semnătură S și are ca output un bit b indicând dacă semnătura este corectă sau nu.

2.7.4.4 Jucătorul și adversarul

Jocul nostru include următorii jucători: un combinator, un set l de servere P_i , un adversar, userii care cer semnături.

Considerăm următorul scenariu:

1. La faza de inițializare, serverele folosesc algoritmul distribuit de generare de chei pentru crearea cheilor private, publice și a cheilor de verificare. Cheia publică $\langle N, e \rangle$ și cheile de verificare $v'_u S$ și $v'_{u,i} S$ sunt publicate și fiecare server obține partajarea d_i a cheii secrete d .
2. Pentru a semna un mesaj m , combinatorul trimite mai departe m către servere. Folosind cheia privată d_i și cheile de verificare $v_u, v_{u,1}$ pentru $u \in [1, k]$ pentru fiecare server rulează algoritmul de partajare de semnături și are ca output o semnătură partajată S_i împreună cu o dovadă de validitate și o dovadă de validitate proof_i . În final, combinatorul utilizează algoritmul de combinare de generare de semnături, și scoate câte semnături partajate sunt valabile și valide.

2.7.4.5 Proprietăți

După cum am mai menționat, robustețea garantează faptul că dacă avem un număr t de jucători malițioși care trimit semnături false, schema tot va returna o semnătură corectă. Această proprietate este folosită doar în prezența adversarilor activi.

Proprietatea care asigură faptul că nu poate fi falsificat , spune că pentru orice subset de $t + 1$ jucători pot genera o semnătură S , dar nu permite generarea a mai puțin de t jucători. Această proprietate exprimă securitatea unei scheme de securitate și este folositoare în prezența atacurilor adversarilor pasivi.

2.7.4.6 Generarea cheii și protocolul de semnare

În această secțiune, vom descrie noțiunile de securitate pentru generarea de chei și protocolul pentru semnături. Am arătat că informațiile arătate în timpul generărilor de chei și semnării nu eliberează informații secrete adversarilor.

- Jocul pentru generarea de chei:

Corectitudinea pentru generarea cheilor necesită ca probabilitatea cheilor secrete d, P, Q și cheilor publice $\langle N, e \rangle$ să fie uniform distribuite către adversari

Dacă \exists un adversar A care corupe cel mult t servere la începutul jocului, atunci el nu poate obține informația ținută de jucătorii necorupți

- Jocul pentru protocolul de semnare:

Secretul semnării înseamnă că dacă \exists un adversar A care corupe cel mult t servere la începutul jocului chiar dacă el poate obține informații, el nu poate forța o semnătură pe un nou mesaj.

2.7.4.7 Metoda distribuită de generare de chei în protocolul lui Shoup

De îndată ce N este generat, fie e primul număr mai mare decât $4\delta^2$ în așa fel încât serverul îl poate calcula. Apoi, protocolul Catalano [8] este rulat pentru a genera cheia partajată într-o manieră distribuită. La finalul protocolului, fiecare server calculează cheia ei de verificare în felul următor:

$$v_{u,i} = v_u^{\delta d_i}$$

Pentru v_u aleator calculat ca și $y_u^2 \bmod N$ unde y_u e concatenarea lui $H(N|i)$ pentru suficient de mulți i pentru a colecta corect parametri de securitate.

2.7.5 Concluzia asupra schemei lui Shoup

Am arătat cum trebuie să evităm numerele prime sigure în schema lui Shoup în așa fel încât dovada să rămână corectă. Am considerat că mediile în care am lucrat sunt cu grad de securitate

ridicat(ex scheme de vot electronic).

Pe scurt, folosim diferite tehnici pentru a dovedi că:

1. Grupul pătratelor este ciclic.
2. Generăm P și Q astfel încât P' și Q' să nu conțină factori primi mici
3. Generăm un set de generatori Q_N generând aleator diferiți generatori Q_N

Capitolul 3

Criptanaliză RSA

3.1 Criptanaliză pe RSA clasic [5]

Deși securitatea algoritmului RSA constă în legătura dintre aceasta și factorizarea întregilor, el trebuie folosit cu grijă în implementări, deoarece, în caz de folosire eronată, sistemele bazate pe RSA pot fi atacate în anumite maniere care ocolesc factorizarea efectivă a modului, atacatorul ajungând să obțină mesajul clar sau cheia secretă.

3.1.1 Atac cu criptotext ales

În cazul acestui atac, adversarul dispune de cheia publică a entității atacate (exponentul de criptare e și modulul N), și interceptează mesaje criptate trimise acestuia. Pentru a obține mesajul clar m dintr-un mesaj criptat c atacatorul poate proceda în felul următor:

1. Calculează $x = (c \cdot 2^e) \bmod N$
2. Trimite entității atacate spre semnare pe x , obținând $y = x^d \bmod N$
3. Se observă că:

$$\begin{aligned}x &= c \cdot 2^e \bmod N \\&= m^e \cdot 2^e \bmod N \\&= (2m)^e \bmod N\end{aligned}$$

4. Se rezolvă ecuația $y = 2m \bmod N$

Atacatorul obține astfel mesajul criptat. Există mai multe feluri de tipul atac cu criptotext ales, dar sunt câteva moduri de apărare împotriva lor. Unele pot fi evitate dacă pur și simplu entitatea protejată cu chei secrete refuză să semneze texte arbitrare trimise de terți. Dacă acest lucru nu este posibil (ca de exemplu în cazul unui notar public care trebuie să semneze documente electronice prezentate de persoane străine), atunci atacul poate fi prevenit prin folosirea unei perechi de diferite chei pentru criptare și pentru semnătura electronică. De asemenea, este necesar să se folosească și un padding aleator pentru mesaj înainte de criptare sau, în cazul semnăturii, să nu semneze mesajul clar, ci un has al acestuia. De asemenea, atacul poate fi evitat și dacă se impune o anumită structură predefinită a mesajelor primite spre semnare.

3.1.2 Mesaje necriptate

Întrucât RSA se bazează pe ridicarea la putere (modulo un număr N), există părți de mesaje care nu sunt criptate, părți rezultate în urma împărțirii mesajului pe blocuri. Astfel de mesaje sunt mesajele m cu proprietatea că $m = m^x \bmod N$ oricare ar fi x , ca de exemplu $m = 0, m = 1, m = N - 1$. Numărul exact al acestor mesaje decriptate este $(1 + \gcd(e - 1, P - 1))((1 + \gcd(e - 1, Q - 1))$, și deși este minim 0 (deoarece e, p și q sunt impare). Pentru a micșora numărul de astfel de părți de mesaj, este util să se folosească un exponent public e cât mai mic.

3.1.3 Exponentul de criptare mic

În unele aplicații, se folosește un exponent de criptare e mic, de exemplu $e = 17$, pentru a mări performanța, dar și pentru a rezolva unele probleme de securitate. Dacă mai multe entități care comunică folosesc același exponent public (dar fiecare are propriul modul și deci propria cheie secretă), atunci același mesaj trimis mai multor destinatari are următoarele valori:

$$\begin{cases} c_1 = m^e \bmod N_1 \\ c_2 = m^e \bmod N_2 \\ c_3 = m^e \bmod N_3 \end{cases}$$

Unde N_i sunt modulele elor trei destinatari, e este exponentul comun acestora iar m este mesajul trimis celor trei. Un atacator poate folosi algoritmul lui Gauss pentru a descoperi o soluție mai mică

decât N_1, N_2, N_3 a unui sistem compus din următoarele ecuații

$$\begin{cases} x = m^e \bmod N_1 \\ x = m^e \bmod N_2 \\ x = m^e \bmod N_3 \end{cases}$$

Această soluție este, conform teoremei chineze a resturilor, cubul mesajului m . Soluția pentru această problemă este cea denumită sărarea mesajului (din engleză salting), adică adăugarea unui padding format din numere pseudoaleatoare, padding diferit pentru fiecare expediție a mesajului.

3.1.3.1 Atac de tip mesaj stereotip

Când o parte din plaintext este cunoscută, este posibil să se recupereze tot plaintextul inițial dacă dimensiunea exponentului este destul de mică.

În cazuri extreme, presupunem că noi cunoaștem că plaintextul este mic relativ la dimensiunea modulului. Asta înseamnă că cei mai semnificativi biți ai plaintextului vor fi 0. Dacă plaintextul $m < N^{1/e}$ este cryptat cu exponentul e , atunci cu siguranță plaintextul va fi recuperat ușor deoarece:

$$c = m^e \bmod N = m^e$$

Doar luând e rădăcina unui criptotext, vom putea descoperi un plaintext. Pentru un plaintext aleator m în \mathbb{Z}_N este foarte improbabil ca și acesta să fie mic.

Așadar, dacă dimensiunea lui N ar fi de 1024 de biți iar $e = 3$, atunci plaintextele mai mici de 342 de biți pot fi recuperate foarte ușor. Padând biți random unui plaintext în așa fel încât $m > N^{1/e}$ împiedică acest atac.

Acum să presupunem că avem un plaintext de dimensiune mare, asta ar însemna că acest atac nu ar putea funcționa, dar totuși noi cunoaștem o bucată din acest text, spre exemplu:

Astăzi la licență am prezentat ????????

unde partea pe care nu o cunoaștem este destul de mică. În această situație, Copper-smith [9] [10] ne arată că dacă partea necunoscută a unui text este destul de mică, este posibil să recuperăm întreg plaintextul. Rezultatul de bază este dat de următoare teoremă:

Teorema 1 Fie $\langle N, e \rangle$ o cheie RSA publică, și m plaintextul. Cheia publică e este folosită pentru calculul criptotextului $c = m^e \bmod N$, dacă toate sau măcar o fracție $1/e$ de biți consecutivi este cunoscută, atunci tot plaintextul m poate fi calculat în timpul $\log(N)$.

Demonstrație: Deoarece măcar o fracție $1/e$ de biți consecutivi este cunoscută din m , putem scrie plaintextul ca și:

$$m = m_2 \cdot 2^{k_2} + m_1 \cdot 2^{k_1} + m_0$$

Unde toți acești termeni sunt cunoscuți, mai puțin m_1 . Mai departe știm că dimensiunea lui m_1 satisface următoarea relație $|m_1| < N^{1/e}$. Acest lucru sugerează că trebuie să ne uităm după soluții mici ale polinomului monic de grad e $f_N(x) \in \mathbb{Z}_N$ (un polinom monic este un polinom univariat al cărui coeficient al termenului X^n este egal cu 1, iar un polinom univariat este un polinom care are o singură variabilă x) dat de următoarea formulă:

$$f_N(x) = 2^{-k_1 \cdot e} ((m_2 \cdot 2^{k_2} + x \cdot 2^{k_1} + m_0)^e - e) \bmod N$$

deoarece $f_N(m_1) = 2^{-k_1 \cdot e} (m^e - e) = 0 \bmod N$. Deoarece $|m_1| < N^{1/e}$, rezultatul Copersmith pentru găsirea soluțiilor mici pentru ecuațiile modulare univariate poate fi folosit pentru a calcula m_1 , care va face ca întreg plaintextul să fie descoperit.

Pentru teorema ca teorema prezentată mai sus să poată fi folosit eficient în practică, exponentul public trebuie să fie relativ mic. Acest lucru urmărește 2 condiții:

1. Dacă $e > \log_2(N)$ atacul necesită ca toți biții din plaintext să fie cunoscuți.
2. Costul unui astfel de atac va crește odată cu creșterea exponentului public. Spre exemplu, pentru valuarea $e = 2^{16} + 1$ atacul nu va funcționa.

3.1.3.2 Atac de tip mesaje relaționale

Când două plaintexte sunt criptate cu aceeași cheie publică e , este posibil să recuperăm plaintextele dacă exponentul public este mic iar relația liniară dintre plaintexte este cunoscută. Un atac inițial a fost făcut când exponentul public e era $e = 3$, de către Franklin și Reiter [24] după care a fost preluată de Coppersmith [11], iar rezolvarea este dată de următoarea teoremă:

Teorema 2 Fie $< N, e >$ o cheie publică RSA, cu $e = 3$. Fie m_1 și m_2 două plaintexte astfel încât $m_2 = am_1b$. Date $a, b, c = m_1^3 \bmod N, c_2 = m_2^3 \bmod N$, atât m_1 cât și m_2 pot fi calculat în timpul $\log(N)$.

Acest lucru este dovedid în următorul calcul:

$$\frac{b(c_2 + 2a^3c_1 - b^3)}{a(c_2 - a^3c_1 + 2b^3)} \bmod N$$

$$= \frac{m_1(3a^3bm_1^2 + 3a^2b^2m_1 + 3ab^3)}{3a^3m_1^2 + 3a^2b^2m_1 + 3ab^3} \bmod N = m_1$$

De îndată ce m_1 este cunoscut, este ușor să calculăm $m_2 = am_1 + b$

3.1.3.3 Atac prin padare aleatoare

Când două plaintexte sunt legate între ele printr-o relație liniară $m_2 = am_1 + b$, unde b este necunoscut, tot este posibil să recuperăm plaintextul dacă b este suficient de mic. Acest atac a fost făcut pentru $e = 3$ de către Coppersmith [10], folosind următoare teoremă:

Teorema 3 Fie $\langle N, e \rangle$ cheia RSA publică și $e = 3$. Fie m_1 și m_2 două plaintexte care satisfac relația $m_2 = am_1 + b$. Dat $c_1 = m_1^3 \bmod N$, $c_2 = (m_1 + b)^3 \bmod N$, dacă $|b| < N^{1/9}$ atunci plaintexte m_1 și m_2 pot fi calculate în timp polinomial.

3.1.4 Exponent de decriptare mic

Dacă exponentul de decriptare (cel secret) este mic, pe lângă faptul că multe părți din mesaj nu se criptează, așa cum s-a arătat mai sus, există un algoritm rapid de găsim a lui d , cunoscând informațiile e și N . Acest algoritm nu este eficient dacă d este de același ordin de mărime cu N , deci dacă e este mic, acesta fiind unul din motivele pentru care se alege în general e un număr mic, pentru ca d să fie cât mai mare.

3.1.4.1 Atacul fracțiilor continue: Atacul Wiener

Primul atac semnificativ asupra exponenților privați mici [3]. Dată doar cheia publică RSA $\langle N, e \rangle$, atacul factorizează modulul folosind doar informații obținute din convergența fracției continue e/N . Atacul Wiener folosește următoarea teoremă:

Teorema 4 Fie $N = PQ$ modulul RSA cu e cheia de criptare publică și d cheia privată definită ca

și $\lambda(N)$. Fie k să fie un număr întreg care satisface următoarele:

$$ed = 1 + k\lambda(N)$$

$$g = \text{cmmdc}(P-1, Q-1)$$

$$g_0 = \frac{g}{\text{cmmdc}(g, k)}$$

$$k_0 = \frac{k}{\text{cmmdc}(k, g)}$$

Dacă d satisface:

$$d < \frac{PQ}{2(P+Q-1)g_0k_0} = \frac{N}{2(P+Q-1)g_0k_0}$$

Atunci N poate fi factorizat în timpul $\log(N)$

Demonstrație: folosim N modulul RSA, unde $N = PQ$

$$\begin{aligned}\lambda(N) &= \text{cmmdc}(P-1, Q-1) = \frac{\phi(N)}{\text{cmmdc}(P-1, Q-1)} = \frac{PQ - (P+Q-1)}{g} \\ &= \frac{N - (P+Q-1)}{g}\end{aligned}$$

Vom nota cu s conținutul parantezii, deci rezultă:

$$\lambda(N) = \frac{N-s}{g}$$

De aici rezultă că ecuația cheii va fi:

$$ed = 1 + k\lambda(N) = 1 + \frac{k \cdot \phi(N)}{\text{cmmdc}(P-1, Q-1)}$$

Cunoaștem faptul că $g_0 = \frac{g}{\text{cmmdc}(g, k)}$ și $k_0 = \frac{k}{\text{cmmdc}(k, g)}$ de unde rezultă că $g = g_0 \cdot \text{cmmdc}(g, k)$ și $k = k_0 \cdot \text{cmmdc}(k, g)$, prin urmare $\frac{g}{k} = \frac{g_0}{k_0}$ de unde rezultă:

$$\begin{aligned}1 + \frac{k \cdot \phi(N)}{\text{cmmdc}(P-1, Q-1)} &= 1 + \frac{k(N-s)}{g} \\ &= 1 + \frac{k_0}{g_0}(N-s)\end{aligned}$$

Dacă vom împărți ecuația $ed = 1 + \frac{k_0}{g_0}(N-s)$ la dN vom obține următoarea relație:

$$\frac{e}{N} = \left(1 + \frac{k_0N - k_0s}{g_0}\right) \cdot \frac{1}{dN}$$

$$\frac{e}{N} = \frac{1}{dN} + \frac{k_0}{g_0d} + \frac{k_0s}{dNg_0}$$

De unde rezultă:

$$\left| \frac{e}{N} - \frac{k_0}{dg_0} \right| = \left| \frac{1}{dN} - \frac{k_0s}{dg_0N} \right|$$

De aici putem considera faptul că :

$$\left| \frac{1}{dN} - \frac{k_0s}{dg_0N} \right| < \frac{k_0s}{dg_0N}$$

Folosind următoarea relație:

$$d < \frac{N}{2(P+Q-1)g_0k_0}$$

$$d < \frac{N}{2sg_0k_0}$$

Din această relație putem ajunge la următoarea formulă:

$$\frac{d}{N} < \frac{1}{2sg_0k_0} \Rightarrow \frac{k_0s}{N} < \frac{1}{2dg_0}$$

Înmulțind în ambele părți cu $\frac{1}{dg_0}$ obținem:

$$\frac{k_0s}{dg_0N} < \frac{1}{2(dg_0)^2}$$

Din această relație, ajungem la concluzia că:

$$\left| \frac{e}{N} - \frac{k_0}{dg_0} \right| < \frac{1}{2(dg_0)^2}$$

Știm din teorema fracțiilor continue, că $\frac{k_0}{dg_0}$ este una din convergențe din fracția continuă $\frac{e}{N}$. Dacă avem $c_i = \frac{a_i}{b_i}$ să fie convergența i a lui $\frac{e}{N}$ atunci știm că $\frac{k_0}{dg_0} = \frac{a_j}{b_j}$ pentru un anumit j .

Fracții continue [22]

Fie a și b numere naturale, $b \neq 0$, folosind algoritmul lui Euclid obținem:

$$a = q_1 \cdot b + r_1, \quad 0 < r_1 < b$$

$$b = q_2 \cdot r_1 + r_2, \quad 0 < r_2 < r_1$$

$$r_1 = q_3 \cdot r_2 + r_3, \quad 0 < r_3 < r_2$$

...

$$r_{k-2} = q_k \cdot r_{k-1} + r_k, \quad 0 < r_k < r_{k-1}$$

$$r_{k-1} = q_{k+1} \cdot r_k$$

Fracția $\frac{a}{b}$ poate fi scrisă după cum urmează:

$$\begin{aligned}\frac{a}{b} &= \frac{q_1 b + r_1}{b} \\ &= q_1 + \frac{1}{\frac{b}{r_1}} \\ &= q_1 + \frac{1}{q_2 + \frac{1}{\frac{b}{r_1}}} \\ &\dots \\ &= q_1 + \frac{1}{q_2 + \frac{1}{\ddots q_k + \frac{1}{q_{k+1}}}}\end{aligned}$$

Ultimul termen va fi referit ca fracția continuă asociată fracției $\frac{a}{b}$ și va fi notat prin $[q_1, q_2, \dots, q_{k+1}]$. Expresiile $[q_1, q_2, \dots, q_i]$, $1 \leq i \leq k+1$, vor fi numite convergențele fracției continue $[q_1, q_2, \dots, q_{k+1}]$. Considerăm numerele naturale α_i și β_i date prin

$$\begin{aligned}\alpha_1 &= q_1, & \beta_1 &= 1 \\ \alpha_2 &= q_1 \cdot q_2 + 1, & \beta_2 &= q_2 \\ \alpha_i &= q_i \cdot \alpha_{i-1} + \alpha_{i-2}, & \beta_i &= q_i \cdot \beta_{i-1} + \beta_{i-2}\end{aligned}$$

Pentru orice $3 \leq i \leq k+1$

Atunci are loc relația

$$\begin{aligned}[q_1, q_2, \dots, q_i] &= \frac{\alpha_i}{\beta_i} \\ (\alpha_i, \beta_i) &= 1\end{aligned}$$

Pentru orice $1 \leq i \leq k+1$

Exemplu Pentru $a = 4, b = 11$ obținem:

$$4 = 0 \cdot 11 + 4$$

$$11 = 2 \cdot 4 + 3$$

$$4 = 1 \cdot 3 + 1$$

$$3 = 3 \cdot 1$$

$$\text{și } [0, 2, 1, 3] = 0 + \frac{1}{2 + \frac{1}{1 + \frac{1}{3}}}, [0] = \frac{0}{1}, [0, 2] = \frac{1}{2} \text{ și } [0, 2, 1] = \frac{1}{3}$$

Acum să s-a înțeleas conceptul de fracție continuă, vom încerca să scriem ecuația $ed = 1 + k\lambda(N)$ sub o altă formă folosindu-ne de faptul că $\lambda(N) = \frac{N-s}{g}$

$$\begin{aligned} ed &= 1 + k \cdot \frac{N-s}{g} \\ &= 1 + \frac{kN - ks}{g} \\ &= 1 + \frac{k}{g}(N-s) \\ &= 1 + \frac{k}{g}(PQ - P - Q + 1) \\ &= 1 + \frac{k}{g}\phi(N) \end{aligned}$$

Am demonstrat și faptul că $\frac{k_0}{g_0} = \frac{k}{g}$. Deci ecuația finală va fi:

$$ed = 1 + \frac{k_0}{g_0}\phi(N)$$

De aici putem observa următorul lucru:

$$\begin{aligned} \phi(N) &= (ed - 1) \frac{g_0}{k_0} \\ &= ed \frac{g_0}{k_0} - \frac{g_0}{k_0} \\ &= e \left(\frac{dg_0}{k_0} \right) - \frac{g_0}{k_0} \\ &= e \left(\frac{b_j}{a_j} \right) - \frac{g_0}{k_0} \end{aligned}$$

Așadar, putem calcula $\phi(N)$ dacă cunoaștem convergența corectă $c + j$ și putem afla valorile $\frac{g_0}{k_0}$. Știm că putem calcula $\phi(N)$ în momentul în care putem să-l factorizăm pe N deoarece $\phi(N) = (P-1)(Q-1)$ și $N = PQ$.

Pentru ușurința în calcul, vom presupune că $g = 2$ și k este un număr suficient de mic, deci $\frac{k_0}{g_0} \approx k$, atunci $ed = 1 + k\phi(N)$ de unde rezultă:

$$\begin{aligned} ed - k\phi(N) &= 1 \\ \frac{e}{\phi(N)} - \frac{k}{d} &= \frac{1}{d\phi(N)} \\ \frac{e}{N} &\approx \frac{k}{d} \end{aligned}$$

Deoarece $PQ - (P + Q) + 1 \approx N$ Pentru găsirea lui $\frac{e}{N}$ trebuie să găsim fracția $\frac{k}{d}$. Trebuie să ținem cont de următoare observații:

1. Deoarece $ed \equiv 1 \pmod{\phi(N)}$ și $\phi(N)$ va fi un număr par, d trebuie să fie impar. Deci dacă d este par, vom continua la următoarea convergență.
2. Deoarece $\phi(N)$ este un număr întreg, vom verifica $\frac{ed-1}{k}$. Dacă este un număr întreg, vom trece la următoarea convergență.

Ecuția pătratică

Din

$$\phi(N) = N - (P + Q) + 1$$

$$P + Q = N - \phi(N) + 1$$

Considerăm ecuația de gradul 2 $(x - P)(x - Q)$ cu rădăcinile P și Q . Atunci vom avea:

$$(x - P)(x - Q) = 0$$

$$x^2 - (P + Q)x + PQ = 0$$

$$x^2 - (N - \phi(N) + 1)x + N = 0$$

Dacă valoarea lui $\phi(N)$ este corectă, atunci rădăcinile ecuației de gradul 2 vor fi numere întregi, numerele fiind chiar factorii lui N .

Exemplu vom avea modulul RSA $N = 64741$ și exponentul public $e = 42667$. Exponentul de decriptare fiind d .

Trebuie să ne amintim faptul că $\frac{e}{N} \approx \frac{k}{d}$, deci va trebuie să găsim aproximarea fracției $\frac{e}{N} = \frac{42667}{64741}$.

Vom folosi algoritmul lui Euclid pentru a găsi convergențele succesive.

$$\frac{42667}{64741} = 0 \text{ rest } 42667 \Rightarrow \frac{k}{d} \approx 0$$

Dar $k = 0, d = 1$ clar nu vor funcționa pentru decriptare, deci vom trece la următoarea convergență.

$$\frac{64741}{42667} = 1 \text{ rest } 22074 \Rightarrow \frac{k}{d} \approx 0 + \frac{1}{1}$$

Din nou $k = 1, d = 1$ nu va funcționa pentru decriptare, deci vom trece din nou la următoarea convergență.

$$\frac{42667}{22074} = 1 \text{ rest } 20593 \Rightarrow \frac{k}{d} \approx 0 + \frac{1}{1 + \frac{1}{1}} = \frac{1}{2}$$

Acum avem $k = 1, d = 2$ dar am menționat mai sus că d trebuie să fie un număr impar, deci vom trece la următoarea convergență

$$\frac{22074}{20593} = 1 \text{ rest } 1481 \Rightarrow \frac{k}{d} \approx 0 + \frac{1}{1 + \frac{1}{1}} = \frac{2}{3}$$

De aici ne rezultă $k = 2, d = 3$, acesta fiind un rezultat posibil. Acum trebuie să verificăm dacă $\phi(N)$ este un număr întreg:

$$\phi(N) = \frac{ed - 1}{k} = \frac{42667 \cdot 3 - 1}{2} = 64000$$

Am trecut de ambele condiții, acum trebuie să vedem dacă ecuația de gradul 2 $x^2 - (N - \phi(N) + 1)x + N = 0$ are soluții întregi.

$$x^2 - (N - \phi(N) + 1)x + N = 0$$

$$x^2 - (64741 - 64000 + 1)x + 64741 = 0$$

$$x^2 - 742x + 64741 = 0$$

$$\Delta = b^2 - 4ac = 742^2 + 4 \cdot 64741 = 550564 - 258964 = 291600 = 540^2$$

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} = \frac{742 + 540}{2} = 641$$

$$x_2 = \frac{-b - \sqrt{\Delta}}{2a} = \frac{742 - 540}{2} = 101$$

Cum $641 \cdot 101 = 64741$, înseamnă că l-am factorizat corect pe N , deci $d = 3$

3.1.4.2 Apărarea împotriva atacului Wiener

Este destul de clar că atunci când implementăm RSA, vom dori să evităm acest atac. Avem următorii parametri:

- Alegem P, Q cu $N = PQ$
- Găsim ed astfel încât $ed \equiv 1 \pmod{\phi(N)}$

Putem dovedi că, dacă $q < p \leq 2q$ și $d \leq \frac{1}{3}\sqrt[4]{N}$ atacul Wiener va reuși.

Astea înseamnă că noi trebuie să alegem $d > \frac{1}{3}\sqrt[4]{N}$. Și așa, există o șansă ca atacul Wiener să reușească.

3.1.4.3 Numere prime parțial cunoscute

La acest atac, trebuie să ne bazăm pe faptul că N este o aproximare a lui $\phi(N)$. În particular, atacurile depind de inecuația $|N - \phi(N)| = |s| < 3N^{1/2}$. Pentru îmbunătățirea acestui atac, va trebui să găsim o aproximare mai bună pentru $\phi(N)$.

Când măcar unul din cele două numere prime ale modulului RSA, are cel mai semnificativ bit cunoscut, atunci noi putem să formăm o aproximare mai bună pentru $\phi(N)$. Fie P' o aproximare pentru P care satisface următoarea relație:

$$|P - P'| < N^\gamma$$

cu $\frac{1}{4} < \gamma \leq \frac{1}{2}$. Cu P' cunoscut, îl putem calcula foarte ușor și pe Q'

$$|Q - Q'| = \left| \frac{N}{P} - \frac{N}{P'} \right| = \left| \frac{N(P - P')}{PP'} \right| < 5N^\gamma$$

Folosind aceste relații, vom putea calcula și o aproximare a lui N notată N' care corespunde următoarei formule:

$$N' = N - P' - Q' + 1$$

Iar de aici putem scoate o aproximare și pentru s'

$$\begin{aligned} |\phi(N) - N'| &= |(P-1)(Q-1) - (N - P' - Q' + 1)| \\ &= |PQ - P - Q + 1 - N + P' + Q' - 1| \\ &= |P - P' + Q - Q'| \end{aligned}$$

Știm că $|P - P'| < N^\gamma$ și $|Q - Q'| < 4N^\gamma$ rezultă

$$|P - P' + Q - Q'| < 5N^\gamma$$

Având toate aceste date, ecuația finală va fi:

$$ed = 1 + k\phi(N) = 1 + k(N' - s')$$

Folosind această relație, putem dezvolta următorul atac de tip exponent de decriptare mic. În realizarea acestui atac, vom folosi următoarea teoremă:

Teoremă fie $N = PQ$ modulul RSA, e exponentul de criptare și d exponentul de decriptare. Fie k un număr întreg care satisface relația $1 + k\lambda(N), g = \text{cmmdc}(P-1, Q-1), g_0 = \frac{g}{\text{cmmdc}(g,k)}$ și

$k_0 = \frac{k}{\text{cmmdc}(k,g)}$. Folosind P' cu proprietatea $|P - P'| < N^\gamma$, exponentul de decriptare are următoarea proprietate:

$$d < \frac{N'}{2s'g_0k_0}$$

Totuși, această primă relație este posibil să nu fie suficientă pentru a putea dezvolta atacul, și vom avea nevoie de o inecuația care să-l conțină și pe N^γ .

$$N' = N - P' - Q' + 1 \Rightarrow N' < N$$

$$s' = |P - P' + Q - Q'| < 5N^\gamma$$

Împărțind aceste două relații vom obține:

$$\frac{N'}{s'} < \frac{N}{5N^\gamma} \Rightarrow \frac{N'}{s'} < \frac{N^{1-\gamma}}{5}$$

Înmulțind în ambele părți cu $\frac{1}{2g_0k_0}$ obținem

$$\frac{N'}{2s'g_0k_0} < \frac{N^{1-\gamma}}{10g_0k_0}$$

Așadar, am aflat că

$$d < \frac{N'}{2s'g_0k_0} < \frac{N^{1-\gamma}}{10g_0k_0}$$

Folosind această inecuație, îl vom putea factoriza pe N în timp polinomial. De asemenea, dacă vom considera un parametru δ astfel încât $d = N^\delta$, atunci atacul se va simplifica la

$$\delta \leq \begin{cases} \frac{1}{4} - \frac{\alpha}{2} - \epsilon, & \lambda = 1/2 \\ 1 - \frac{\alpha}{2} - \frac{\lambda}{2} - \epsilon, & \lambda < 1/2 \end{cases}$$

Pentru un exponent arbitrat fixat va rezulta:

$$\delta \leq \begin{cases} \frac{3}{4} - \epsilon, & \lambda = 1/2, \alpha = 1 \\ \frac{1}{2} - \frac{\lambda}{2} - \epsilon, & \lambda < 1/2, \alpha = 1 \end{cases}$$

Toate atacurile de tip "latice-based" bazate pe exponenți de decriptare mici, au fost îmbunătățite de către Sarkar, Maitra și Sarka [26]. Pentru a generaliza atacul Blomer și May inspirat din [26], vom considera:

Pentru $\forall \epsilon > 0, \exists n_0$ astfel încât $n > n_0$ unde n este lungimea în biți a lui N . Fie $N = PQ$ modulul

RSA cu e cheia publică și $d = N^\gamma$ cheia privată, ce corespunde modulului $\phi(N)$. Dată cheia publică $\langle N, e \rangle$ și λ cu $\frac{1}{4} < \lambda \leq \frac{1}{2}$ și

$$\delta \leq \frac{2}{5} - \frac{6}{5}\lambda + \frac{2}{5}\sqrt{4\lambda^2 - \lambda + 1} - \varepsilon$$

, atunci modulul N poate fi calculat în timp polinomial.

O altă formă, este dată de următoarea teoremă

Teoremă $\forall \varepsilon > 0, \exists n > n_0$. Fie $N = PQ$ modulul RSA cu e exponentul public și $d = N^\delta$ exponentul privat definit modulo $\phi(N)$. Date fiind cheia publică $\langle N, e \rangle$ și λ cu $\frac{1}{4} < \lambda \leq \frac{1}{2}$ satisfac următoarea relație

$$1 - 2\lambda < \delta \leq 1 - \sqrt{\lambda} - \varepsilon$$

atunci putem găsi modulul RSA în timp polinomial.

3.2 Criptanaliză pe variante RSA

3.2.1 CRT-RSA

Puterea computațională necesară decriptării RSA-ului poate fi redusă exploatând factorizarea modulului N . Fie $\langle N, e \rangle$ cheia publică RSA și $\langle d, P, Q \rangle$ cheia privată unde exponenții sunt definiți modulo $\lambda(N)$. Pentru un criptotext $c = m^e \bmod N$ decriptarea standard se obține făcând calculul

$$m = c^d \bmod N$$

Acest calcul fiind o singură exponențiere pe n biți. Totuși, plaintextul poate fi obținut prin decriptări parțiale, una modulo P și alta modulo Q . Folosind teorema Chineză a resturilor va trebui să calculăm:

$$m_P = c^d \bmod P$$

$$m_Q = c^d \bmod Q$$

Deoarece P și Q sunt numere prime (înseamnă că sunt și relativ prime), combinând aceste două ecuații folosind teorema Chineză a resturilor, vom putea obține rezultatul. Folosind algoritmul lui Garner [?], atunci plaintextul se calculează astfel:

$$m = m_P + P((m_Q - m_P)p^{-1} \bmod Q)$$

Totuși, când exponentul de decriptare este mai mare decât \sqrt{N} , un exponent redus poate fi folosit.

Fie d_P și d_Q astfel încât:

$$d_P = d \bmod (P - 1)$$

$$d_Q = d \bmod (Q - 1)$$

De unde rezultă:

$$m_P = c^{d_P} \bmod P$$

$$m_Q = c^{d_Q} \bmod Q$$

Îi vom numi pe d_P și d_Q exponenții CRT. Din ecuațiile de mai sus și din faptul că $ed = 1 + k\phi(N)$ următoarele:

$$ed_P \equiv 1 \bmod (P - 1)$$

$$ed_Q \equiv 1 \bmod (Q - 1)$$

De asemenea, $\exists k_P$ și k_Q astfel încât:

$$ed_P = 1 + k_P(P - 1)$$

$$ed_Q = 1 + k_Q(Q - 1)$$

3.2.1.1 Spargerea CRT-RSA

Calculând exponenții CRT este suficient pentru a sparge această variantă de RSA. Dacă avem ambii exponenți d_P și d_Q putem obține următoarea relație:

$$ed_P = 1 + k_P(P - 1) \Rightarrow ed_P - 1 = k_P(P - 1)$$

$$ed_Q = 1 + k_Q(Q - 1) \Rightarrow ed_Q - 1 = k_Q(Q - 1)$$

Înmulțind aceste două relații obținem:

$$\begin{aligned} (ed_P - 1)(ed_Q - 1) &= [k_P(P - 1)][k_Q(Q - 1)] \Rightarrow \\ e^2 ed_Q - ed_P - ed_Q + 1 &= k_P(P - 1)k_Q(Q - 1) \Rightarrow \\ e(ed_P d_Q - d_P - d_Q) + 1 &= k_P(P - 1)k_Q(Q - 1) \end{aligned}$$

Unde, fiecare membru din partea stângă este cunoscut. Așadar, putem calcula un multiplu de $\phi(N)$ ce ne va permite într-un mod probabilistic să factorizăm modulul folosind Miller [28].

Cunoscând doar un singur exponent CRT, există totuși o șansă ca modulul N să poate fi factorizat. Să presupunem că d_P este termenul cunoscut. Dacă d cheia privată este mai mică decât fiecare număr prim, astfel încât $d = d_P = d_Q$ atunci $ed_P - 1$ este un multiplu de $\lambda(N)$ și modulul poate fi factorizat folosind Miller [28]. Dacă exponentul privat d este mai mare decât fiecare dintre numerele prime, atunci vom observa folosind $ed_P = 1 + k_P(P - 1)$ că $m^{ed_P} \equiv m \pmod{P}$, pentru orice plaintext $m \in \mathbb{Z}_P$, atunci:

$$m^{ed_P} - m = cp$$

unde c este un număr întreg. Fie $M = (m^{ed_P} - m) \pmod{N}$. Când c nu este un multiplu al numărului prim Q , factorizarea modulului este dată de $\text{cmmdc}(M, N) = P$. Atunci când c este un multiplu al lui q atunci $m^{ed_P} \equiv m \pmod{N}$. De aici rezultă că $\forall m \in \mathbb{Z}_P^*$, atunci modulul N poate fi factorizat, deoarece d_P este un exponent valid de decriptare și $ed_P - 1$ dezvăluie un multiplu al lui $\lambda(N)$.

3.2.1.2 Reconstruirea cheii folosind erori aleatoare

Aceasta este o altă modalitate pentru reconstrucția cheii atunci când se folosește CRT-RSA. Este menționat în [4], că atacul poate fi îmbunătățit folosind mai multe informații despre cheia privată. Întrădevăr, există un atac îmbunătățit realizat de Heninger și Shacham [13] pentru exponenți publici mici, folosesc aproximări pentru fiecare P, Q, d, d_P, d_Q , din CRT-RSA și exploatează următoarele 4 ecuații:

$$N = PQ$$

$$ed = k(N - P - Q + 1) + 1$$

$$ed_P = k_P(P - 1) + 1$$

$$ed_Q = k_Q(Q - 1) + 1$$

care conțin 8 necunoscute. Deoarece exponentul public este mic, $2^{10} + 1$ spre exemplu, constantele k, k_P și k_Q poate fi ușor determinate ceea ce reduce numărul de necunoscute la 5.

Pentru determinarea lui k ne folosim de faptul că $1/2$ din cei mai semnificativi biți al exponentului privat sunt arătați de către $\lceil (kN + 1)/e \rceil$, calculând această valoare pentru fiecare $2 \leq k' < e$, valoarea corectă a lui k este arătată atunci când un număr semnificativ de biți este același ca și în ecuația $\lceil (k'M + 1)/e \rceil$.

Cu k cunoscut, Heninger și Shacham demonstrează că se pot afla foarte ușor k_P și k_Q , folosind următoarele ecuații:

$$\begin{aligned}
-1 &\equiv k(P-1)(Q-1) \bmod e \\
-1 &\equiv k_P(P-1) \bmod e \\
-1 &\equiv k_Q(Q-1) \bmod e
\end{aligned} \tag{3.1}$$

Vom observa că dacă înmulțim ultimele două ecuații vom obține $1 \equiv k_P k_Q (P-1)(Q-1) \bmod e$. Comparând rezultatul cu prima ecuația vom constata următorul lucru:

$$k \equiv -k_P k_Q \bmod e$$

Modificând prima ecuație, vom obține:

$$\begin{aligned}
-1 &\equiv k(N-P-Q+1) \bmod e \Rightarrow \\
-1 &\equiv kN - kP - kQ + k \bmod e \Rightarrow \\
-1 &\equiv kN - kP - kQ + k - k + k \bmod e \Rightarrow \\
-1 &\equiv (kN - k) - (kP - k) - (kQ - k) \bmod e \Rightarrow \\
-1 &\equiv k(N-1) - k(P-1) - k(Q-1) \bmod e \Rightarrow \\
-1 &\equiv k(N-1) - k(P-1+Q-1) \Rightarrow
\end{aligned}$$

Putem substitui:

$$\begin{aligned}
k &\equiv -k_P k_Q \bmod e \\
(P-1) &\equiv -1/k_P \bmod e \\
(Q-1) &\equiv -1/k_Q \bmod e
\end{aligned}$$

și obținem:

$$\begin{aligned}
-1 &\equiv k(N-1) - (-k_P k_Q) \left(-\frac{1}{k_P} - \frac{1}{k_Q} \right) \Rightarrow \\
-1 &\equiv k(N-1) - \left(\frac{k_P k_Q}{k_P} + \frac{k_P k_Q}{k_Q} \right) \Rightarrow \\
0 &\equiv k(N-1) - (k_P + k_Q) + 1 \Rightarrow \\
0 &\equiv k(N-1) - \left(k_P - \frac{k}{k_P} \right) + 1
\end{aligned}$$

Înmulțind această ecuație cu k_P , obținem următoarea ecuație de gradul 2:

$$k_P k(N-1) - (k_P^2 - k) + k_P \equiv 0 \Rightarrow$$

$$k_P [k(N-1) + 1] - k_P^2 + k \equiv 0 \Rightarrow$$

$$k_P^2 - k_P [k(N-1) + 1] - k \equiv 0$$

Această ecuație, cu exonentul public e fiind un număr prim va avea următoarele soluții: $k_P \bmod e$ și $k_Q \bmod e$. Deoarece $k_P, k_Q < e$ înseamnă ca valorile exacte pentru k_P și k_Q vor fi descoperite.

Cu constantele $< k, k_P, k_Q$ fiind descoperite, reconstruirea cheii înseamnă doar enumerarea cheilor posibile $< P, Q, d, d_P, d_Q >$ care respectă ecuația (2.1)

3.2.2 Multi-Prime RSA

Multi-Prime RSA este o simplă extensie a criptosistemului, în care modulul N este format din produsul a mai multor numere prime.

În această prezentare, vom folosi r ca și cardinal pentru numerele prime.

Pentru Multi-Prime RSA cu r numere prime, $N = P_1 P_2 \dots P_r$, așadar modulul N este un simplu produs de r numere prime. Dacă sortăm aceste numere prime, în așa fel încât $P_i < P_{i+1}$ pentru $i = 1, \dots, r-1$, atunci putem afirma că:

$$4 < \frac{1}{2} N^{\frac{1}{r}} < P_1 < N^{\frac{1}{r}} < P_r < 2 N^{\frac{1}{r}} \quad (3.2)$$

Generarea cheilor pentru algoritmul Multi-Prime RSA este în general același ca și pentru RSA clasic, diferența fiind doar faptul că vom avea r numere prime distincte. Exponentul public și cel private sunt definite ca și inverse modulare modulo $\phi(N)$, deci:

$$ed \equiv 1 \bmod \phi(N)$$

Iar din această afirmație putem scoate ecuația:

$$ed = 1 + k\phi(N)$$

unde k este un număr întreg pozitiv. Ca și în RSA, putem înlocui $\phi(N)$ cu $N - s$. Dezvoltând $\phi(N)$,

vom observa că s poate fi scris ca:

$$\begin{aligned}
 s &= N - \phi(N) \\
 &= N - \prod_{i=1}^r (P_i - 1) \\
 &= \sum_{i=1}^r \frac{N}{P_i} - \sum_{\substack{i,j=1 \\ i < j}}^r \frac{N}{P_i P_j} + \sum_{\substack{i,j,k=1 \\ i < j < k}}^r \frac{N}{P_i P_j P_k} + \cdots + (-1)^r
 \end{aligned}$$

După cum a fost prezentat în [27], această expresie pentru un anumit s , combinată cu condiția (2.1), implică o nouă dimensiune pentru s

$$|s| < (2r - 1)N^{1-1/r} \quad (3.3)$$

Deoarece, $\phi(N)$ și N au aproximativ $(r - 1)/r$ din cei mai semnificativi biți în comun, atunci N este o bună aproximare pentru $\phi(N)$. Cu cât cardinalul numerelor prime este mai mare, numărul de biți pe care acestea îl au în comun scade. Așadar, majoritatea atacurilor pot fi extinse ca și extensii pentru multi-prime RSA folosind pur și simplu marginea lui $|s|$ când sunt r numere prime.

Algoritmul de criptare pentru multi-prime RSA este identic ca și cel clasic. Pentru orice plaintext m , criptotextul lui va fi $c = m^e \bmod N$. Exponentul public e va fi notat ca și $e = N^\alpha$ cu $0 < \alpha < 1$. La fel ca și cu variante clasică, există două tipuri de decriptări pentru acest algoritm, abemele variante fiind identice ca și la RSA clasic.

3.2.2.1 Spargerea Multi-Prime RSA

Vom considera o instanță a lui Multi-Prime RSA să fie spartă când factorizarea lui N este cunoscută. Pentru RSA clasic, a fost suficient să recuperăm exponentul privat pentru a putea calcula $\phi(N)$. Pentru Multi-Prime RSA o astfel de abordare s-a reușit încă. Așadar, nu există vreun algoritm determinist care poate factoriza modulul folosind exponentul privat sau $\phi(N)$. Totuși, odată ce un multiplu de $\phi(N)$ este cunoscut, rezultatul lui Milner [28] poate fi folosit pentru a putea calcula modulul în mod probabilistic. Deoarece $ed - 1 = k\phi(N)$, este suficient să obținem exponentul privat pentru a putea calcula în mod probabilistic modulul

3.2.3 Factorizarea Modulului

3.2.3.1 Exponent de decriptare mic

Toate atacurile cunoscute de tip exponent de decriptare mic pentru Multi-Prime RSA sunt de fapt extensii pentru RSA clasic.

Extensie atacului lui Wiener pentru Multi-Prime RSA pare să fie unul dintre cele mai slabe atacuri, dar totuși este eficient în implementare. Teorema folosită în atacul clasic poate fi transformată în următoare teoremă:

Teorema 5 *Pentru fiecare număr întreg $r > 1$ corespunde următorul lucru: Fie N modulul RSA format din numere prime balansate, fie e exponentul public pentru criptare, și d exponentul privat pentru decriptare. Cu cheia publică*

$\langle N, e \rangle$ dată, exponentul privat satisface următoarea relație:

$$d < \frac{N^{\frac{1}{2r}}}{\sqrt{2(2r-1)}}$$

Apoi modulul poate fi factorizat în mod probabilistic în timp polinomial

Demonstrație pentru Teorema 2:

Vom generaliza atacul lui Wiener pentru cazul Multi-Prime. Fie $ed = 1 + k\phi(N)$ și

$$\begin{aligned} 0 < \frac{k}{d} - \frac{e}{N} &= \\ &= \frac{kN - ed}{dN} \\ &= \frac{kN - 1 - k\phi(N)}{dN} \\ &= \frac{k(N - \phi(N))}{dN} \\ &= \frac{ks - 1}{dN} \end{aligned}$$

Folosind ecuația (2.3) și faptul că $k < d$, ne vom da seama că partea din stânga este strict mai mică

decât $\frac{2r-1}{N^{1/r}}$ rezultă:

$$\begin{aligned}\frac{2r-1}{N^{1/r}} &< \frac{1}{2d^2} \Rightarrow \\ \frac{2(2r-1)}{N^{1/r}} &< \frac{1}{d^2} \Rightarrow \\ \frac{N^{1/r}}{2(2r-1)} &> d^2 \Rightarrow \\ \frac{N^{1/2r}}{\sqrt{2(2r-1)}} &> d\end{aligned}$$

Dacă ultima inegalitate este adevărată, putem aplica modelul fracțiilor continue pentru $\frac{e}{N}$, vom obține $\frac{k}{d}$ ca fiind convergența. Pentru a obține convergența, adversarul va trebui doar să testeze toate convergențele de forma $\frac{k'}{d'}$ ale lui $\frac{e}{N}$.

Deoarece algoritmul pentru fracții continue are timp polinomial, numărul de convergențe alui factorului $\frac{e}{N}$ este $O(\log(N))$, iar costul pentru testarea fiecărui candidat $\frac{k'}{d'}$ este de asemenea timp polinomial.

3.2.4 Multi-Power RSA

Multi-Power RSA reprezintă o altă variantă RSA unde $N = P^{b-1}Q$ cu $b \geq 3$.

Când exponentul public și cel privat sunt definite modulo funcției lui Euler $\phi(N)$, variantele sunt foarte asemănătoare cu cele asupra multi-prime RSA și majoritatea atacurilor de la multi-prime RSA se aplică direct la multi-power RSA.

Focusul principal va fi o variantă multi-power RSA, varianta lui Takagi, unde exponentul public și cel privat sunt definiți :

$$\lambda'(N) = \text{cmmm}(P-1, Q-1)$$

în loc de $\phi(N) = P^{b-2}(P-1)(Q-1)$. Decriptarea acestei variante folosește lema lui Hensel și teorema Chineză a resturilor, și este cea mai rapidă metodă de decriptare pentru toate variantele atunci când un exponent public mic este folosit.

3.2.4.1 Spargerea multi-power RSA (schema lui Takagi)

Vom considera că o instanța a schemei lui Takagi să fie spartă atunci când factorizarea modului este cunoscută. Pentru RSA, a fost suficient să recuperăm exponentul privat sau să calculăm modulul $\phi(N)$ deoarece există algoritmi determinați ce pot rezolva problema în timp polinomial, dacă

deținem măcar unul din aceste două necunoscute. Pentru schema lui Takagi, odată ce un multiplu de $\lambda'(N)$ este cunoscut, rezultatul lui Miller [28] poate fi folosit pentru a factoriza modulul în mod probabilistic. Deoarece $ed = 1 + k\lambda'(N)$, este suficient să obținem exponentul privat pentru a putea factoriza modulul. Când exponentul este definit sub forma $(P-1)(Q-1)$, în loc de $\lambda'(N)$, a fost arătat de către Kunihiro și Kurosawa [32] că modulul $N = P^{b-1}Q$ poate fi factorizat deterministic în timp polinomial când exponentul privat d este cunoscut.

Similar cu CRT-RSA, cunoașterea unui exponent CRT ne va permite să factorizăm modulul. Din definiția exponenților CRT, este de așteptat ca pentru un mesaj $m \in \mathbb{Z}_N^*$,

$$cmmdc(m^{ed_P} - m, N) = P$$

$$cmmdc(m^{ed_Q} - m, N) = Q$$

,așadar cunoașterea lui d_P sau d_Q ar trebui să ne aducă factorizarea modulului

3.2.5 Factorizarea modulului

Securitatea schemei lui Takagi, la fel ca și la multi-prime RSA, se bazează pe dificultatea factorizării modulului. Aici, dificultatea factorizării unui modul în schema lui Takagi nu ar trebui să fie mai greu decât factorizarea unui modul pe n biți. Dificultatea factorizării unui modul de forma $N = P^{b-1}Q$, pentru o dimensiune fixată a unui modul scade odată cu creșterea lui b când folosim un algoritmul de genul ECM(Lenstra's elliptic curve factorization), așadar valoarea lui b trebuie aleasă cu foarte mare grijă. Orice număr b care satisface condiția este cunoscut ca și număr sigur. Deoarece algoritmul ECM depinde foarte multe de dimensiunea celui mai mic factor al lui N .

3.2.5.1 Exponent de decriptare mic

Sunt două tipuri de atac prin exponent de decriptare mic, fiecare fiind mai puternic decât cel prezentat în multi-prime RSA. Ambele atacuri au fost decoperite de May [33], care se aplică asupra schemei lui Takagi când o parte din biții CRT sunt cunoscuți.

Primul atac se aplică asupra instanțelor de forma $N^{(b-1)/b^2}$ și necesită o parte din cei mai semnificativi biți a unei exponențieri CRT. Atacul lui May [33] este dat de următoarea teoremă:

Teorema 6 *Fie $N = P^{b-1}Q$ să fie modulul RSA, pentru $b \geq 3$. Fie $e = N^\alpha \leq N^{(b-1)/b^2}$ un exponent public valid pentru schema lui Takagi, iar d_P și d_Q să fie exponenții CRT ce satisfac condiția*

$ed \equiv 1 \pmod{(P-1)}$ și $ed \equiv 1 \pmod{(Q-1)}$. Dat d_P' astfel încât

$$|d_P - d_P'| \leq N^{\frac{b-1}{b^2} - \alpha}$$

sau d_Q' astfel încât

$$|d_Q - d_Q'| \leq N^{\frac{1}{b^2} - \alpha}$$

Atunci modulul poate fi factorizat în timpul polinomial n

Următorul atac poate fi inițiat când o parte din cei mai puțin semnificativi biți al exponențerii CRT sunt cunoscuți. Totuși, exponentul public trebuie să fie foarte mic pentru ca acest atac să fie eficient, deoarece atacul are complexitatea liniară e . Acest atac folosește următoarea teoremă

Teorema 7 Fie $N = P^{b-1}Q$ modulul RSA cu $b \geq 3$. Fie $e \ll N^{1/b}$ (\ll însemnând că este mult mai mic) exponentul public valid pentru schema lui Takagi, iar d_P și d_Q să fie exponentul CRT ce satisface relațiile $ed \equiv 1 \pmod{(P-1)}$ și $ed \equiv 1 \pmod{(Q-1)}$ Date oricare din relațiile:

1. $M_P \geq N^{1/b^2}$ și $d_P' \equiv d_P \pmod{M_P}$
2. d_P' astfel încât $|d_P - d_P'| \leq N^{(b-1)/b^2}$
3. $M_Q \geq N^{(b-1)/b^2}$ și $d_Q' \equiv d_P \pmod{M_Q}$
4. d_Q' astfel încât $|d_Q - d_Q'| \leq N^{1/b^2}$

Atunci modulul poate fi factorizat în timpul liniar e

Capitolul 4

Aplicabilitate RSA - KeePass Plugin

4.1 Introducere

4.1.1 Ce este KeePass

KeePass este un manager de parole open source care te ajută să îți stochezi parolele într-un mod securizat. Se pot stoca toate parolele într-o singură bază de date care este blocată cu ajutorul unei chei sau al unui fișier în care se află cheia. Așadar, veți fi nevoiți să rețineți doar o singură parolă master sau să selectați acel fișier pentru a debloca accesul la baza de date. Parolele din interiorul acestei baze de date sunt criptate cu ajutorul algoritmilor AES sau RSA.

4.1.2 Minusurile programului

KeePass nu permite și o modalitate prin care un utilizator își poate exporta parolele stocate în baza de date. Deoarece majoritatea persoanelor folosesc dispozitive multiple(laptop, desktop, tabletă, smartphone), ar trebui să existe o modalitate prin care acestea pot fi sincronizate.

4.1.3 Soluția problemei

O soluție pentru rezolvarea acestei probleme este implementarea unui plugin care va adăuga taburi noi în meniul aplicației. Astfel, un utilizator își va putea adăuga parolele într-un serviciu denumit Vault secret service, iar ulterior, acesta va putea să le și importe într-o altă bază de date.

4.2 Tehnologii Utilizate

4.2.1 .Net Framework C#. Windows System Forms

.Net Framework [35] este un framework produs de Microsoft, folosit la dezvoltarea softwarului care rulează preponderent pe Microsoft Windows, și include o serie de limbaje pentru care acesta oferă suport. În cadrul aplicațiilor, orice limbaj ce face parte din acest framework poate folosi codul scris în orice alt limbaj ce face parte din framework.

Pentru această aplicație, s-a folosit limbajul C# - un limbaj ce face parte din .Net, orientat obiect, ce include o gamă largă de funcționalități, instrumente și mecanisme - printre cele folosite în cadrul aplicației prezentate fiind [36]:

- **ToolStrip** este clasa de bază pentru MenuStrip, StatusStrip și ContextMenuStrip. Acesta este asociat cu Windows Forms pentru a crea toolbar-uri care pot arăta ca și Windows XP, Office, Internet Explorer sau alte aparențe custom, cu sau fără iteme. Clasa ToolStrip conține mai mulți membri care pot face manage pentru desenare, cu input de la mouse sau tastatură și funcționalitate drag and drop. Folosim clasa ToolStripRender și clasa ToolStripManager pentru a avea și mai mult control asupra customizării în momentul desenării. Următorii itemi sunt speciali proiectați pentru a lucra cu ToolStripSystemRenderer și ToolStripProfessionalRenderer: ToolStripButton, ToolStripSeparator, ToolStripLabel, ToolStripDropDownButton, ToolStripSplitButton, ToolStripTextBox, ToolStripComboBox.
- Windows System Forms namespace conține clase pentru a crea aplicații windows-based care conțin interfețe foarte bogate valabile în sistemele de operare windows.
- RSACryptoserviceprovider este implementarea standard a criptosistemului RSA. Această bibliotecă suportă chei de dimensiuni între 384 și 16384 de biți. Această librerie a fost folosită la comunicarea TCP dintre clientul de KeePass și serverul din Python.

4.2.2 Vault secret service

Vault secret service [12] stochează și controlează accesul la tokeni, parole, certificate, chei API și alte secrete. Printr-un API unificat, userii pot accesa un storage criptat, să genereze credențiale AWS IAM/STS sau baze de date SQL/NoSQL, certificate X.509 sau credențiale SSH.

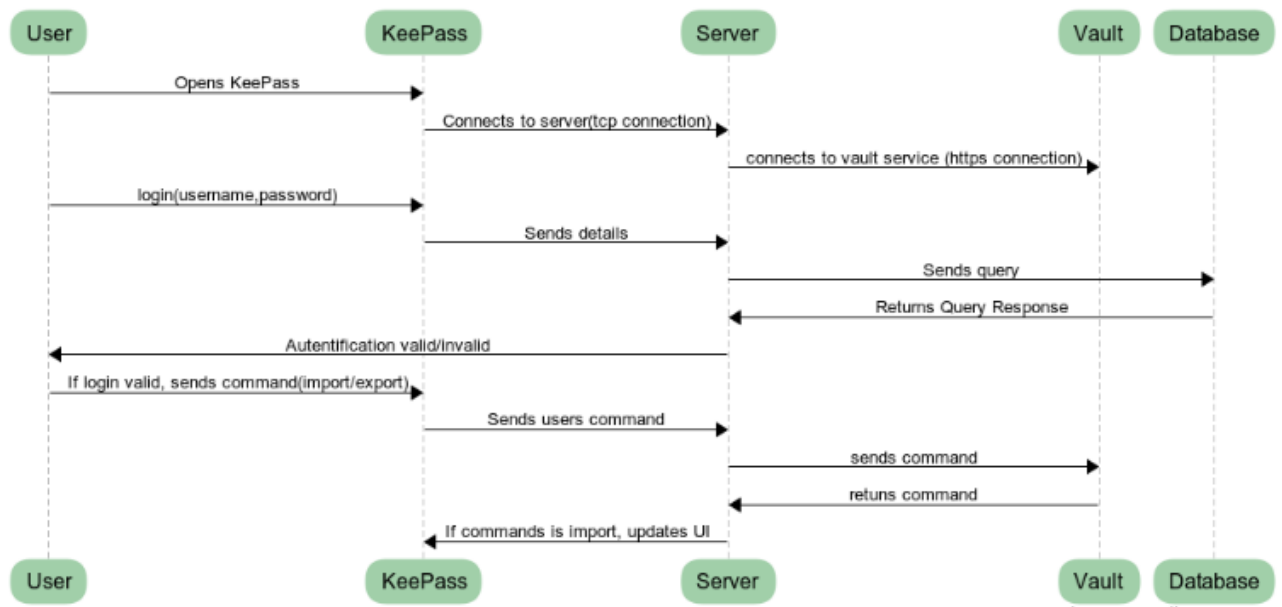
- Vault poate păstra secrete existente, sau poate genera dinamic secrete noi pentru a controla accesul la resurse third-party sau să furnizeze credențiale limitate de timp pentru infrastructură. Toate datele din Vault sunt criptate. Dinamic, sunt generate secrete temporare, iar după o anumită durată de timp, vault le va șterge. Se folosesc și politici de control al accesului pentru a furniza control strict asupra persoanelor care au acces la secrete.
- Secretele ținute în Vault pot fi updatate oricând, la orice oră. Dacă se folosește „encryption-as-a-service”, cheile pot fi transformate oricând, păstrând abilitatea de a decrypta valori criptate în trecut cu o altă cheie.
- De asemenea, Vault conține loguri de audit pentru interacțiunea cu clienții autentificați: tokeni de autentificare, creare de tokeni, acces la secrete, revocarea secretelor.

4.2.3 Python

Python [37] este un limbaj de programare creat în 1989 de programatorul olandez Guido van Rossum. Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul orientat obiect, dar permite și programare imperativă, funcțională sau procedurală.

4.3 Arhitectura aplicației

Aplicația se bazează pe arhitectura client-server, în sensul că utilizatorul va interacționa cu aplicația desktop (KeePass), iar aceasta va trimite cererile clientului către serverul TCP de Python urmând ca acesta să folosească API-ul de la Vault secret service, acest lucru fiind posibil doar dacă utilizatorul se autentifică cu succes.



Comunicarea dintre clientul de KeePass și serverul în Python fiind TCP, datele se vor trimite sub formă de flux de octeți, iar din scriptul de Python către serviciul vault, datele se vor trimite sub formă de Json.

Autentificarea unui user se va face folosind o bază de date PostgreSQL. Am ales această variantă deoarece în interiorul serverului de baze de date nu vor fi ținute decât username-ul utilizatorului și parola acestuia (asupra parolei se va aplica o funcție hash sha256). După autentificare, userii vor putea să exporte parolele care au fost selectate în KeePass iar la export, toate parolele care sunt ținute în serviciu vor fi aduse către aplicație. Pentru a scrie și a citi din serviciul vault, fiecare user va avea nevoie de un backend, acesta reprezentând locul în care se vor afla parolele (fiecare backend va reprezenta chiar numele contului utilizatorului).

La înregistrare, în serviciul vault se va monta un nou sistem de fișiere în care sunt ținute secretele acestuia.

Pentru ca scriptul de Python să aibă acces la instanța de Vault, acesta trebuie să se autentifice folosind un token generat de vault, și va avea nevoie să treacă de o barieră. Barierea reprezintă faptul că la inițializare, serviciul vault nu poate fi accesat de nimic, doar dacă se îndeplinește o anumită condiție. Serverul la pornire va genera 5 chei iar pentru a accesa serviciul, este necesară cunoașterea a măcar 3 dintre aceste chei. Un astfel de exemplu este prezentat în imaginea de mai jos:


```

C:\Users\stfcr>vault init
Unseal Key 1: V2fklNd9PKk8Qk8igPVw5uOwWslw+LveQyQAQ009EQ7Dx
Unseal Key 2: Id9V/DLMGMkAGcdhfjkXt67taZrp/7+ZXC0bhzPr0XVz
Unseal Key 3: C4eD7T2JfIFiuWmvwMIw6wNDvo/950R7CcJacHr8HPSA
Unseal Key 4: 6TbdQwg0L4DHtru7VfY6qOnmP2fyQowni20waKt6edZ3
Unseal Key 5: IDp7lH1rUJfZWufSGiHmeL+H+8kA7H5jWwNDVedQQ4KT
Initial Root Token: 8ddaa675-5efa-ab3b-8055-46075af1a0da

Vault initialized with 5 keys and a key threshold of 3. Please
securely distribute the above keys. When the vault is re-sealed,
restarted, or stopped, you must provide at least 3 of these keys
to unseal it again.

Vault does not store the master key. Without at least 3 keys,
your vault will remain permanently sealed.

C:\Users\stfcr>vault status
Sealed: true
Key Shares: 5
Key Threshold: 3
Unseal Progress: 0
Unseal Nonce:
Version: 0.7.3

High-Availability Enabled: true
Mode: sealed

```

Serviciul poate fi deblocat de la linia de comandă folosind comanda *vault unseal* dar și din cod folosind funcția *client.unseal(cheia)*. În interiorul backend-ului parolele vor fi ținute în clar, dar acest lucru nu va reprezenta o problemă deoarece datele nu pot fi accesate decât de entitatea care reușește să treacă pragul celor 3 chei de decriptare. Datele din storage arată în felul următor

```

C:\Users\stfcr>vault read secret/Stefanlala1
Key          Value
---          -
refresh_interval 768h0m0s
value          [default1,default2]

```

Pentru ca datele din KeePass să ajungă până în serviciul Vault, datele din KeePass trebuie să fie întâi selectate, după care trebuie apăsat butonul de Export Passwords. Pentru a putea selecta acele date trebuie să folosim următoarele lucruri:

```
private IPluginHost m_host = null;
```

Așa vom inițializa obiectul necesar operațiilor asupra aplicației KeePass. În acest obiect se află toate metodele necesare modelării aplicației.

```

1 private void OnMenuExport(object sender, EventArgs e)
2     {
3         msg.SendMessage("Export My Passwords");
4         PwEntry [] pwEntries = m_host.MainWindow.GetSelectedEntries();
5         for(int i=0; i<pwEntries.Length; i++)
6         {
7             string title = pwEntries[i].Strings.Get(PwDefs.TitleField).ReadString();

```

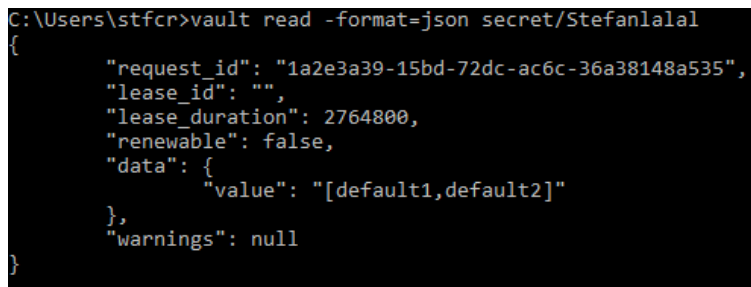
```

8         string username = pwEntries[i].Strings.Get(PwDefs.UserNameField).ReadString();
9         ProtectedString protectedString = pwEntries[i].Strings.GetSafe(PwDefs.PasswordField);
10        byte[] bytes = protectedString.ReadUtf8();
11        UTF8Encoding encoding = new UTF8Encoding();
12        string passwords = encoding.GetString(bytes);
13
14        msg.SendMessage(title);
15        msg.SendMessage(username);
16        msg.SendMessage(passwords);
17    }
18    }

```

La apăsarea butonului Export Passwords se va apela metoda OnMenuExport, care prin apelul funcției GetSelectedEntries() va returna un vector cu acele obiecte care sunt selectate. A urmat interacția prin acele obiecte, formatarea lor și trimiterea lor către server.

Pentru ca datele din Vault să ajungă în KeePass, va trebui doar să apăsăm butonul de Import Passwords. Datele vor fi scoate din Vault într-un format JSON, datele arătând exact ca în formatul de mai jos:



```

C:\Users\stfcr>vault read -format=json secret/Stefanlala
{
  "request_id": "1a2e3a39-15bd-72dc-ac6c-36a38148a535",
  "lease_id": "",
  "lease_duration": 2764800,
  "renewable": false,
  "data": {
    "value": "[default1,default2]"
  },
  "warnings": null
}

```

Tot ce a trebui, a fost parsarea acelui JSON, și trimiterea datelor pe rând înapoi clientului. La primirea datelor, am adăugat fiecare obiecte primit (titlu, username, parolă) în câte o listă, după care vom apela o altă metodă din m_host pentru a updata UI-ul din KeePass.

Codul prin care se primesc datele este următorul:

```

1  private void OnMenuImport(object sender, EventArgs e)
2  {
3      msg.SendMessage("ImportMyPasswords");
4      FieldManipulator field = new FieldManipulator();
5      string answer = msg.ReceiveMessage();
6      int length = Int32.Parse(answer);
7      List<string> titles = new List<string>();
8      List<string> usernames = new List<string>();
9      List<string> passwords = new List<string>();
10     for (int i = 0; i < length; i++)
11     {

```

```

12         string title = msg.ReceiveMessage();
13         string username = msg.ReceiveMessage();
14         string password = msg.ReceiveMessage();
15         titles.Add(title);
16         usernames.Add(username);
17         passwords.Add(password);
18     }
19     for(int i = 0; i < length; i++)
20     {
21         field.writeEntry(m_host, titles[i], usernames[i], passwords[i]);
22         Thread.Sleep(2000);
23     }
24     msg.closeClient();
25
26 }

```

Iar pentru adaptarea UI-ului:

```

1 public void writeEntry(IPluginHost m_host, string title, string username, string password)
2     {
3         PwEntry entry = new PwEntry(true, true);
4         entry.Strings.Set(PwDefs.TitleField, new ProtectedString(false, title));
5         entry.Strings.Set(PwDefs.UserNameField, new ProtectedString(false, username));
6         entry.Strings.Set(PwDefs.PasswordField, new ProtectedString(false, password));
7
8         m_host.Database.RootGroup.AddEntry(entry, true);
9         m_host.MainWindow.UpdateUI(false, null, true, m_host.Database.RootGroup, true, null, true);
10
11     }

```

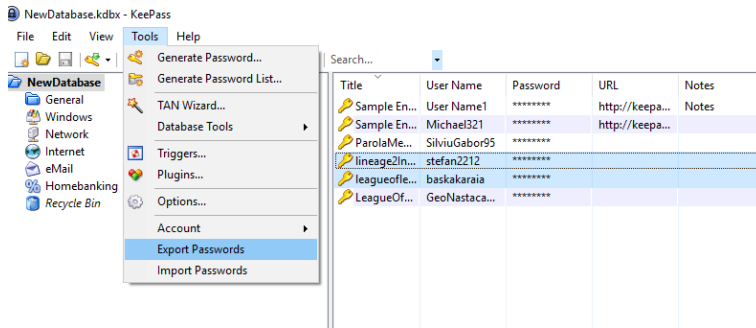
4.4 Utilizarea aplicației

În această secțiune, vor fi prezente și anumite lucruri despre KeePass, nu doar despre plugin. La inițializarea unei noi baze de date, veți fi nevoiți să introduceți o parolă de criptare. Această parolă trebuie ținută minte deoarece cu ea se poate decripta baza de date locală și se pot vizualiza parolele.

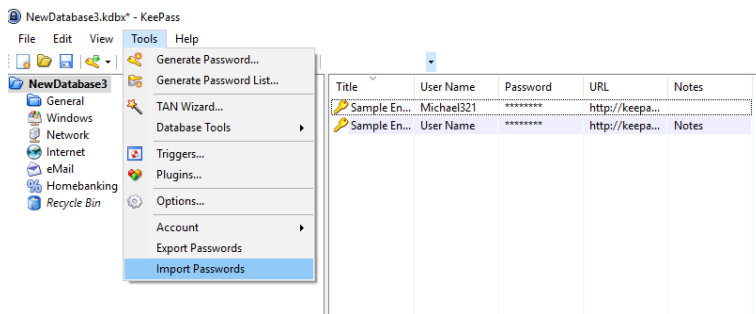
Pentru folosirea pluginului, utilizatorul va fi nevoit să se ducă în dreptul meniului de tools, iar acolo se va observa butonul de account. De îndată ce veți duce cursorul mouse-ului către butonul de account, vor apărea login și register.

Odată logat, utilizatorul va putea să înceapă să selecteze parolele (acest lucru se poate face ținnd apăsată tasta ctr + click stânga pe itemul dorit). După ce Aveți selectate itemele, puteți apăsa

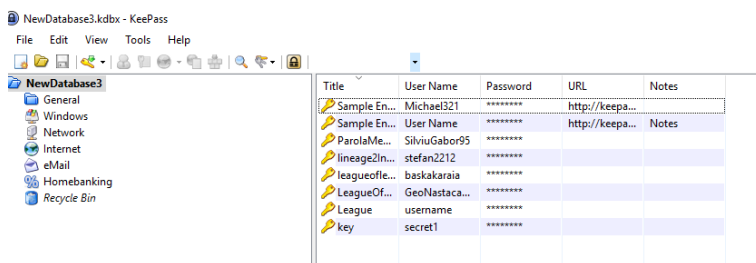
butonul de export.



După ce vom trece pe o altă bază de date, parolele care se aflau în cealaltă bază de date nu mai sunt vizibile și aici, dar cu opțiunea de import vom putea obține din nou parolele.



Importul va dura puțin mai mult, deoarece UI-ul trebuie să se reactualizeze la primirea fiecărei parole noi, dar după câteva secunde, baza de date va avea toate parolele dorite



Concluzii

Pe parcursul acestor capitole, am detaliat o serie de derivări ale algoritmului RSA, moduri de implementare ale acestuia, cum depindem foarte mult de modul în care generăm cheile (numere prime cât mai mari, exponentul de criptare să nu fie nici foarte mare, nici foarte mic), îmbunătățiri, precum și vulnerabilități și modalități prin care putem să le exploatăm.

De asemenea, am reușit să înțelegem și o parte din algoritmi fundamentali care aparțin de teoria numerelor (TCR, Euclid-extins, invers modularul, ecuație diofantică etc), precum și alte aspecte matematice (polinoame, grupuri).

Am reușit să detaliem și o problemă matematică care este ușoară într-un sens, dar intractabilă în celălalt (factorizarea unui număr), deci am reușit să vedem cât de greu este, chiar și cu puterea computațională actuală, să reușim să factorizăm un număr foarte mare.

Scopul acestei lucrări a fost de a-l introduce pe cititor în domeniul criptografiei, și de a-i forma o bază solidă în ceea ce privește criptografia cu chei publice, mai precis RSA.

Pe partea de aplicație am reușit integrarea unui plugin de KeePass cu un serviciu Vault, această implementare fiind prima datorită versiunii vechi de .NET (versiunea 3.5 care la momentul lansării nu suporta comunicarea cu un serviciu). Am văzut cum se poate construi o aplicație folosind atât tehnologii low level (windows system forms, sockets) dar și o parte high level (API-ul în Python ce permite comunicarea cu serviciul vault). Pe lângă toate acestea, am văzut și cum poate fi folosit algoritmul RSA pentru securizarea unei aplicații.

Aplicația poate constitui un punct de plecare pentru alte pluginuri similare ce doresc comunicarea cu un serviciu. De asemenea, această aplicație poate fi extinsă și în alte domenii de cercetare, unul dintre acestea fiind „Încrederea în Rețele”. În acest caz vom avea un cluster de noduri (fiecare nod reprezentând un user) iar la adăugarea unui nou nod în cluster vom putea să-i distribuim anumite informații doar dacă acel nod este considerat de încredere.

Bibliografie

- [1] <https://ro.wikipedia.org/wiki/Criptografie>
- [2] Y.Desmedt and Y.Frankel. Share Generation of Authenticators and Signature. In Crypto 91, LNCS 576, paginile 457-469.
- [3] M. J. Wiener. Cryptanalysis of short RSA secret exponents. IEEE Transactions on Information Theory, 36(3):553–558, May 1990
- [4] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In P. C. van Oorschot, editor, USENIX Security Symposium, pages 45–60. USENIX Association, 2008.
- [5] <https://ro.wikipedia.org/wiki/RSA>
- [6] D.Boneh și M.Franklin. Efficient Generation of Shared RSA keys. In Crypto 97, LNCS 1233, paginile 425-439. Springer-Verlag, 1997
- [7] A. Menezes, P. Van Oorschot, and S. Vanstone. Handbook for applied cryptography.
- [8] D.Catalanoși S.Halevi. Computing Inverses over a Shared Secret Modulus. In Eurocrypt '00, LNCS 1355, paginile 190-207. Springer Verlag, 2000.
- [9] D.Coppersmith. Finding a small root of a univariate modular equation. In U. M. Maurer, editor, EUROCRYPT, volume 1070 of Lecture Notes in Computer Science, pagini 155-165. Springer, 1996
- [10] D. Coppersmith. Small solution to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, paginile 233-260, 1997.

- [11] D. Coppersmith, M. K. Franklin, J. Patarin, and M. K. Reiter. Low-exponent RSA with related messages. In U. M. Maurer, editor, EURO-CRYPT, volume 1070 of Lecture Notes in Computer Science, paginile 1–9. Springer, 1996
- [12] <https://www.hashicorp.com/blog/Vault-announcement/>
- [13] N. Heninger and H. Shacham. Improved RSA private key reconstruction for cold boot attacks. Cryptology ePrint Archive
- [14] V. Shoup. Practical Threshold Signatures. In Eurocrypt '00, LNCS 1807, paginile 207–220. Springer-Verlag, 2000
- [15] D. Boneh and H. Shacham. Fast variants of RSA. Cryptobytes, 5(1):1–9, 2002.
- [16] A. Fiat. Batch RSA. In Proceedings of Crypto '89, volume 435, paginile 175–185. Springer-Verlag, 1989
- [17] T. Collins, D. Hopkins, S. Langford, and M. Sabin. Public Key Cryptographic Apparatus and Method. US Patent, 1997
- [18] A. Shamir. How to Share a Secret. Communications of the ACM, 22:612–613, November 1979
- [19] https://en.wikipedia.org/wiki/Lenstra_elliptic_curve_factorization
- [20] M. Wiener. Cryptanalysis of short RSA secret exponents. IEEE Trans. on Info. Th., 36(3):553–558, 1990.
- [21] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0.292}$. IEEE Trans. on Info. Th., 46(4):1339–1349, Jul. 2000
- [22] <https://profs.info.uaic.ro/siftene/Wiener.pdf>
- [23] [https://ro.wikipedia.org/wiki/Grup_\(matematică\)](https://ro.wikipedia.org/wiki/Grup_(matematică))
- [24] M. K. Franklin and M. K. Reiter. A linear protocol failure for RSA with exponent three. CRYPTO '95 rump session, 1995.
- [25] T. Takagi. Fast RSA-type cryptosystem modulo $p \cdot k \cdot q$. In Advances in cryptology, volume 1462, paginile 318–326. Springer-Verlag, 1998.

- [26] S.Sarkar și S.Maitra. Partial key exposure attacks on RSA and its variants by guessing a few bits of one of the prime factors. To appear in B-KMS, the Bulletin of Korean Mathematical Society.
- [27] M. J. Hinek, M. J. Low și E. Teske. On some attack on multi-prime RSA. K.Nyberg și H. M. Heys, editorii, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, paginile 385-404. Springer, 2002.
- [28] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.
- [29] Ferucio Laurențiu Țiplea. *Introducere în criptografie*. Anul II semestrul 1, Universitatea Alexandru Ioan Cuza, facultatea de Informatică.
- [30] <https://profs.info.uaic.ro/fltiplea/> *Fundamente algebrice ale informaticii*. Anul I semestrul 2, Universitatea Alexandru Ioan Cuza, facultatea de Informatică.
- [31] T. Takagi. A fast RSA-type public-key primitive modulo pq using Hensel lifting. *IEICE Transactions*, 87-A(1):94–101, 2004.
- [32] N. Kunihiro and K. Kurosawa. Deterministic polynomial time equivalence between factoring and key-recovery attack on Takagi's RSA. In T. Okamoto and X. Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, paginile 412–425. Springer, 2007.
- [33] A. May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, paginile 213–219. Springer, 2004.
- [34] https://ro.wikipedia.org/wiki/Criptografie_asimetrică
- [35] https://en.wikipedia.org/wiki/.NET_Framework
- [36] [https://msdn.microsoft.com/en-us/library/system.windows.forms.form\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.forms.form(v=vs.110).aspx)
- [37] <https://ro.wikipedia.org/wiki/Python>