

**UNIVERSITATEA DE MEDICINĂ, FARMACIE, LUCRARE DE LICENȚĂ ȘTIINȚE ȘI
TEHNOLOGIE „G.E. PALADE” DIN TÂRGU MUREȘ**

**FACULTATEA DE INGINERIE ȘI
TEHNOLOGIA INFORMAȚIEI**

Candidat: Ciucălău Alexandru

Programul de studii : Informatică

Anul absolvirii : 2022

**Coordonator științific :
Lector dr. ing. Bogdan Crainicu**

Viza facultății

a) Tema lucrării de licență :

Infrastructură bazată pe chei publice implementată în sistemul medical

b) Problemele principale tratate :

- Implementarea componentelor necesare pentru abstractizarea datelor
- Folosirea componentelor în vederea implementării unei soluții care să administreze datele necesare creării unei rețete medicale.

c) Bibliografia recomandată :

- W. Diffie. The first ten years of public-key cryptography. Proceedings of the IEEE, 76(5): 560-577, 1988. Doi: 10.1109/5.4442
- J. A. Buchmann. Introduction to public key infrastructures. 2013. doi: 10.1007/978-3-642-40657-7
- Leighton Johnson. Security Controls Evaluation, Testing and Assessment HandBook(Second Edition), pages 471-536, 2020. URL <https://www.sciencedirect.com/science/article/pii/B9780128184271000112>.

d) Termene obligatorii de consultații :

de 2 ori pe lună

e) Locul și durata practicii :

laboratoarele specifice programului de studiu

Primit tema la data de : 19.01.2023

Termen de predare : 20.01.2023

Semnătura Director Departament

Semnătura coordonatorului

Semnătura candidatului

UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI
TEHNOLOGIE "GEORGE EMIL PALADE" DIN TÂRGU MUREȘ
FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI
Programul de studii: INFORMATICĂ



LUCRARE DE LICENȚĂ

Arhitectura unui sistem de securitate bazat pe chei publice

Student:

Ciucălău Alexandru

Coordonator științific:

Lector universitar dr. ing. Crainicu Bogdan

Februarie 2022

Conținut

1	Introducere	2
1.1	Introducere în tematica lucrării	3
1.2	Contribuții personale	4
2	Criptografia asimetrică	5
2.1	Criptarea asimetrică a cheii	6
2.2	Algoritmi asimetrici de criptare	8
2.3	Principii de funcționare a criptării RSA	8
2.3.1	Generare cheie publică	10
2.3.2	Generare cheie privată	11
3	Infrastructuri bazate pe chei publice	13
3.1	Securitate Web	13
3.2	Securizare email	14
3.2.1	S/MIME	14
3.2.2	PGP	16
3.3	Semnare de software	18
3.4	VPN	19
3.5	Semnături electronice	22
4	Proiectarea unui sistem de semnare digitală bazat pe chei publice	23
4.1	Certificatele X.509	23
4.2	Implementarea certificatelor în sistemul medical	24
4.3	Administrarea datelor în cadrul infrastructurii medicale	25
4.4	Autoritatea de certificate	27
4.5	Înregistrarea cererilor de rețetă a pacienților	30
4.6	Generarea certificatelor digitale	33
4.7	Generarea semnăturilor digitale	34
4.8	Verificarea semnăturilor digitale	35
5	Concluzii	39
	Bibliografie	40

Abstract

Criptografia bazată pe chei publice a fost inventată în urmă cu mai mult de 30 de ani. Rolul acestui tip de proces este de a ascunde adevăratul conținut al unor informații, iar entitățile capabile de a descifra mesajul sunt exclusiv expeditorul și destinatarul. Pentru a reuși securizarea datelor în acest mod este nevoie de dispozitive computaționale care pot rezolva operații matematice complexe, folosind numere mari, pentru a cripta și respectiv, decripta datele. Ca și instrumente auxiliare, procesul de criptare asimetrică are nevoie de o pereche de chei pentru a funcționa. Din punct de vedere etimologic, termenul de *criptografie* provine din îmbinarea a două cuvinte care provin din limba greacă, și anume *ascuns* și *scris*. Abia în timpurile actuale criptografia a început să fie pusă în practică de publicul larg, până în anii '70 neavând parte de atenția meritată, în domeniul securității. Printre principalele cauze se numără problema partajării cheilor deoarece comunicarea la distanță nu era foarte dezvoltată din punct de vedere aplicativ. Înainte ca acest proces de criptografie care folosește chei publice să fie inventat, securizarea datelor se realiza cu aceeași cheie pentru criptare și pentru decriptare. Odată ce această problemă care implică siguranța datelor a fost rezolvată cu apariția criptografiei asimetrice, a apărut conceptul de *pereche de chei*, dintre care una publică, ce poate fi partajată în mod deschis și una privată ce trebuie menținută secretă.

Capitolul 1

Introducere

În mod tradițional, criptografia se baza pe schimbul unor chei secrete înainte de orice comunicare securizată, ceea ce a făcut ca aplicarea criptografiei în rețele deschise, ca și internetul, să fie foarte dificilă. Ca și alternativă, criptografia bazată pe chei publice asigură comunicarea sigură a unor entități care nu au avut niciun contact anterior. În contextul societății actuale, cum internetul are mai mult de 2 miliarde de utilizatori, acest lucru este extrem de important. Pe lângă acest avantaj, criptografia bazată pe chei publice permite tehnici care nu se găsesc în criptografia tradițională, cel mai important concept fiind semnăturile digitale. De fapt, securitatea pe internet nu ar fi putut exista fără semnături digitale, ca exemplu avem necesitatea de a autentifica descărcările de software sau actualizările. Acest lucru ne asigură că în zilele noastre sau în viitor nu este sau nu va exista securitate IT fără criptografia bazată pe chei publice.

Deși criptografia bazată pe chei publice nu constă în schimbul unei perechi de chei secrete, administrarea corectă a cheilor este de o importanță vitală pentru securitate. În criptografia bazată pe chei publice se folosesc perechi de chei private și publice. Primul aspect al administrării corecte ale cheilor este menținerea cheilor private secrete. Acest lucru este mai ușor decât protejarea cheilor secrete în criptografia tradițională, deoarece nu este nevoie de trimiterea cheilor private pe canale care nu sunt sigure. Protecția acestor chei rămîne totuși o provocare, dat fiind faptul că încă există miliarde de calculatoare care au stocate chei private în memoria lor. Al doilea aspect al administrării corecte ale cheilor este garantarea autenticității cheilor publice, lucru la fel de important ca păstrarea confidențialității cheilor private. De exemplu, în cazul în care cheia de verificare a unei semnături publice aparținând unui comerciant de software ar putea fi înlocuită cu cheia publică a unei firme concurente, semnăturile de software nu ar mai avea întrebuințare, întrucât firma adversă ar putea semna software-ul în numele comerciantului.

Pentru a înțelege complet criptografia asimetrică, este necesar studiul infrastructurii care administrează perechile de chei în cadrul criptografiei bazate pe chei publice, așa numitele infrastructuri de chei publice (PKIs). [1]

1.1 Introducere în tematica lucrării

O sarcină importantă a unei infrastructuri de chei publice (PKI) este de a putea oferi dovada autenticității cheilor publice. O modalitate importantă pentru a putea justifica aceste dovezi este folosirea certificatelor de autenticitate.

CertIFICATELE sunt structuri de date care fac legătura dintre cheile publice și entități și sunt semnate de o parte terță. Conceptul de certificat de autenticitate poate fi exemplificat folosind următoarea situație: presupunem că utilizatorul Alice dorește să verifice semnătura digitală emisă de utilizatorul Bob. Astfel, ea interoghează serviciul de directoare, obține cheia publică a lui Bob și trebuie să se convingă de autenticitatea acelei chei. În acest fel, încrederea utilizatorului care dorește să verifice autenticitatea cheii publice se reduce la a avea încredere în autoritatea de certificare care a produs cheia.

În continuare vom lista componentele minime din care este compus un certificat.

1. Numele subiectului de care aparține cheia publică din certificat. Acesta poate fi și un pseudonim.
2. Cheia publică care corespunde entității.
3. Algoritmul criptografic cu care trebuie să fie folosită cheia publică.
4. Numărul de serie al certificatului.
5. Perioada de valabilitate a certificatului
6. Numele persoanei sau companiei care a semnat certificatul.
7. Restricțiile care se aplică folosirii cheii publice din certificat. De exemplu, întrebuințarea poate fi redusă la verificarea identității acelei entități.

După cum vom observa mai departe, certificatele pot conține mult mai multe informații. Conținutul unui certificat este semnat de partea terță care a emis certificatul, iar semnătura este anexată acestuia.

În general, cel care emite un astfel de certificat este diferit de subiectul care îl va folosi. Totuși, există certificate în care subiectul care îl folosește și autoritatea care l-a emis sunt aceleași.

Acestea sunt, ca exemplu, folosite în cazul în care cine a produs certificatul și-a schimbat politica. Un caz special de certificate emise de către cel ce le și folosește sunt certificatele autosemnate care au o proprietate adițională ca acea cheie publică confirmată este, de asemenea, și cheia publică necesară pentru a verifica semnătura de pe certificat. Certificatele autosemnate sunt folosite pentru a face cheile publice aparținând unei entități care le-a emis, să fie disponibile unei aplicații ce poate procesa chei publice aparținând unor certificate.

1.2 Contribuții personale

Pentru a crea o infrastructură bazată pe chei publice voi alege un cadru de aplicabilitate unde este necesară validarea documentelor pentru a demonstra autenticitatea datelor transmise.

În domeniul medical, de exemplu, fluxul de date pentru obținerea unei rețete medicale care să folosească în scopul obținerii unui tratament avizat de către medic poate fi securizat folosind certificatele digitale.

Îndeplinirea scopului acestei lucrări reprezintă crearea unei prescripții medicale care să fie autorizată de cele 3 părți, și anume, pacienți, doctori și farmacii.

Gestionarea certificatelor digitale se va face în cadrul bazei de date destinate stocării tuturor certificatelor generate în cadrul aplicației în felul următor:

1. Certificatele farmaciilor sau a doctorilor sunt create în momentul înregistrării în aplicație. De asemenea, dacă perioada de valabilitate expiră se poate actualiza certificatul, generându-se altă pereche de chei.
2. Certificatele pacienților sunt create de către aplicație în momentul în care doctorul își înregistrează un pacient care dorește să beneficieze de funcționalitățile aplicației și să înainteze o cerere pentru obținerea unei rețete.
3. Emitentul certificatelor folosite în cadrul aplicației este infrastructura medicală creată, care va administra semnăturile digitale ale utilizatorilor care vor să includă în sistemul lor această digitalizare.
4. Revocarea dreptului de a folosi certificatul pentru a obține o semnătură digitală se va face în momentul în care verificarea nu are succes, verificare efectuată în fiecare etapă a fluxului de date, după efectuarea cererii pacientului. Revocarea acestui drept are ca și consecință ștergerea din baza de date a entității înregistrate.

Capitolul 2

Criptografia asimetrică

Criptografia asimetrică, numită și criptografie bazată pe chei publice, este o alternativă a criptografiei clasice. Această formă de criptografie este folosită într-o arie foarte largă a comunicării, în continuă expansiune, în care informațiile sunt transmise frecvent între diferite părți de comunicare. Principiile de funcționare ale criptografiei asimetrice sunt următoarele:

1. Fiecare utilizator are două chei: o cheie publică și o cheie privată.
2. Ambele chei sunt asociate matematic.
3. Cheile publice sunt disponibile pentru toată lumea. Cheia privată este menținută secretă.
4. Ambele chei sunt necesare pentru a realiza o operație. De exemplu, informațiile criptate folosind cheia privată sunt decriptate cu ajutorul cheii publice. Viceversa, informațiile criptate folosind cheia publică sunt decriptate cu ajutorul cheii private.
5. Criptarea datelor cu cheia privată creează o *semnătură digitală*. Aceasta asigură faptul că mesajul a venit de la emitătorul stabilit, deoarece numai el are acces la cheia privată pentru a putea crea acea semnătură.
6. Un plic digital reprezintă semnarea mesajului cu cheia publică a destinatarului. Acest plic digital are ca scop controlul accesului prin asigurarea faptului că numai destinatarul, care deține cheia privată necesară, poate deschide plicul, concept cunoscut și ca *autentificarea destinatarului*.
7. În cazul în care cheia privată a fost descoperită, o nouă pereche de chei trebuie generată.

[2]

Criptografia asimetrică este deseori utilizată în schimbul de chei secrete, urmând să fie folosită criptografia simetrică pentru a cripta datele. În cazul unui schimb de chei, o parte creează cheia

secretă și o criptează folosind cheia publică a destinatarului, urmând ca acesta să decripteze datele cu cheia lui privată. Restul comunicării va continua folosind cheia secretă pe post de cheie de criptare. Criptarea asimetrică este folosită în schimbul de chei, securitatea email-urilor, securitate Web și alte sisteme de criptare care necesită schimbul de chei într-o rețea publică.

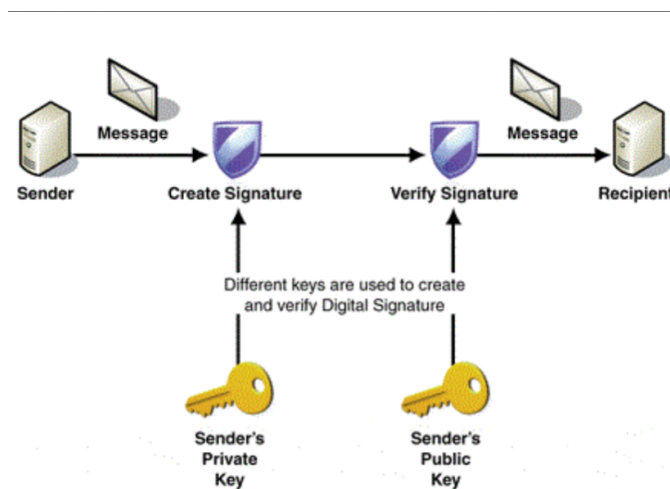


FIGURE 2.1: Criptografie asimetrică

[3]

Ideea de bază prin care acest sistem criptografic oferă o siguranță puternică în diferitele arii de aplicare se datorează în principal pe funcțiile matematice care se rezolvă ușor într-un sens, însă, necunoscând cheia secretă, rezolvarea funcțiilor în sens opus este foarte dificilă. Faptul că cele două chei sunt generate simultan și sunt interconectate matematic poate reprezenta în același timp și o vulnerabilitate, astfel relația dintre cele două chei trebuie să fie una cât mai complexă pentru a preveni aflarea cheii private folosind-o cea publică.

2.1 Criptarea asimetrică a cheii

Presupunem că o entitate care deține o pereche de chei dorește criptarea unui mesaj folosind cheia publică a destinatarului, prin urmare, mesajul va putea fi decriptat exclusiv cu acea cheie privată asociată. Altfel, dacă realizăm criptarea datelor folosind cheia privată, vom avea nevoie de cheia publică pentru a decripta datele. Această situație nu exemplifică o criptare, dar în acest mod ne convingem de autenticitatea expeditorului sau a datelor transmise. În continuare vom explica structura algoritmică după care funcționează criptarea asimetrică:

1. Date + Cheie publică = Date criptate
2. Date criptate + Cheie privată = Date Decriptate

3. Date + Cheie privată = Date semnate
4. Date semnate + Cheie publică = Autentificare

Spre deosebire de criptografia simetrică care oferă confidențialitate, criptarea asimetrică pune la dispoziție alte avantaje precum: integritate, autenticitate și non-repudiare. Pentru a realiza o criptare a datelor care să ofere aceste proprietăți este necesară folosirea unui algoritm bine structurat prin care sunt legate cele două chei. Pentru a crea o relație matematică puternică se folosesc în general chei de o dimensiune mai mare, precum 128 biți sau 256 biți, cu scopul de a face aflarea cheii nerealizabilă, cu toate că este cunoscută cheia publică asociată. De asemenea, algoritmul utilizat trebuie să fie specific pentru scopul în care se realizează criptarea. Astfel, există diferiți algoritmi de criptare asimetrică pentru diferite întrebuințări:

Acordul cheie Diffie-Hellman	Face posibil pentru 2 entități care nu se cunosc să partajeze o cheie secretă sau o informație pe un canal nesigur.
RSA (Rivest Shamir Adleman)	Permite criptarea cheilor publice și este folosit pentru comunicarea informațiilor secrete și pentru semnături digitale.
ECC (criptografie curbă eliptică)	Proprietățile sunt asemănătoare cu RSA, se bazează pe o pereche de chei pentru criptarea și decriptarea traficului web.
El Gamal	Este folosit ca parte dintr-un criptosistem hibrid, pentru a cripta o cheie simetrică. Pentru schimbul de semnături și chei digitale.
DSA (Algoritmul semnăturii digitale)	Folosit pentru a crea o semnătură electronică, astfel ajută la generarea unei perechi de numere folosite ca semnătura digitală.

[4] [5]

În criptografia asimetrică, algoritmi au un timp de execuție mai mare, față de algoritmi simetrici care folosesc operații matematice simple pe biți pentru criptare sau decriptare. Pentru a promite siguranța este necesară efectuarea acestor funcții matematice de mai multe ori, setul de biți trecând printr-o suită de combinații și permutări în cazul criptografiei simetrice.

Ca eficiență, în comparație cu criptografia simetrică, algoritmi asimetrici sunt mai lenti în execuție din cauza relațiilor și funcțiilor matematice mai complexe, care folosesc mai mult din capacitatea CPU-ului. Cu toate acestea, criptografia asimetrică reușește să ofere avantaje diverse, printre care și un sistem de comunicare a cheilor mai ușor de controlat decât un sistem simetric. Cauza acestui avantaj este faptul că, în momentul când se dorește transmiterea cheii, se folosește o singură cheie publică pentru a fi trimisă către fiecare destinatar, în locul unei chei secrete pentru fiecare.

2.2 Algoritmi asimetrici de criptare

În general, construcția unui sistem criptografic bazat pe chei publice se realizează folosind ecuații matematice cu complexitate ridicată, care calculează o funcție inversă a altei funcții. În termeni matematici, acest tip de operații se numesc *unidirectionale*, parcurgerea acestora în sens invers fiind foarte dificilă, aproape nerealizabilă. În principiu, pentru a cripta mesajul pentru destinatar se folosește cheia publică a acestuia, iar pentru calculul funcției inverse (decriptare), se folosește cheia lui secretă. Pentru a se ajunge la cifruri folositoare în practică nu există multe operații matematice care să atingă scopurile descrise anterior, însă putem considera mai departe câteva din cele mai folosite sisteme cu cheie publică.

1. RSA : În acest scenariu de criptare se va folosi problema factorizării, numită și calculul factorilor primi ai unui număr întreg mare. Acest algoritm se bazează pe înmulțirea a două numere prime cu cifre mari.
2. El-Gamal : Se bazează pe problema logaritmului discret într-un câmp finit.
3. Curba eliptică : Se bazează pe problema logaritmului discret pe curbe eliptice într-un câmp finit.

Pentru calculul invers al factorizării sau al logaritmului discret este necesar un studiu și o rezolvare amplă a ecuațiilor, dar în cazul în care sunt folosite numere cu o lungime mare de cifre, calculul devine aproape imposibil.

Modalitatea de criptare RSA este în contextul societății actuale un standard fundamental al sistemelor de securitate și protecție a informațiilor. Aceasta metoda este, de asemenea, utilizată și recomandată în multiple standarde internaționale, precum X.509, S-HTTP, STT, SWIFT, SSL etc., dar și în cadrul sistemelor bancare de carduri sau pentru protecția de protocoale de rețea.

Cercetările științifice care au avut ca subiect metodele de criptare RSA și El-Gamal, au fost într-un număr ridicat, studiindu-se în detaliu capacitatea acestor modalități de criptare împotriva atacurilor, iar rezistența criptografică fiind determinată de lungimea cheii și diferiți parametrii. În urma cercetărilor s-a ajuns la concluzia că cei doi algoritmi de criptare sunt echivalenți în puterea criptografică și au viteza de operare asemănătoare. Considerând faptul că algoritmul bazat pe curba eliptică nu a fost îndeajuns de mult testat împotriva hacking-ului, folosirea modalităților RSA sau El Gamal rămâne preferabilă.

2.3 Principii de funcționare a criptării RSA

Principiul de bază al criptării RSA este acela de calcul unidirecțional, care este determinat de calculul numerelor prime mari folosite în acest algoritm. Un exemplu simplu poate fi construit

în urma înmulțirii a două numere prime mari, care va avea ca rezultat un număr mare ai cărui factori primi sunt foarte dificil de descoperit. Ca exemplu concret vom lua drept factori primi numerele 1013 și 1231, iar rezultatul operației 1013×1231 va fi 1.247.003, un număr care nu sugerează nimic ce ne poate îndrepta către aflarea factorilor primi inițiali, în afară de căutarea exhaustivă a acestora.

Din pricina faptului că operația dintre două numere prime este ușor de calculat într-un sens, dar exponențial de dificil în direcție opusă, această ecuație are numele de *funcție ușă capcană*. În același timp, putem foarte ușor realiza ideea că în timp ce pentru oameni rezolvarea în sens invers a ecuației este aproape imposibilă pentru oameni, puterea computațională a unui calculator îi oferă posibilitatea de a rezolva ușor ecuația exemplului de mai sus. Astfel, pentru a îmbunătăți capacitatea algoritmului de a rezista împotriva atacurilor malițioase realizate prin intermediul unui computer, algoritmul RSA a fost conceput să folosească un număr foarte mare, în funcție de numărul de biți folosiți în implementarea acestuia. De exemplu, folosirea a 2048 de biți în criptarea RSA determină chei cu o mărime de 617 cifre, care ar reduce considerabil capacitatea calculatorului de a afla informația necesară prin calculul invers al ecuației.

În continuare vom explica modalitatea de generare a numerelor prime folosite pentru criptarea asimetrică a cheii. Anterior am precizat folosirea *funcțiilor ușii capcană* ca și concept de bază a sistemelor criptografice care folosesc o pereche de chei, astfel putem spune despre aceste ecuații că ne permit prin caracteristicile lor partajarea cheilor publice fără a reprezenta un pericol pentru aflarea cheii private sau decriptarea mesajului. Totodată, aceste proprietăți asigură criptarea informațiilor cu o cheie într-o manieră care permite decriptarea acestora folosind exclusiv cheia asociată.

Criptarea RSA a unui mesaj începe cu generarea cheilor, iar pentru a le obține vom porni de la alegerea a **două numere prime p și q** rezultate dintr-un test de primordialitate precum evaluarea Miller-Rabin care este o problemă de probabilitate ce determină dacă un număr este posibil să fie prim.

Ca regulă de debut în alegerea numerelor p și q , e necesar ca ele să fie mari, iar diferența dintre cele două să fie de asemenea mare, pentru ca cele două să nu fie apropiate pentru a îngreuna cât mai mult aflarea lor. Cu toate acestea, vom relua totuși exemplul anterior în care folosim numere mai mici pentru a avea o ilustrare mai ușor de înțeles. Astfel, presupunem că în urma testului de primordialitate cele două numere vor fi 1013 și 1231. Al doilea pas al algoritmului este calculul modulului (n) folosind formula $n = p \times q$, unde $p = 1013$, iar $q = 1231$. Prin urmare, rezultatul obținut, n , va fi egal cu 1.247.003, acest număr urmând să fie folosit în **funcția totală a lui Carmichael**.

$$\gamma(n) = LCM(p-1, q-1)$$

În continuare vom explica această funcție împreună cu o exemplificare. Pentru a înțelege ecuația trebuie să menționăm ce reprezintă fiecare termen, așa că, $\gamma(n)$ este *totientul* lui Carmichael în funcție de n , iar LMC reprezintă *cel mai mic multiplu comun al lui $p - 1$ și $q - 1$* . Astfel, continuând exemplul de mai sus vom avea ca rezultat pentru $\gamma(1.247.003) = LCM(1013 - 1, 1231 - 1) = LCM(1012, 1230)$, iar cel mai mic multiplu comun al acestor doua numere este $\gamma(1.247.003) = 622380$.

2.3.1 Generare cheie publică

Odata ce obținem totientul celor două numere putem începe aflarea cheii publice. După standardele RSA, o cheie publică este formată dintr-un număr prim e ai cărui valoare este cuprinsă în intervalul 1 și rezultatul ecuației $\gamma(n)$, în exemplul anterior având valoarea 622380.

Datorită expunerii publice a acestei chei, alegerea aleatoare a numărului e nu este obligatorie. În general, în alte criptosisteme asemănătoare, valoarea este stabilită la 65537 din pricina faptului că alegând numere mai mari se reduce eficiența algoritmului. Pentru a continua exemplificarea eficient vom alege $e = 13$.

La final, informațiile criptate se vor numi *cifru text* c , iar derivarea mesajului în text integral m folosind cheia publică și următoarea formulă.

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

Revenind la exemplul nostru, alegem ca mesajul criptat să fie doar un număr, adică $m = 5$, astfel:

$$c = 5^{13} \bmod 1.247.003$$

După efectuarea operațiilor, cifrul text va avea valoarea $c = 1.134.191$. În final, după ce am folosit criptarea RSA a mesajului $m = 5$ folosind cheia publică vom obține textul criptat 1.134.191. Prin urmare, am avut mesajul initial "5" care dorim să rămână secret, iar după aplicarea algoritmului și a cheii publice am obținut rezultatul criptat 1.134.191. Acest mesaj poate fi transmis în siguranță către destinatarul care deține perechea de chei, iar mesajul poate fi decriptat doar folosind cheia privată pentru a vedea mesajul original.

2.3.2 Generare cheie privată

Din momentul în care mesajul a fost transformat în cifră text folosind cheia publică acesta poate fi decriptat doar cu cheia privată asociată. Pentru a compune cheia privată avem nevoie de două componente: d și n . Din calculele precedente am obținut n -ul, iar pentru a afla valoarea lui d vom folosi următoarea formulă:

$$d = 1/e * \text{mod} \gamma(n)$$

În exemplul nostru concret am ales ca e să fie egal cu 13, iar $\gamma(n)$ este 622380 obținut din *funcția totală a lui Carmichael*. Luând pas cu pas operația, prima dată va trebui să calculăm $1/e \text{ mod}$ care sugerează faptul că trebuie efectuat calculul **invers modular** de e , care în exemplul nostru este 13.

Prin urmare, acest lucru sugerează că nu vom mai folosi operația de modulo și vom folosi inversul modular care tradițional se rezolvă cu algoritmul euclidian extins. Astfel, înlocuind totul în formula noastră vom obține:

$$d = 1/13 * \text{mod} 622.380$$

După efectuarea operației de invers modular am aflat valoarea lui d care este 239.377, iar odată ce am obținut această valoare putem începe decriptarea mesajului cu ajutorul următoarei formule de calcul:

$$m = c^d \text{ mod } n$$

Înlocuind fiecare variabilă din exemplu, ecuația se reduce la rezolvarea următoarei relații:

$$m = 1.134.191^{239.377} \text{ mod } 1.247.003$$

După rezolvarea ecuației folosind un calculator RSA online obținem mesajul care era inițial de criptat, adică informațiile inițiale $m = 5$.

Supply Modulus: N

Supply Encryption Key and Plaintext message M:

Encryption Key: e

Plaintext Message to encode:

Encrypt

Plaintext Message in numeric form:

Encrypted Message in numeric form:

OR

Supply Decryption Key and Ciphertext message C:

Decryption Key: d

Ciphertext Message in numeric form:

1134191

Decrypt

Decrypted Message in numeric form:

5

Decrypted Message in text form:

FIGURE 2.2: Decriptare mesaj

Capitolul 3

Infrastructuri bazate pe chei publice

Proprietățile criptografiei asimetrice oferă acestui concept un rol important în schimbul de informații și securitate, astfel, în continuare vom descrie aplicații practice pentru care s-au dezvoltat diferite infrastructuri de chei publice (PKI).

3.1 Securitate Web

Pentru a garanta siguranța web, în mediul online sunt folosite protocoale de internet precum SSL și TLS, care sunt formate cu ajutorul criptografiei cu chei publice. Aceste protocoale reușesc să ofere confidențialitate utilizatorului și a datelor lui personale în timpul comunicării acestuia cu diferite servere cu care intră în contact.

O aplicație a protocolului TLS poate fi exemplificată în cadrul unei pagini web creată pentru a prezenta și vinde diferite produse. În acest caz, pentru a oferi un trafic sigur al utilizatorului pe site, organizația care deține magazinul deține un certificat X.509 și pune la dispoziție protocolului cheia privată asociată pentru a crea un canal sigur de comunicare între utilizator și pagină web. Deoarece schimbul de informații între cele două entități trebuie să fie confidențial, când o conexiune TLS este stabilită, server-ul oferă clientului certificatul, pe care acesta îl verifică înainte de a avea încredere directă în integritatea server-ului. Astfel, după ce validarea a avut loc, serverul este autentificat față de utilizator, care poate avea încrederea de a oferi date precum nume, detalii card, locație etc. fiind protejate de accesul neautorizat. Autentificarea poate fi făcută și din partea clientului, care în acest caz, ar avea nevoie la rândul său de un certificat, aceasta operațiune fiind necesară în cazul altor tipuri de comunicații online, precum aplicațiile bancare. Pentru a reuși autentificarea, utilizatorul primește un certificat digital și o cheie privată asociată din partea băncii, stocate pe un card.

De asemenea, protocolul securizat de transfer hypertext (HTTPS), este o variantă sigură a protocolului HTTP și este obținut prin combinarea acestui protocol cu TLS. Astfel, folosirea acestui protocol realizează autentificarea site-urilor. În figura ?? putem observa pagina web securizată a publicației Springer, fapt indicat de către navigatoarele web în dreptul URL-ului.

În cazul în care se face click pe iconița care garantează securitatea site-ului web, sunt afișate diferite informații despre siguranța conexiunii, printre care și algoritmi criptografici folosiți, după cum putem observa în 3.1.

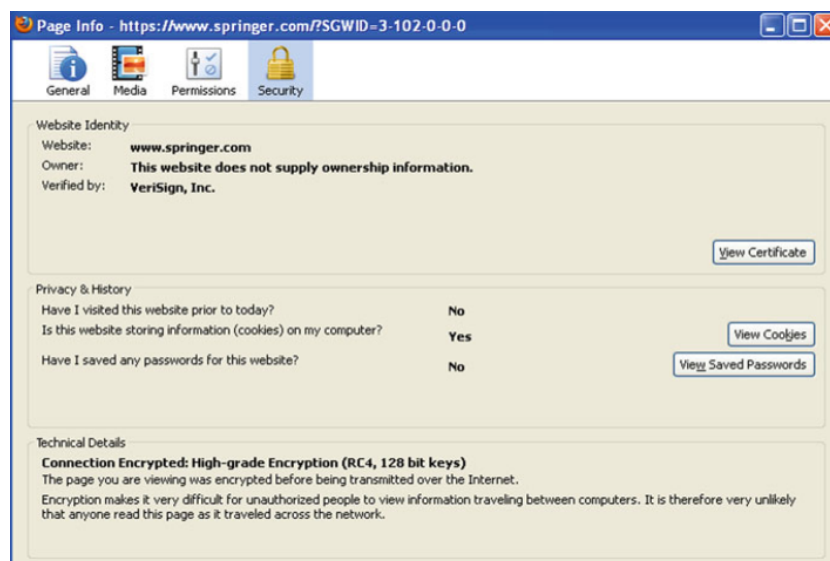


FIGURE 3.1: Informații securitate conexiune site

[3]

3.2 Securizare email

Infrastructurile care folosesc chei publice oferă în cazul comunicațiilor prin intermediul email-urilor proprietățile de confidențialitate, autenticitate și non-repudiare. Aceste caracteristici construiesc conceptul numit securitate *end-to-end*, concept pe care îl putem regăsi în două standarde pe care le vom explica în continuare.

3.2.1 S/MIME

Printre cele mai folosite standarde de securizare a email-urilor este *extensia securizată de internet pentru mail* (S/MIME). Această tehnologie permite părților de comunicare criptarea mesajului dintr-un email astfel încât acesta să nu fie vulnerabil atacurilor cibernetice. Această modalitate funcționează în momentul în care un certificat X.509 este instalat în clientul de email

al ambelor părți, iar astfel, când un mesaj electronic este trimis, emițătorul criptează mesajul folosind cheia publică a destinatarului, care va decripta informațiile folosindu-și cheia sa privată.

Pentru a explica principiile de funcționare ale acestui standard vom folosi un exemplu în care două entități, Alice și Bob, folosesc clientul de email Thunderbird iar amândoi dețin un certificat emis de către autoritatea de certificate CDC-CA. După cum putem observa în 3.2, utilizatorul Alice își are instalat certificatul X.509, cu cheia sa privată stocată în spațiul destinat certificatelor proprii, iar în 3.3 remarcăm faptul că și-a instalat și certificatul lui Bob în categoria pentru certificatele altor persoane. Totodată, și certificatul root CDC-CA trebuie instalat în calitate de autoritate.

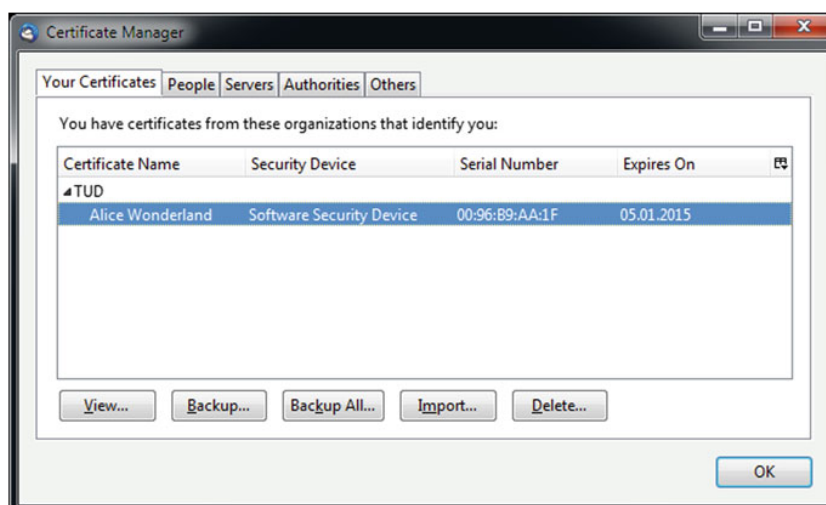


FIGURE 3.2: Certificate personale

[3]

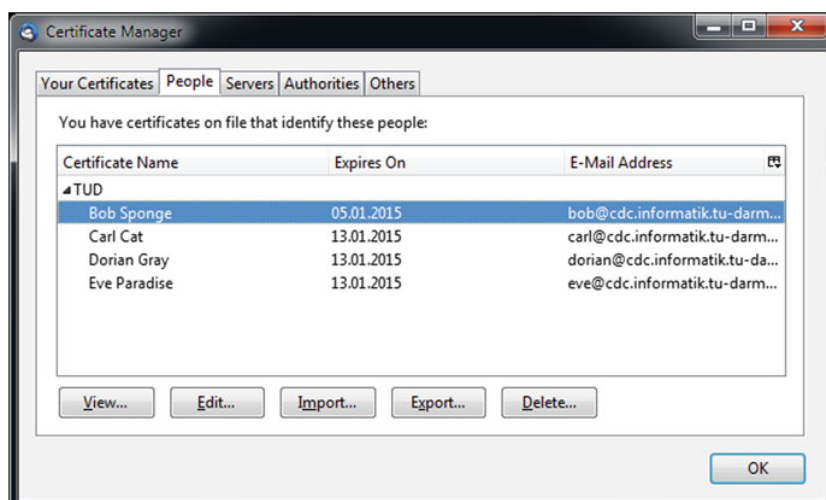


FIGURE 3.3: Certificatele altor persoane

[3]

Odată instalate toate componentele necesare, în 3.4 Alice are posibilitățile de a cripta mesajul pentru a preveni accesul neautorizat al acestuia și respectiv de a semna informațiile transmise.

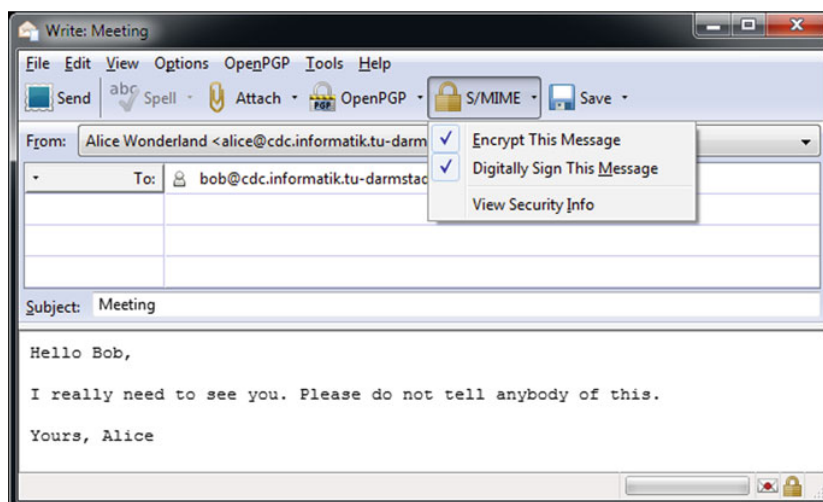


FIGURE 3.4: Trimiterea unui email securizat S/MIME

[3]

3.2.2 PGP

De asemenea, un standard important de securitate al poștei electronice este standardul PGP. Abrevierea PGP provine de la numele de *pretty good privacy* este reprezentat printr-un program folosit pentru criptarea, decriptarea și autentificarea email-urilor prin intermediul semnăturilor digitale și a criptării de fișiere. Ca și mod de acțiune, standardul PGP oferă diferite modalități de criptare a datelor, printre care:

- PGP/Inline - Printre modalitățile de criptare a unui mesaj folosind PGP există și opțiunea de a cripta fiecare element al email-ului separat. Prin urmare, corpul mesajului și atașamentele sunt individual criptate și semnate. Acest tip de criptare este avantajoasă în cazul în care destinatarul folosește un client de email care nu utilizează criptare PGP, iar pentru decriptarea mesajului se vor folosi aplicații terțe, precum *gpg*, pentru a decripta mesajul. Din nefericire, această metodă are ca și consecințe vulnerabilitatea informațiilor despre tipul și numele fiecărui atașament.
- PGP/MIME - În antiteză cu PGP/Inline, această modalitate este folosită pentru a cripta și a semna mesajele împreună cu atașamentele ca un întreg. Ca și avantaj, alternativa PGP/MIME e reprezentată de faptul că structura mesajului, precum atașamentele, nu sunt predispuse spre aflarea acestora de către alte persoane.

Din punct de vedere aplicativ, un email securizat prin intermediul protocolului PGP este la fel de ușor de trimis precum ar fi un email tradițional, nesecurizat, odată ce au fost instalate

certIFICATELE necesare. După cum putem observa în 3.5, utilizatorul Alice are posibilitatea de a trimite către Bob un email semnat și criptat cu ajutorul PGP folosind meniul destinat pentru securizarea mesajului, *OpenPGP*, și alegerea acțiunilor asupra mesajului. De asemenea, în colțul din dreapta jos, fereastra email-ului afișează opțiunile de securitate alese de către Alice, imaginea creionului exprimă semnarea mesajului, iar simbolul de cheie reprezintă criptarea cu succes a mesajului.

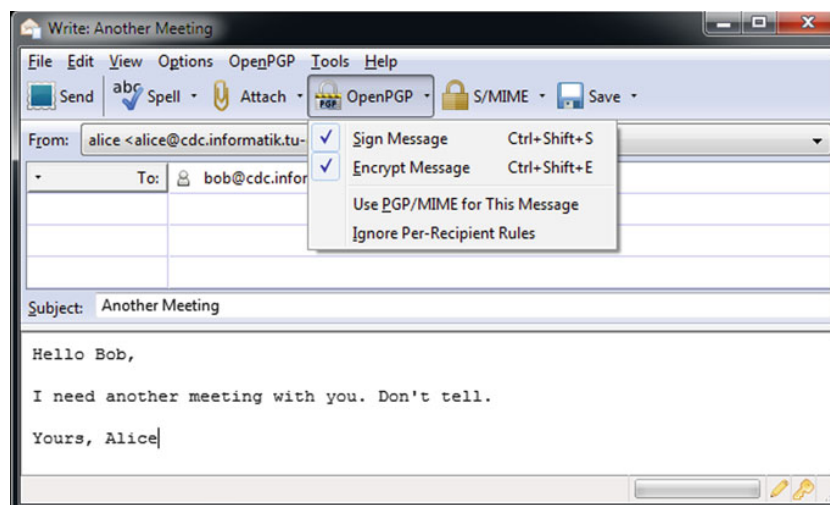


FIGURE 3.5: Semnarea sau criptarea unui email folosind PGP

[3]

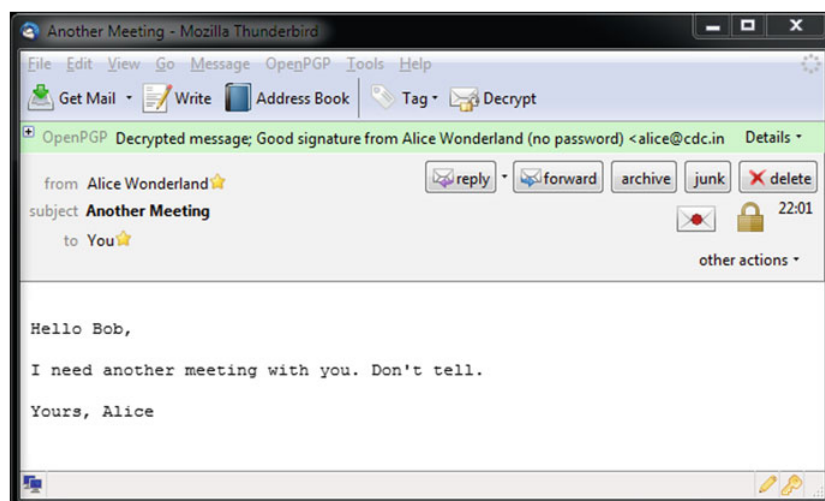


FIGURE 3.6: Citirea mesajului criptat

[3]

Odată trimis un email securizat, utilizatorul Bob are posibilitatea de a accesa conținutul mesajului. După cum este ilustrat în 3.6, un email semnat este detectat și verificat automat, mesajul criptat fiind afișat ca și cel original, iar în mailbox va fi menținut criptat. În cazul în care

semnarea nu s-ar fi efectuat cu succes, în locul mesajului de confirmare ar fi fost afișat un mesaj de atenționare cu privire la securizarea email-ului. De asemenea, starea de securitate a unui mesaj este afișată cu ajutorul simbolurilor de creion și cheie de securitate, menționate anterior.

3.3 Semnare de software

În cadrul distribuției de programe în mediul online, pentru prevenirea atacurilor cibernetice care folosesc viruși, este foarte important ca firmele de software să își autentifice produsele oferite prin semnarea lor.

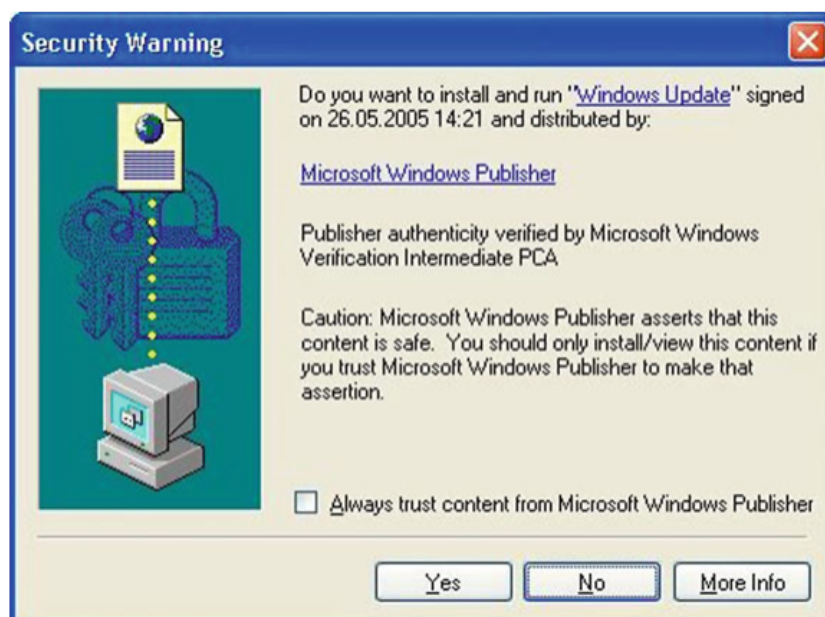


FIGURE 3.7: Instalarea unui sistem de operare semnat

[3]

Pentru a instala în siguranță un program semnat, trebuie să ne convingem de autenticitatea semnăturii. În 3.7 ne este prezentată protecția oferită de către compania Microsoft Windows, legată de actualizarea sistemului de operare. Asemănător se prezintă majoritatea sistemelor de operare, însă un utilizator poate fi neglijent asupra altor programe care nu au semnătura verificată și astfel ar risca compromiterea datelor și a calculatorului.

Semnarea codului este folosită, de altfel, și în cazul programelor mai puțin complexe, cum ar fi o aplicație auxiliară Java care ar rula într-un browser. De exemplu, o aplicație semnată poate avea accesul la scrierea sau citirea din sistemul de fișiere al calculatorului, de a comunica cu porturi externe sau pentru a stabili conexiunea la internet.



FIGURE 3.8: Avertizări cu privire la verificarea semnăturii unui program

[3]

În cadrul certificatelor care dovedesc siguranța programelor este necesară stabilirea unei extensii a cheii, în care ar trebui să fie inclusă OID-ul semnăturii codului “1.3.6.1.5.5.7.3.3”. Lipsa acestui cod va face ca certificatul să fie nefolositor în acest caz.

3.4 VPN

Protejarea rețelelor virtuale private (VPN-uri) se folosește, de asemenea, de infrastructuri bazate pe chei publice. În mediul online, acest tip de rețele au ca rol asocierea angajaților sau a clienților unor companii la rețele locale de internet (LAN), astfel încât aceștia să figureze ca parte din rețeaua respectivă.

Pentru a implementa securizarea VPN-urilor sunt folosite protocoale precum IPsec, SSH sau TSL respectiv SSL, programul cel mai des folosit fiind OpenVPN. Acest program se bazează pe protocoalele TSL/SSL și folosesc certificate X.509 pentru verificare și autentificarea utilizatorului la server.

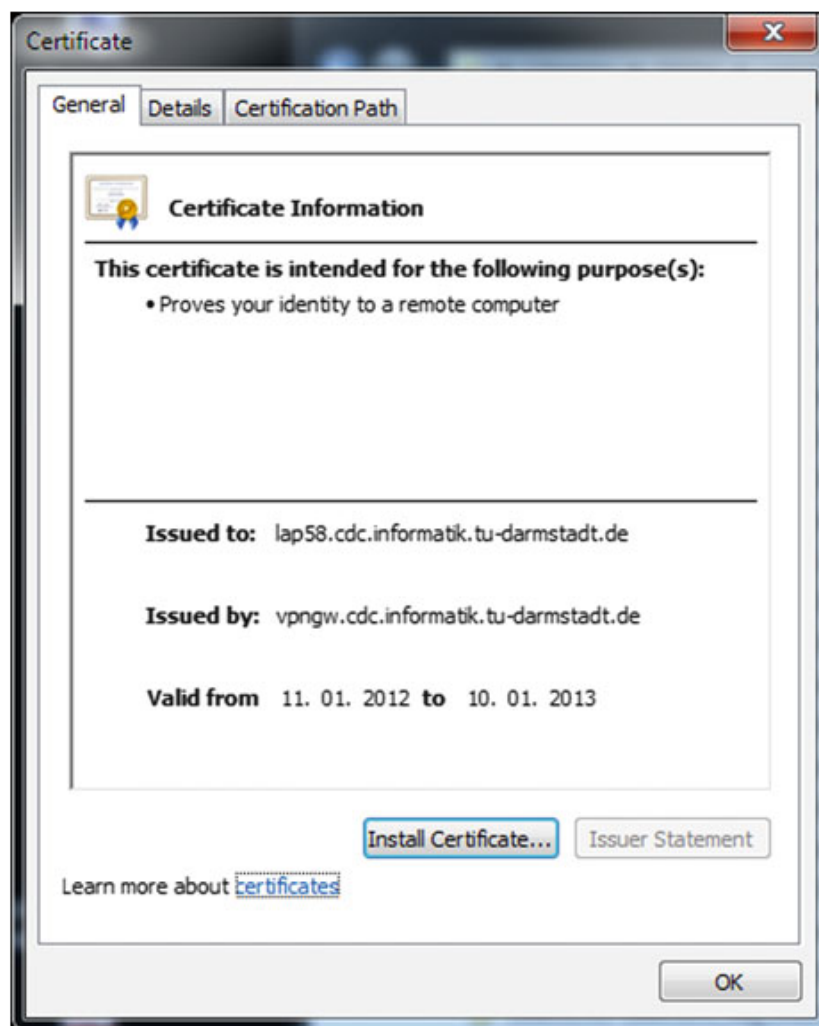


FIGURE 3.9: Certificatul VPN al unui utilizator

[3]

În 3.9 avem un exemplu în care ne este prezentat certificatul folosit de un utilizator pentru a se conecta de la distanță la LAN-ul universității, folosind OpenVPN. În acest exemplu putem observa faptul că certificatul nu este emis către utilizator, ci este emis pentru laptopul pe care îl folosește *lap58*. Certificatul root folosit de utilizator pentru autentificarea la serverul VPN este prezentat în 3.10 și este folosit împreună cu certificatul menționat anterior pentru a stabili conexiunea VPN.

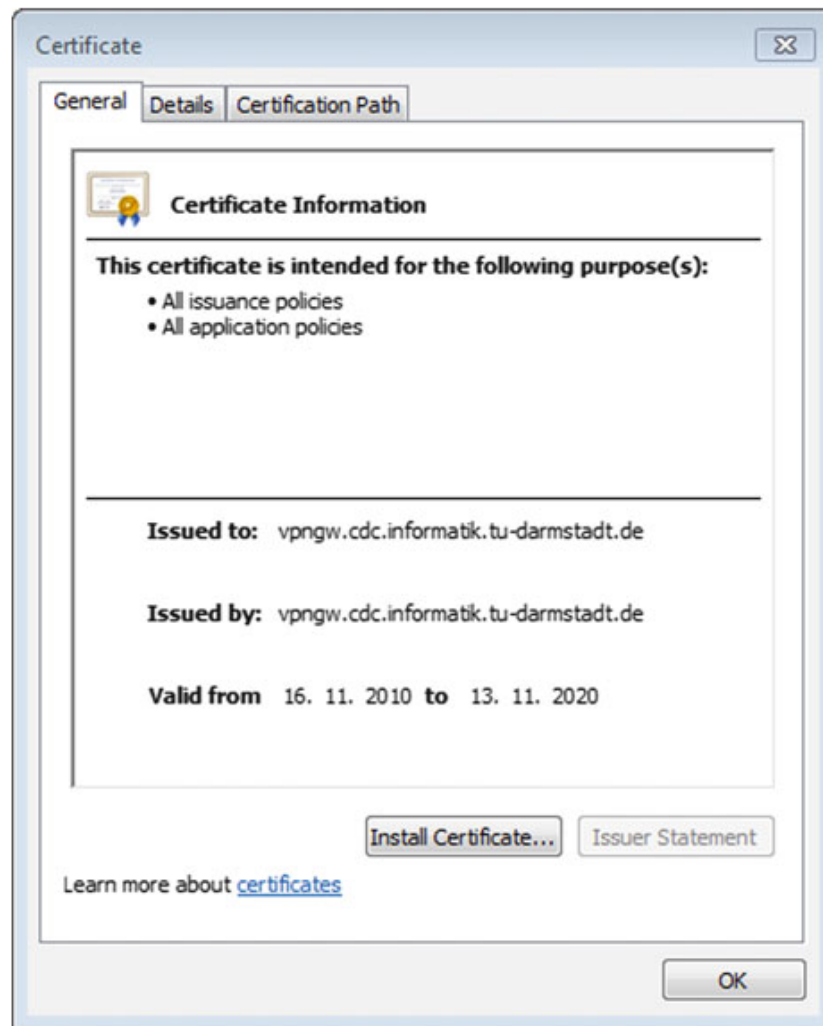
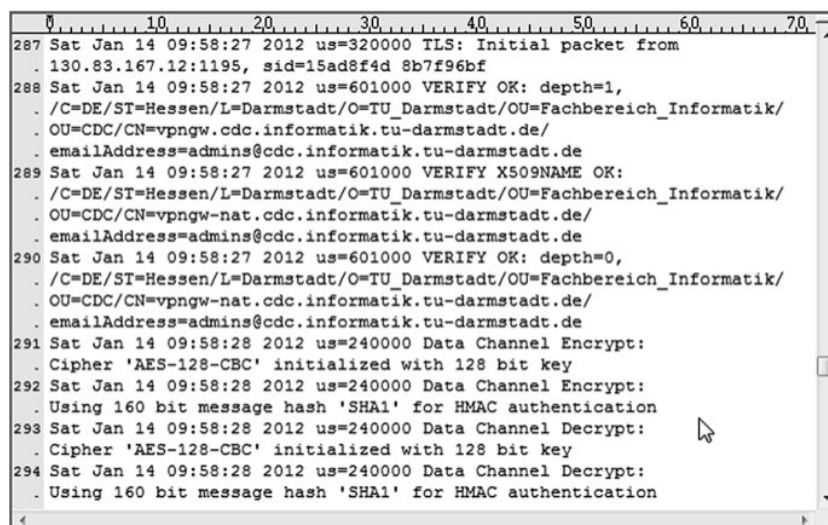


FIGURE 3.10: Certificatul root VPN

[3] Dacă am accesa fișierele de configurație ale certificatelor ne vor fi prezentate detalii despre subiect, și anume numele fișierului din laptopul utilizatorului și cheia lui privată și DN-ul serverului VPN la care se dorește stabilirea conexiunii. În fișierul de log care a fost rezultat în urma conectării reușite vom observa confirmarea verificării certificatului VPN și cifrul care definește mecanismele criptografice folosite pentru canalul TLS. Acestea sunt *AES-128-CBC* și *SHA1* pentru ambele direcții față de server, sugerând criptarea AES pe 128 de biți și fiind autenticată folosind algoritmul hash SHA1.



```
287 Sat Jan 14 09:58:27 2012 us=320000 TLS: Initial packet from
. 130.83.167.12:1195, sid=15ad8f4d 8b7f96bf
288 Sat Jan 14 09:58:27 2012 us=601000 VERIFY OK: depth=1,
. /C=DE/ST=Hessen/L=Darmstadt/O=TU_Darmstadt/OU=Fachbereich_Informatik/
. OU=CDC/CN=vpngw.cdc.informatik.tu-darmstadt.de/
. emailAddress=admins@cdc.informatik.tu-darmstadt.de
289 Sat Jan 14 09:58:27 2012 us=601000 VERIFY X509NAME OK:
. /C=DE/ST=Hessen/L=Darmstadt/O=TU_Darmstadt/OU=Fachbereich_Informatik/
. OU=CDC/CN=vpngw-nat.cdc.informatik.tu-darmstadt.de/
. emailAddress=admins@cdc.informatik.tu-darmstadt.de
290 Sat Jan 14 09:58:27 2012 us=601000 VERIFY OK: depth=0,
. /C=DE/ST=Hessen/L=Darmstadt/O=TU_Darmstadt/OU=Fachbereich_Informatik/
. OU=CDC/CN=vpngw-nat.cdc.informatik.tu-darmstadt.de/
. emailAddress=admins@cdc.informatik.tu-darmstadt.de
291 Sat Jan 14 09:58:28 2012 us=240000 Data Channel Encrypt:
. Cipher 'AES-128-CBC' initialized with 128 bit key
292 Sat Jan 14 09:58:28 2012 us=240000 Data Channel Encrypt:
. Using 160 bit message hash 'SHA1' for HMAC authentication
293 Sat Jan 14 09:58:28 2012 us=240000 Data Channel Decrypt:
. Cipher 'AES-128-CBC' initialized with 128 bit key
294 Sat Jan 14 09:58:28 2012 us=240000 Data Channel Decrypt:
. Using 160 bit message hash 'SHA1' for HMAC authentication
```

FIGURE 3.11: Fișierul log de conectare

[\[3\]](#)

3.5 Semnături electronice

În contextul societății actuale, înlocuirea semnăturilor tradiționale, pe hârtie, cu cele digitale a devenit din ce în ce mai necesară. În cazul în care un contract este generat în format electronic, este posibilă semnarea acestuia folosind o semnătură digitală, acțiunea fiind specificată în legile din diferite țări, cu referire la semnarea documentelor.

De asemenea, în cadrul spațiului European, există o sumedenie de țări care au legi compatibile cu directiva 1999/93/EC a Uniunii Europene. În această directivă este specificată noțiunea de *semnătură electronică avansată*, care trebuie să fie unic asociată de cel care aplică semnătura, să permită verificarea acestuia și să nu permită modificarea datelor semnate. Emiterea acestor certificate trebuie să fie realizată de către o autoritate de certificate care să fie capabilă de obținerea și verificarea identității subiectului care deține certificatul, cât și revocarea certificatului. Proprietarul certificatului poate alege să folosească un pseudonim, pe care autoritatea trebuie să fie capabilă să îl asocieze cu identitatea proprietarului. Nu este specificat în această lege tipul de certificat care trebuie folosit, însă, în general, sunt folosite certificatele X.509 ca și standard.

Ca exemplu de aplicabilitate, în legea Germană, acest tip de semnături sunt numite *semnături calificate* care au ca și proprietate de securitate regula ca aceste certificate să fie create folosind dispozitive sigure, specificate de autoritatea Germană. În practică, semnăturile digitale fac gestionarea documentelor mult mai eficientă, de exemplu, întreprinderile germane permit aplicarea dosarelor de angajare prin intermediul documentelor semnate electronic.

Capitolul 4

Proiectarea unui sistem de semnare digitală bazat pe chei publice

4.1 Certificatele X.509

În cadrul domeniului criptografic, certificatele digitale sunt utilizate pentru securizarea comunicațiilor, protocoale web precum TLS/SSL sau aplicații offline, ca de exemplu semnăturile digitale. O introducere a conceptului de certificat de autenticitate este prezentată în [1.1](#). Ca și model de proiectare a certificatului care va fi generat de către autoritatea de certificare pentru utilizator voi folosi tipul de certificat X.509, care este formatul standard de certificate bazate pe chei publice, aprobat de ITU (International Telecommunication Union). [\[6\]](#) Ca și principiu de funcționare, certificatele X.509 beneficiază de o structură formată din două chei, una privată și una publică, care reușesc să îndeplinească diferite aplicații ca și decriptarea sau criptarea mesajelor, oferind autenticitate sau foarte des întâlnite în construirea protocoalelor HTTPS pentru o navigare sigură a internetului. De asemenea, arhitectura certificatelor X.509 permite semnarea diferitelor produse de software sau semnături digitale.

Din punct de vedere istoric, prima versiune a apărut în anul 1988 când sectorul de standardizare a comunicațiilor al ITU a construit un sistem ierarhic, în asociere cu certificatele X.500, pentru a genera certificate digitale. Ca și antiteză a unui sistem creat pe baza certificatelor X.509 avem rețeaua de încredere PGP, în care garantarea validității și a autenticității certificatelor putea fi exprimată și de alte persoane sau companii, în afară de o autoritate de certificare. În 1996, a 3-a versiune a apărut și a adus odată cu ea multe extensii care sunt utile și în zilele noastre pentru dezvoltarea diverselor aplicații și site-uri care vor să asigure o utilizare securizată. La momentul actual, în uz se află o nouă versiune a standardului, versiunea 9, apărută în Octombrie 2019.

4.2 Implementarea certificatelor în sistemul medical

Pentru a demonstra funcționalitatea certificatelor X.509, am creat o infrastructura bazata pe chei publice in felul următor:

- Am construit o aplicație Windows Forms cu ajutorul Visual studio pentru administrarea rețetelor medicale. Pentru a demonstra autenticitatea cererilor din partea pacienților, a rețetelor prescrise de catre medici si a facturii oferite de către farmacii se vor folosi certificatele digitale pentru a crea semnături electronice.
- Pacienții care doresc digitalizarea rețetelor au posibilitatea de a transmite doctorului dorința de a fi înregistrați, aceștia creându-le un cont în baza de date a pacienților și un certificat digital odată cu înregistrarea.
- Medicii au opțiunea de a deschide lista cu cereri pentru a le analiza în scopul oferiții medicamentatiei necesare sau, alternativ, înștiințarea pacientului cu privire la necesitatea unei consultații.
- Farmaciile pot analiza toate cererile, însă prioritate pentru onorarea rețetei va avea farmacia aleasa de către pacient ca și farmacie preferată. Dacă nu sunt disponibile toate medicamentele prescrise de către medic, altă farmacie poate alege sa trimită medicamentele împreună cu rețeta si factura către pacient.
- În fiecare etapă a înregistrării într-o bază de date documentele se semnează digital folosind cheia privată generată odată cu crearea certificatului digital. Această semnătura va fi verificată cu ajutorul cheii publice pentru a realizarea autentificării.
- Pentru elementele de criptografie care asigură autenticitatea și integritatea rețetelor medicale am folosit librăria *Bouncy Castle* pentru a implementa mecanismele de criptare si semnare digitală a datelor.

Pentru a realiza autentificarea am folosit extensia *C # Entity Framework* care stochează în baza de date codul de parafa al doctorului și parola în momentul înregistrării ca ulterior să fie comparate cu datele de login introduse de acesta.

După autentificarea utilizatorului, acesta are opțiunea de a crea o cerere care va fi trimisă catre medicul său.

Ca și medic, putem vedea cererile disponibile si putem verifica semnătura digitala a pacientului înainte de a decide aprobarea rețetei. În cazul în care formatul cererii este corect, iar semnătura este verificată se înregistrează rețeta in baza de date corespunzătoare, pentru a fi vizualizată de către farmacii.

4.3 Administrarea datelor în cadrul infrastructurii medicale

Pentru atingerea scopului propus, și anume de a crea o infrastructură care să administreze cu succes certificatele digitale ale utilizatorilor am creat 6 baze de date în cadrul contextului `AutoritateContext`, și anume:

- **Pacienți** - în care salvăm CNP-ul pacientului și datele sale personale.
- **Doctori** - folosită pentru stocarea personalului care poate accesa cererile de creare a rețetelor.
- **Farmacii** - în această bază de date se vor afla înregistrate farmaciile care vor avea la dispoziție baza de date a rețetelor pentru onorarea lor.
- **Cereri** - această bază de date a fost folosită pentru a stoca cererile în vederea aprobării acestora de către doctori.
- **Rețete** - în acest context sunt salvate rețetele create, fiind asociate cu CNP-ul pacientului și codul de parafă a medicului pentru a fi disponibile farmaciilor.
- **Certificate** - bază de date destinată înregistrărilor de certificate, care pot fi folosite ulterior pentru a înregistra cereri și pentru a demonstra identitatea utilizatorului care înregistrează un set de date.

Următoarea parte de cod realizează înregistrarea în baza de date a cererilor unui pacient:

```
private void butonInregistrare_Click(object sender, EventArgs e)
{
    using(AutoritateContext ac = new AutoritateContext())
    {
        CereriReteta c = new CereriReteta();
        c.Nume = textBoxNume.Text;
        c.Prenume = textBoxPrenume.Text;
        c.CNP = textBoxCNP.Text;
        c.Patologie = textBoxPatologie.Text;
        c.SemnaturaDigitala = textBoxSemnatura.Text;
        c.CheiePublica = textBoxCheie.Text;
        c.FarmaciaPreferata = textBoxFarmacie.Text;
        ac.CereriRetete.Add(c);
        ac.SaveChanges();
    }
}
```

Iar afișarea cererilor în vederea aprobării s-a realizat prin parcurgerea tuturor cererilor existente în baza de date:

```
private void butonAfisare_Click(object sender, EventArgs e)
{
    BindingSource bi = new BindingSource();
    var query = from c in ac.CereriRetete
                orderby c.id
                select new {c.id, c.Nume, c.Prenume, c.CNP, c.Patologie,
                           c.SemnaturaDigitala, c.CheiePublica, c.FarmaciaPreferata};
    try { bi.DataSource = query.ToList(); }
    catch(Exception ex)
    {
        string aux = ex.ToString();
        MessageBox.Show(aux + "Nu exista cereri.");
    }
    bi.DataSource = query.ToList();
    dataGridView1.DataSource = bi;
    dataGridView1.Refresh();
}
```

Pentru a putea folosi bazele de date în cadrul acestui proiect, am construit o clasă auxiliară, *autoritateContext*. De asemenea, manipularea datelor a fost posibilă prin utilizarea extensiei Entity Framework.

```
class AutoritateContext : DbContext
{
    public DbSet<Pacienti> Pacientis {get; set;}
    public DbSet<Doctori> Doctoris {get; set;}
    public DbSet<Farmacii> Farmaciis {get; set;}
    public DbSet<CereriReteta> CereriRetete {get; set;}
    public DbSet<Retete> Retetes {get; set;}
    public DbSet<Certificate> Certificates {get; set;}
}
```

Principalul obiectiv al folosirii acestui context de date este de a gestiona protejarea datelor personale în sistemul medical pentru obținerea unei rețete medicale. Această digitalizare a sistemului oferă o alternativă a modului tradițional a procesului de obținere a documentelor necesare pentru procurarea tratamentului. În acest flux de informații din cadrul aplicației fiecare entitate va putea înregistra date pe care să le semneze digital pentru asumarea identității. Scopul final este de a crea o ierarhie în care utilizatorii aplicației care participă în acest schimb de informații garantează originalitatea și proveniența rețetei primite de pacient.

După ce datele au fost semnate iar fiecare entitate a garantat pentru semnătura plasată anterior pe cerere sau pe rețetă, s-a realizat o scrisoare medicală, iar pacientul poate verifica fiecare semnătura primită.

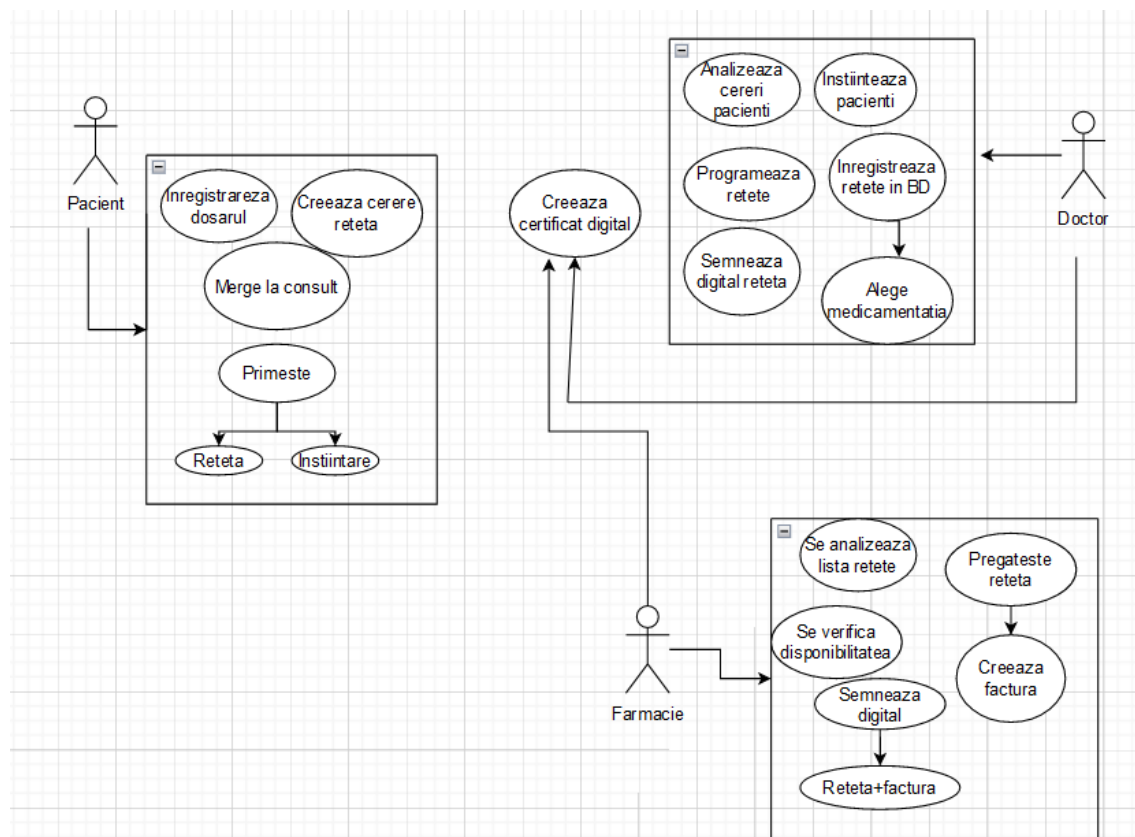


FIGURE 4.1: Rolurile entităților

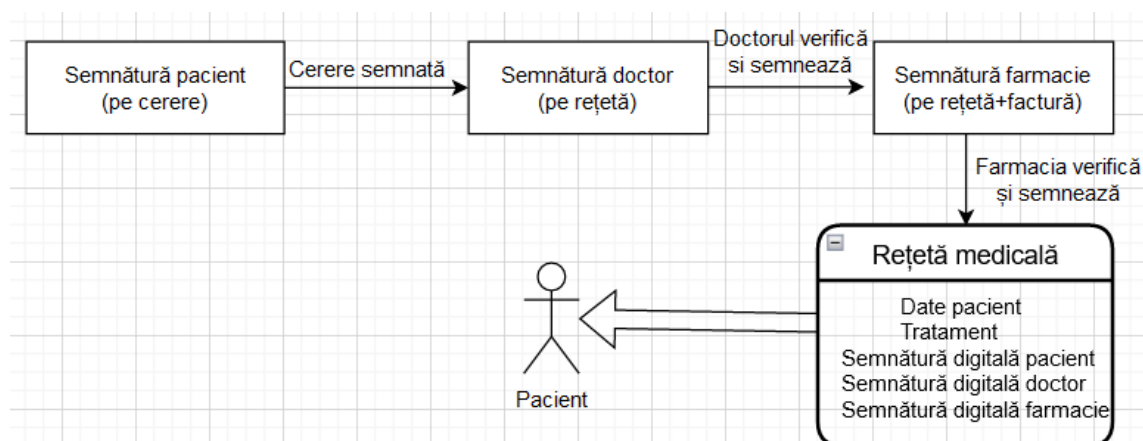


FIGURE 4.2: Rețeta rezultată

4.4 Autoritatea de certificate

O autoritate de certificate reprezintă o entitate care emite certificatele digitale și stabilește o comunicare sigură între participanții din schimbul de date. Aceste certificate au în general scopul de a verifica identitatea persoanei sau companiei care trimite informații într-o tranzacție digitală și de a cripta datele.

Aplicația are ca și modalitate de administrare a certificatelor o bază de date din contextul `AutoritateContext`, în care doctorii și farmaciile pot salva certificatele proprii și respectiv, a pacienților. Prin urmare, autoritatea de certificate este integrată în aplicație, fiecare utilizator având încredere în certificatele emise în acest context.

În momentul creării unui certificat, utilizatorului îi sunt salvate local fișierele PEM care conțin cheia publică și cea privată, pe care le folosesc la semnarea și verificarea documentelor, de asemenea și certificatul X509 care este emis de autoritatea reprezentată de infrastructura de chei publice creată.

Generarea certificatelor se realizează după apăsarea butonului *buttonGenerareCertificatClick* care apelează funcția definită în clasa de Criptografie.

```
private void buttonGenerareCertificat_Click(object sender, EventArgs e)
{
    try
    {
        AsymmetricCipherKeyPair CheiCertificat;
        //generam certificatul root
        X509Certificate2 X509RootCert = Criptografie.CreateCertificate(textBox2.Text,
            textBox1.Text, (int)numericUpDown1.Value, out CheiCertificat);

        //scriem certificatele in folder-ul ales
        File.WriteAllBytes(textBox3.Text + "\\\" + \"X509Cert.der\", X509RootCert.RawData);
        string FisierPEM_public = textBox3.Text + "\\\" + \"X509Cert-public.pem\";
        string FisierPEM_privat = textBox3.Text + "\\\" + \"X509Cert-privat.pem\";
        string cheiePublica = X509RootCert.GetPublicKeyString();
        cheiePublica = cheiePublica.Remove(0, 18);
        cheiePublica = cheiePublica.Remove(cheiePublica.Length - 10);
        textBoxCheie.Text = cheiePublica;
        RSA cheie_privata = X509RootCert.GetRSAPrivateKey();

        //Cream si fisierele PEM
        using (TextWriter textWriter = new StreamWriter(FisierPEM_public, false))
        {
            PemWriter pemWriter = new PemWriter(textWriter);
            pemWriter.WriteObject(CheiCertificat.Public);
            pemWriter.Writer.Flush();
        }

        using (TextWriter textWriter = new StreamWriter(FisierPEM_privat, false))
        {
            PemWriter pemWriter = new PemWriter(textWriter);
            pemWriter.WriteObject(CheiCertificat.Private);
            pemWriter.Writer.Flush();
        }

        MessageBox.Show(\"Certificatele au fost generate cu succes.\", \"Succes\",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
    }
```

```

        MessageBox.Show("A aparut o eroare in generarea mesajelor. "
            + ex.Message, "Eroare", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    using (AutoritateContext ac = new AutoritateContext())
    {
        CertificateUtilizator certificat = new CertificateUtilizator();
        certificat.nume_user = textBox5.Text;
        certificat.cheie_publica = textBox6.Text;
        ac.certificates.Add(certificat);
        ac.SaveChanges();
    }
}

```

Prin urmare, codul asociat acestui buton generează la apăsarea lui un certificat digital în felul următor:

1. Se crează un obiect de tipul *AsymmetricCipherKeyPair* numit "CheiCertificat" și generează, cu ajutorul funcției din clasa criptografică, un certificat X509 prin trimiterea de valori înregistrate de utilizator în textBox-urile și controlul de numeric up-down al formului ca și parametrui.
2. Certificatul returnat este salvat în folder-ul specificat de utilizator ca și fișier DER, cheile publice și private fiind salvate în fișiere PEM în același director. Pentru a putea salva cheia publică, cu scopul de a fi accesibilă oricărei entități care dorește să verifice semnătura, am extras cheia publică din fișierul PEM și am salvat-o într-un textBox.
3. La finalul funcției se crează un obiect nou de tipul certificat căruia i se atribuie elementele generate anterior pentru a completa câmpurile necesare acestei clase *Certificat*. După completarea noului obiect generat se înregistrează în baza de date de unde poate fi vizualizat de către doctori sau farmacii.

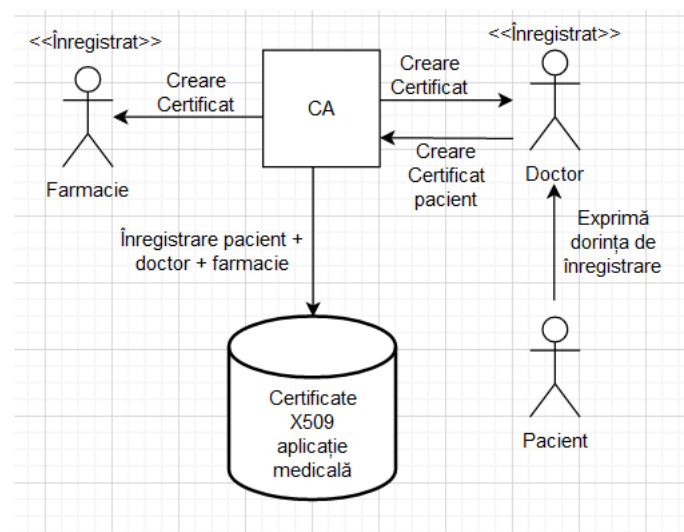


FIGURE 4.3: Administrare certificate

După cum putem observa în figura anterioară, accesul la baza de date a certificatelor este exclusiv destinat farmaciilor, la înregistrare, și respectiv, doctorilor care au posibilitatea de a își înregistra pacienții pentru a putea depune cerere, sau de a se autoînregistra în sistem.

Acest mod de prelucrare a certificatelor oferă o securitate pacienților, și anume, îi asigură că numai medicul care i-a înregistrat pot vedea datele lor personale și farmacia care va onora rețeta creată.

4.5 Înregistrarea cererilor de rețetă a pacienților

În cadrul aplicației Windows Forms care se ocupă de gestionarea și înregistrarea în bazele de date aferente tipurilor de date transmise am creat o pagină web care să fie accesibilă pacienților care au deja cont deschis de către medic, pentru înregistrarea cererilor.

Crearea site-ului web a posibilă prin folosirea unei variabile private de tip *Socket* care reprezintă un socket de server ce folosește protocolul HTTP pentru a comunica cu clienții. Această clasă *Socket* este o parte a bibliotecii Java și este utilizată pentru a crea conexiuni către o rețea, în cazul infrastructurii noastre având rolul de a asculta pentru conexiuni de intrare din partea pacienților și de a gestiona cereri și răspunsuri http.

Această clasă a fost folosită într-un form responsabil de pornirea server-ului, accesibil doctorilor, pentru cazul în care există o breșă de securitate și trebuie închisă conexiunea cu server-ul.

Pornirea server-ului se face folosind metoda următoare

```
private void startServerBtn_Click(object sender, EventArgs e)
{
    serverLogsTb.Text = "";

    try
    {
        httpServer = new Socket(SocketType.Stream, ProtocolType.Tcp);

        try
        {
            serverPort = int.Parse(textBox1.Text.ToString());

            if (serverPort > 65535 || serverPort <= 0)
            {
                throw new Exception("Server Port not within the range");
            }
        }
        catch (Exception ex)
        {
            serverPort = 80;
            serverLogsTb.Text = "Server Failed to Start on Specified Port \n";
        }
    }
}
```

```
        thread = new Thread(new ThreadStart(this.connectionThreadMethod));
        thread.Start();

        // Disable and Enable Buttons
        startServerBtn.Enabled = false;
        stopServerBtn.Enabled = true;
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error while starting the server");
        serverLogsTb.Text = "Server Starting Failed \n";
    }
    serverLogsTb.Text = "Server Started";
}
```

În momentul în care această funcție este apelată și inițializează un nou socket `httpServer`, care are tipul de socket setat pe *Stream*, iar protocolul are tipul de protocol *Tcp*. În continuarea acestei părți de cod se încearcă obținerea numărului de port de la user prin intermediul unui `textBox` și atribuie valoarea obținută variabilei `serverPort` ca și număr întreg. Validarea acestui număr reușește dacă se află în intervalul 1 - 65535, iar dacă validarea nu are succes se atribuie valoarea 80 portului. Se crează un nou fir de execuție căruia i se atribuie metoda *connectionThreadMethod* ce va fi executată când thread-ul începe execuția și va funcționa în paralel cu programul principal, care va fi capabil de a se ocupa de sarcini multiple în același timp.

Metoda atribuită firului de execuție este responsabilă de gestionarea conexiunilor cu clienții și funcționează în următorul mod:

1. Crează un nou obiect de tipul *IPEndPoint*, care reprezintă o adresă de internet și un număr de port, folosește proprietatea *IPAddress.Any* ce sugerează faptul că serverul va recepționa conexiunile ce vor urma pe toate interfațele de rețea disponibile și va asigura numărul de port variabilei `serverPort` folosind constructorul *IPEndPoint*.
2. Folosind metoda socket-ului, *Bind*, se face legătura dintre socket și endpoint, iar prin urmare se vor aștepta conexiuni la endpoint-ul specificat.
3. În final, se apelează metoda *startListeningForConnection()* care va începe procesul de așteptare pentru conexiuni următoare ale clienților.

Funcția care începe ascultarea conexiunilor următoare menționate anterior începe printr-o repetiție în care se afișează timpul curent și se inițializează un *string data* și un vector de biți cu o mărime de 2048. Ulterior se folosește metoda *Accept()* a socket-ului, care așteaptă o nouă conectare și returnează pacientul când acesta accesează site-ul. Se citește într-un alt loop datele pacientului folosind metoda *Receive* a socket-ului care citește datele în vectorul de biți menționat anterior și returnează numărul de bytes primiți.

Folosind metoda *Invoke* aparținând unui textbox se încarcă textul afișat cu datele primite și timpul curent, într-o modalitate sigură a firului de execuție.

Compunerea paginii web se va face prin crearea a două string-uri, unul care va conține un header HTTP, iar al doilea va conține un form de introducere a datelor și o funcție javascript pentru a înregistra formularul. Trimiterea răspunsului către client se realizează folosind metoda *Send*, aparținând socket-ului.

```
private void startListeningForConnection()
{
    while (true)
    {
        DateTime time = DateTime.Now;

        String data = "";
        byte[] bytes = new byte[2048];

        Socket client = httpServer.Accept();
        while (true)
        {
            int numBytes = client.Receive(bytes);
            data += Encoding.ASCII.GetString(bytes, 0, numBytes);
            if (data.IndexOf("\r\n") > -1)
                break;
        }

        serverLogsTb.Invoke((MethodInvoker)delegate {
            serverLogsTb.Text += "\r\n\r\n";
            serverLogsTb.Text += data;
            serverLogsTb.Text += "\n\n----- End of Request -----";
        });

        String resHeader = "HTTP/Content-Type: text/html; charset: UTF-8\r\n\r\n";
        String resBody = "<html>" +
            "<head>" +
            "<title>My Server</title></head>" +
            "<body>" +
            "<h4>Server Time is: " + time.ToString() + "</h4>" +
            "<form onsubmit=\"submitForm()\" accept-charset=\"UTF-8\">" +
            "<div style=\"display: flex; flex-direction: column;\">" +
            "  <label for=\"nume\">Nume:</label> " +
            "  <input type=\"text\" id=\"nume\" name=\"nume\" required><br>" +
            "  <label for=\"prenume\">Prenume:</label> " +
            "  <input type=\"text\" id=\"prenume\" name=\"prenume\" required><br>" +
            "  <label for=\"cnp\">CNP:</label> " +
            "  <input type=\"text\" id=\"cnp\" name=\"cnp\" required><br>" +
            "  <label for=\"patologie\">Patologie:</label> " +
            "  <input type=\"text\" id=\"patologie\" name=\"patologie\" required><br>" +
            "  <label for=\"semnatura\">Semnatura:</label> " +
            "  <input type=\"text\" id=\"semnatura\" name=\"semnatura\" required><br>" +
            "  <label for=\"cheiepublica\">Cheie publica:</label> " +
            "  <input type=\"text\" id=\"cheiepublica\" name=\"cheiepublica\" required><br>" +
            "  <input type=\"submit\" value=\"Submit\" >" +
            "</div>" +
```

```

    "</form>" +
    "<script type=\"text/javascript\">" +
    "function submitForm() {" +
    "    var nume = document.getElementById(\"nume\").value;" +
    "    var prenume = document.getElementById(\"prenume\").value;" +
    "    var cnp = document.getElementById(\"cnp\").value;" +
    "    var patologie = document.getElementById(\"patologie\").value;" +
    "    var semnatura = document.getElementById(\"semnatura\").value;" +
    "    var cheiepublica = document.getElementById(\"cheiepublica\").value;" +
    "    var formData = new FormData();" +
    "    formData.append(\"nume\", nume);" +
    "    formData.append(\"prenume\", prenume);" +
    "    formData.append(\"cnp\", cnp);" +
    "    formData.append(\"patologie\", patologie);" +
    "    formData.append(\"semnatura\", semnatura);" +
    "    formData.append(\"cheiepublica\", cheiepublica);" +
    "    var xhr = new XMLHttpRequest();" +
    "    xhr.open(\"POST\", \"my_server\");" +
    "    xhr.send(formData);" +
    "}" +
    "</script>" +
    "</body>" +
    "</html>";

    byte[] resBytes = Encoding.ASCII.GetBytes(resHeader + resBody);
    client.Send(resBytes);
    client.Close();
}
}

```

4.6 Generarea certificatelor digitale

Pentru generarea certificatului am utilizat extensia Bouncy Castle în cadrul clasei de criptografie, care pe baza datelor introduse în cererea utilizatorului va crea fișierele necesare. Clasa conține o metodă principală *CreateCertificate* care va avea ca și parametrii 2 câmpuri de tip string, care vor reprezenta informațiile legate de subiect și emitentul care este prestabilit, perioada de valabilitate, mărimea cheii (2048 de biți) și un parametru care va reprezenta perechea de chei generată. În cadrul acestei metode create am folosit următoarele clase auxiliare din cadrul extensiei Bouncy Castle:

- **CryptoApiRandomGenerator** care folosește Microsoft CryptoAPI pentru a genera numere random. Am generat o instanță a acestei clase în scopul creării unui număr aleatoriu, care va fi transmis ca și parametru clasei *SecureRandom*. Aceasta fiind clasa care folosește un generator de numere pentru a crea valori noi.

- **X509V3CertificateGenerator** pentru a crea un generator de certificate cărui îi vom atribui câmpurile necesare, și anume numărul de serie, numele unic al entității și al emitentului, limitele de valabilitate, algoritmul de semnare a certificatului (SHA256WithRSA) și cheia publică obținută. Acest generator folosește o metoda pentru crearea unor numere întregi mari într-un anumit interval, care va fi atribuit numărului de serie al certificatului. Folosind funcțiile *SetIssuerDN()* și *SetSubjectDN()* se vor completa numele emitentului și respectiv, al subiectului. Ulterior, se setează valabilitatea certificatului și se specifică rolul perechii de chei prin adăugarea unei extensii la certificat care are Id-ul setat pe true indicând faptul că extensia este necesară. Acest cod este cel care generează certificatul X509, reprezentând standardul pentru demonstrarea identității unui utilizator sau a unui calculator. În cazul nostru, certificatul este creat specificând numărul de serie, numele emitentului și a subiectului, perioada de valabilitate și scopul folosirii cheilor.
- **RsaKeyPairGenerator** această clasă este responsabilă de generarea unei perechi de chei și de setarea cheii publice a subiectului în certificat, care este semnat folosind algoritmul *SHA256WithRSA* împreună cu cheia privată a emitentului.

După ce fiecare câmp a fost atribuit generatorului, certificatul este creat și fișierul PEM conținând cheia privată poate fi folosit împreună cu textul din document pentru generarea semnăturii.

4.7 Generarea semnăturilor digitale

A doua clasă importantă pe care am folosit-o pentru generarea unei semnături pentru un text al utilizatorului este clasa *SemnareDigitală* care conține următoarele metode din clasa *Criptografie*:

- **RSASignWithPrivateKey** Această metodă folosește librăria Bouncy Castle pentru a crea o semnătură digitală folosind algoritmul RSA. Se începe cu calcularea hash-ului SHA-256 al biților primiți folosind clasa *Sha256Digest*. Apoi se crează un obiect de tipul *PssSigner* care este folosit pentru a crea semnătura RSA folosind cheia privată din perechea de chei (*KeyPair*) primită. În final, folosind mecanismul RSA pentru operațiile matematice și obiectul *sha256Digest* pentru funcția de calcul a hash-ului se returnează semnătura sub forma de vector de biți.

```
public static byte[] RSASignWithPrivateKey(AsymmetricCipherKeyPair KeyPair,
                                           byte[] BytesToSign)
{
    //calcularea has-ului SHA256 din bitii primiti pentru a fi semnati
    SHA256Digest sha256Digest = new Sha256Digest();
    byte[] TheHash = new byte[sha256Digest.GetDigestSize()];
    sha256Digest.DoFinal(TheHash, 0);
```

```

        PssSigner Signer = new PssSigner(new RsaEngine(),
            sha256Digest.GetDigestSize());
        Signer.Init(true, KeyPair.private);
        Signer.BlockUpdate(TheHash, 0, TheHash.Length);
        byte[] Signature = Signer.GenerateSignature();

        return Signature;
    }

```

- **RSASignWithPemPrivateKey** care are ca și parametrii numele fișierului PEM generat de clasa *criptografie* și textul din documentul care se dorește să fie semnat. Pentru a începe pregătirea semnăturii, această funcție face conversia textului în biți folosind codificarea UTF-8 și citește cheia privată din fișierul PEM folosind clasa *PemReader*. Ulterior, cheia privată este transmisă împreună cu biții de semnat metodei *RSASignWithPrivateKey* pentru a obține semnătura. Semnătura rezultată este transformată în baza 64 ca și string și returnată.

```

public static string RSASignWithPEMPrivateKey(string PrivateKeyPEMFileName, string Text)
{
    byte[] BytesToSign = Encoding.UTF8.GetBytes(Text);
    AsymmetricCipherKeyPair KeyPair = null;
    TextReader reader = File.OpenText(privateKeyPEMFileName);
    KeyPair = (AsymmetricCipherKeyPair)new PemReader(reader).ReadObject();

    byte[] Signature = RSASignWithPrivateKey(KeyPair, BytesToSign);
    string Result = Convert.ToBase64String(Signature);

    return Result;
}

```

4.8 Verificarea semnăturilor digitale

Pentru verificarea semnăturilor, componentele de farmacii și doctori vor avea la dispoziție un form *Verificare semnătură* care va fi folosit pentru determinarea autenticității mesajului emittentului și pentru aplicarea semnăturii pentru continuarea fluxului securizat de documente.

Această funcție are nevoie de trei informații pentru validarea semnăturii, și anume, cheia publică a emitentului, textul asociat semnăturii și semnătura digitală care ar trebui obținută pe baza textului și a cheii publice. Se citește cheia publică din fișierul PEM folosind *PemReader* și crează un obiect de tipul *AsymmetricKeyParameter*. Ulterior se crează un obiect *Sha256Digest* și aplică hash-ul peste biții textului folosind această instanță. Datele respective sunt manipulate de către funcția *VerifySignature* care calculează hash-ul obținut din biții textului introdus și folosește o instanță a clasei *PssSigner* pentru a moșteni funcția de verificare a semnăturii aparținând acestei clase și a compara semnătura primită cu cea care este de așteptat să fie rezultată.

```

public static bool VerifySignature(string PublicKeyPEMFileName,
                                   string Text, string ExpectedSignature)
{
    byte[] BytesToSign = Encoding.UTF8.GetBytes(Text);
    byte[] ExpectedSignatureBytes = Convert.FromBase64String(ExpectedSignature);

    TextReader reader = File.OpenText(PublicKeyPEMFileName);
    AsymmetricKeyParameter KeyPair =
        s(AsymmetricKeyParameter)new PemReader(reader).ReadObject();

    Sha256Digest sha256Digest = new Sha256Digest();
    byte[] TheHash = new byte[sha256Digest.GetDigestSize()];
    sha256Digest.BlockUpdate(BytesToSign, 0, BytesToSign.Length);
    sha256Digest.DoFinal(TheHash, 0);

    PssSigner Signer = new PssSigner(new RsaEngine(), new Sha256Digest(),
        sha256Digest.GetDigestSize());
    Signer.Init(false, KeyPair);
    Signer.BlockUpdate(TheHash, 0, TheHash.Length);
    return Signer.VerifySignature(ExpectedSignatureBytes);
}

```

La momentul încărcării acestui form de verificare se încarcă textul care a fost semnat de către pacient care este format din datele lui personale și semnătura generată pe baza cheii lui publice. De asemenea funcția va returna pe baza rezultatului obținut din compararea celor două semnături, cea așteptată și cea primită, un mesaj care asigură utilizatorul de identitatea celui care a înregistrat cererea.

```

private void ButonVerificare_Click(object sender, EventArgs e)
{
    try
    {
        bool Result = Criptografie.VerifySignature(textCheiePublica.Text,
            textBoxInput.Text, textBoxSemnatura.Text);
        string Output = Result ?
            "Semnatura se potrivește, verificarea a fost efectuată cu succes"
            : "Semnatura nu se potrivește. Verificarea a eșuat.";
        MessageBox.Show(Output, Result ? "Success" : "Error");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Eroare");
    }
}

private void VerificareSemnatura_Load(object sender, EventArgs e)
{
    string DateSemnate = "";
    string varsta = "";
    using (AutoritateContext ac = new AutoritateContext())
    {
        foreach (var d in ac.Pacientis)

```

```

    {
        if (d.CheiePublica == FormUserPrincipal.CheiePacient)
        {
            varsta = d.Varsta.ToString();
        }
    }
    foreach(var r in ac.CereriRetete)
    {
        if(r.id == FormUserPrincipal.idCerere)
        {
            DateSemnate = r.Nume + r.Prenume + r.CNP + varsta + r.Patologie;
        }
    }
}
textCheiePublica.Text = FormUserPrincipal.CheiePacient;
textBoxSemnatura.Text = FormUserPrincipal.SemnaturaPacient;
textBoxInput.Text = DateSemnate;
}

```

Funcția de verificare va fi folosită în form-ul menționat anterior, iar la apăsarea butonului de *Verificare semnătură* se va returna rezultatul obținut în urma executării funcției.

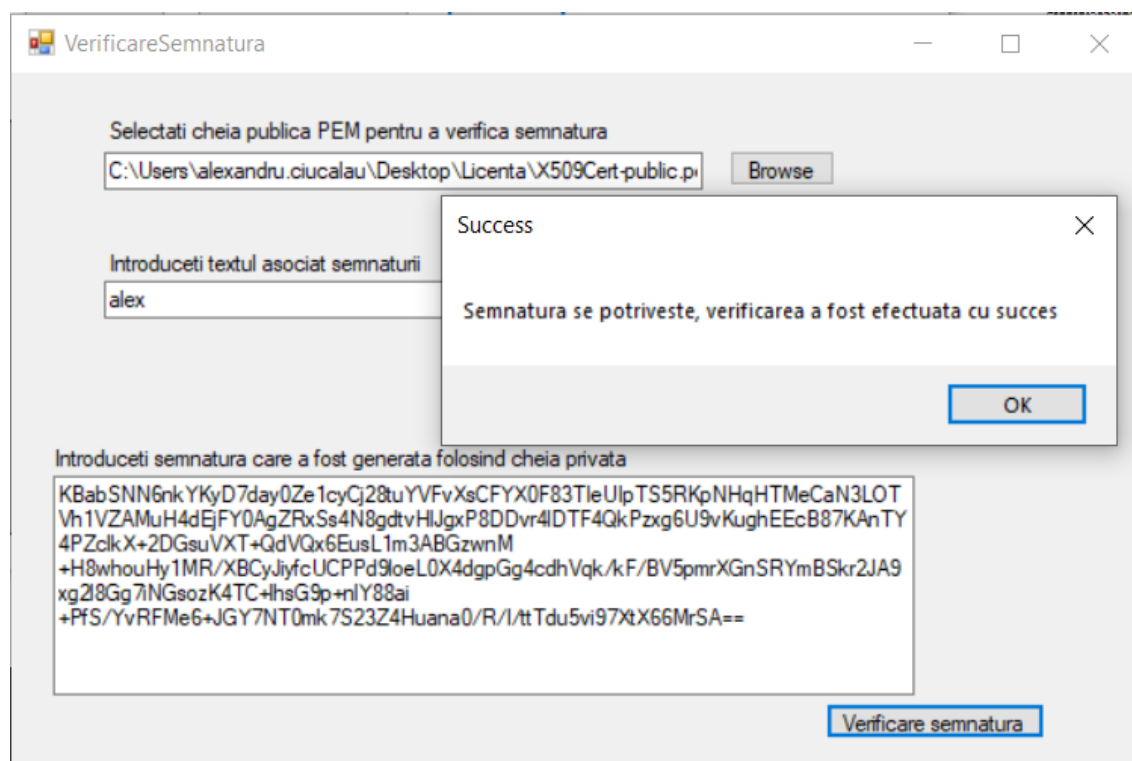


FIGURE 4.4: Rezultat verificare semnătură

În final, s-a realizat un flux de date în care fiecare utilizator al aplicației este convins de proveniența datelor medicale înregistrate, fiecare informație având o semnătură digitală care dovedește identitatea emitentului.

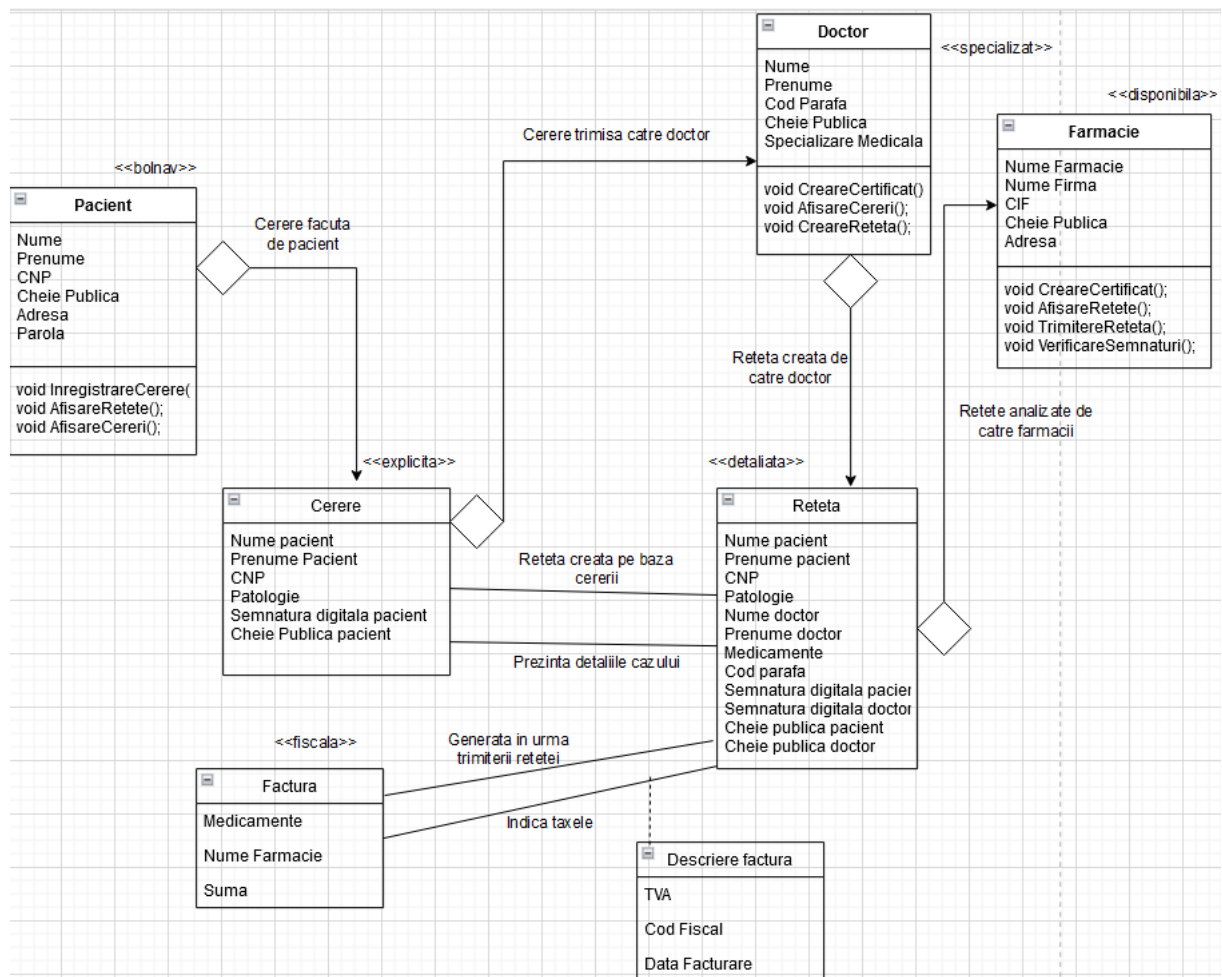


FIGURE 4.5: Diagrama De Clasa

Capitolul 5

Concluzii

După analiza problemelor de securitate existente în contextul societății actuale am reușit să descoperim o alternativă sigură a comunicării la distanță, care reușește să asigure proprietățile pe care trebuie să le ofere criptografia asimetrică, și anume, autentificare, non-repudiare, confidențialitate și integritatea datelor. Acest scop a fost atins prin documentarea asupra subiectului de criptografie asimetrică, reușind astfel să înțelegem problema siguranței în mediul online, iar căutând algoritmi matematici am realizat criptarea unui mesaj folosind cheia privată generată, asociată cheii publice.

De asemenea, digitalizarea sistemului medical în scopul obținerii unei prescripții medicale poate ajuta la evoluția acestei instituții ale statului care se ocupă de sănătatea populației, din punct de vedere administrativ și într-un mod care garantează siguranța datelor personale.

Astfel, am atins scopul de abstractizare a datelor într-un mod în care dacă o entitate care vrea să transmită un mesaj secret, să aibă posibilitatea de a își demonstra autenticitatea folosind o semnătură generată prin utilizarea algoritmului RSA, pentru generarea unui certificat X.509 împreună cu fișierele care conțin câte o cheie separată, legate matematic între ele. Administrarea acestor elemente care construiesc opțiunea modernă și sigură de a semna mesaje și a dovedi autenticitatea lor se realizează prin intermediul programului creat, care conține 6 baze de date, salvând informațiile obținute.

Într-un final, scopul de a asigura încrederea destinatarului în emițător a fost stabilită prin partajarea textului scris de către creatorul mesajului, în cazul nostru, pacient, doctor sau farmacie, împreună cu semnătura digitală creată de aplicație pe baza textului abstractizat, peste care a fost efectuată o criptare folosind cheia privată. Demonstrarea acestei autenticități poate fi realizată prin verificarea pusă la dispoziție oricărei entități dintre cele trei, care va asigura recepționarul mesajului de identitatea celui care a trimis informațiile.

Bibliografie

- [1] W. Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5): 560–577, 1988. doi: 10.1109/5.4442.
- [2] . URL <https://ro.education-wiki.com/5679799-asymmetric-encryption>.
- [3] J. A. Buchmann. Introduction to public key infrastructures. 2013. doi: 10.1007/978-3-642-40657-7.
- [4] A. S. Arnold, J. S. Wilson, and M. G. Boshier. A simple extended-cavity diode laser. *Review of Scientific Instruments*, 69(3):1236–1239, March 1998. URL <http://link.aip.org/link/?RSI/69/1236/1>.
- [5] Leighton Johnson. *Security Controls Evaluation, Testing, and Assessment Handbook (Second Edition)*, pages 471–536, 2020. URL <https://www.sciencedirect.com/science/article/pii/B9780128184271000112>.
- [6] . URL <https://sectigo.com/resource-library/what-is-x509-certificate>.