SIGURNOST I BEZBEDNOST ELEKTROENERGETSKOG SOFTVERA

PRAKTIKUM

2021/2021

Profesor : **Imre Lendak**
Asistenti: **Zorana Babić**
Jelena Sekulić
Stefan Ruvčeski

Sadržaj

Uvod

Polaganje predmeta:

- Projekat – 60 bodova
- Test – 20 bodova
- Usmeni – 20 bodova

Vežbe :

- Alat koji se koristi prilikom izrade zadataka je Visual Studio (verzija po želji).
- Prisustvo na vežbama je obavezno (broj maksimalnih izostanaka je 2).
- Podela projekata će se održati online u terminu vežbi (29.11. i 30.11).
- Poslednje tri nedelje (od 13.12. do 09.01) se izrađuje projekat.
- Odbrane projekata će se održati u poslednjoj nedelji semestra 10.01. i 11.01.
- Projekti moraju biti okačeni na git (poslednji commit 09.01. u 23:59.
- Odbrana projekata će se održati u prostorijama FTN-a, nije moguća online odbrana.
- Projekat može ranije da se brani uz mogućnost nagradnih poena. ☺
- Postoje tri vrste projekata:
  Projekat za 6
  - Maksimalni broj bodova koji može da se osvoji je 36.5
  - Minimalan broj bodova koji mora da se osvoji da bi student položio je 30.5
  - Radi se u timovima od po dva studenta

  Redovan projekat
  - Maskimalan broj bodova koji može da se osvoji je 60
  - Minimalan broj bodova koji mora da se osvoji da bi student položio je 30.5
  - Radi se u timovima od po četiri studenta
  - Super projekat ☺
    - Maksimalan broj bodova koji može da se osvoji je 80
    - Minimalan broj bodova koji mora da se osvoji da bi student položio je 55
    - Studenti koji polože ovaj projekat ne moraju da izlaze na testić
    - Radi se u timovima od po dva studenta
    - Teorija koja je neophodna za ovaj projekat može da izlazi iz opsega teorije koja se radi na vežbama
    - Super projekat predstavlja projekat koji je osmišljen od strane studenata, studenti dolaze sa idejom šta bi radili, dok asistenti pomažu da se ta ideja razvije
- Odbrana projekata se sastoji od prikaza urađenog zadatka i dodele bodova na osnovu urađenog zadatka, to je maksimalan broj bodova koji može da osvoji svaki od članova tima, nakon toga svako od članova tima pojedinačno “brani” projekat i dobija broj bodova u zavisnosti od svog znanja i urađenog projekata.
- Oblasti koje će se obrađivati :
  - Autentifikacija, Impersonifikacija, Sertifikati, digitalni potpisi, Autorizacija, RBAC, Kriptografija, Auditing

Vezba 1 - Autentifikacija

Cilj ove vežbe je upoznavanje sa osnovnim elementima bezbedne komunikacije u .NET razvojnom okruženju, a zatim i sa mehanizmom autentifikacije.

Autentifikacija je jedan od osnovnih bezbednosnih mehanizama kojim se obezbeđuje validacija identiteta u okviru informacionog sistema. Da bi entitet mogao da pristupi sistemu potrebno je da dokaže da je on upravo onaj za koga se izjašnjava. Podaci kojima se korisnici predstavljaju sistemu i potvrđuju identitet nazivaju se kredencijali i mogu se podeliti u tri kategorije:

1) nešto što korisnik zna (npr. šifra),

2) nešto što korisnik ima (npr. pametna kartica),

3) nešto što korisnik jeste (npr. otisak prstiju).

Na primer, putnici se na carini autentifikuju pomoću pasoša tako što posedovanjem pasoša potvrđuju svoj identitet. Proces validacije pasoša, kao i procena sličnosti putnika sa slikom u pasošu predstavlja način autentifikacije kojim putnik dokazuje svoj identitet. U računarskim sistemima, entitet može biti korisnik, ali i računar, servis, aplikacija ili bilo koji uređaj. Autentifikacija treba da obezbedi validaciju identiteta i tako spreči pristup nevalidnim korisnicima.

Bezbedna komunikacija u .NET-u

Prvi korak prilikom uspostavljanja komunikacije između dva entiteta je definisanje komunikacionog protokola kako bi podaci bili poslati u formatu koji će druga strana moći da razume.

Bezbedna komunikacija uključuje skup bezbednosnih mehanizama koji će omogućiti zaštitu komunikacionog kanala, koji može da prolazi kroz neobezbeđene sisteme, od trećeg entiteta. Ovo se postiže izvršavanjem sledećih koraka:

- Definisanje autentifikacionog protokola kada se učesnici u komunikaciji dogovaraju na koji način će biti izvršena validacija identiteta svakog entiteta.
- Definisanje mera zaštite podataka od:
    - neovlašćenog pristupa, čitanja i otkrivanja (poverljivost) - definisanje algoritma za šifrovanje podataka, a zatim i razmena ključeva za dešifrovanje.
    - neovlašćenih izmena ili brisanja (integritet) – definisanje algoritma za detekciju izmene podataka.

U WCF .NET razvojnom okruženju se za povezivanje učesnika u komunikaciji koristi ***Binding.*** WCF nudi širok spektar ugrađenih bindinga, u zavisnosti od specifičnih zahteva aplikacije njihova podešavanja moguće je izmeniti ili definisati specifične custom bindinge.

Bindingom se se definiše:

- komunikacioni protokol: TCP, HTTP, IPC
- protokol za autentifikaciju: Windows autentifikacioni protokol ili autentifikacija upotrebom sertifikata
- bezbednosni mod kojim se definiše:
    - ***Transport Security*** mod koji obezbeđuje zaštitu uspostavljenog komunikacionog kanala, odnosno *point-to-point* zaštitu podataka
    - ***Message Security*** mod obezbezbeđuje zaštitu na nivou poruka koje se razmenjuju u komunikaciji.

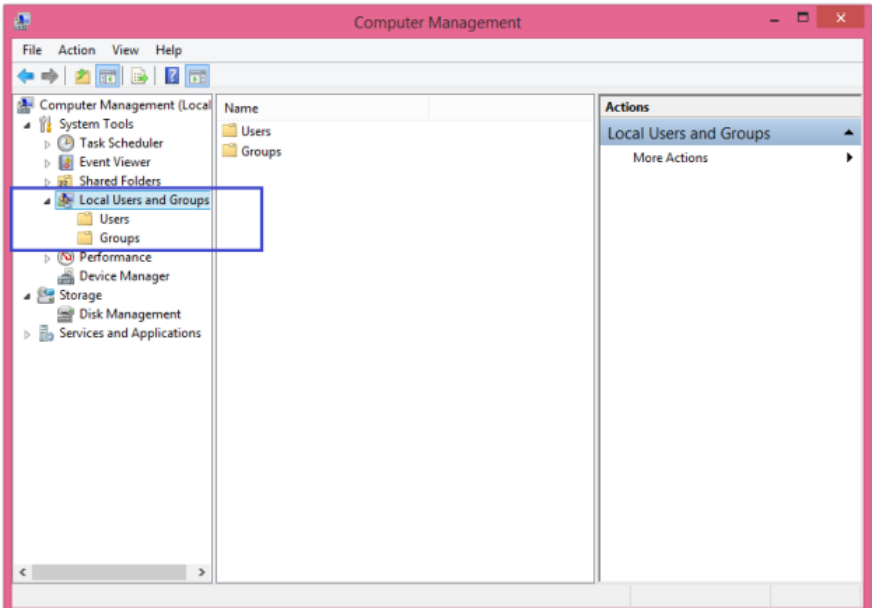
```
NetTcpBinding binding = new NetTcpBinding();
binding.Security.Mode = SecurityMode.Transport;

binding.Security.Transport.ClientCredentialType = TcpClientCredentialType.Windows;

binding.Security.Transport.ProtectionLevel =
System.Net.Security.ProtectionLevel.EncryptAndSign;
```

U nastavku je dat primer podešavanja različitih parametara ugrađenog ***NetTcpBinding*** tipa bindinga:

Zadatak 1.1 Na primeru jednostane WCF klijent-servis aplikacije, u okviru koje WCF servis pruža mogućnost poziva metode AddUser, implementirati Windows autentifikaciju, i obezbediti zaštitu poverljivosti i integriteta podataka u okviru uspostavljene komunikacije.

Zadatak 1.2 Klijentsku i serversku aplikaciju iz prethodnog primera proširiti tako da prilikom startovanja ispisuju infomacije o identitetu korisnika (npr. ime) koji je pokrenuo proces. Zatim, podesiti radno okruženje tako da klijentska i serverska aplikacija budu pokrenute kao različiti korisnici.

Napomena: Za upravljanje lokalnim korisničkim nalozima i korisničkim grupama se koristi *Computer Management* Windows konzola prikazana na slici 1.



Slika 1- Computer managament

Zadatak 1.3 Proširiti metodu Read iz prethodnog zadatka tako da ispiše podatke o klijentu koji je pozvao ovu metodu. Potrebno je ispisati sledeće informacije: ime klijenta, njegov jedinstveni identifikator (*SecurityIdentifier*), informacije o tipu autentifikacije i o korisničkim grupama kojima dati korisnik pripada.

Autentifikacioni protokoli zasnovani na šiframa

Windows autentifikacioni model je zasnovan na SPNEGO (*Simple and Protected GSS API negotiation mechanism*) mehanizmu za pregovaranje između različitih realnih autentifikacionih mehanizmana u zavisnosti od okruženja. SSPI (*Security Support Provider Interface*) je Windows API koji implementira SPNEGO i predstavlja zajednički interfejs za različite Windows autentifikacione protokole (*Secure Service Provider*). Negotiate SSP je aplikativni protokol kojim je implementirano pregovaranje u Windows-u. Trenutno podržani protokoli su NTLM i Kerberos:

NTLM (*NT Lan Manager*) je autentifikacioni protokol zasnovan na *challenge-response* autentifikacionoj šemi, čime je omogućena autentifikacija bez slanja poverljivih podataka (šifre). Iako challenge-response spada u jake autentifikacione šeme jer nema razmene poverljivih podataka, problem ovakvih protokola je činjenica da servis mora da zna originalnu šifru svakog klijenta kako bi mogao da validira pristigli response. Dodatno, u ovako definisanom autentifikacionom protokolu izostaje verifikacija servisnog identiteta od strane klijenta, odnosno ovakav protokol **ne omogućava obostranu autentifikaciju.**

Da bi se obezbedila dvosmerna/obostrana autentifikacija u okviru challenge-response protokola, potrebno je da po istom principu kao što klijent dokazuje identitet servisu, i servis dokaže svoj identitet klijentu. Međutim, autentifikacioni protokoli koji se zasnivaju na obostranoj autentifikaciji i koji podrazumevaju razmenu poruka između dva učesnika u autentifikaciji na potpuno isti način su generalno nesigurni protokoli. Kako bi se obezbedili sigurniji protokoli, uvodi se treća strana od poverenja odnosno entitet kome veruju svi ostali učesnici u komunikaciji.

***Kerberos** je dvosmerni autentifikacioni protokol koji* se zasniva na trećoj strni od poverenja i razmeni **ticketa** u cilju uspostavljanja bezbedne obostrane autentifikacije učesnika u komunikaciji bez razmene

šifri. Kerberos je namenjen za domenska okruženja gde uslugu treće strane od poverenja ima posebno konfigurisani server, tzv. domen kontroler (DC). DC predstavlja autoritet na nivou celokupnog domena kome pripada skup računara i korisničkih naloga.

Tipično, Kerberos je siguran, dvosmerni autentifikacioni protokol preko koga Negotiate prvo pokušava da autentifikuje korisnike. Ukoliko iz bilo kog razloga Kerberos autentifikacija nije moguća, NTLM protokol će se koristiti.

Autentifikacija preko NTLM protokola neće biti realizovana u slučaju da je Kerberos autentifikacija pokušana, ali je bila neuspešna.

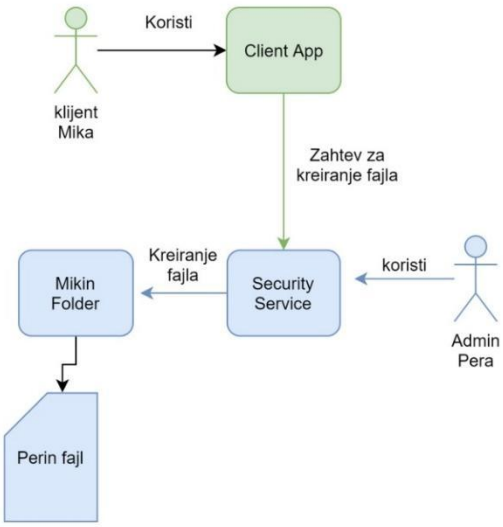
Zadatak 2.1 Proširiti WCF klijent-servis aplikaciju iz Zadatka 2. tako da prilikom uspostavljanja komunikacije bude definisan identitet servisa (*EndpointIdentity*).

Zadatak 2.2 Obezbediti da u slučaju kada Kerberos autentifikacija nije moguća, NTLM ne bude dozvoljen.

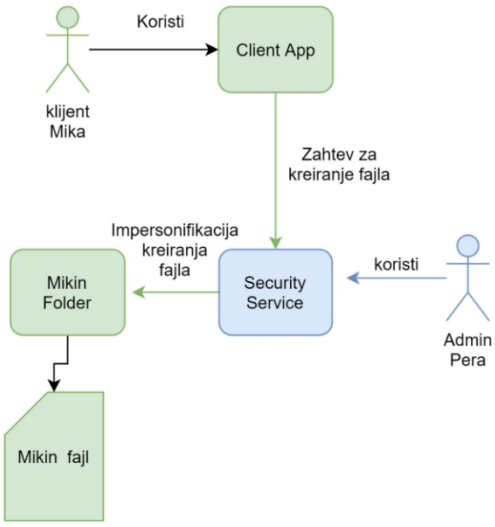
Vezba 2 – Impersonifikacija, Sertifikati

Impersonifikacija

Sposobnost niti da izvrši korišćenje različitih bezbednosnih informacija od procesa u čijem je vlasništvu nit. Obično se nit u aplikaciji servera impersonifikuje kao klijent. Ovo omogućava da nit servera deluje u ime tog klijenta da pristupi objektima na serveru ili da proveri valjanost pristupa objektima klijenta.



Dijagram 1- Kreiranje fajla, bez impersonifikacije



Dijagram 2 - Kreiranje fajla, sa impersonifikacijom

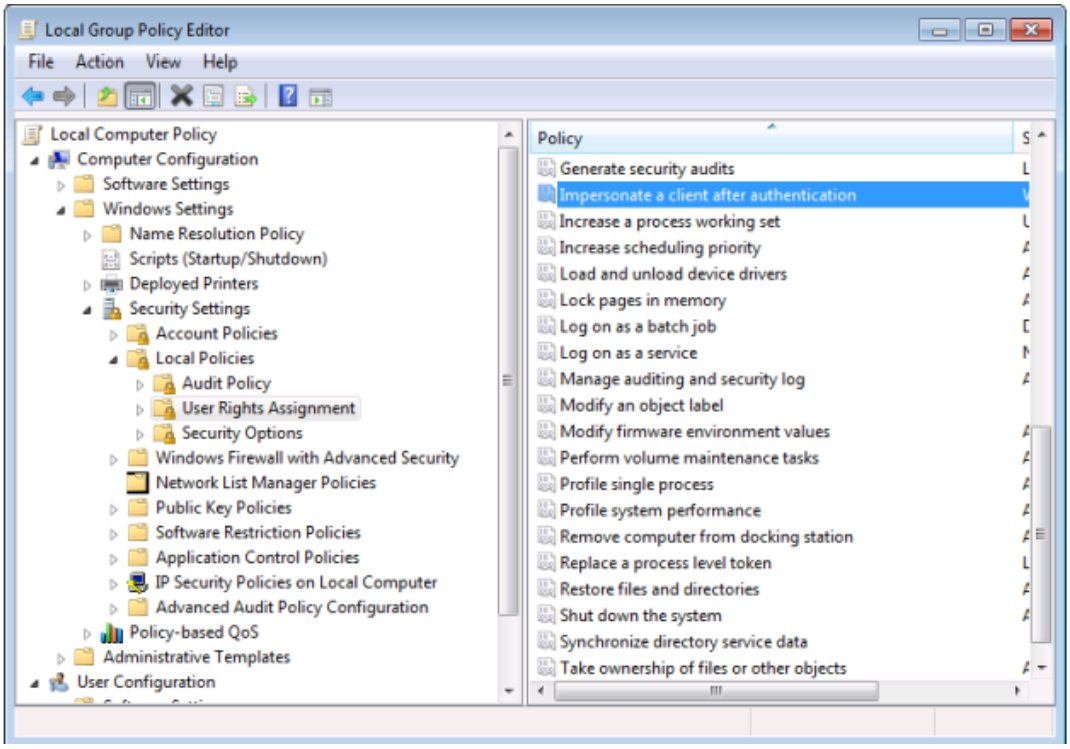
Na dijagramu 1. se može videti postupak kreiranja fajla od stane klijenta Mike bez impersonifikacije, dok se na dijagramu 2. koristi impersonifikacija. Bitno je primetiti razliku u tome ko je kreirao date fajlove sa i bez upotrebe imersonifikacije na servisnoj strani.

Zadatak 3.1. Proširiti zadatak rađen na vežbi 1 tako da bude omogućen poziv metode *CreateFile(string fileName)* u okviru koje se kreira datoteka unutar proizvoljnog foldera nad kojim samo servisni nalog ima pravo pristupa. Potvrditi da je fajl uspešno kreiran. U slučaju neuspešnog pokušaja kreiranja fajla, baciti izuzetak *SecurityException* i obraditi ga na klijentskoj strani.

Zadatak 3.2. Izmeniti metodu CreateFile tako da se izvršava pod klijentskim kredencijalima i potvrditi da će nakon toga kreiranje fajla unutar foldera nad kojim samo servisni nalog ima pravo biti neuspešno. Dodatno, verifikovati da je impersonifikacijom izmenjen identitet procesa.

Napomena: Za potrebe impersonifikacije potrebno je:

- Impersonifikovati odgovarajući deo koda.
- Podesiti proxy tako da klijent dozvoljava impersonifikaciju (TokenImpersonationLevel enumeracija definiše moguće nivoe).
- Grupnom polisom koja je prikazana na slici 2 je potrebno dozvoliti da servisni nalog impersonifikuje pozive klijenta.



Slika 2- Group Policy Editor

Zadatak 3.3. Implementirati metodu koja ima istu logiku kao *CreateFile* metoda, sa tom razlikom da se umesto implicitnog zahteva za impersonifikaciju koristi deklarativni način.

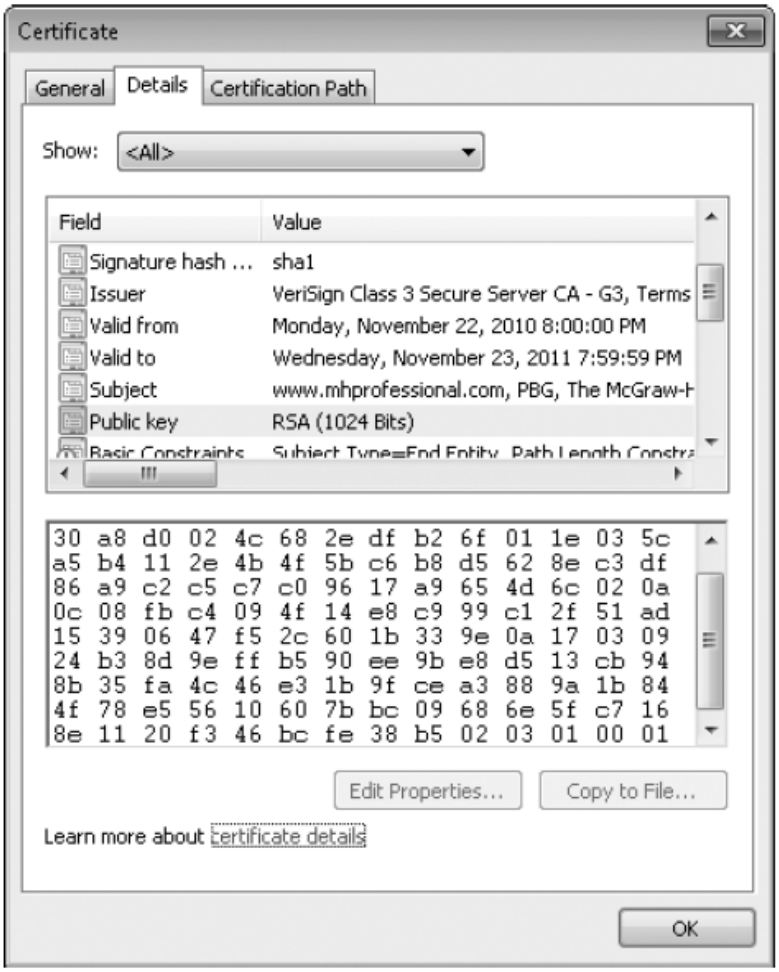
Napomena: Deklarativna impersonifikacija se definiše uz pomoć OperationBehavior atributa na sledeći način: *[OperationBehavior(Impersonation = ImpersonationOption.Required)]*

Sertifikati

Cilj ove vežbe je upoznavanje sa autentifikacionim modelom koji se zasniva na sertifikatima.

Sertifikat predstavlja digitalni identitet korisnika izdat od strane sertifikacionih tela (*certification authority, CA*). Na slici 3. je prikazan primer sertifikata. Sertifikat sadrži različite podatke: podaci o vlasniku sertifikata (*Subject*), validnost odnosno period važenja sertifikata (*ValidFrom, ValidTo*), informacije o izdavaocu sertifikata (*Issuer*). U sertifikat se ugrađuje javni ključ korisnika (uz identifikator algoritma primenjenog za generisanje ključa, npr. *RSA*), dok se tajni ključ ne razmenjuje.

Svaki sertifikat je digitalno potpisan od strane sertifikacionog tela koje ga izdaje čime se potvrđuje da sertifikat zaista pripada podnosiocu zahteva. Na ovaj način je takođe moguće detektovati izmene u okviru samog sertifikata jer digitalni potpis obezbeđuje integritet podataka.



Slika 3. - Primer sertifikata

Postupak izdavanja sertifikata

Sertifikaciono telo je komponenta zadužena za izdavanje i upravljanje sertifikatima. Odgovornosti svakog sertifikacionog tela su sledeće:

- **Verifikacija identiteta** (Verification of a certificate requestor) – Pre nego što izda sertifikat, odgovornost ovlašćenog lica, odnosno CA-a, je verifikacija identiteta onoga ko šalje zahtev. Kada korisnik šalje zahtev za sertifikat, on mora da pošalje sve neophodne informacije koje CA zatim ugrađuje u sertifikat. Tip sertifikata određuje njegov sadržaj, kao i namenu. Npr, IPSec sertifikat može da se koristi samo za autentifikaciju klijenta i servisa u okviru IPSec komunikacije.
- **Izdavanje sertifikata** (Issuing certificates to requestors) – Nakon uspešne verifikacije identiteta korisnika, računara, servisa, mrežnog uređaja CA izdaje digitalno potpisani sertifikat. Odnosno, sertifikat sa svim potrebnim informacijama će dodatno biti potpisan privatni ključem sertifikacionog tela kako bi se omogućila detekcija neovlašćenih izmena sadržaja sertifikata, ali i potvrda da je upravo određeni CA izdao sertifikat.
- **Povlačenje sertifikata** (Certificate revocation) – U nekim situacijama kao što je kompromitovanje sertifikata, potrebno je povući sertifikat iako je još uvek validan. CA ima listu povučenih sertifikata odnosno njihovih serijskih brojeva, uključujući i razlog zbog čega je svaki sertifikat povučen – CRL.

Uputsvo za izdavanje sertifikata

Potrebni alati:

- **makecert** – alat za generisanje sertifikata. Osim sertifikata sa odgovarajućim javnim ključem u okviru ***.cer*** fajla, moguće je generisati i dodatni ***.pvk*** fajl koji sadrži tajni ključ. Sertifikati kreirani na ovaj način su digitalno potpisani od strane testnih CA-eva.
- **pvk2pfx** – alat za generisanje .pfx fajla na osnovu javnog i privatnog ključa, odnosno testni sertifikat instaliran na osnovu .pfx fajla sadržaće i javni i privatni ključ. Za pristup privatnom ključu dodatno je potrebno dodeliti prava odgovarajućem korisniku nad prethodno instaliranim sertifikatom.
  Lokacija : *“ C:\Program Files (x86)\Windows Kits\10\bin\[brojVerzije – razlicit kod svakoga]\x86“ (ova dva alata se nalaze na istoj lokaciji)*
- **certmgr.msc** – mmc za rad sa sertifikatima. Pruža mogućnost rada sa sertifikatima specifičnim za trenutno logovanog korisnika na Windowsu (*currentuser*), kao i sa sertifikatima skladištenim u okviru određenog računara (npr. *localmachine*).
  Napomena*: Komanda certmgr.msc otvara storage za current user, dok se storage za local machine otvara na sledeći način:* ***Start → mmc.exe → File → Add/Remove Snap-in → Certificates → Add → Computer Account***

Potrebni korisnički nalozi

Korisnički nalozi: Kreirati lokalne korisničke naloge pod kojima će biti pokrenuta servisna, odnosno klijentska aplikacija. Lokalni nalozi se kreiraju u okviru “Computer Management“ konzole.

- **wcfservice** – servisni nalog
- **wcfclient** – klijentski nalog

Primer generisanja sertifikata korišćenjem makecert alata

Napomena : Command Prompt pokrenuti kao Administrator

1. Generisanje TestCA sertifikata

```
> makecert -n "CN=TestCA" -r -sv TestCA.pvk TestCA.cer
```

Generisan je self-signed sertifikat, koji je potrebno instalirati na "Trusted Root Certification Authorities" lokaciji.

2. Generisanje WCFService sertifikata za uspostavljanje komunikacije

2.1. Generisanje .pvk i .cer fajla

```
> makecert -sv WCFService.pvk -iv TestCA.pvk -n "CN=wcfservice" -pe -ic TestCA.cer WCFService.cer -sr localmachine -ss My -sky exchange
```

Namena (uspostavljanje komunikacije) je definisano argumentom –sky exchange.

2.2. Generisanje .pfx fajla

```
> pvk2pfx.exe /pvk WCFService.pvk /pi 1234 /spc WCFService.cer /pfx WCFService.pfx
```

Karakteri koji se nalaze posle reči pi predstavljaju sifru sertifikata.
Ovaj fajl je potrebno instalirati (ili importovati iz certmgr.msc) na "Personal" lokaciji, koristeći opciju "Mark key as exportable".

2.3. Dodeljivanje prava pristupa konkretnom korisničkom nalogu

U okviru certification managera neophodno je otici na sertifikat koji sadrzu privatni kljuc (.pfx).
Desni klik na sertifikat - > All Tasks - > Manage Private Keys i izvrsiti dodavanje korisnika kom treba da damo dozvolu da moze da koristi ovaj privatni ključ.
Napomena: **Ova komanda se izvršava nakon instaliranja sertifikata.**

3. Generisanje WCFClient sertifikata za uspostavljanje komunikacije

Na isti način kao u koraku 2 se generišu sertifikati koji služe za komunkikaciju, ono što je neophodno promeniti jeste subjectName "CN=wcfclient", i korisnički nalog kom se dodeljuje pristup je wcfclient.

4. Generisanje sertifikata za digitalno potpisivanje

4.1. Generisanje .pvk i .cer fajla

```
> makecert -sv Sign1.pvk -iv TestCA.pvk -n "CN=wcfclient_sign" -pe -ic TestCA.cer Sign1.cer -sr localmachine -ss My -sky signature
```

Namena (digitalno potpisivanje) je definisano argumentom –sky signature.
Ostali koraci se izvode kao i u koraku 3. s razlikom subjecName-a i naloga kom se dodeljuje pravo pristupa.
Napomena : Ovi sertifikati će se koristiti prilikom digitalnog potpisivanje koje će biti obrađivano u kasnijim lekcijama

Vezbe 3 – Autentifikacija upotrebom sertifikata

Obostrana autentifikacija uz pomoć sertifikata

Zadatak 4.1

Potrebno je obezbediti obostranu autentifikaciju uz pomoć sertifikata, pri čemu se validacija sertifikata zasniva na lancu poverenja (*chain trust*). Rešenje verifikovati pozivom metode *TestCommunication()*.

Prvi korak u realizaciji rešenja je generisanje sertifikata. S obzirom da je u pitanju obostrana autentifikacija sertifikatima, potrebno je izgenerisati sertifikat za svakog od učesnika u komunikaciji. *Sertifikate generisati tako da subjectName odgovara korisničkom imenu naloga*.

U nastavku su nabrojane komponente koje učestvuju u komunikaciji i čije sertifikate je neophodno instalirati na odgovarajuću lokaciju na računaru, a zatim za odgovarajuće korisnike dodeliti prava pristupa privatnom ključu odgovarajućeg sertifikata:

- ***TestCA*** – komponenta od poverenja zadužena za izdavanje ostalih testnih sertifikata. Prilikom instalacije kod korisnika izdatog sertifikata samo je javni ključ dozvoljeno instalirati. Privatni ključ je vlasništvo CA komponente i **nikad se ne instalira**. Instalira se u okviru ***Trusted Root Certification Authorities*** lokacije.
- ***WCFService*** – servisni sertifikat za autentifikaciju. Privatni ključ se instalira samo kod vlasnika sertifikata (odnosno na mašini na kojoj je pokrenut servis). Instalira se u okviru lokacije ***Personal***.
- ***WCFClient*** – klijentski sertifikat za autentifikaciju. Privatni ključ se instalira samo kod vlasnika sertifikata (odnosno na mašini na kojoj je pokrenuta klijentska aplikacija). Instalira se u okviru lokacije ***Personal***.

Potrebno je omogućiti čitanje sertifikata iz skladišta sertifikata, kao i iz fajlova (.cer i .pfx) za potrebe servisne i klijentske aplikacije. Implementacija ovih metoda treba da bude u posebnom CertificateManager DLL-u. Zaglavlja metoda su opisane u sledećem kodu:

```
public static X509Certificate2 GetCertificateFromStorage(StoreName storeName,
StoreLocation storeLocation, string subjectName)

public static X509Certificate2 GetCertificateFromFile(string fileName)

public static X509Certificate2 GetCertificateFromFile(string fileName, SecureString
pwd)
```

Prvi korak je podešavanje bindinga tako da podrži autentifikaciju uz pomoć sertifikata. Ovim je definisan tip kredencijala koji se očekuje od drugog učesnika u komunikaciji.

Zatim je potrebno da svaki učesnik u komunikaciju podesi svoj sertifikat kojim se predstavlja drugim učesnicima u komunikaciji.

- Svaka komponenta mora da podesi .pfx sertifikat, jer na taj način dokazuje da je vlasnik tog sertifikata (ima privatni ključ);
- Servis podešavanjem sertifikata na hostu: ***host.Credentials.ServiceCertificate.Certificate;***
- Klijent podešavanjem proxy-a: ***proxy.Credentials.ClientCertificate.Certificate;***
- .NET podržava X509 standard implementiran klasom ***X509Certificate2*** iz *System.Security.Cryptography.X509Certificates* namespace.

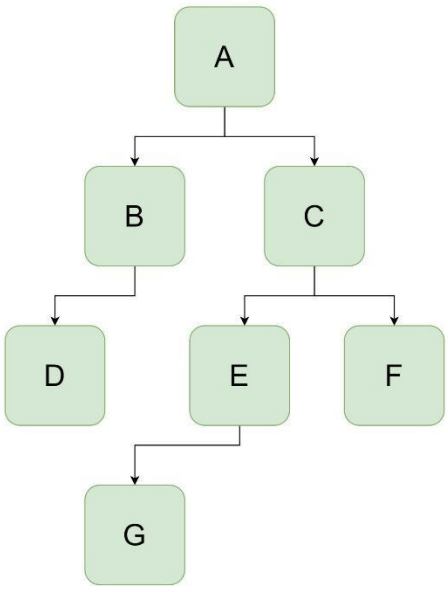
Kao deo obostrane autentifikacije, neophodno je ***podesiti identitet servisa koji klijent očekuje*** prilikom uspostavljanja komunikacije. Za to je potrebno proslediti *EndpointIdentity* prilikom uspostavljanja komunikacije. Objekat tipa *EndpointAddress* može se kreirati na sledeći način:

```
EndpointAddress address = new EndpointAddress(new Uri("net.tcp://localhost:9999/Receiver"),
new X509CertificateEndpointIdentity(srvCert)); //srvCer je .cer servisnog sertifikata
```

Napomena: Klijent mora instalirati serverski sertifika(.cer) na svoju mašinu u folder Thrusted People

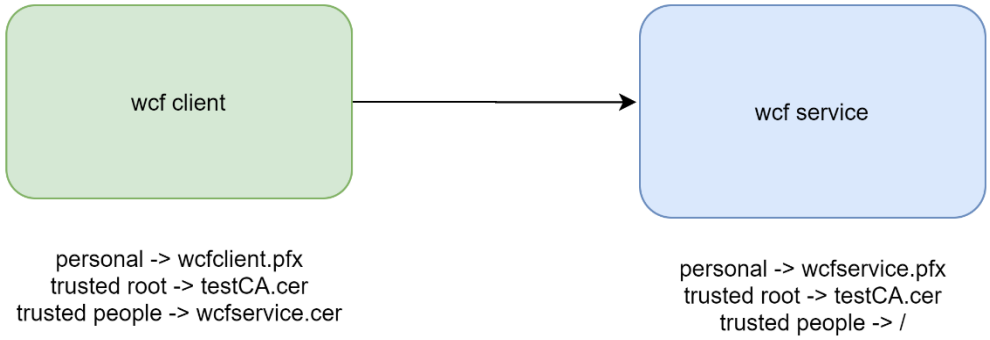
Poslednji korak je definisanje tipa validacije sertifikata kod klijenta i servera.

```
Credentials.ServiceCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.ChainTrust;
Credentials.ServiceCertificate.Authentication.RevocationMode =
       X509RevocationMode.NoCheck;

host.Credentials.ClientCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.ChainTrust;
host.Credentials.ClientCertificate.Authentication.RevocationMode =
       X509RevocationMode.NoCheck;
```



Kod ChainTrust validacije se na osnovu izdavaoca sertifikata zaključuje da li će se verovati određenoj strani, odnosno ukoliko je sertifikat sa kojim se neko predstavlja izdat od strane nekoga kome određena strana veruje taj sertifikat će se smatrati vallidnim.

Na slici 4 možemo da vidimo jedan primer izdavanja sertifikata. Ukoliko koristimo ChainTrust validaciju možemo da kažemo da strana D i strana F mogu da veruju jedna drugoj jer je F izdao C, koji je izdao A, dok je D izdao B, kojeg je izdao A. I sam tim zaključujemo da su oba ova sertifikata potekla od istog sertifikata kom veruju obe strane.



Slika 5 - Instalacija sertifikata

Sa slike 5 možemo da vidimo na kojim sve lokacijama i koji sve sertifikati su nam neophodni za realizaciju zadatka.

Zadatak 4.2

Potrebno je izmeniti prethodni zadatak tako da se umesto *ChainTrust* validacije sertifikata koristi Custom validacija.

Za potrebe ovog primera, implementirati validatore (*nasleđuju klasu X509CertificateValidator*):

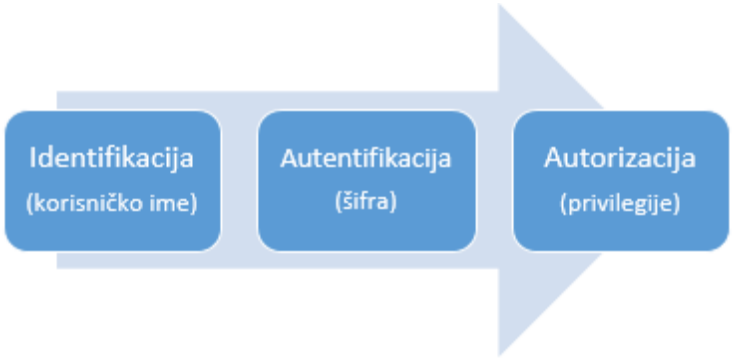
- ***ServiceCertValidator*** klasa – klijentski sertifikat je validan ukoliko je potpisan od strane istog sertifikacionog tela kao i servisni sertifikat;
- **ClientCertValidator** klasa – sevisni sertifikat je validan ukoliko nije self-signed.

```
Credentials.ServiceCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.Custom;
Credentials.ServiceCertificate.Authentication.CustomCertificateValidator =
       new ClientCertValidator();

host.Credentials.ClientCertificate.Authentication.CertificateValidationMode =
       System.ServiceModel.Security.X509CertificateValidationMode.Custom;
host.Credentials.ClientCertificate.Authentication.CustomCertificateValidator =
       new ClientCertValidator();
```

Vezba 4 - Autorizacija

Cilj ove vežbe je upoznavanje sa mehanizmom autorizacije. Mehanizam autentifikacije treba da obezbedi validaciju identiteta i tako spreči pristup nevalidnim korisnicima. Međutim, različiti korisnici mogu imati različita ovlašćenja u sistemu kome pristupaju. Na primeru putnika koji se na carini autentifikuje tako što posedovanjem pasoša potvrđuju svoj identitet, to bi značilo da uspešno autentifikovani putnici mogu imati različita prava u zemlji u koju ulaze, jedni su državljani, drugima su prava definisana vizom pa tako možemo imati putnike koji smeju da borave u zemlji mesec dana, putnike kojima je dozvoljen boravak godinu dana, itd. Autentifikacijom nije moguće definisati različit nivo ovlašćenja za validne korisnike u sistemu. Proces definisanja ovlašćenja validnim entitetima u sistemu, kao i proces odlučivanja kojim resursima tog sistema entitet može da pristupi i koje operacije nad resursima može da izvrši se naziva autorizacija.

Iz prethodno opisanog primera mogu se uočiti tri procesa u okviru provere prava pristupa resursima u sistemu: identifikacija, autentifikacija i autorizacija. Ova tri procesa zajedno čine mehanizam kontrole pristupa kojim se definiše da li i na koji način korisnici mogu pristupiti resursima u sistemu, odnosno "ko šta može da radi u sistemu". Na slici je konceptualno prikazan sistem kontrole pristupa. Identifikacija je proces u okviru koga se utvrđuje da li je predstavljeni entitet poznat sistemu. Entitet se predstavlja sistemu korišćenjem jedinstvenog identifikatora (npr. korisničko ime), ali se ne utvrđuje verodostojnost ove tvrdnje. Autentifikacija je proces validacije identiteta u sistemu. Da bi entitet mogao da pristupi sistemu potrebno je da dokaže da je on upravo onaj za koga se izjašnjava (npr. šifra). Autorizacija je proces provere ovlašćenja u sistemu, odnosno proces odlučivanja kojim resursima entitet može da pristupi i koje operacije može da izvršava nad resursima u sistemu.



Slika 6 - Sistem za kontrolu pristupa

U okviru ove vežbe, studenti će prvo biti upoznati sa osnovnim konceptima mehanizma kontrole pristupa koji je podržan u .NET razvojom okruženju, a zatim će biti implementiran model kontrole pristupa zasnovan na korisničkim ulogama (eng. *role-based access control – RBAC*).

Osnovni mehanizam kontrole pristupa u .NET-u

Zadatak 5.

U okviru ovog zadatka potrebno je obezbediti proveru prava korisnika koji pristupaju servisu pre nego što se dozvoli izvršvanje pozvanih metoda na sledeći način.

Servis nudi tri metode:

- Metoda **`Read`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Reader`** grupe.
- Metoda **`Modify`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Modifier`** grupe.
- Metoda **`Delete`** – ovu metodu je moguće izvršiti ukoliko je korisnik član **`Admin`** grupe.

U slučaju neuspešnog pokušaja pristupanja metodi, potrebno je baciti izuzetak (SecurityException) i obraditi ga na klijentskoj strani. U okviru poruke, izuzetak treba da vrati sledeće informacije: korisnik koji je pokušao da pristupi servisu, metoda servisa kojoj je pokušao da pristupi, grupa kojoj korisnik treba da pripada da bi mogao da pristupi metodi i vreme poziva.

Prilikom pokretanja aplikacija koristiti lokalne korisničke naloge i korisničke grupe. Podesiti radno okruženje tako da klijentska i serverska aplikacija budu pokrenute kao različiti korisnici, npr. servisna aplikacija da bude pokrenuta kao **`wcfservice`** korisnik, a klijentska aplikacija kao **`wcfclient`** korisnik.

Napomena: Kontrolu pristupa na WCF servisu moguće je implementirati na dva načina:

- Deklarativna provera privilegija koristeći **`PrincipalPermissionAttribute`** atribut.
- Imperativna provera privilegija pozivom metode **`IsInRole`** **`IPrincipal`** interfejsa.

Zadatak 5.1 Deklarativna provera privilegija

Deklarativna provera privilegija se izvršava pre ulaska u pozvanu metodu.

```
[PrincipalPermission(SecurityAction.Demand, Role="Ime_privilegije")]
void MetodaServisa()
{ ... }
```

Zadatak 5.2 Imperativna provera privilegija

Imperativna provera privilegija podrazumeva eksplicitan poziv metode za proveru privilegija na proizvoljnom mestu u telu metode.

Zadatak 5.3

Izmeniti Zadatak 5 tako da ukoliko korisnik nije član **`Reader`** grupe ne može da pristupi nijednoj metodi. Ukoliko je korisnik član **`Reader`** grupe, proveriti prava pristupa metodama servisa kao što je opisano u Zadatku 5.

Napomena: WCF nudi mogućnost centralizovane kontrole pristupa u slučajevima kada postoji zajednički preduslov za pristup svim metodama servisa. **`ServiceAuthorizationManager`** je ugrađena .NET klasa nudi mogućnost override-a **`CheckAccessCore`** metode u okviru koje se definiše ponašanje servisa prilikom poziva metoda koje on izlaže. Podrazumevano ponašanje ove metode je da vraća vrednost **`true`**, odnosno da je pristup dozvoljen.

Jedan način registracije objekta `ServiceAuthorizationManager` klase na WCF servisu je:

```
ServiceHost host = new ServiceHost(typeof(WCFService));
host.AddServiceEndpoint(typeof(WCFService), binding, address);

host.Authorization.ServiceAutorizationManager = new MyAuthorizationManager();
```

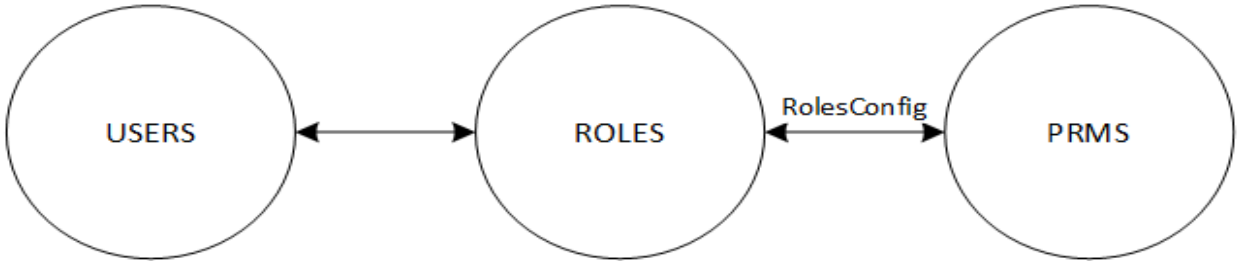
Vežba 5 – RBAC i ACL

RBAC – Role-Based Acces Control

RBAC model je model koji se koristi prilikom implementacije autorizacije u sistemima. Sam model je zasnovan permisijama koje se dodeljuju određenim ulogama (slika 7). U RBAC modelu su definisana tri osnovna pravila :

1. Dodeljivanje uloga
2. Autorizacja uloga
3. Ovlašćenje za dozvolu

Korsniku sistema može biti dodeljeno više uloga, samim tim skup njegovih permisija je skup permisija svih njegovih uloga. Dobra praksa RBAC modela jeste da se za jednu rolu vezuje više permisija, jer ukoliko se za jednu ulogu vezuje samo jedna permisija to nije dobra implementacija, onda bismo mogli da koristimo samo ugrađene grupe bez dodavanja permisija.



Slika 7 – RBAC model

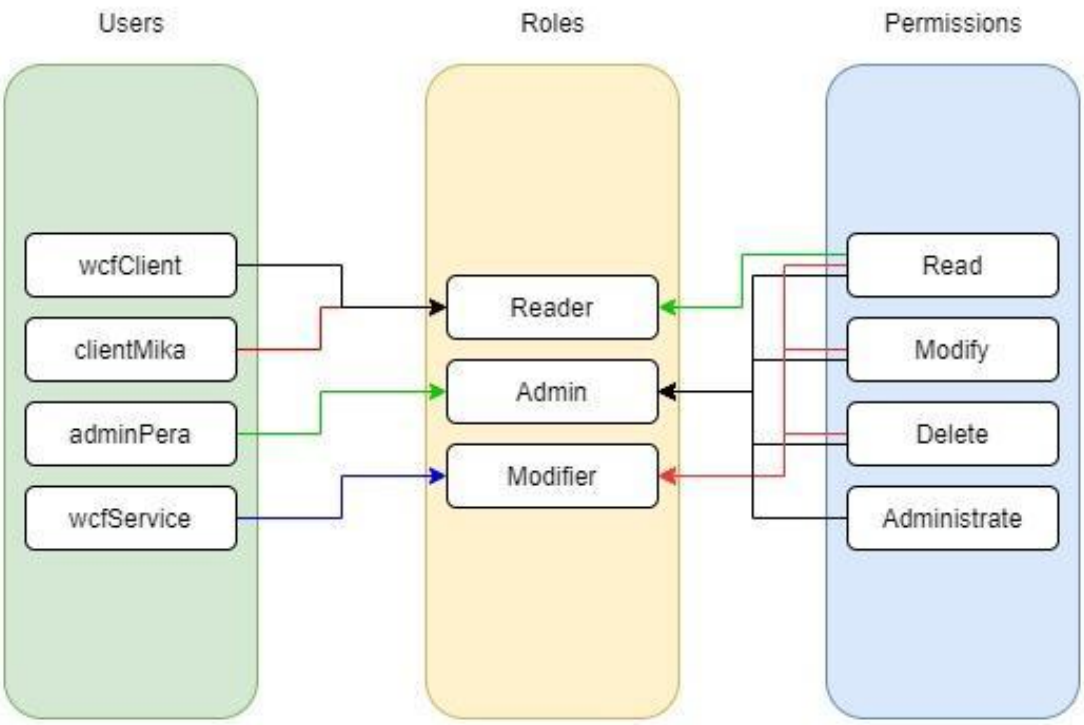
Zadatak 6.

Izmeniti zadatak 5. tako da se umesto ***WindowsPrincipal*** objekta koristi novodefinisana klasa ***CustomPrincipal*** koja implementira specifično ponašanje ***IPrincipal*** interfejsa. Kroz ovu klasu je potrebno implementirati RBAC model (slika 7) sa sledećom bezbednosnom politikom:

- **USERS** – skup korisnika koji pristupaju servisu.
- **ROLES** – skup korisničkih uloga u sistemu su predstavljenje Windows grupama koje se dodeljuju korisnicima {Reader, Modifier, Admin}
- **PRMS** – skup privilegija (permisije) kojima se definiše pristup metodama servisa {Read, Modify, Delete, Administrate}. Privilegije se definišu na aplikativnom nivou, odnosno privilegije nisu korisničke grupe kreirane na Windows OS.

Korisnicima se dodeljuju isključivo korisničke uloge, a ulogama se dodeljuju privilegije. Direktno uvezivanje korisnika sa privilegijama nije dozvoljeno, već ih korisnik dobija kroz pripadnost ulogama u sistemu.

U sistemu su definisane sledeće tri uloge (slika 8): Reader {Read}, Modifier {Read, Modify,Delete} i Admin {Read, Modify,Delete,Administrate}



Slika 8- Odnos permisija i rola

Kako bi se omogućilo korišćenje klase ***CustomPrincipal*** na serverskoj strani je neophodno podesiti ***PrincipalPermissionMode*** na Custom, takođe je neophodno podesiti autorizacionu polisu, odnosno ***CustomAuthorizationPolicy*** klasu koja implementira interfejs ***IAuthorizationPolicy***. U okviru klase ***CustomAuthorizationPolicy*** u metodi ***Evaluate*** se definiše koji identitet će se koristiti, odnosno na kontekst se podešava onaj objekat koji se kasnije koristi kao identitet.

Pošto je za sve metode neophodna permisija Read neophodno je implementirati i klasu ***CustomAuthorizationManager*** i override-ovati metodu ***CheckAccesCore***, voditi računa da se radi sa CustomPrincipal objektom.

Zadatak 6.1

Implementirati metode WCF servisa za izmenu RBAC konfiguracije (dodati/obrisati novu privilegiju, dodati/obrisati novu korisničku ulogu, ažurirati relacije između privilegija i korisničkih uloga). Ovu metodu može da izvrši samo visoko-privilegovani korisnik kom je dodeljena permisija ***Administrate***.

ACL – Access Control List

Model listi kontrole pristupa (eng. *access control list - ACL*) je model u kojem je pristup objektima definisan na osnovu korisničkog identiteta i autorizacionih pravila kojima je za svakog korisnika ili korisničku grupu određen skup dozvoljenih operacija nad datim objektom. ACL se tipično koristi kao mehanizam kontrole pristupa na operativnim sistemima kao što su Unix/Linux i Windows za upravljanje sistemom fajlova.

Zadatak 7.

Implementirati WCF servis koji pruža interfejs **`IFileControl`** za upravljanje pristupom sistemu fajlova. Pokretnjem klijentske aplikacije pod različitim korisničkim nalozima pokazati da samo ovlašćeni korisnici mogu da kreiraju, modifikuju i brišu fajlove.

```
[ServiceContract]
public interface IFileControl
{
        [OperationContract]
        [FaultContract(typeof(SecurityException))]
        void CreateFolder(string name, string path);

        [OperationContract]
        [FaultContract(typeof(SecurityException))]
        void DeleteFolder(string name, string path);

        [OperationContract]
        [FaultContract(typeof(SecurityException))]
        void CreateTxtFile(string name, string path, string text);

        [OperationContract]
        [FaultContract(typeof(SecurityException))]
        void UpdateTxtFile(string name, string path, string text, bool append);

        [OperationContract]
        [FaultContract(typeof(SecurityException))]
        void DeleteTxtFile(string name, string path);

}
```

Napomena: **DirectorySecurity** je .NET klasa iz **System.Security.AccessControl** namespace-a za upravljanje bezbednošću fajl sistema.

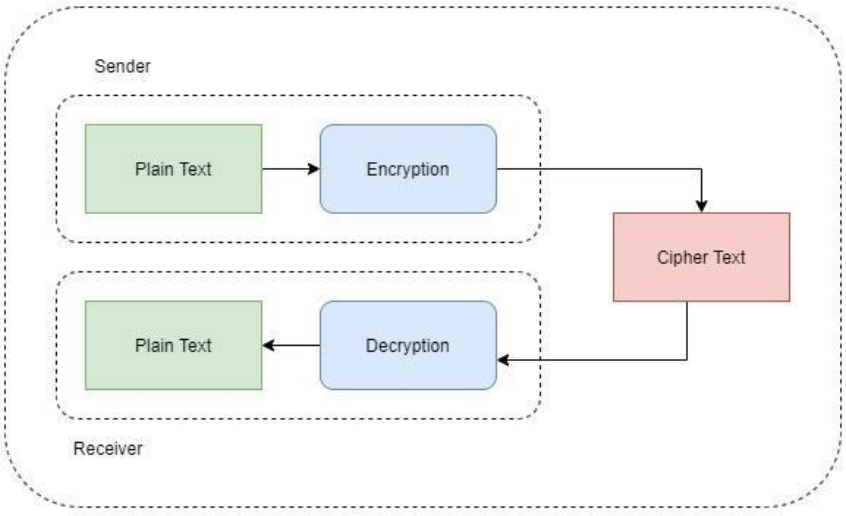
Vežba 6 – Kriptografija

Osnovni pojmovi u kriptografiji

Sa pojavom računara, otvorile su se i nove mogućnosti kada je u pitanju kriptografija. Računari su postajali sve brži, radeći po nekoliko stotina, a kasnije i nekoliko miliona operacija u sekundi čime je omogućeno probijanje šifri za sve manje vremena, ali i definisanje kompleksnijih i sigurnijih algoritama koji rade nad nizovima bajtova. S druge strane, kriptografija danas ima veliku važnost za bezbednost računarskih sistema, s obzirom da se informacije prenose raznovrsnim otvorenim i nesigurnim komunikacionim kanalima koje nije moguće u potpunosti fizički zaštititi, tako da je širok spektar mogućnosti za neovlašćeno pristupanje i čitanje, modifikaciju ili brisanje podataka, a time i narušavanje sigurnosti celokupnog sistema. Iz tog razloga, jaki bezbednosni mehanizmi za zaštitu nesigurnih komunikacionih kanala postaju najvažniji oblik ostvarenja bezbednosti, a jedan od najvažnijih zadataka je očuvanje poveljivosti podataka, kako u toku prenosa tako i dok su skladišteni.

Kriptografija je nauka koja se bavi metodama očuvanja poverljivosti podataka. Čitljiv tekst (plaintext) je informacija kojoj je potrebno očuvati poverljivost (tajnost), dok šifrovana poruka (cipher text) predstavlja nečitljiv tekst dobijen transformacijom čitljivog teksta. Transformacija čitljivog teksta u nečitljiv format je kriptovanje (encryption), dok je dekriptovanje (decryption) postupak vraćajna šifrovanog teksta u čitljiv oblik. Tajni ključ se koristi prilikom kriptovanja tako da samo korisnici koji znaju vrednost tajnog ključa mogu dekriptovati šifrovane podatke. Na slici 9 je dat šematski prikaz šifrovanja i dešifrovanja. Uopšteno, kriptografski sistem se može definisati kao petorka (P, C, K, E, D), gde je:

- P – skup čitljivih poruka,
- C – skup šifrovanih poruka,
- K – skup ključeva,
- E (P,K) ⟶ C – funkcija šifrovanja
- D (C, K) ⟶ P – funkcija dešifrovanja



Slika 9 - Šematski prikaz procesa kriptovanja i dekriptovanja

Osim očuvanja poverljivosti, kriptografija ima primenu i u očuvanju integriteta podataka, kao i obezbeđivanju autentifikacije i neporecivosti. Dakle, osnovni zadaci kriptografije se odnose na:

- **Poverljivost/Tajnost podataka (Confidentiality)** – prevencija od neovlašćenog pristupa podacima, tako da samo autorizovani učesnici u komunikaciji imaju pristup sadržaju podataka
- **Integritet podataka (Integrity)** – očuvanje sadržaja poruke od neovlašćenih izmena ili brisanja
- **Autentifikacija (Authentication)** – verifikacija identiteta učesnika u komunikaciji, odnosno potvrđivanje porekla i odredišta poruke (digitalni potpisi)
- **Neporecivost (Non-repudiation)** – nemogućnost pošiljaoca poruke da negira slanje poruke koju je poslao.

Vrste kriptografskih algoritama

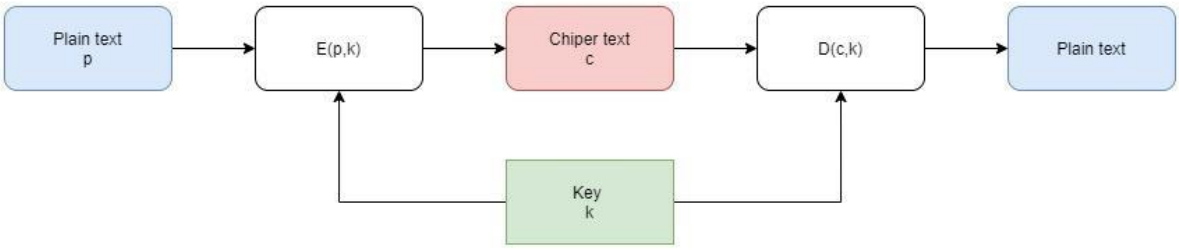
Kriptografski algoritam je skup dve matematičke funkcije – funkcija kriptovanja i funkcija dekriptovanja. Ranije su se metode šifrovanja zasnivale na tajnosti algoritama, ali su se takve metode pokazale kao vrlo nepouzdane. Bezbednost današnjih metoda šifrovanja se ne zasniva na tajnosti algoritma, detalji algoritama su javni, već se bezbednost zasniva na tajnosti ključa. U zavisnosti od načina korišćenja ključa, postoje dve grupe kriptografskih algoritama:

- **Simetrični algoritmi** - isti ključ se koristi za enkripciju i dekripciju poruke (shared secret key – symmetric cryptography)
- **Asimetrični algoritmi** - ključevi za enkripciju i dekripciju su različiti, od kojih je jedan javni i poznat svima, a drugi tajni

Simetričnih kriptografskih algoritama

Simetrični kriptografski algoritmi (Shared secret key) su algoritmi koji koriste isti ključ za kriptovanje i dekriptovanje podataka. Šematski prikaz šifrovanja i dešifrovanja simetričnim algoritmima dat je slici 10.

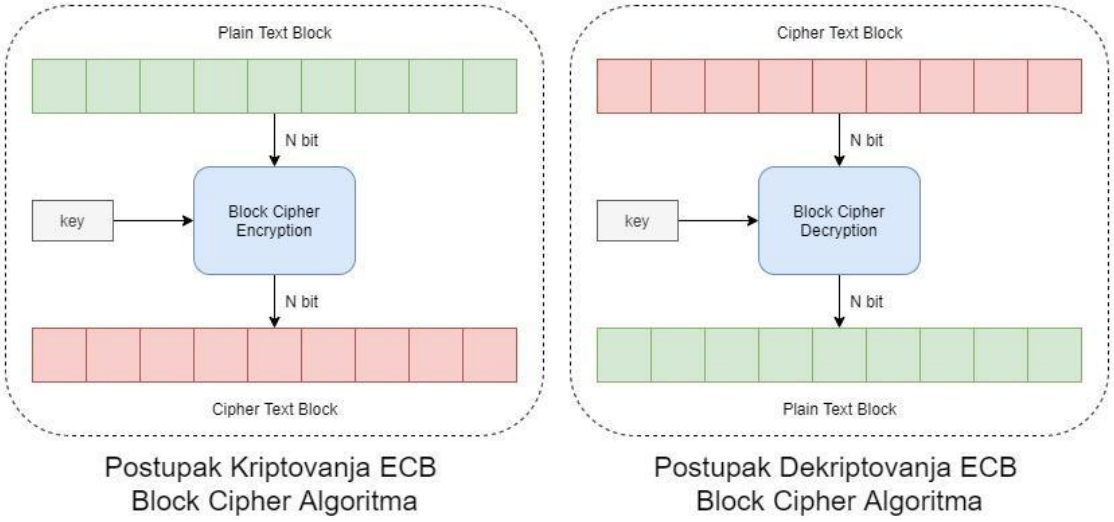
Osnovna prednost simetričnih algoritama jeste to što nisu računski intenzivni, tako da se velike količine podataka mogu brzo kriptovati/dekriptovati. Mana simetričnih algoritama je to što je potrebno obezbediti bezbedan način za distribuciju tajnog ključa, tj. neophodno je obezbediti bezbedan kanal za razmenu ključeva između zainteresovanih strana. Ukoliko bi takav kanal postojao, kriptografija ne bi bila ni potrebna - jednostavno bi se sami podaci slali preko takvog kanala.



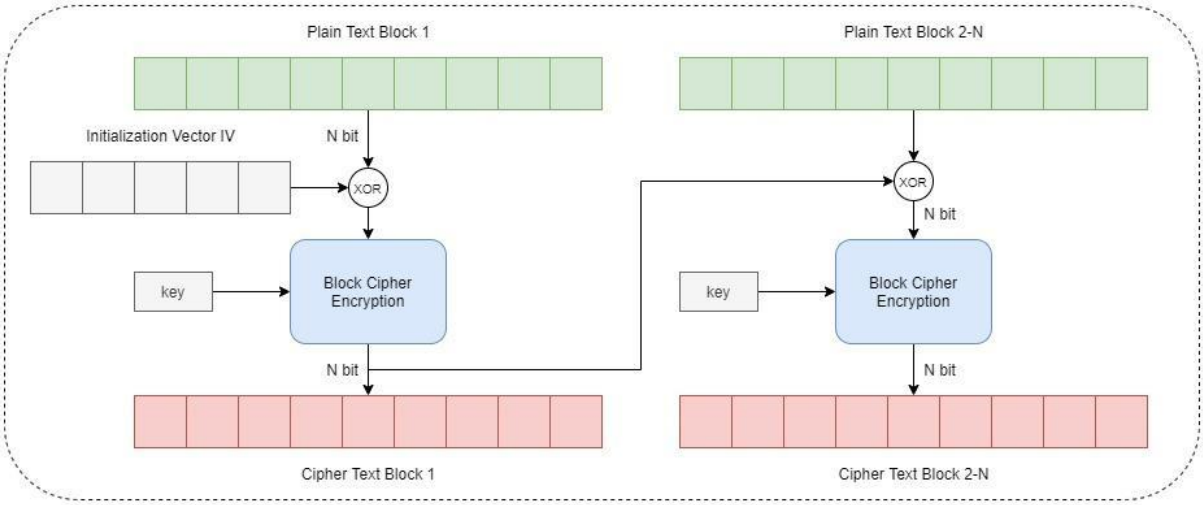
Slika 10 - Šematski prikaz simetričnih kriptografskih algoritama

Simetrični kriptografski algoritmi mogu da se podele na dve grupe, na osnovu toga na koji način se kriptuju podaci :

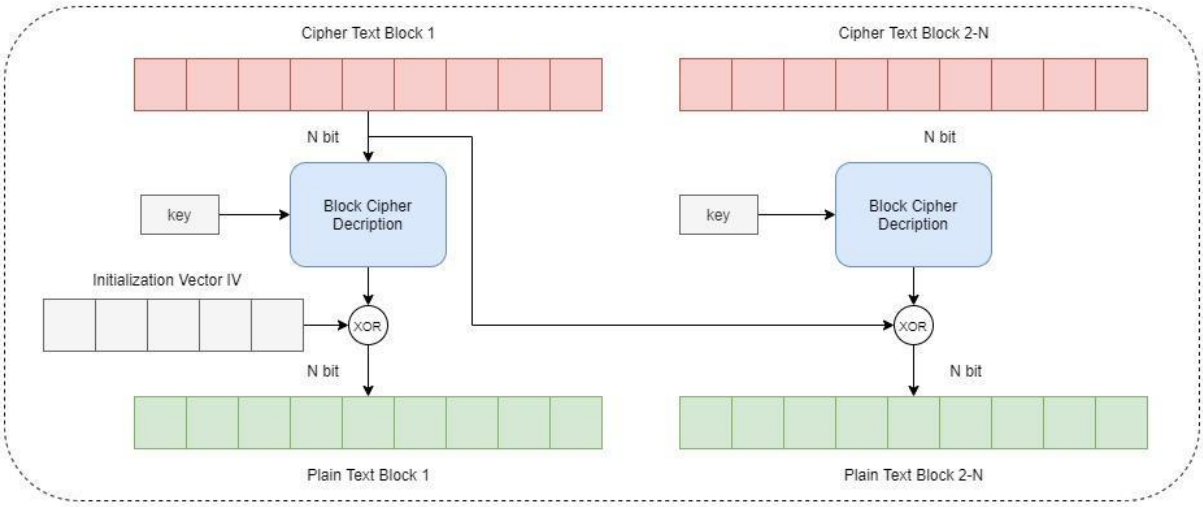
- **Stream cipher**
  - Algoritam koji radi nad bitovima podataka. Šifruje se svaki bit poruke posebno, čime je moguće šifrovati poruke proizvoljne dužine.
  - Šifrovanje se vrši bit po bit primenom operacije XOR (“ekskluzivno ili”) sa odgovarajućim bitom *keystream*, a dobijena šifrovana poruka će biti iste dužine kao i *plaintext*.
  - Postupak dekriptovanja se vrši na isti način - primenom XOR operacije između bitova šifrovanog teksta i keystream vrednosti.
  - Najpoznatiji algoritam ovog tipa je RC4
- **Blok cipher**
  - Rade nad blokovima podataka fiksne dužine (poruka se deli na n-bitne blokove, a ukoliko je poslednji blok manji dopunjava se tako do n bita).
  - Nad svakim blokom podataka se primenjuje algoritam koji je matematički dosta kompleksiniji od stream cipher algoritama (kombinovane operacije zamene i permutacije).
  - Algoritam se primenjuje u iterativnim postupcima, odnosno rundama.
  - Rezultat je takođe blok podataka iste dužine kao i ulazni blok.
  - Dva tipična moda block cipher algoritama su:
    - ECB - Electronic Codebook mod (slika 11)
    - CBC - Cipher Block Chaining mod (slika 12)
  - Neki od poznatijih algoritama su AES, DES, 3DES



Slika 11- ECB mode



Postupak Kriptovanja CBC Block Cipher Algoritma



Postupak Dekriptovanja CBC Block Cipher Algoritma

Slika 12 - CBC

Zadatak 8. Poređenje ECB i CBC moda kripto-algoritama u .NET razvojnom okruženju

Korišćenjem proizvoljnog block cipher algoritma pokazati veću sigurnost CBC moda u odnosu na ECB mod.

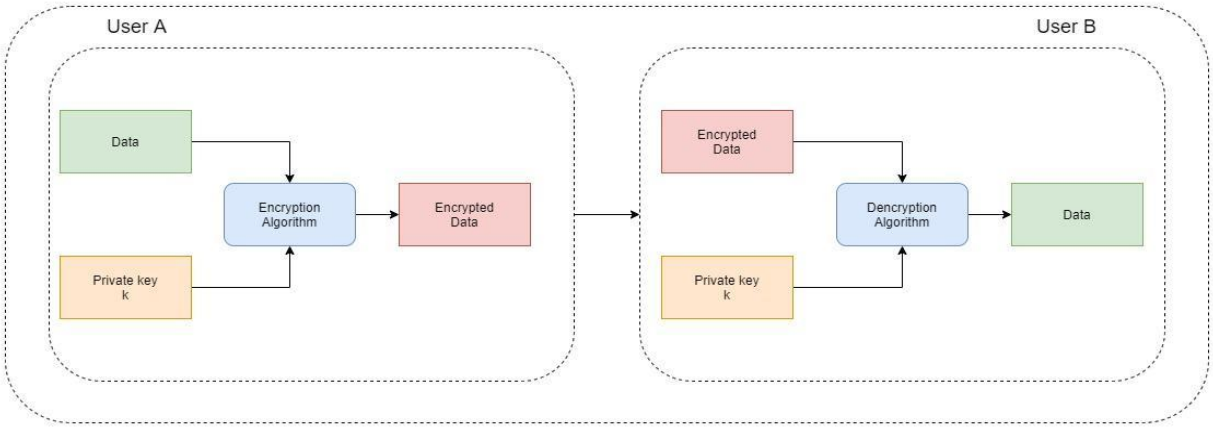
1. Unapred zadatu sliku u bitmap formatu kriptovati koristeći ECB mod proizvoljnog block cipher algoritma. Rezultat kriptovanja sačuvati u bitmap fajlu.
2. Obrnutim postupkom na osnovu rezultata iz koraka 1. dobiti prvobitni bitmap fajl.
3. Unapred zadatu sliku u bitmap formatu (kao iz koraka 1.) kriptovati koristeći CBC mod proizvoljnog block cipher algoritma. Rezultat kriptovanja sačuvati u bitmap fajlu.
4. Obrnutim postupkom na osnovu rezultata iz koraka 3. dobiti prvobitni bitmap fajl.

5. Uporediti rezultate kriptovanja iz koraka 1. i 3. Šta se može zaključiti?

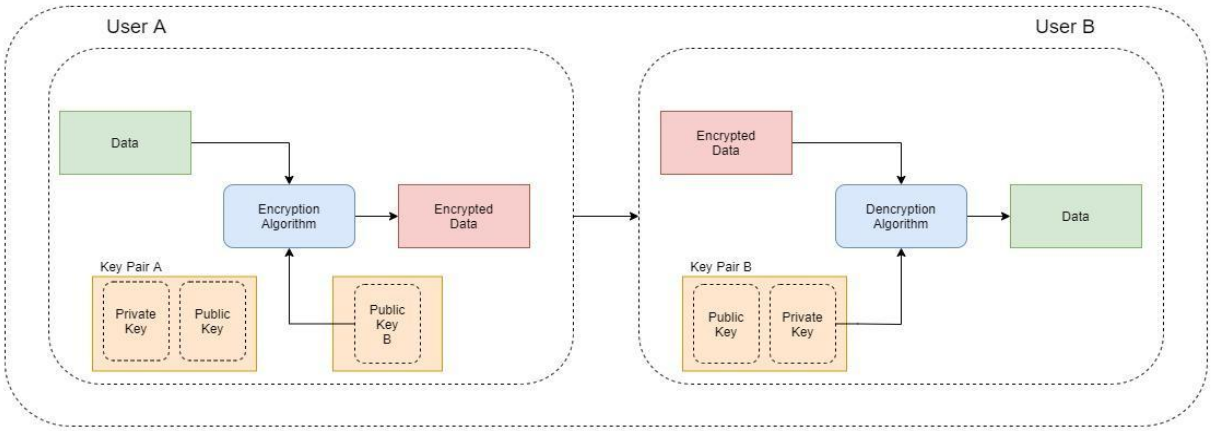
Vežba 7 - Digitalni Potpisi

Asimetricni kriptografski Algoritmi

Asimetrični kriptografski algoritmi su algoritmi koji, umesto jednog tajnog ključa za kriptovanje i za dekriptovanje podataka, koriste dva različita ključa – jedan ključ se koristi za kriptovanje poruke, a drugi za dekriptovanje. Jedan ključ je tajni (private) i poseduje ga samo jedan od učesnika u komunikaciji, a drugi ključ je javni (public) i on je dostupan svim učesnicima zainteresovanim za komunikaciju. Ovim je rešen problem sigurne razmene tajnih ključeva kod simetričnih algoritama. Na slikama 13 i 14 je konceptualno prikazana osnovna razlika između simetričnih i asimetričnih algoritama.



Slika 13 - Simetrični algoritmi



Slika 14- Asimetrični algoritmi

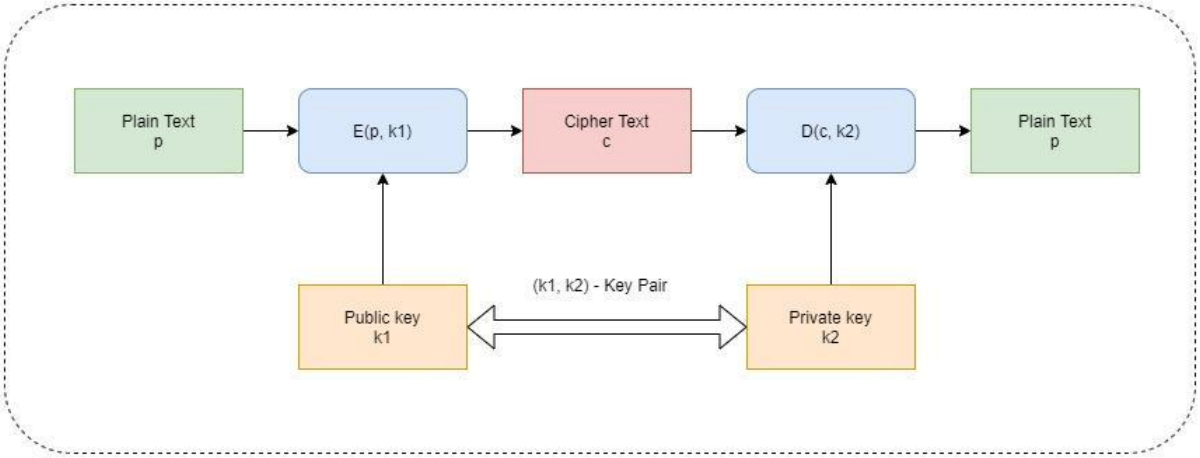
Kriptografija javnog ključa kako se još nazivaju asimetrični kripto-algoritmi, se zasniva na kriptovanju i dekriptovanju poruke sa dva različita ključa pri čemu moraju da važe sledeća pravila:

- Poruka kriptovana javnim ključem može se dekriptovati samo odgovarajućim tajnim ključem.
- Poruka kriptovana tajnim ključem može se dekriptovati samo odgovarajućim javnim ključem.

- Tajni ključ je poznat samo jednom učesniku u komunikaciji i nikad se ne razmenjuje preko mreže.
- Javni ključ je dostupan svima, odnosno može biti poznat svima.

Šematski prikaz šifrovanja i dešifrovanja asimetričnim algoritmima dat je slici 15. Koristi se par matematički povezanih ključeva k1 i k2, pri čemu se jedan koristi za kriptovanje (za kriptovanje E se koristi ključ k1), a drugi za dekriptovanje (za dekriptovanje D se koristi ključ k2). Bitno je da se jedan od ova dva ključa ne može izvesti iz drugog u "razumnom" vremenu, odnosno da napadač ne može u razumnom vremenu na osnovu javno dostupnih informacija (algoritam za generisanje ključeva i javni ključ) doći do tajnog ključa.

Asimetrični kriptografski algoritmi se baziraju na tzv. jednostranim funkcijama sa vratancima (trapdoor one-way functions), odnosno funkcijama koje se jednostavno računaju u jednom smeru, a ne mogu se lako izračunati u suprotnom smeru. Namena trapdoor one-way funkcija jeste generisanje para javnog i tajnog ključa tako da napadač ne bi mogao da iskoristi javno dostupne informacije da bi došao do tajnog ključa. Primer trapdoor one-way funkcije je množenje dva velika prosta broja, N = p * q – množenje brojeva p i q je jednostavna računska operacija, dok je faktorizacija broja N znatno kompleksnija. Javni ključ se ugrađuje u okviru sertifikata tako da asimetrični kriptografski sistemi zahtevaju PKI (public key infrastructure), odnosno infrastrukturu za rad sa sertifikatima (kreiranje, rukovanje, distribuiranje, validacija, povlačenje sertifikata).



Slika 15 - Šematski prikaz asumetričnih algoritama

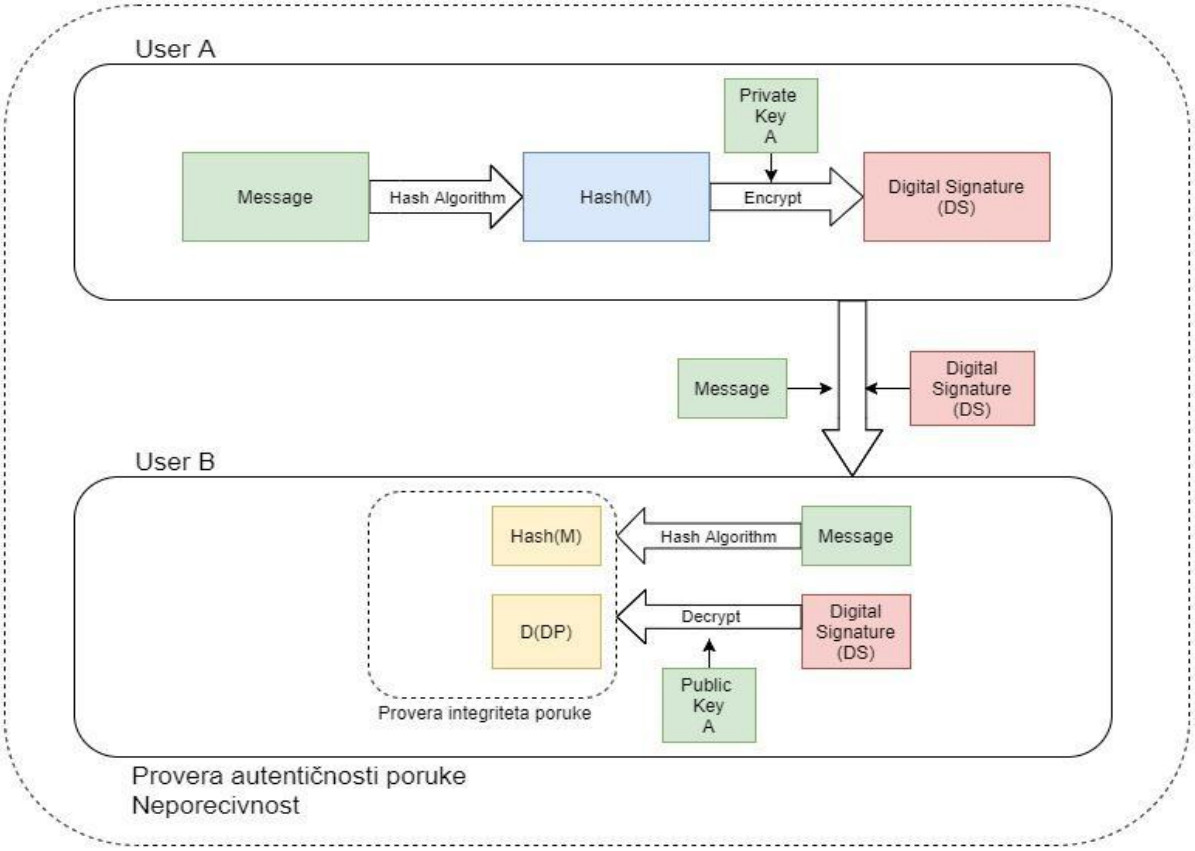
RSA Algoritam

RSA (Rivest Shamir Adelman) algoritam je najpoznatiji i najviše primenjivan asimetrični algoritam. Prvenstveno je namenjen šifrovanju podataka, ali se koristi i u sistemima elektronskog potpisa. Sigurnost RSA algoritma se zasniva na računskoj složenosti postupka faktorisanja velikih brojeva. Naime, javni i tajni ključ se generišu postupkom množenja velikih prostih brojeva. Za razliku od generisanja ključeva, proces faktorizacije velikog broja kako bi se dobili javni i tajni ključ je znatno komplikovaniji postupak. Odnosno, sigurnost RSA algoritma se zasniva na činjenici da je dobijanje otvorenog teksta na osnovu javnog ključa i šifrovane poruke jednako složeno kao i faktorisanje proizvoda dva velika prosta broja.

Digitalno potpisivanje

Kriptografija javnog ključa zajedno sa hash algoritmima ima važnu primenu u oblasti digitalnog potpisivanja (digital signing) koje predstavlja važan aspekat sigurne razmene poruka. Digitalni potpis predstavlja elektronski potpis pošiljaoca poruke ili podatka kojim se obezbeđuje integritet i autentičnost poruke, kao i neporecivost od strane pošiljaoca da je upravo on poslao poruku. Naime, digitalni potpis predstavlja hash vrednost izračunatu nad sadržajem poruke koja se zatim kriptuje privatnim ključem pošiljaoca. Primalac poruke, dekriptovanjem digitalnog potpisa javnim ključem pošiljaoca potvrđuje autentičnost poruke, jer samo pošiljalac ima odgovarajudi privatni ključ. Ovim je istovremeno omogućeno da pošiljalac neće moći da porekne da je on poslao poruku (neporecivost).

Rezultat dekriptovanja digitalnog potpisa je hash vrednost poslate poruke, a primalac poruke primenom istog algoritma može da izračuna hash pristigle poruke i uporedi tu vrednost sa dekriptovanom vrednošću, i na taj način proveri integritet poruke. Proces kreiranja digitalng potpisa na strani pošiljaoca poruke, kao i proces verifikacije od strane primaoca je prikazan na slici 15.



Slika 16 - Digitalni potpis

Uspešnom proverom digitalnog potpisa garantuje se:

- **Integritet** - upoređivanjem izračunatog i dekriptovanog hash-a poruke utvrđuje se da li je poruka modifikovana. Mada se kriptovanjem skriva sadržaj poruke, moguće je slučajno izmeniti originalnu poruku bez razumevanja šta je stvarno promenjeno. Ukoliko je poruka digitalno potpisana, svaka promena poruke će promeniti potpis. Ne postoji efikasan način da se poruka i njen potpis izmene i da se stvori nova poruka sa validnim potpisom.
- **Autentifikacija** - pouzdanost identiteta pošiljaoca, što je posledica činjenice da je hash poruke koji je šifrovan tajnim ključem, moguće uspešno dešifrovati samo primenom odgovarajućeg javnog ključa.
- **Neporecivost** - pošiljalac ne može da porekne slanje poruke pošto je potpisana njegovim tajnim ključem.

Važno je pomenuti da digitalni potpisi ne pružaju zaštitu tajnosti podataka od neovlašćenog čitanja odnosno zaštitu poverljivosti, jer se svi podaci šalju u svom originalnom (nepromenjenom) obliku.

Zadatak 9.

Na primeru WCF klijent-servis aplikacije demonstrirati rad sa digitalnim potpisima. Prilikom slanja poruke u okviru metode *SendMessage()* potrebno je verifikovati integritet i autentičnost poslate poruke na servisnoj strani. Poruka koja se verifikuje je proizvoljna.

Prvi korak je generisanje sertifikata čija namena je digitalno potpisivanje. Subject name sertifikata treba da bude u formatu "*<clientName>_sign*", odnosno servisna komponenta će prilikom verifikacije digitalnog potpisa tražiti sertifikat sa *Subject name* u tom format.

Napomena: vrednost *<clientName>* ne sme biti fiksirana u kodu.

Drugi korak je izmena metode klijenta i servisa tako da klijent kreira digitalni potpis pre slanja svake poruke, a servis validira integritet i autentičnost poruke validacijom digitalnog potpisa.

Nakon toga, generisati sertifikat čija je vrednost Subject Name "*wrong_sign*" koji će klijent koristiti za digitalno potpisivanje. Verifikovati da će (sa neizmenjenom logikom na servisu) poruka biti nevalidna na servisnoj strani.

Vežba 8 - Auditing

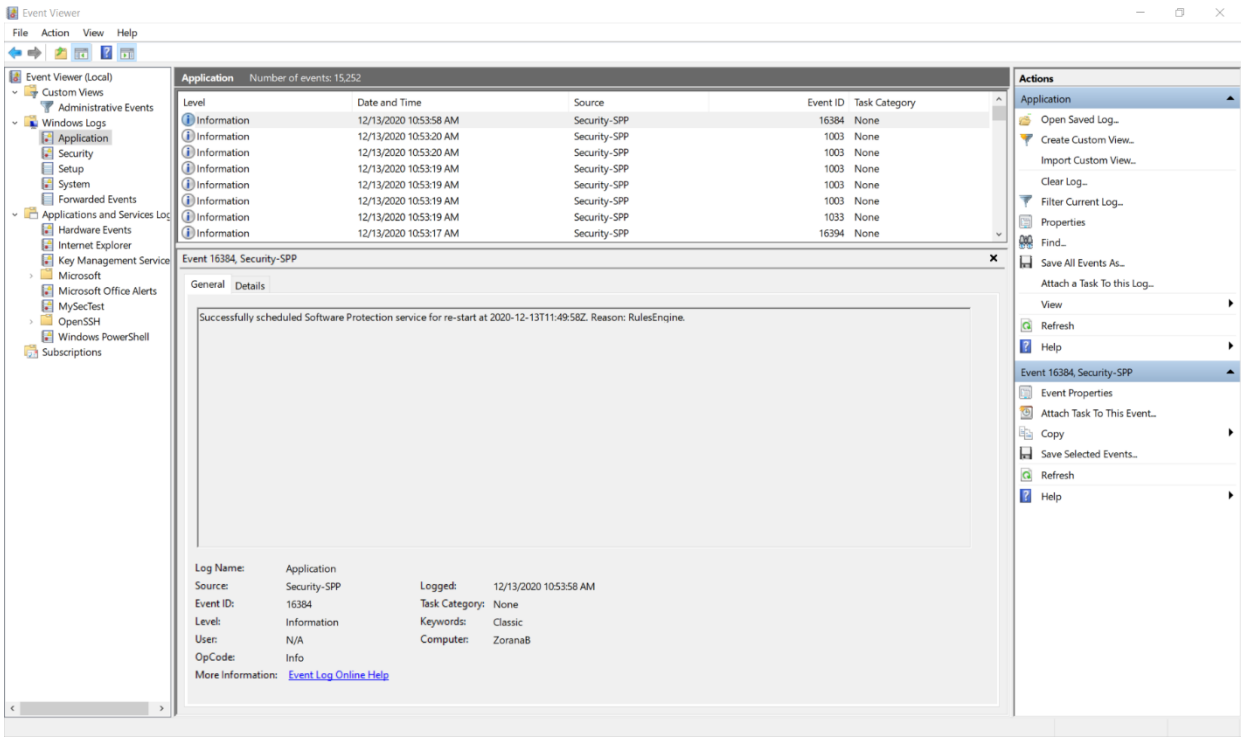
Cilj ove vežbe je upoznavanje sa mehanizmom logovanja i audita bezbednosnih događaja.

“AAA” (*eng. AAAs of security*) predstavlja akronim za tri osnovna bezbednosna mehanizma koji zajedno funkcionišu kako bi se obezbedio kontrolisan pristup informacionom sistemu i podacima (eng. *AAA = Authentication, Authorization, Accounting = Auditing*). Auditing se odnosi na proces praćenja, snimanja/logovanja, analize i izveštavanja o bezbednosnim događajima u sistemu.

Bezbednosni događaji mogu biti kako uspešno izvršene akcije u sistemu, tako i neuspešni pokušaji pristupa resursima. Audit log predstavlja zapis bezbednosno relevantnih događaja u sistemu. S obzirom da audit log predstavlja vremenski obeležene zapise o aktivnostima u sistemu, učesnici ne mogu naknadno poricati izvršene akcije čime se obezbeđuje neporecivost. Integritet, odnosno tačnost podataka koje sadrže audit logovi se obezbeđuje primenom mehanizama kontrole pristupa logovima, digitalnim potpisima, itd.

Analizom prikupljenih informacija moguće je detektovati kako uspešne tako i neuspešne pokušaje kako redovnih tako i malicioznih aktivnosti u sistemu, odnosno naknadno utvrditi uzroke grešaka ili otkaza u sistemu.

Ugrađeni .NET mehanizam za logovanje i audit koristi log datoteke Windows operativnog sistema za zapis različitih tipova događaja: sistemskih (generisanih od strane operativnog sistema), aplikativnih i bezbednosnih – tzv. *Microsoft Windows Event Log*. Dodatno .NET pruža mogućnost jednostavnog kreiranja novih log datoteka. *Windows Event Viewer* je alat za prikaz i upravljanje event logovima Windows operativnog sistema.



Slika 17 - Event viewer

Na slici 17 je prikazan Event Viewer, gde se mogu se uočiti dve kategorije log fajlova:

- **Windows Logs** – gde se zapisuju događaji na nivou operativnog sistema;
- **Applications and Services logs** – logovi generisane od strane specifičnih aplikacija unutar sistema.

Za svaki događaj definisan je jedinstveni identifikator (*EventID*), datum kad se događaj desio (*Date and Time*), izvor (aplikacija) koji je generisao događaj (*Source*) i kategorija događaja (*Level*).

WCF pruža mogućnost konfigurisanja **`ServiceHost`** objekata da podrže zapisivanje bezbednosnih događaja (uspešni i/ili neuspešni pokušaji autentifikacije i autorizacije) definisanjem ponašanja (behaviors) WCF servisa:

```
ServiceHost host = new ServiceHost();
ServiceSecurityAuditBehavior newAuditBehavior = new ServiceSecurityAuditBehavior();
...
host.Description.Behaviors.Remove<ServiceSecurityAuditBehavior>();
host.Description.Behaviors.Add(new AuditBehavior);
```

Objektom klase **`ServiceSecurityAuditBehavior`** (definiše ponašanje WCF servisa) moguće je specificirati:

- Da li će se logovati uspešni i/ili neuspešni pokušaju autentifikacije (na message nivou);
- Da li će se logovati uspešni i/ili neuspešni pokušaju autorizacije;
- Tip log datoteke gde će se događaji zapisivati;
- Šta se dešava u slučaju neuspešnog audit poziva, npr. ukoliko aplikacija pokušava da zapiše događaj u log za koji nema ovlašćenja - default vrednost je ***true*** i znači da će aplikacija nastaviti normalno sa radom, dok vrednost ***false*** znači da se neće nastaviti sa obradom.

Zadatak 10.

Potrebno je konfigurisati WCF servis tako da omogući zapisivanje bezbednosnih događaja u Windows Aplikativnu log datoteku.

Potrebno je obezbediti upis sledećih bezbednosnih događaja u Windows Event Log. Skup bezbednosnih događaja, kao i format poruke koje je potrebno podržati u ovom zadatku je prikazan u tabeli 1, a svi događaji se zapisuju u Windows Event log datoteku posebno kreiranu za potrebe rada ovog servisa.

Tip događaja	Format poruke
AuthenticationSuccess	User {0} is successfully authenticated. {0} – ime korisnika koji je uspešno autentifikovan
AuthorizationSuccess	User {0} successfully accessed to {1}. {0} – ime korisnika koji je uspešno autorizovan {1} – ime metode kojoj je korisnik uspešno pristupio
AuthorizationFailure	User {0} failed to access {1}. Reason: {2}. {0} – ime korisnika koji nije uspešno autorizovan {1} – ime metode kojoj je korisnik pokušao da pristupi {2} – razlog zbog čega pristup nije dozvoljen (očekivana privilegija)

Tabela 1 – Tipovi događaja u zadtku

Napomena: Za manipulaciju Windows Event Logom moguće je koristiti ugrađenu .NET klasu **EventLog** iz **System.Diagnostics** namespace. Prilikom inicijalizacije EventLog objekta, neophodno je registrovati aplikaciju – izvor događaja (na nivou Windows operativnog sistema) kao što je navedeno ispod:

```
If (!EventLog.Sourceexists(SourceName))
{
        EventLog.CreateEventSource(SourceName, LogName);
}
newLog = new EventLog(LogName, Environment.MachineName, SourceName);
```