Fakulteten för ekonomi, kommunikation och IT

Säker Hälsa

Datum: 2007-10-20

Namn: Henrik Bäck
Mathias Andersson

Kurs: DVGC03

Innehållsförteckning

Inledning **3**
Tjänstens utformning **3**
Integritet 3
Användartyper 3
Server 3
Klient 4
Krypteringsalgoritmer 4
Nyckeldistribution 5
Signering 5
Kommandon 5
Testförfaranden 6
Prototyp **8**
Prototypens omfattning 8
Funktionalitet 8
Säkerhet 8
Nyckeldistribution 9
Kommunikation 9
Databas 9
Klient 9
Server 10
Bilagor **12**
Databas - loginData 12
Databas - patientJournals 12
Databas - userPatientRelation 13
Kod - Klient 15
Kod - Server 29
Datafiler 46

Inledning

SäkerHälsa är en tjänst för att lagra information om patienter och deras vårdtillfällen. Tjänsten måste vara säker på det sätt att information inte skall kunna vara åtkomlig av någon annan än patienten själv och dess vårdare. Datan som strömmar mellan klient och server måste också vara säker på det sättet att det inte skall gå att lyssna av trafiken och få reda på information om patienten.

Tjänstens utformning

Tjänsten utformas som en Klient/Serverlösning. Tjänsten består av en server som har en databas med information om patienter och användare samt en klient som ansluter till servern för att hämta information.

Informationen mellan Klienten och Servern går via en krypterad anslutning lämpligen med hjälp av SSL.

Integritet

Tjänsten är utformad på det sätt att det, via databasen, inte går att ta reda på vilka patienter som är associerade med vilka vändare. Det går inte att koppla samman en patientjournal med en användare eftersom information om detta ligger krypterad. För att kunna erhålla en lista över en användarens patienter behövs användarens lösenord för att kunna avkryptera denna information.

Användartyper

Systemet bygger på att enbart en journal kan skrivas för patienten och att denna journal innehåller all information som kan tänkas behövas. Detta gör att systemet inte definierar några användartyper utan definierar istället användare som har tillgång till utvalda journaler med läs eller läs och skrivrättigheter.

System att utvidgas genom att exempelvis lägga till fält för provsvar, vårdtillfällen med mera. Dessa fält kan sedan individuellt rättighetsbestämmas för varje användare. Det gör systemet mer flexibelt än att definiera ett antal användarroller.

Server

Servern har tillgång till någon form av databas. Detta kan antingen exempelvis vara en fil på disk eller en databasserver.

Databas

Det finns tre separata tabeller i databasen. En för lagring av patientuppgifter, en för lagring av användare samt en tabell för hantering av relationen mellan patientuppgifter och användare i systemet.

Varje patientjournal lagras krypterad i databasen, med en för varje journal unik krypteringsnyckel. För varje användare som har tillgång till en patients journal lagras information om denna användare samt patientjournalens krypteringsnyckel tillsammans med flagga för läs och skrivrättighet i journalen, i en specifik tabell. För att kunna finna journalen lagras här också en unik identifierare. Denna information lagras krypterad med användarens lösenord. Detta gör det möjligt att med hjälp av ett användarnamn och det lösenord låsa upp de krypteringsnycklar som gäller för de patienter som användaren har tillgång till. Detta leder till att användaren får tillgång till patientjournalen i klartext genom att låsa upp den låses upp med denna krypteringsnyckel. Här måste ser-

vern hålla reda på ifall användaren har rättighet till att ändra uppgifter eller bara visa dem eftersom krypteringsnyckeln enbart används till att avkoda journalen i databasen.

Det finns även en tabell som lagrar information om alla användare i systemet. Tabellen lagrar lösenordet på varje användare krypterat med ett envägchiffer. Tabellen används enbar till att kontrollera ifall en användare har tillgång till systemet eller inte.

Kommunikationen med databasen sker okrypterad eftersom den information som lagras i databasen redan är krypterad så när som på användarnamn. Självklart kan även denna trafik, med fördel krypteras, men det anses för närvarande vara onödigt.

Kommunikation
En klient upprättar en anslutning mot server via en krypterad länk. Över denna länk skickar sedan klienten över kommandon som denne vill utföra. Mer om de tillgängliga kommandon finns längre ned i detta dokument.

Vid varje förfrågan skickas användarens lösenord över den krypterade tunneln. Lösenordet används sedan för att först hämta krypteringsnyckeln för de patientjournaler som begärts. Den krypteringsnyckel som sedan erhålls används för att hämta patientjournalen i klartext.

Alla meddelanden som skickas från servern signeras så att klienten kan kontrollera att den får meddelande från rätt servern. Nyckeln för att kontrollera signaturen är inbyggt i klienten från start.

Inloggning
Servern bergränsar antalet inloggningar från varje klient och för varje användare för att undvika att ett brute-force-attak utförs mot servern. En klient som förbjudits mot servern måste manuellt låsas upp igen för att kunna ansluta.

Klient
Klienten skickar förfrågningar till servern. Varje förfrågning innehåller namnet på frågan, frågedata, användarnamnet samt lösenordet för den aktuella användaren. Informationen skickas som objekt till servern, detta gör att servern direkt kan arbeta med datan som ankommer.

Klienten har ett grafiskt användargränssnitt där användaren först får mata in sitt användarnamn och lösenord. I klientmjukvaran visas, efter inloggning, en lista över de journaler som användaren har tillgång till.

Krypteringsalgoritmer
För kommunikationen mellan server och klient används asymmetrisk kryptering med RSA. De publika nycklar som används delas ut vid förhandlingen mellan applikationerna och är nya för varje gång en klient ansluter till servern. Inga nycklar återanvänds.

Lösenord lagras med MD5 i databasen. Dessa lösenord bör också "saltas" för att försvåra tekniken att upptäcka mönster i informationen och därigenom härleda ett lösenord.

Övrig data i databasen sparas med AES.

Nyckeldistribution

När klienten först ansluter till servern skickar den över sin publika krypteringsnyckel. Servern sparar undan denna nyckel eftersom all annan kommunikation kommer att ske krypterad med denna sedan. Efter detta skickar servern över sin publika krypteringsnyckel till klienten.

Både servern och klienten genererar nya nycklar varje gång en anslutning upprättas. Detta är viktigt för att ingen skall kunna få tag på de korrekta nycklarna och lyssna på trafiken mellan server och klient. Det gör också att inga nycklar behöver delas ut i förväg.

Efter att nycklarna har utbytts kommer all kommunikation mellan server och klient att vara krypterad. Detta gör att enbart mottagaren kan läsa meddelandenas innehåll vilket är viktigt för integriteten och säkerheten i tjänsten.

Förfarandet gör dock fortfarande att alla kan kommuncera med servern på detta sätt. Det som skyddar data från att lämnas ut är en authentiering med ett användarnamn och lösenord.

Signering

Alla meddelanden som skickas från servern innehåller en signatur. Denna signatur används av klienten för att verifiera att kommunikation sker med korrekt server.

Kommandon

Klienten skickar data i form av XML till servern och samma gäller som svar från servern till klienten. Servern känner till flera olika metoder som kan skickas till den, bland annat

```
PUBKEY-SESSION
ACKDATA
LOGIN
REQUEST_JOURNALS
REQUEST_JOURNAL_DATA
SAVE_JOURNAL_DATA
```

Klienten känner till flera olika metoder som kan skickas till den, bland annat

```
PUBKEY-SESSION
ACKDATA
LOGIN-OK
LOGIN-FAILD
RESPONSE_JOURNALS
RESPONSE_JOURNAL_DATA
JOURNAL_DATA_SAVED
JOURNAL_DATA_NOT_SAVED
```

Innan data skickas måste `STARTDATA` skickas, därefter kan ett antal sammanhörande datapaket skickas. Efter att sändningen är klar skickas `STOPDATA` . När detta kommando påträffas börjas behandlingen av datan som skickats.

Varje förfrågan och svar bör innehålla ett fält med ett slumpartat tal, detta för att likadana meddelanden i samma session inte ska ha samma utseende efter kryptering. Ett exempel på ett meddelande som alltid har samma utseende är ACKDATA.

Testförfaranden

Hämta en patients journal, inloggning korrekt

En användare startar klientmjukvaran och startskärmen visas. Användaren förser sedan mjukvaran med sitt användarnamn och lösenord och klickar på knappen "Log in".

En anslutnign upprättas till servern och klienten skickar över sin publika sessionsnyckel Efter att detta gjorts skickar klienten följande meddelande, krypterat, till servern

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
     <string>LOGIN</string>
     <string>Användarnamn</string>
     <string>Lösenord</string>
</array>
</plist>
```

Servern kontrollerar användarnamn och lösenord mot databasen och returnerat ett meddelande baserat på hur inloggning gick. I detta exempel lyckas inloggningen och detta meddelande skickas från servern till klienten.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<string>LOGIN-OK</string>
</plist>
```

Klienten begär sedan att få en lista över de patienter/journaler associerade med den samma. Klienten skickar då följande meddelande. Användarnamnet och lösenordet skickas över än en gång för att kontrollera att servern fortfarande talar med samma användare.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
     <string>REQUEST_JOURNALS</string>
     <string>Användarnamn</string>
     <string>Lösenord</string>
</array>
</plist>
```

Servern kommer att behandla detta meddelande och skicka tillbaka en lista, i form av ett objekt, till klienten. Detta objekt kan sedan tillämpas av klienten för att visa en lista över patientet/journaler.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
     <string>RESPONSE_JOURNALS</string>
     <array>
          <string>PatientName1</string>
          <string>PatientName1</string>
          ...
     </array>
</array>
</plist>
```

Efter detta visas programmets huvudskärm och användaren kan välja vilken patient/ journal denna vill visa. Användaren kan klicka på en patient/journal för att hämta data om den, endast patienter/journaler som är kopplade till användaren visas.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
     <string>REQUEST_JOURNAL_DATA</string>
     <string>PatientID(ej patientnamn)</string>
     <string>Användarnamn</string>
     <string>Lösenord</string>
</array>
</plist>
```

Servern kommer att behandla detta meddelande och skicka tillbaka en lista, i form av ett objekt, till klienten. Detta objekt kan sedan tillämpas av klienten för att visa patientens journal.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
     <string>RESPONSE_JOURNAL_DATA</string>
     <string>Patientens namn</string>
     <string>Journaltext</string>
     <string>Behörighet att ändra</string>
</array>
</plist>
```

Hämta en patients journal och uppdatera texten, inloggning felaktig

En användare startar klientmjukvaran och startskärmen visas. Användaren förser sedan mjukvaran med sitt användarnamn och lösenord och klickar på knappen "Log in".

En anslutnign upprättas till servern och klienten skickar över sin publika sessionsnyckel till servern. Server svarar med att skicka över sin publika sessionsnyckel till klienten. Efter att detta gjorts skickar klienten följande meddelande, krypterat, till servern

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```
<plist version="1.0">
<array>
     <string>LOGIN</string>
     <string>Användarnamn</string>
     <string>Lösenord</string>
</array>
</plist>
```

Servern kontrollerar användarnamn och lösenord mot databasen och returnerat ett meddelande baserat på hur inloggning gick. I detta exempel lyckas inte inloggningen och detta meddelande skickas från servern till klienten.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<string>LOGIN-FAILD</string>
</plist>
```

Klienten visar ett meddelande om att inloggningen misslyckades. Därefter kopplar klienten ned anslutningen mot servern. Vid nästa inloggning kommer nya sessionsnycklar att genereras innan en ny inloggning provas.

Prototyp

En prototyp för tjänsten har implementerats i Objective-C med hjälp av ramverket Cocoa. Cocoa ger möjlighet, via Apple Interface Builder, att snabbt skapa ett användargränssnitt för programmet. Samt erbjuder en mängd smidiga byggstenar för att underlätta programmeringen.

Strukturen på programkoden i prototypen är inte helt och hållet den bästa. Detta berodde på bristande kunskap när uppstarten av projektet skedde. Efter prototypens färdigställande finns betydligt mer kunskap om hur strukturen borde ha lagts upp. Dock fanns dessvärre inte tid till att göra om strukturen.

Programkoden för prototypen finns som bilaga, men saknar koden för de grafiska elementen. Den hela programkoden kan hämtas på http://it4.baeck.se/DVGC03/SakerHalsa. Programkoden måste kompileras under XCode 2.5 eller senare samt MacOS 10.4 eller senare.

Prototypens omfattning

Prototypen visar på hur kommunikation över ett oskyddat nätverk kan skyddas samt hur data lagras säkert. Prototypen visar också hur integriteten skyddas och att det inte finns några synliga kopplingar mellan användare och patienter.

Funktionalitet

Prototypen tar upp de grundläggande delarna i systemet. En användare har möjlighet att logga in mot servern, lista journaler och uppdatera utvalda journaler. Användaren kan dessutom logga ut.

Säkerhet

Prototypen upprättar ingen riktigt SSL-koppling mellan klient och server. Istället upprättas en session mellan klient och server där de utbyter publika nycklar. Dessa nycklar används sedan för att kryptera datafragmet innan de skickas på den osäkra kommunik-

tionskanalen. Endast mottagaren kan avkryptera meddelanden och visa innehållet. Krypteringen som används är RSA och nyckelns storlek är 1024 bitar.

Servern måste ha minst ett sätt att långsiktigt lagra information om patienter, journaler och användare. Informationen som lagras krypteras med AES, 128 bitar.

Nyckeln till denna information om en användares patientlista är en MD5-hash av användarens lösenord. I denna information finns ytterligare nycklar för att komma åt journaldata för de specifika patienterna.

Nyckeldistribution

Klienten öppnar en anslutning mot servern och skickar med en gång över sin publika nyckel. Servern tar emot denna nyckel och lagrar den temporärt under den aktuella sessionen. Servern svarar med att skicka sin publika nyckel till klientent som gör samma sak.

Efter att nycklarna har utbytts sker all kommunikation via krypterad data.

Kommunikation

Både klient och server skapar ett objekt med data som skall skickas. Detta objekt serialiseras till en mängd data som sedan portioneras upp i 512 bitars stora block. Dessa block krypteras individuellt med RSA och skickas sedan ut på länken.

På mottagarsidan tas dessa block emot, avkrypteras, samlas i ett dataobjekt och därefter skickas en bekräftelse på att datan mottagits till sändaren. Detta gör att sändarsidan skickar nästa datablock om sådant finns. När alla datablock har mottagits och avkrypterats så deserialiseras de. Mottagaren finner de data som skall behandlas i detta objekt.

Databas

Servern lagrar data i from av tre stycken serialiserade objekt dessa är sparade XML-filer. XML-filerna är i sig inte krypterade men däremot är innehållet i dess entiteter krypterade. Anledningen till att XML-filerna i sig inte krypteras är att det krävs extra steg för detta. I denna prototyp ansågs det vara fullt gott att lagar dem som de var.

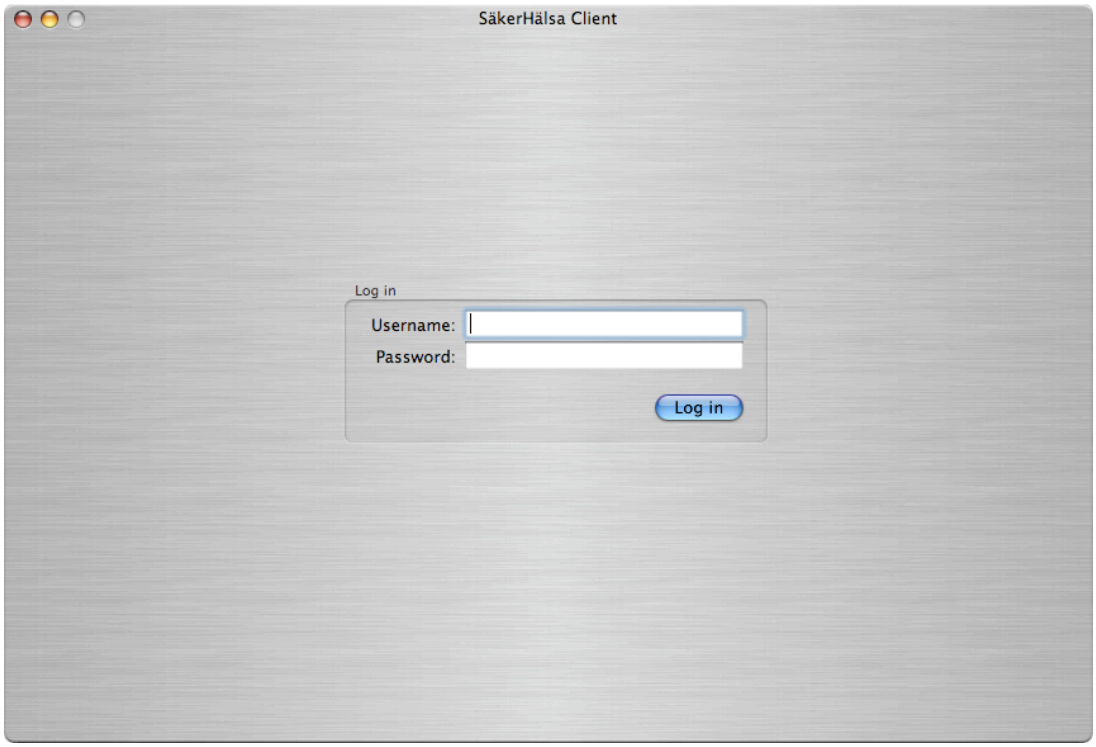
Det finns ingen sätt att, genom att titta på XML-filerna, härleda en patient till en användare eller vice versa. Däremot är det möjligt att se hur många patienter en användare har samt vilka användare som finns i systemet.

Denna information kan göras otillgänglig genom att kryptera de serialiserade objekten innan de lagras på disk.

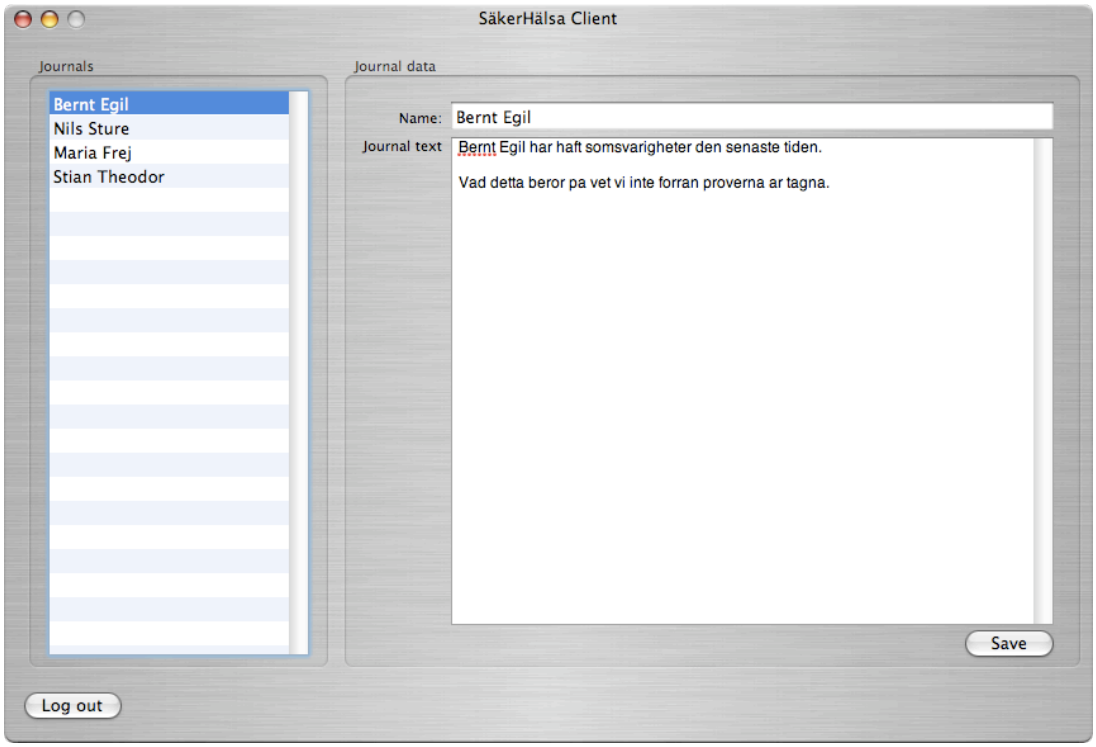
Klient

Klienten är implementerad för att vara simpel för användaren men ingen extra tid på att lagts på att anpassa användarvänligheten. Detta är tros allt en prototyp.

Prototypklitenen tillåter inloggning, uppvisning av patienter/journaler samt att läsa och uppdatera journaltext.



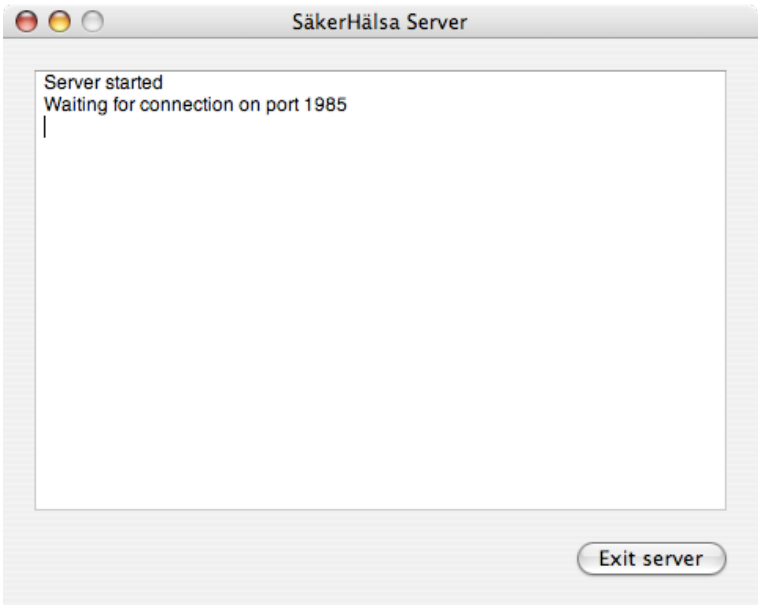
Inloggningsfönster



Vy över patientjournaler samt journaldata.

Server

Servern är även den grafiskt uppbyggt. En grafisk server är inte nödvändigt men kan vara trevligt. Prototypserver tillåter inloggning, uppvisning av patienter/journaler samt att läsa och uppdatera journaltext.



Serverfönster

Bilagor

Databas - loginData

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>BeckOS</key>
     <string>81dc9bdb52d04dc20036dbd8313ed055</string>
     <key>Mathias</key>
     <string>81dc9bdb52d04dc20036dbd8313ed055</string>
     <key>Thijs</key>
     <string>81dc9bdb52d04dc20036dbd8313ed055</string>
     <key>Maria Frej</key>
     <string>81dc9bdb52d04dc20036dbd8313ed055</string>
</dict>
</plist>
```

Databas - patientJournals

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>1</key>
     <data>

ZrCcxHKWEymukcVaIJd7SlP8iUq5PQNYAJlgKmlv3pbheq5Z2dpkp+5uIv8UAahIXW2
1wZ+SF60A7TOENd7S1DsZKCmmpW7u7m9AdmM5Mvy9ev8HZi+WIff0y49y2WwZx8rAtl
Dw0xIVX9CkAs2pbfcv7QhgybMzuoKIYluLPhE8p+jnxj46Id89C6qP5Gsg3Vh3wXoJ0
4NG7vhH18+13bjGpuyo1kT33LdsKLMQa3iGeCSOLXVXFLz0OH/VDCM2q4U6auFFyuM3
/Q9SO3qcIt0XUNo7Jl4kOI8cJzJ5pY5e73Aj7ftTVNRHmz9sf5Pxa/hsHqY0AbmAsGw
dMJLJg2qgozZ5eH7IpolRFAIf2biue4wg+B7U2LOjO6LHMkK0RcrdywR5WnbU2NL6c
FZSssbPirsjwdHc8el7ZheNrhF9dcLheMnp+oeHMqaRP6Bk6f3rXiA5sbXY6pQ==
     </data>
     <key>2</key>
     <data>
/uyjpHLYwsnnEy7qztLMMyl9S/5gXF+6EKVemTi6XScwquSf28wUsdemtg8mCAylvD3
H6fNA9OBLcFSG2xRMffb5UecxN4crbJ7OYQ3ze9VFK1nEhfl41IGiTgUf7wcMvU8Ekp
7rC59ULcp+41AnRboIp529bI+hSwVY3QulxsfiD8Rk/Dl/Qiyy2M2XdAwRFJBx+3CVo
I9qg91J6euNyOIdVopg7B6TpkUJxjnuTRChNt3PcdXFfQvgRtiEgT05J8LYUerPM/Qi
DDkyJL5SPU8Kvt09gH0lsbe7ocLVgnUR6hWgsPpEGui2HM1fdzcsjufrG9W5pnCk2Ha
MsfP/lpwpDR7i530rVs1XQUL2rwvwa+uXvxHiVOFwW+YsmxW0qZ0w+L1Qfp+us1Kpjo
5cAEqKrYo64WKxLmoiViMG7MkG98E+BzmqH8rCLETuDyQUdv2BOlGMfDB/gI6nAh6BY
tvENBUe0BOAPnpw=
     </data>
     <key>3</key>
     <data>

TTlvtzdb68RB+63qceymhTIwKbmQa83zEGNc57NRtILY5TbIIA39KzKL2DqJA6g7+BH
CyXQhCDajoISV7y/jUalt4AgfzEuYf0B2oIbeV0bx4dWc0rO0YOp4FUpkEs+Y7uHI4I
h0ja3Mt5VjL9Zs/mjk1L/7dfIrEyNqi2o4Gn+i5zrh9ohwKMhL2OUCrYfDCb6tHrnWr
zGWpVHckT8HrmXkTRwM4p6byppBJpqLRQmfKa5hnvmAuVFUA2jTv+jJFRCQQVk4rYI9
+5OlxB6mBnoZsYVBUHwjHKRilz42kvA/svdDTmHTyUhG5V/lZvIi7kO1JZS2s4KBhcg
ypUEcpuVxetvjYdjmbaZjjmfMtKioQxjamm2ebKatKowWi3R4yCEGJhaL8pHBNr3kAQ
==
     </data>
     <key>4</key>
```

```
     <data>
FD2hCqWdvVMbiu89YgQuObOJca3Zf8DA35anFQLlk4mHgG5wjKR+iaQvV9wPWDDTkIQ
J/WnWEhlV7rnB2k/Bmsltrr2el6qLMnofEbxXkyxM02Caz+en1tzEb6kc1l/OWAKDmp
SuzIaQRJSm9O0SDjruHrsa04xryAnYN2VPvO3Sz8SFvF/jycbZehnoE1ppgSDxVAjYf
Vw037Ncb9G/5GkTyDWNrfA25KIl4rOwOlDA74Y7sKmtnY9IN4eiQRp1kSMDQ/DNSeNg
07GT0bnrbj0e9sAzypwFyAceSsySLSTh0eiGhtszTLZXMkpZQSTjvVX/5K8stgSqHfy
Zvt/upoP34Vq3lqYjZc6de6SmV+8rpARoR/lDzcXhnXwcFPJaqzlkVVCiJtQiNcaYyr
l/R4Ryqt/lqurkorwvqqs=
     </data>
</dict>
</plist>
```

Databas - userPatientRelation

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>BeckOS</key>
     <array>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnu6K3IT
2nAirc19oKM7pGaAGNr2MIzlW5ZEhv+qkZLkcuZa9CarD8NaSRY7Km+eoXZP
9qhZLjN85I9cEio49WF1rR1kFCgl4U+k59KK76YuESRQnSZnDSTJ99YZWizd
G+wi+uPoxdxyJUrtODn3UTT+F6XG0tZnP8Wn26yvX6nsDQ==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnuBGV6k
DDhJOdK2EJj3Jgtpp0+yziS5xYp23lcs0fL4L2C80TgxfoA3UJKvkoO5cP1R
OhDhgQLjVdT+KorNvhhsdjYXyu0A0u+Yhei5K2yRwXwtOqZc/7Yi5OeBRpPu
ywNBOpxYntEUhckCshfifiB+9vAAxHKTED3mzbvjZVcYw==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksntfJNef
bSdFB5jIkclLV6flfXbGfdzElfBTR73iImg2AuJ14/87s3NPnvaFo9KEK4ZO
H3K2A1kS1oNtpFLNJ+u1JS3c7E+3HB03ivpFLOUI8Ke2awYC+hBhB9nAzson
qWMepiIlh71aLMwdGIsrKzERq6iVFB9wrv/FEfFW03qeuA==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnsllc4k
FpGWdAswVG0lt2ckpStCvsEJb+LuO/lS0ZVSjfAKMU6WECY5KwVSDBGvMJpO
YAjL0diuCfFajHMIwYPTxZvI8rMl/eonyyx8i+xbEnCOSR52R/jzpkxf6WZN
hwEtUbLM3m7+2FF8mTX0tqsfs9qzeMDDN/N5t9TMPgxojw==
          </data>
     </array>
```

```
     <key>Mathias</key>
     <array>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnuBGV6k
DDhJOdK2EJj3Jgtpp0+yziS5xYp23lcs0fL4L2C80TgxfoA3UJKvkoO5cP1R
OhDhgQLjVdT+KorNvhhsdjYXyu0A0u+Yhei5K2yRwRMRYs1BRM+qJ/YgKD4p
5aWdoVpXpdXdm2SeI4jUjETzj4ps40cjZiqYMuCUa2Q0mw==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnu6K3IT
2nAirc19oKM7pGaAGNr2MIzlW5ZEhv+qkZLkcuZa9CarD8NaSRY7Km+eoXZP
9qhZLjN85I9cEio49WF1rR1kFCgl4U+k59KK76YuESRQnSZnDSTJ99YZWizd
G+wi+uPoxdxyJUrtODn3UTT+F6XG0tZnP8Wn26yvX6nsDQ==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnsllc4k
FpGWdAswVG0lt2ckpStCvsEJb+LuO/lS0ZVSjfAKMU6WECY5KwVSDBGvMJpO
YAjL0diuCfFajHMIwYPTxZvI8rMl/eonyyx8i+xbEnCOSR52R/jzpkxf6WZN
hwEtUbLM3m7+2FF8mTX0tqsfs9qzeMDDN/N5t9TMPgxojw==
          </data>
     </array>
     <key>Thijs</key>
     <array>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnuBGV6k
DDhJOdK2EJj3Jgtpp0+yziS5xYp23lcs0fL4L2C80TgxfoA3UJKvkoO5cP1R
OhDhgQLjVdT+KorNvhhsdjYXyu0A0u+Yhei5K2yRwXwtOqZc/7Yi5OeBRpPu
hywNBOpxYntEUhckCshfifiB+9vAAxHKTED3mzbvjZVcYw==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksntfJNef
bSdFB5jIkclLV6flfXbGfdzElfBTR73iImg2AuJ14/87s3NPnvaFo9KEK4ZO
H3K2A1kS1oNtpFLNJ+u1JS3c7E+3HB03ivpFLOUI8GixFg3faK/7g6zyIrch
/Jr1FUB+0fcm7khrBIGW/m4Q+D3BrwvQg8rHWrq2gW8ObQ==
          </data>
          <data>

SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksnsllc4k
```

```
FpGWdAswVG0lt2ckpStCvsEJb+LuO/lS0ZVSjfAKMU6WECY5KwVSDBGvMJpO
YAjL0diuCfFajHMIwYPTxZvI8rMl/eonyyx8i+xbEg4cxaFXWcpAO6DHJWSE
          4SS0fXaLQERKrJrOFPG61hfvEe1aRG8Nj94jtrNfJaiF4Q==
          </data>
     </array>
     <key>Maria Frej</key>
     <array>
          <data>
SDn/5akod6k20JZWrhZLyUynVwd6nzbVCDi59q96Qpj8TJF9J/8Ug/557pfI
UpWJtnZdTJMU7ZgXUW2GC1aPILcBpHo7nc1yiQNmDpm7XiYSlcImVuzTC6Fl
YbP8Nhe5rLePKqASspPki1wwYtBr5Yxl0Itz1ruPkWEYwdkEgJYgHMyl23Gu
j9rKaExUDnIOuwTWpR/lAZ1tKd7N9qGI6DS/Jg0JRfKaiSWqsCfksntfJNef
bSdFB5jIkclLV6flfXbGfdzElfBTR73iImg2AuJ14/87s3NPnvaFo9KEK4ZO
H3K2A1kS1oNtpFLNJ+u1JS3c7E+3HB03ivpFLOUI8GixFg3faK/7g6zyIrch
/Jr1FUB+0fcm7khrBIGW/m4Q+D3BrwvQg8rHWrq2gW8ObQ==
          </data>
     </array>
</dict>
</plist>
```

Kod - Klient

```
//
//  main.m
//  Client
//
#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc,  (const char **) argv);
}
```

```
//
//  windowHandler.h
//  client
//

#import <Cocoa/Cocoa.h>
#import "NetSocket.h"
#import <SSCrypto/SSCrypto.h>

@interface WindowHandler : NSObject
{
     //Handtag till objekt i användargränssnittet
    IBOutlet NSBox *journal;
    IBOutlet NSTextField *journalPatientName;
    IBOutlet NSTableView *journalTable;
    IBOutlet NSTextView *journalText;
    IBOutlet NSBox *login;
    IBOutlet NSTextField *loginName;
    IBOutlet NSTextField *loginPassword;
    IBOutlet NSTextView *logWindow;
    IBOutlet NSProgressIndicator *progressWheel;
     IBOutlet NSProgressIndicator *journalLoadProgress;
     IBOutlet NSBox *journalLoadBox;
     IBOutlet NSTextField *loginText;
     IBOutlet NSBox *journalsBox;
     IBOutlet NSWindow *currentWindow;
     IBOutlet NSButton *logOutButton;
     IBOutlet NSButton *saveButton;

    //Övriga objekt för denna klass
    NetSocket*    mSocket;
```

```
    SSCrypto *crypto;
    BOOL isCrypt;
    BOOL isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal;
    BOOL isStarted;
    float currentAlpha;
    NSString *currentPatient;
    NSData *publicKeyData ;
    NSData *privateKeyData;
    NSData *serverPublicKeyData;
    NSMutableData *reciveBuffer;
    NSMutableArray *toSend;
    NSMutableArray *journals;
    NSTimer *alphaTimer;

}

//Konstruktör
//Pre: True
//Post: Ett nytt objekt har allokerats och startvärden satts
- (id)init;

//Destruktör
//Pre:
//Post: Objektet har deallokerats
- (void)dealloc;

//Funktion för att köra objekt när gränssnittet laddats klart
//Pre: Användargränssnittet just laddats
//Post: Satt information för objekt
- (void)awakeFromNib;

//Sätter loggtext i ett loggfönster
//Pre: True
//Post: Skrivit texten på en rad i loggen
-(void) setText: (NSString*) insText;

//Hanterar mottagning av data
//Pre: Data vänter på socket inNetSocket
//Post: Rätt åtgärd har tagits för inkommande data
-(void) handleRequest:(NetSocket*)inNetSocket;

//Skicka nästa data i kön
//Pre: !isStarted
//Post: Skickat nästa dataportion till servern
-(void) sendNextInQueue;

//Skicka data på socket
//Pre: True
//Post: Data har skikcats på socket
-(void) sendToSocket: (NSData*)dataToSen;

//Delar upp och skickar data på socket, med STARTDATA före
//   och STOPDATA efter datamängden.
//Pre: True
//Post: Data har delats upp och lagds på sändkö
-(void) sendData: (NSData*)dataToSend;

//Genererar nya sessionsnycklar
//Pre: True
//Post: Nycklarna finns i publicKeyData resp privateKeyData
-(void) generateKeys;

//Utför inloggning mot server
//Pre: True
//Post: Inloggningsdata har skickats till servern (via kö)
-(void) doLogin;
```

```
//Begär en lista över journaler/patienter
//Pre: Användare inloggad korrekt
//Post: Skickat förfrågan om journaler till server (via kö)
-(void) requestJournalsList;

//Bekräfta mottagen data
//Pre: Data har mottagits
//Post: Ett ack har skickats
-(void) ackData;

//Hämtar journaldata för patient med ett visst id
//Pre: True
//Post: En förfrågan har skickats till server (via kö)
-(void) requestJournalDataForPatientWithId: (NSString*)patientID;

//Spara journaldata
//Pre: Data har ändrats och rättighet för ändring medges
//Post: Om rättighet medges har data sparats, annars ej.
-(void) saveJournalDataForPatientWithId: (NSString*)patientID;

//Spela upp ett talat meddelande
//Pre: True
//Post: Meddelandet uppspelat på standardljudenheten
-(void) say:(NSString*) toSay;

//Responder vid klick på knapp "Log in"
-(IBAction)logIn:(id)sender;

//Responder vid klick på "Save"
-(IBAction)saveJournal:(id)sender;

//Responder vid klick i listan över journaler
-(IBAction)tableViewSelected:(id)sender;

//Responder för mSocket
- (void)netsocket:(NetSocket*)inNetSocket dataAvailable:(unsigned)inAmount;

//Responder för mSocket
- (void)netsocketDisconnected:(NetSocket*)inNetSocket;

@end


//
//  windowHandler.m
//  client
//

#import "WindowHandler.h"

@implementation WindowHandler


- (id)init
{
    if( ![super init] )
        return nil;

    mSocket = nil;
    isCrypt = NO;
    isStarted = NO;
    isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal = NO;

    toSend = [[NSMutableArray alloc] init];
    reciveBuffer = [[NSMutableData alloc] init];

    //currentAlpha = 0.3;
```

```
    currentAlpha = 1.0;

    return self;
}

- (void)dealloc
{
    [mSocket release];
    mSocket = nil;
    [toSend dealloc];
    [reciveBuffer dealloc];
    [super dealloc];
}

- (void)awakeFromNib
{
    [journalText setBackgroundColor:[NSColor colorWithDeviceWhite:1.0
alpha:currentAlpha]];
    [journalPatientName setBackgroundColor:[NSColor
colorWithDeviceWhite:1.0 alpha:currentAlpha]];
    [journalText setEditable:NO];
    [journalPatientName setEditable:NO];
}

- (IBAction)logIn:(id)sender
{
    //Visa inloggnisngsvy
    [progressWheel setHidden:NO];
    [login setHidden:YES];
    [loginText setHidden:NO];

    //Animera snurran
    [progressWheel startAnimation:self];

    //Igonrera felaktiga uppkopplingar
    [NetSocket ignoreBrokenPipes];

    //Sätt anslutnign till server (borde vara under Preferences men inte än
så länge)
    mSocket = [[NetSocket netsocketConnectedToHost:@"192.168.0.4"
port:1985] retain];

    //Lägg mSockets funktioner på runLoop för att tillåta aktivitet i nib
    [mSocket scheduleOnCurrentRunLoop];

    //Delegera funktioerna från mSocket till denna klass
    [mSocket setDelegate:self];
    [saveButton setHidden: YES];

}

- (IBAction)logOutButton:(id)sender
{
    //Spela upp talad sekvens
    [self say: [@"Good bye " stringByAppendingString:[loginName stringVa-
lue]]];

    //Hantera gränssnitt
    [saveButton setHidden: YES];
    [login setHidden:NO];
    [journal setHidden:YES];
    [journalTable setHidden:YES];
    [loginText setHidden:YES];
    [journalsBox setHidden:YES];
    [logOutButton setHidden:YES];
    [loginName setStringValue:@""];
    [loginPassword setStringValue:@""];
```

```
    [journalPatientName setStringValue:@""];
    [journalText setString:@""];

    //Koppla ned anslutningen
    [mSocket release];
    isCrypt = NO;
    isStarted= NO;

}

- (IBAction)tableViewSelected:(id)sender
{

        if([sender selectedRow] != -1 &&
!isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal)
        {
            //Lås
            isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal
= YES;
            //Hämta data för PatientID
            [self requestJournalDataForPatientWithId: [[journals
objectAtIndex:[sender selectedRow]] objectAtIndex:0]];
        }
        else
        {
            //Spela upp standardljud om me försöker klicka under datahämt-
ning
            //eller klicka på en tom rad
            NSBeep();
        }

}

- (IBAction)saveJournal:(id)sender
{
    isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal = YES;
    [journalLoadBox setHidden:NO];
    [journalLoadProgress startAnimation:self];
    [journalPatientName setEditable:NO];
    [journalText setEditable:NO];
    [self saveJournalDataForPatientWithId: currentPatient];
}


- (void)setText: (NSString*)insText
{
    [logWindow insertText: insText];
    [logWindow insertText: @"\n"];
}

- (void)netsocketConnected:(NetSocket*)inNetSocket
{
    [self setText:@"Connected"];

    //Generera nya nycklar
    [self generateKeys];

    //Läs in nycklar
    NSString *publicKeyPath  = @"/Developer/client-public.pem";
    NSString *privateKeyPath = @"/Developer/client-private.pem";

    //Spara undan nycklar i dataobjekt och skapa krypteringsobjektet
    publicKeyData  = [NSData dataWithContentsOfFile:publicKeyPath];
    privateKeyData = [NSData dataWithContentsOfFile:privateKeyPath];

    [privateKeyData retain];
```

```
    //Skapa ett "paket" att skicka till mottagaren
    NSData *dataToSend;
    NSMutableArray *dataArr;

    dataArr = [[NSMutableArray alloc] init];

    NSString * method;
    method = [[NSString alloc] initWithString:@"PUBKEY-SESSION"];

    //Lägg först till metoden
    [dataArr addObject: method];

    //Sedan nyckel
    [dataArr addObject: publicKeyData];

    //Skapa ett serialiserat objekt att skicka till servern
    dataToSend = [NSPropertyListSerialization dataFromPropertyList:dataArr
format:NSPropertyListXMLFormat_v1_0 errorDescription:nil];

    //I det första meddelandet till servern skickar vi vår publika nyckel -
okrypterat
    [self sendToSocket: dataToSend];
    [self setText:@"Sent PUBLIC key to server"];

}

-(void) sendNextInQueue
{

    if ([toSend count] > 0)
    {
        //Skicka nästa i kön
        [self sendToSocket: [toSend objectAtIndex:0]];
        [toSend removeObjectAtIndex:0];

        isStarted = YES;

    }
    else
    {
        //Kön är tom
        isStarted = NO;
    }

}

- (void)netsocket:(NetSocket*)inNetSocket dataAvailable:(unsigned)inAmount
{
    //Hantera inkommen data
    [self handleRequest: inNetSocket];
}

- (void)netsocketDisconnected:(NetSocket*)inNetSocket
{
    //Hantera gränssnitt
    [progressWheel stopAnimation:self];
    [progressWheel setHidden:YES];
    [login setHidden:NO];
    [loginText setHidden:YES];
    [journalsBox setHidden:YES];
    [journal setHidden:YES];

    NSString *msg = @"Server unreachable, please check network connection";

    //Meddela audiovisuellt
    [self say: msg];
```

```
    //Meddela visuellt
    NSBeginAlertSheet(@"Communication
problem",@"OK",nil,nil,currentWindow,self,nil,NULL,nil,msg);
    isCrypt = NO;
    isStarted= NO;
    [crypto release];

    [self setText:@"Disconnected"];

}

-(void) handleRequest:(NetSocket*)inNetSocket
{

    if(!isCrypt)
    {
        NSData * key;
        NSMutableArray *dataArr;
        key = [inNetSocket readData];
        NSPropertyListFormat format;

        dataArr = [NSPropertyListSerialization propertyListFromData:key
mutabilityOption:NSPropertyListImmutable format:&format
errorDescription:nil];

        //Vi får en publik nyckel för klienten
        if([[dataArr objectAtIndex:0] isEqualToString:@"PUBKEY-SESSION"])
        {

            //Spara publika nyckeln
            serverPublicKeyData = [dataArr objectAtIndex:1];

            //Sätt den publika nyckeln till vårt krypteringsobjekt
            crypto = [[SSCrypto alloc]
initWithPublicKey:serverPublicKeyData privateKey:privateKeyData];

            //Skriv information
            [self setText:@"Recived PUBLIC key from server"];
            //[self setText: [[NSString alloc]
initWithData:serverPublicKeyData encoding: NSUTF8StringEncoding]];

            isCrypt = YES;
            [self setText:@"Entering encryption mode"];

            [self doLogin];

        }
        else
        {

            [self setText:@"Expected Key, but got something unreadable."];

        }
    }
    else
    {

        //Vi får krypterat dataobjekt från servern

        NSData*  data;
        data = [inNetSocket readData];

        [crypto setCipherText:data];
        NSData *decryptedTextData = [crypto decrypt];

        NSString *textFromIncoming = [[NSString alloc]
initWithData:decryptedTextData encoding:NSUTF8StringEncoding];
```

```
        //Servern talar om att den mottagit föregående data (för att und-
vika flooding av socket)
        if([textFromIncoming isEqualToString:@"ACKDATA"])
        {
            //Sicka nästa paket i kön
            [self sendNextInQueue];
        }
        else if([textFromIncoming isEqualToString:@"LOGIN-OK"])
        {
            //Servern talade om att inloggningen var okej, spela meddelan-
de och begär en lista över patienter
            [self say: [@"Welcome " stringByAppendingString:[loginName
stringValue]]];
            [self requestJournalsList];

        }
        else if([textFromIncoming isEqualToString:@"LOGIN-FAILD"])
        {
            //Inloggningen misslyckades
            [progressWheel stopAnimation:self];
            [progressWheel setHidden:YES];
            [login setHidden:NO];
            [loginText setHidden:YES];
            NSString *msg = @"Login faild, check username and password.";
            [self say:msg];
            NSBeginAlertSheet(@"Login
faild",@"OK",nil,nil,currentWindow,self,nil,NULL,nil,msg);
            isCrypt = NO;
            isStarted= NO;
            [inNetSocket release];
        }
        else if([textFromIncoming isEqualToString:@"STARTDATA"])
        {
            //Tala om att vi är redo att fortsätta att ta emot data
            [self ackData];

        }
        else if([textFromIncoming isEqualToString:@"STOPDATA"])
        {
            //Vi har fyllt upp ett dataobjekt, vi sätter samman och dese-
rialiserar det.
            NSMutableArray *toHandle;
            NSPropertyListFormat format;
            NSString *errere = nil;

            //Översätt från XML till en Array
            toHandle = [NSPropertyListSerialization
propertyListFromData:reciveBuffer mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:&errere];
            [reciveBuffer setLength: 0];

            //Kontrollera metodfältet
            if([[toHandle objectAtIndex:0]
isEqualToString:@"RESPONSE_JOURNALS"])
            {
                //Vi har fått svar på vilka patienter som finns
                [progressWheel stopAnimation:self];
                [progressWheel setHidden:YES];
                [login setHidden:YES];
                [journal setHidden:NO];
                [journalTable setHidden:NO];
                [loginText setHidden:YES];
                [journalsBox setHidden:NO];
                [logOutButton setHidden:NO];
```

```
                    //Sätt det objekt som vi får med journaler/patienter till
listan som syns i användargränssnittet
                    journals = [[toHandle objectAtIndex:1]retain];

                    //Ladda om listan så att datan syns.
                    [journalTable reloadData];
               }
               else if([[toHandle objectAtIndex:0]
isEqualToString:@"RESPONSE_JOURNAL_DATA"])
               {

                    //Tala om att vi inte väntar på journaldata löngre

isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal = NO;

                    //Kontroller rättighet (enbart för visuella behov en kon-
troll gärs även på servern om man
                    //försöker spara utan rättigheter ör det
                    if([[toHandle objectAtIndex:3] isEqualToString:@"NO"])
                    {
                         [saveButton setHidden:YES];
                         [journalText setEditable:NO];
                         [journalPatientName setEditable:NO];
                    }
                    else
                    {
                         [saveButton setHidden:NO];
                         [journalText setEditable:YES];
                         [journalPatientName setEditable:YES];
                    }

                    [journalLoadBox setHidden:YES];
                    [journalLoadProgress stopAnimation:self];
                    [journalText setString: [toHandle objectAtIndex:2]];
                    [journalPatientName setStringValue:[toHandle
objectAtIndex:1]];
               }

               //Tala om att vi har mottagit ACK-datan
               [self ackData];

          }
          else if([textFromIncoming isEqualToString:@"JOURNAL_DATA_SAVED"])
          {

isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal = NO;
                    [journalLoadBox setHidden:YES];
                    [journalLoadProgress stopAnimation:self];
                    [journalPatientName setEditable:YES];
                    [journalText setEditable:YES];
                    [self requestJournalsList];
          }
          else if([textFromIncoming
isEqualToString:@"JOURNAL_DATA_NOT_SAVED"])
          {
                    [self logOutButton:self];
          }
          else
          {
               //Lägg till data i bufferten, data kommer portionsvis
               [self setText:@"    Append data to end of buffer"];
               [reciveBuffer appendData:decryptedTextData];

               //Begär nästa dataobejkt
               [self ackData];
          }
```

```
    }


}


-(void) ackData
{

    NSString *start = @"ACKDATA";
    NSData *encryptedTextData = [crypto encrypt];
    [crypto setClearTextWithString:start];
    encryptedTextData = [crypto encrypt];

    //Skicka ett krypterat ACK-till servern
    [mSocket writeData:encryptedTextData];

}


-(int)numberOfRowsInTableView:(NSTableView *)tableView
{
    //Returnera antalet
    return [journals count];
}

-(id)tableView: (NSTableView *)tableView
objectValueForTableColumn:(NSTableColumn *)tableColumn row:(int)row
{
    //Returnera det riktiga ID:t för journalen
    return [[journals objectAtIndex:row] objectAtIndex:1];
}

-(id)tableView: (NSTableView *)tableView
shouldEditTableColumn:(NSTableColumn*) atb row:(int)rowi
{
    return NO;
}

-(BOOL)tableView: (NSTableView *)tableView shouldSelectRow:(int)rowi
{

    if(!isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal)
    {
            [journalLoadBox setHidden:NO];
            [journalLoadProgress startAnimation:self];
            [alphaTimer invalidate];
        //  alphaTimer = [NSTimer scheduledTimerWithTimeInterval:0.02
target:self selector:@selector(changeAlphaDown) userInfo:NULL repeats:YES];
    }

    return !isWaitingForAnswerOfDataFromServerWhenWeHaveKlickedOnAJournal;

}

-(void)sendToSocket: (NSData*)dataToSend
{
    //Skicka data på socket
    [self setText:@"Sent DATA to server"];
    [mSocket writeData:dataToSend];
}

-(void)sendData: (NSData*)dataToSend
{

    //Storlek på datablock
    #define dataSize 512
```

```
    [self setText:@"Sending DATA to server"];

    NSString *start = @"STARTDATA";
    NSString *stop = @"STOPDATA";
    NSData *encryptedTextData = [crypto encrypt];

    //Skicka ett STARTDATA för att tala om att mer data kommer
    [crypto setClearTextWithString:start];
    encryptedTextData = [crypto encrypt];

    //Lägg på sändkö
    [toSend addObject:encryptedTextData];

    //Påbörja styckning av data
    NSData *test;
    NSData *temp;// = [[NSData alloc] init];
    NSMutableArray *dataCont = [[NSMutableArray alloc] init];
    int timesInLoop = 0;

    while ([dataToSend length] > timesInLoop*(dataSize/8))
    {

            if ([dataToSend length]-timesInLoop*(dataSize/8) >= dataSize/
8)
            {
                test = malloc(dataSize/8);
                NSRange rangeCopy;
                rangeCopy.length = dataSize/8;
                rangeCopy.location = (dataSize/8)*timesInLoop;

                [dataToSend getBytes:test range:rangeCopy];
                temp = [[NSData alloc] initWithBytes:test
length:(dataSize/8)];
                [dataCont addObject:temp];
                free(test);
                test = 0;
            }
            else
            {
                test = malloc([dataToSend length]-timesInLoop*(dataSize/
8));
                NSRange rangeCopy;
                rangeCopy.length = [dataToSend length]-timesInLoop*(data-
Size/8);
                rangeCopy.location = (dataSize/8)*timesInLoop;

                [dataToSend getBytes:test range:rangeCopy];
                temp = [[NSData alloc] initWithBytes:test
length:[dataToSend length]-timesInLoop*(dataSize/8)];
                [dataCont addObject:temp];
                free(test);
                test = 0;
            }
            timesInLoop++;
    }

    while([dataCont count] > 0)
    {

        NSData *objectToEncryptAndSend;
        objectToEncryptAndSend = [dataCont objectAtIndex:0];

        [crypto setClearTextWithData:objectToEncryptAndSend];
        encryptedTextData = [crypto encrypt];

        //Lägg på sändkö
```

```
            [toSend addObject:encryptedTextData];

            [dataCont removeObjectAtIndex:0];

        }


        //Skicka ett STOPDATA för att tala om att inge mer data kommer
        //för detta objekt.
        [crypto setClearTextWithString:stop];
        encryptedTextData = [crypto encrypt];

        //Lägg på sändkö
        [toSend addObject:encryptedTextData];

        [dataCont release];
        [temp release];

        //Starta upp sänding om den inte redan är igång
        if(!isStarted)
            [self sendNextInQueue];

}

-(void) say:(NSString*) toSay
{

    NSTask *genPriv;
    NSTask *genPub;
    genPriv = [[NSTask alloc] init];
    genPub = [[NSTask alloc] init];

    //Vi kör SAY för att generera tal
    [genPriv setLaunchPath:@"/usr/bin/say"];

     //Sätt vad som skall spelas upp
    [genPriv setArguments:[NSArray arrayWithObjects:
          toSay,
          nil]];

    //Kör kommandot
    [self setText: @"Generating sound"];
    [genPriv  launch];
    [genPriv release];
}


-(void) generateKeys
{

    NSTask *genPriv;
    NSTask *genPub;
    genPriv = [[NSTask alloc] init];
    genPub = [[NSTask alloc] init];

    //Vi kör OPENSSL för att generera våra nycklar
    [genPriv setLaunchPath:@"/usr/bin/openssl"];

     //Sätt egenskaper för kommandot
    [genPriv setArguments:[NSArray arrayWithObjects:
          @"genrsa",
          @"-out",
          @"/Developer/client-private.pem",
           @"1024",
          nil]];

     //Kör kommandot
```

```
    [self setText: @"Generating new private key..."];
    [genPriv  launch];
    [genPriv waitUntilExit];
    [self setText:@"  Generation of private key - OK"];
    [genPriv release];

    //Vi kör OPENSSL för att generera våra nycklar
    [genPub setLaunchPath:@"/usr/bin/openssl"];

     //Sätt egenskaper för kommandot
    [genPub setArguments:[NSArray arrayWithObjects:
          @"rsa",
          @"-in",
          @"/Developer/client-private.pem",
           @"-out",
           @"/Developer/client-public.pem",
           @"-outform",
           @"PEM",
           @"-pubout",
          nil]];

     //Kör kommandot
    [self setText: @"Generating new public key..."];
    [genPub  launch];
    [genPub waitUntilExit];
    [self setText:@"  Generation of public key - OK"];
    [genPub release];

}
- (void) doLogin
{

     //Skapa ett "paket" att skicka till mottagaren
     NSData *loginData;
     NSMutableArray *loginArray;

     loginArray = [[NSMutableArray alloc] init];

     NSString * method;
     method = [[NSString alloc] initWithString:@"LOGIN"];

     //Lägg först till metoden
     [loginArray addObject: method];

     //Sedan användarnamn
     [loginArray addObject: [[NSString alloc] initWithString:[loginName
stringValue]]];

     //Sedan lösenord
     [loginArray addObject: [[NSString alloc] initWithString:[loginPassword
stringValue]]];

     //Skapa ett serialiserat objekt att skicka till servern
     loginData = [NSPropertyListSerialization
dataFromPropertyList:loginArray format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

     [self sendData: loginData];


}
- (void) requestJournalsList
{

     //Skapa ett "paket" att skicka till mottagaren
     NSData *sendRequest;
     NSMutableArray *requetArray;
```

```
    requetArray = [[NSMutableArray alloc] init];

    NSString * method;
    method = [[NSString alloc] initWithString:@"REQUEST_JOURNALS"];

    //Lägg först till metoden
    [requetArray addObject: method];

    //Sedan användarnamn
    [requetArray addObject: [[NSString alloc] initWithString:[loginName
stringValue]]];

    //Sedan lösenord
    [requetArray addObject: [[NSString alloc] initWithString:[loginPassword
stringValue]]];

    //Skapa ett serialiserat objekt att skicka till servern
    sendRequest = [NSPropertyListSerialization
dataFromPropertyList:requetArray format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

    [self sendData: sendRequest];

}

-(void) requestJournalDataForPatientWithId: (NSString*)patientID
{

    //Spara undan ID
    currentPatient = patientID;

    //Skapa ett "paket" att skicka till mottagaren
    NSData *sendRequest;
    NSMutableArray *requetArray;

    requetArray = [[NSMutableArray alloc] init];

    NSString * method;
    method = [[NSString alloc] initWithString:@"REQUEST_JOURNAL_DATA"];

    //Lägg först till metoden
    [requetArray addObject: method];

    //Sedan patientID
    [requetArray addObject: patientID];

    //Sedan anvnamn
    [requetArray addObject: [[NSString alloc] initWithString:[loginName
stringValue]]];

    //Sedan lösen
    [requetArray addObject: [[NSString alloc] initWithString:[loginPassword
stringValue]]];

    //Skapa ett serialiserat objekt att skicka till servern
    sendRequest = [NSPropertyListSerialization
dataFromPropertyList:requetArray format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

    [self sendData: sendRequest];

}

-(void) saveJournalDataForPatientWithId: (NSString*)patientID
{
    //Skapa ett "paket" att skicka till mottagaren
```

```
    NSData *sendRequest;
    NSMutableArray *requetArray;

    requetArray = [[NSMutableArray alloc] init];

    NSString * method;
    method = [[NSString alloc] initWithString:@"SAVE_JOURNAL_DATA"];

    //Lägg först till metoden
    [requetArray addObject: method];

    //Sedan patientID
    [requetArray addObject: patientID];

    //Sedan patientNamn
    [requetArray addObject: [journalPatientName stringValue]];

    //Sedan patientData
    [requetArray addObject: [journalText string]];

    //Sedan anvnamn
    [requetArray addObject: [[NSString alloc] initWithString:[loginName
stringValue]]];

    //Sedan lösen
    [requetArray addObject: [[NSString alloc] initWithString:[loginPassword
stringValue]]];

    //Skapa ett serialiserat objekt att skicka till servern
    sendRequest = [NSPropertyListSerialization
dataFromPropertyList:requetArray format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

    [self sendData: sendRequest];

}

@end
```

Kod - Server

```
//
//  main.m
//  server
//

#import <Cocoa/Cocoa.h>
#import "windowHandler.h"

int main(int argc, char *argv[])
{

    return NSApplicationMain(argc,  (const char **) argv);
}

//
//  windowHandler.h
//  server
//

#import <Cocoa/Cocoa.h>
#import "NetSocket.h"
#import <SSCrypto/SSCrypto.h>

@interface windowHandler : NSObject {
```

```
    IBOutlet id logWindow;
    NetSocket* mServerSocket;
    NetSocket* connectedClient;
    BOOL isCrypt;
    BOOL isStarted;
    NSData *publicKeyData;
    NSData *privateKeyData;
    NSData *clientPublicKeyData;
    SSCrypto *crypto;
    NSMutableData *reciveBuffer;
    NSMutableArray *toSend;
    NSMutableArray *journals;

}

//Konstruktör
//Pre: True
//Post: Ett nytt objekt har allokerats och startvärden satts
- (id)init;

//Destruktör
//Pre:
//Post: Objektet har deallokerats
- (void)dealloc;

//Funktion för att köra objekt när gränssnittet laddats klart
//Pre: Användargränssnittet just laddats
//Post: Satt information för objekt
- (void)awakeFromNib;

//Responder för sockets: ny anslutning
//Pre: Ny anslutning på inNetSocket
//Post: Har satt upp en ny ansluting till klienten
- (void)netsocket:(NetSocket*)inNetSocket
connectionAccepted:(NetSocket*)inNewNetSocket;

//Responder för socket: data tillgänglig
//Pre: Data har ankommit
//Post: Hantering av ankommande data har gjorts
- (void)netsocket:(NetSocket*)inNetSocket dataAvailable:(unsigned)inAmount;

//Responde för socket: klient kopplade ner
//Pre:
//Post: anslutningar avslutadade och nycklar rensade
- (void)netsocketDisconnected:(NetSocket*)inNetSocket;

//Sätter loggtext för visuell beskådning
//Pre: True
//Post: Texten har lagts till i loggen
-(void) setLogText: (id)text;

//Hantera en fäfrågan från en klient
//Pre: Det måste finnas data väntade på inNetSocket
//Post: Datan har hanterats
-(void) handleRequest:(NetSocket*)inNetSocket;

//Genererar nya sessionsnycklar
//Pre: True
//Post: Nycklar har genererats och sparats
-(void) generateKeys;

//Skickar en bekräftelse på mottagen data
//Pre: Data mottagen
//Post: Skickat bekräftelse till ansluten klient
-(void) ackData;

//Kontrollera ifall ett användarnman och lösenord är korrekt
```

```
//Pre: True
//Post: Returnerat YES om användarnamn och lösenord är korrekt, annars NO
-(bool) loginUsingName: (NSString*)username
andPassword:(NSString*)passWord;

//Skicka nästa datafragment
//Pre: !isStarted (om den anropas utanför datainsamling)
//Post: Skickat nästa dataelement
-(void) sendNextInQueue;

//Skicka data till klienten
//Pre: TRue
//Post: Datan skickas
-(void)sendData: (NSData*)dataToSend;

//Returnerar en Array över patienter som användaren har
//Pre: True
//Post: Returnerat en lista över patienter för användarnamnet och lösenor-
det
-(NSMutableArray*) getPatientsForUser: (NSString*)userName andPassword:
(NSString*)password;

//Returnerar journaldata för patient
//Pre: True
//Post: Returnerat patientdata för specifik patient om anv. och lösen. var
korrekta
-(NSString*) getJournalDataForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word;

//Returnerar patientnamn för patient
//Pre: True
//Post: Returnerat patientnamn för specifik patient om anv. och lösen. var
korrekta
-(NSString*) getJournalNameForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word;

//Returnerar tillåtelse att redigera journaldata för patient
//Pre: True
//Post: Returnerat "YES" vid tillåtelse att redigera journaldata för pati-
ent annars "NO" (observera NSString)
-(NSString*) getJournalEditForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word;

//Sparar journaldata och namn för en patientjournal
//Pre: [getJournalEditForPatientWithId ... isEqualToString:@"YES"]
//Post: Sparat data i datafil
-(BOOL) saveJournalDataForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word thenSetName: (NSString*)patientname andData:(NSString*)pDAta;

//Responder för kanpp "Exit server"
-(IBAction) endButton: sender;

@end
//
//  windowHandler.m
//  server
//

#import "windowHandler.h"

@implementation windowHandler

- (id)init
```

```
{
    if( ![super init] )
        return nil;

    // Initialize some values
    mServerSocket = nil;
    connectedClient = nil;
    isCrypt = NO;
    isStarted = NO;

    toSend = [[NSMutableArray alloc] init];
    reciveBuffer = [[NSMutableData alloc] init];

    return self;
}

- (void)dealloc
{
    [mServerSocket release];
    mServerSocket = nil;
    [super dealloc];
}

- (void)awakeFromNib
{

    mServerSocket = [[NetSocket netsocketListeningOnPort:1985] retain];
    [mServerSocket scheduleOnCurrentRunLoop];
    [mServerSocket setDelegate:self];

    [self setLogText: @"Server started"];
    [self setLogText: @"Waiting for connection on port 1985"];


}

- (void)netsocket:(NetSocket*)inNetSocket
connectionAccepted:(NetSocket*)inNewNetSocket
{
    [self setLogText: @"\n"];
    [self setLogText: @"New connection on port 1985"];
    connectedClient = [inNewNetSocket retain];

    // Setup socket for use
    [connectedClient open];
    [connectedClient scheduleOnCurrentRunLoop];

    [connectedClient setDelegate:self];
    [self setLogText: @"Connection opened"];

    //Generera nya nycklar
    [self generateKeys];

    //Läs in nycklarna
    NSString *privateKeyPath = @"/Developer/server-private.pem";

    //Spara undan nycklar i dataobjekt och skapa krypteringsobjektet
    privateKeyData = [NSData dataWithContentsOfFile:privateKeyPath];

    [privateKeyData retain];

}

- (void)netsocket:(NetSocket*)inNetSocket dataAvailable:(unsigned)inAmount
{
    [self handleRequest: inNetSocket];
}
```

```
- (void)netsocketDisconnected:(NetSocket*)inNetSocket
{

    [connectedClient release];
    connectedClient = nil;
    isCrypt = NO;
    [self setLogText:@"------------------------------------------------"];
    [self setLogText: @"Exiting encryption mode"];
    [self setLogText: @"Client disconnected"];
    //[NSApp terminate: self];
}

- (IBAction) endButton: sender
{


    [self setLogText: @"Ending..."];
    [NSApp terminate: self];
}

- (void) setLogText: (id)text
{

    [logWindow insertText: text];
    [logWindow insertText: @"\n"];

}


-(void) handleRequest:(NetSocket*)inNetSocket
{

    if(!isCrypt)
    {

        NSData * key;
        NSMutableArray *dataArr;
        key = [inNetSocket readData];
        NSPropertyListFormat format;

        dataArr = [NSPropertyListSerialization propertyListFromData:key
mutabilityOption:NSPropertyListImmutable format:&format
errorDescription:nil];

        if([[dataArr objectAtIndex:0] isEqualToString:@"PUBKEY-SESSION"])
        {

            //Läs in den publika nyckeln
            NSString *publicKeyPath  = @"/Developer/server-public.pem";
            publicKeyData  = [NSData
dataWithContentsOfFile:publicKeyPath];

            clientPublicKeyData = [dataArr objectAtIndex:1];
            crypto = [[SSCrypto alloc]
initWithPublicKey:clientPublicKeyData privateKey:privateKeyData];

            [self setLogText:@"Recived PUBLIC KEY from client"];

        }
        else
        {
            [self setLogText:@"Expected Key, but got something unreadab-
le"];
        }

        //Skapa ett "paket" att skicka till mottagaren
```

```
        NSData *dataToSend;
        NSMutableArray *newDataArr;

        newDataArr = [[NSMutableArray alloc] init];

        NSString * method;
        method = [[NSString alloc] initWithString:@"PUBKEY-SESSION"];

        //Lägg först till metoden
        [newDataArr addObject: method];

        //Sedan nyckel
        [newDataArr addObject: publicKeyData];

        //Skapa ett serialiserat objekt att skicka till servern
        dataToSend = [NSPropertyListSerialization
dataFromPropertyList:newDataArr format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

        //Vi svarar genom att skicka vår egen publika nyckel till servern.
        [inNetSocket writeData:dataToSend];

        [self setLogText:@"Sent PUBLIC key to client"];

        isCrypt = YES;
        [self setLogText:@"Entering encryption mode"];
        [self
setLogText:@"------------------------------------------------"];

    }
    else
    {

        NSData*  data;
        data = [inNetSocket readData];

        [crypto setCipherText:data];
        NSData *decryptedTextData = [crypto decrypt];

        NSString *textFromIncoming = [[NSString alloc]
initWithData:decryptedTextData encoding:NSUTF8StringEncoding];

        if([textFromIncoming isEqualToString:@"ACKDATA"])
        {
            [self sendNextInQueue];

        }
        else if([textFromIncoming isEqualToString:@"STARTDATA"])
        {

            [self setLogText:@"Start reciving dataobject"];
            [self ackData];

        }
        else if([textFromIncoming isEqualToString:@"STOPDATA"])
        {

            [self setLogText:@"End of reciving dataobject"];

            NSMutableArray *toHandle;
            NSPropertyListFormat format;

            toHandle = [NSPropertyListSerialization
propertyListFromData:reciveBuffer mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

            [reciveBuffer setLength: 0];
```

```
            if([[toHandle objectAtIndex:0] isEqualToString:@"LOGIN"])
            {
                    [self setLogText:@"Request: LOGIN"];
                    if([self loginUsingName: [toHandle objectAtIndex:1]
andPassword:[toHandle objectAtIndex:2]])
                    {
                            NSString *start = @"LOGIN-OK";
                            [self setLogText:@"Response: loginOK"];
                            NSData *encryptedTextData = [crypto en-
crypt];

                            [crypto setClearTextWithString:start];
                            encryptedTextData = [crypto encrypt];

                            [connectedClient
writeData:encryptedTextData];
                    }
                    else
                    {
                            NSString *start = @"LOGIN-FAILD";
                            [self setLogText:@"Response: loginFaild"];
                            NSData *encryptedTextData = [crypto en-
crypt];

                            [crypto setClearTextWithString:start];
                            encryptedTextData = [crypto encrypt];

                            [connectedClient
writeData:encryptedTextData];
                    }


            }
            else if([[toHandle objectAtIndex:0]
isEqualToString:@"REQUEST_JOURNALS"])
            {
                 [self setLogText:@"Request: REQUEST_JOURNALS"];

                 // Hämtar en lista med patienter + nycklar + rättigheter
baserat på användanarnaman
                 // Hämtar en jorunal baserat på patient + nyckel

                 //Skapa ett "paket" att skicka till mottagaren
                 NSData *patientsDataToSend;
                 NSMutableArray *message;

                 message = [[NSMutableArray alloc] init];

                 NSString * method;
                 method = [[NSString alloc]
initWithString:@"RESPONSE_JOURNALS"];

                 //Lägg först till metoden
                 [message addObject: method];

                 //Sedan övrig data
                 [message addObject: [self getPatientsForUser: [toHandle
objectAtIndex:1] andPassword: [toHandle objectAtIndex:2]]];

                 //Skapa ett serialiserat objekt att skicka till servern
                 patientsDataToSend = [NSPropertyListSerialization
dataFromPropertyList:message format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

                 [self setLogText:@"Response: RESPONSE_JOURNALS"];
```

```
                    //Skicka objekt
                    [self sendData: patientsDataToSend];



               }
               else if([[toHandle objectAtIndex:0]
isEqualToString:@"REQUEST_JOURNAL_DATA"])
               {
                    [self setLogText:@"Request: REQUEST_JOURNAL_DATA"];

                    // Returnera ett objekt men namn, text och redigerings-
rättigheter

                    //Skapa ett "paket" att skicka till mottagaren
                    NSData *patientsDataToSend;
                    NSMutableArray *message;

                    message = [[NSMutableArray alloc] init];

                    NSString * method;
                    method = [[NSString alloc]
initWithString:@"RESPONSE_JOURNAL_DATA"];

                    //Lägg först till metoden
                    [message addObject: method];

                    //Sedan namn
                    [message addObject: [self
getJournalNameForPatientWithId:[toHandle objectAtIndex:1]
withPermissionOfUserName:[toHandle objectAtIndex:2] andPassword:[toHandle
objectAtIndex:3]]];

                    //Sedan data
                    [message addObject: [self
getJournalDataForPatientWithId:[toHandle objectAtIndex:1]
withPermissionOfUserName:[toHandle objectAtIndex:2] andPassword:[toHandle
objectAtIndex:3]]];

                    //Sedan redigeringstillåtelse
                    [message addObject: [self
getJournalEditForPatientWithId:[toHandle objectAtIndex:1]
withPermissionOfUserName:[toHandle objectAtIndex:2] andPassword:[toHandle
objectAtIndex:3]]];

                    //Skapa ett serialiserat objekt att skicka till servern
                    patientsDataToSend = [NSPropertyListSerialization
dataFromPropertyList:message format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

                    [self setLogText:@"Response: RESPONSE_JOURNAL_DATA"];

                    //Skicka objekt
                    [self sendData: patientsDataToSend];

               }
               else if([[toHandle objectAtIndex:0]
isEqualToString:@"SAVE_JOURNAL_DATA"])
               {
                    [self setLogText:@"Request: SAVE_JOURNAL_DATA"];

                    if ([self saveJournalDataForPatientWithId: [toHandle
objectAtIndex:1] withPermissionOfUserName:[toHandle objectAtIndex:4] and-
Password: [toHandle objectAtIndex:5] thenSetName: [toHandle
objectAtIndex:2] andData:[toHandle objectAtIndex:3]])
                    {
```

```
                    NSString *start = @"JOURNAL_DATA_SAVED";
                    NSData *encryptedTextData = [crypto encrypt];

                    [crypto setClearTextWithString:start];
                    encryptedTextData = [crypto encrypt];

                    [connectedClient writeData:encryptedTextData];

                    [self setLogText:@"Response: JOURNAL_DATA_SAVED"];
                }
                else
                {
                    NSString *start = @"JOURNAL_DATA_NOT_SAVED";
                    NSData *encryptedTextData = [crypto encrypt];

                    [crypto setClearTextWithString:start];
                    encryptedTextData = [crypto encrypt];

                    [connectedClient writeData:encryptedTextData];

                    [self setLogText:@"Response: JOURNAL_DATA_NOT_SA-
VED"];
                }


            }


            [self ackData];

        }
        else
        {

            [reciveBuffer appendData:decryptedTextData];
            [self setLogText:@"       Append data to end of buffer"];
            [self ackData];
        }

    }

}

-(bool) loginUsingName: (NSString*)username andPassword:(NSString*)passWord
{

    NSString *path = @"/Developer/loginData.xml";
    NSData *readed = [[NSData alloc] initWithContentsOfFile:path];
    NSPropertyListFormat format;
    NSDictionary *loginData = [NSPropertyListSerialization
propertyListFromData:readed mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

    SSCrypto* md5a = [[SSCrypto alloc] init];
    [md5a setClearTextWithString:passWord];
    NSData *toMd5 = [md5a digest:@"MD5"];
    NSString *md5passw = [toMd5 hexval];
    [md5a release];

    if ([[loginData objectForKey:username] isEqualToString: md5passw])
    {
        return YES;
    }
    else
    {
        return NO;
```

```
    }

}

-(NSString*) getJournalDataForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word
{

    NSString *toReturn = @"N/A";

    //Kontrollera användarnamn och lösenord
    if ([self loginUsingName:userName andPassword:password])
    {

        NSString *path = @"/Developer/userPatientRelation.xml";
        NSData *db = [[NSData alloc] initWithContentsOfFile:path];
        NSPropertyListFormat format;
        NSDictionary *dbDict = [NSPropertyListSerialization
propertyListFromData:db mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        NSArray *userPatients = [dbDict objectForKey:userName];

        NSEnumerator * enumer = [userPatients objectEnumerator];
        NSData * element;

        NSString *pathToJournal = @"/Developer/patientJournals.xml";
        NSData *journalDB = [[NSData alloc]
initWithContentsOfFile:pathToJournal];
        NSDictionary *journalDBDict = [NSPropertyListSerialization
propertyListFromData:journalDB mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        //Hämta varje patients information
        while(element = [enumer nextObject])
        {
            //Avkryptera låst information
            SSCrypto* md5a = [[SSCrypto alloc] init];
            [md5a setClearTextWithString:password];
            NSData *keyForCipher = [md5a digest:@"MD5"];
            SSCrypto *aesCipher = [[SSCrypto alloc]
initWithSymmetricKey:keyForCipher];
            [aesCipher setCipherText:element];
            NSData * pure = [aesCipher decrypt];

            NSArray *currentPatient = [NSPropertyListSerialization
propertyListFromData:pure mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

            NSString *patientID = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:0]];
            NSString *patientKEY = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:1]];

            if([patientID isEqualToString:pID])
            {
                NSData *patientData = [journalDBDict
objectForKey:patientID];
                SSCrypto *aesCipher2 = [[SSCrypto alloc]
initWithSymmetricKey:[patientKEY dataUsingEncoding:NSASCIIStringEncoding]];
                [aesCipher2 setCipherText:patientData];
                NSData * decryptedPatientData = [aesCipher2 decrypt];

                NSArray *currentPatientArray = [NSPropertyListSerializa-
tion propertyListFromData:decryptedPatientData
```

```
mutabilityOption:NSPropertyListImmutable format:&format
errorDescription:nil];

                    toReturn = [currentPatientArray objectAtIndex:1];
               }

          }

     }

     return toReturn;

}

-(NSString*) getJournalNameForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word
{

     NSString *toReturn = @"N/A";

     //Kontrollera användarnamn och lösenord
     if ([self loginUsingName:userName andPassword:password])
     {

          NSString *path = @"/Developer/userPatientRelation.xml";
          NSData *db = [[NSData alloc] initWithContentsOfFile:path];
          NSPropertyListFormat format;
          NSDictionary *dbDict = [NSPropertyListSerialization
propertyListFromData:db mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

          NSArray *userPatients = [dbDict objectForKey:userName];

          NSEnumerator * enumer = [userPatients objectEnumerator];
          NSData * element;

          NSString *pathToJournal = @"/Developer/patientJournals.xml";
          NSData *journalDB = [[NSData alloc]
initWithContentsOfFile:pathToJournal];
          NSDictionary *journalDBDict = [NSPropertyListSerialization
propertyListFromData:journalDB mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

          //Hämta varje patients information
          while(element = [enumer nextObject])
          {
               //Avkryptera låst information
               SSCrypto* md5a = [[SSCrypto alloc] init];
               [md5a setClearTextWithString:password];
               NSData *keyForCipher = [md5a digest:@"MD5"];
               SSCrypto *aesCipher = [[SSCrypto alloc]
initWithSymmetricKey:keyForCipher];
               [aesCipher setCipherText:element];
               NSData * pure = [aesCipher decrypt];

               NSArray *currentPatient = [NSPropertyListSerialization
propertyListFromData:pure mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

               NSString *patientID = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:0]];
               NSString *patientKEY = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:1]];

               if([patientID isEqualToString:pID])
               {
```

```
                    NSData *patientData = [journalDBDict
objectForKey:patientID];
                    SSCrypto *aesCipher2 = [[SSCrypto alloc]
initWithSymmetricKey:[patientKEY dataUsingEncoding:NSASCIIStringEncoding]];
                    [aesCipher2 setCipherText:patientData];
                    NSData * decryptedPatientData = [aesCipher2 decrypt];

                    NSArray *currentPatientArray = [NSPropertyListSerializa-
tion propertyListFromData:decryptedPatientData
mutabilityOption:NSPropertyListImmutable format:&format
errorDescription:nil];

                    toReturn = [currentPatientArray objectAtIndex:0];
               }

          }


     }

     return toReturn;

}

-(NSString*) getJournalEditForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word
{

     NSString* toReturn = @"NO";

     //Kontrollera användarnamn och lösenord
     if ([self loginUsingName:userName andPassword:password])
     {

          NSString *path = @"/Developer/userPatientRelation.xml";
          NSData *db = [[NSData alloc] initWithContentsOfFile:path];
          NSPropertyListFormat format;
          NSDictionary *dbDict = [NSPropertyListSerialization
propertyListFromData:db mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

          NSArray *userPatients = [dbDict objectForKey:userName];

          NSEnumerator * enumer = [userPatients objectEnumerator];
          NSData * element;

          //Hämta varje patients information
          while(element = [enumer nextObject])
          {
               //Avkryptera låst information
               SSCrypto* md5a = [[SSCrypto alloc] init];
               [md5a setClearTextWithString:password];
               NSData *keyForCipher = [md5a digest:@"MD5"];
               SSCrypto *aesCipher = [[SSCrypto alloc]
initWithSymmetricKey:keyForCipher];
               [aesCipher setCipherText:element];
               NSData * pure = [aesCipher decrypt];

               NSArray *currentPatient = [NSPropertyListSerialization
propertyListFromData:pure mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];
               NSString *patientID = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:0]];

               if([patientID isEqualToString:pID])
                    toReturn = [currentPatient objectAtIndex:2];
```

```
        }

    }

    return toReturn;

}


-(NSMutableArray*) getPatientsForUser: (NSString*)userName andPassword:
(NSString*)password
{

    //Kontrollera användarnamn och lösenord
    if ([self loginUsingName:userName andPassword:password])
    {

        NSString *path = @"/Developer/userPatientRelation.xml";
        NSData *db = [[NSData alloc] initWithContentsOfFile:path];
        NSPropertyListFormat format;
        NSDictionary *dbDict = [NSPropertyListSerialization
propertyListFromData:db mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        NSArray *userPatients = [dbDict objectForKey:userName];

        NSEnumerator * enumer = [userPatients objectEnumerator];
        NSData * element;

        NSMutableArray * toFill = [[NSMutableArray alloc] init];

        NSString *pathToJournal = @"/Developer/patientJournals.xml";
        NSData *journalDB = [[NSData alloc]
initWithContentsOfFile:pathToJournal];
        NSDictionary *journalDBDict = [NSPropertyListSerialization
propertyListFromData:journalDB mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        //Hämta varje patients information
        while(element = [enumer nextObject])
        {
            //Avkryptera låst information
            SSCrypto* md5a = [[SSCrypto alloc] init];
            [md5a setClearTextWithString:password];
            NSData *keyForCipher = [md5a digest:@"MD5"];
            SSCrypto *aesCipher = [[SSCrypto alloc]
initWithSymmetricKey:keyForCipher];
            [aesCipher setCipherText:element];
            NSData * pure = [aesCipher decrypt];

            NSArray *currentPatient = [NSPropertyListSerialization
propertyListFromData:pure mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

            NSString *patientID = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:0]];
            NSString *patientKEY = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:1]];

            NSData *patientData = [journalDBDict objectForKey:patientID];
            SSCrypto *aesCipher2 = [[SSCrypto alloc]
initWithSymmetricKey:[patientKEY dataUsingEncoding:NSASCIIStringEncoding]];
            [aesCipher2 setCipherText:patientData];
            NSData * decryptedPatientData = [aesCipher2 decrypt];

            NSArray *currentPatientArray = [NSPropertyListSerialization
propertyListFromData:decryptedPatientData
```

```
mutabilityOption:NSPropertyListImmutable format:&format
errorDescription:nil];

            [self setLogText:@"      Building patient dataobjekt"];

            //Fyll upp en array att skicka tillbaka
            [toFill addObject:[[NSArray alloc]
initWithObjects:patientID,[currentPatientArray objectAtIndex:0],nil]];

        }

        return toFill;

    }
    else
    {
        NSMutableArray * toFill = [[NSMutableArray alloc]
initWithObjects:@"N/A"];
        return toFill;
    }
}

-(BOOL) saveJournalDataForPatientWithId: (NSString*)pID
withPermissionOfUserName:(NSString*)userName andPassword: (NSString*)pass-
word thenSetName: (NSString*)patientname andData:(NSString*)pDAta
{

    BOOL toReturn = NO;

    //Kontrollera användarnamn och lösenord först, rättigheten kollas sen
    if ([self loginUsingName:userName andPassword:password])
    {

        NSString *path = @"/Developer/userPatientRelation.xml";
        NSData *db = [[NSData alloc] initWithContentsOfFile:path];
        NSPropertyListFormat format;
        NSDictionary *dbDict = [NSPropertyListSerialization
propertyListFromData:db mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        NSArray *userPatients = [dbDict objectForKey:userName];

        NSEnumerator * enumer = [userPatients objectEnumerator];
        NSData * element;

        NSString *pathToJournal = @"/Developer/patientJournals.xml";
        NSData *journalDB = [[NSData alloc]
initWithContentsOfFile:pathToJournal];
        NSDictionary *journalDBDict = [NSPropertyListSerialization
propertyListFromData:journalDB mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];

        //Hämta varje patients information
        while(element = [enumer nextObject])
        {
            //Avkryptera låst information
            SSCrypto* md5a = [[SSCrypto alloc] init];
            [md5a setClearTextWithString:password];
            NSData *keyForCipher = [md5a digest:@"MD5"];
            SSCrypto *aesCipher = [[SSCrypto alloc]
initWithSymmetricKey:keyForCipher];
            [aesCipher setCipherText:element];
            NSData * pure = [aesCipher decrypt];

            NSArray *currentPatient = [NSPropertyListSerialization
propertyListFromData:pure mutabilityOption:NSPropertyListImmutable
format:&format errorDescription:nil];
```

```
            NSString *patientID = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:0]];
            NSString *patientKEY = [[NSString alloc]
initWithString:[currentPatient objectAtIndex:1]];

            if([patientID isEqualToString:pID] && [[currentPatient
objectAtIndex:2] isEqualToString:@"YES"] )
            {
                //Hämta objektet
                NSData *patientData = [journalDBDict
objectForKey:patientID];
                SSCrypto *aesCipher2 = [[SSCrypto alloc]
initWithSymmetricKey:[patientKEY dataUsingEncoding:NSASCIIStringEncoding]];
                NSData * decryptedPatientData;

                //Vi skapar en ny array med de data som nu är aktuella
                NSArray* newPatientArray = [[NSArray alloc]
initWithObjects:patientname, pDAta, nil];

                //Serialisera data och kryptera
                decryptedPatientData = [NSPropertyListSerialization
dataFromPropertyList:newPatientArray format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];
                [aesCipher2 setClearTextWithData:decryptedPatientData];
                patientData = [aesCipher2 encrypt];

                //Uppdatera Dict
                [journalDBDict  setValue:patientData forKey:patientID];

                //DICTEN är nu i Dataformat
                journalDB = [NSPropertyListSerialization
dataFromPropertyList:journalDBDict format:NSPropertyListXMLFormat_v1_0
errorDescription:nil];

                //Spara journalDBDict som XML
                NSFileHandle *save = [NSFileHandle
fileHandleForWritingAtPath:pathToJournal];
                [save writeData:journalDB];

                toReturn = YES;
            }

        }

    }

    return toReturn;

}


-(void) ackData
{

    NSString *start = @"ACKDATA";
    NSData *encryptedTextData = [crypto encrypt];

    //Skicka ett ACK
    [crypto setClearTextWithString:start];
    encryptedTextData = [crypto encrypt];

    [connectedClient writeData:encryptedTextData];

}

-(void) generateKeys
```

```
{

    NSTask *genPriv;
    NSTask *genPub;
    genPriv = [[NSTask alloc] init];
    genPub = [[NSTask alloc] init];

    //Vi kör OPENSSL för att generera våra nycklar
    [genPriv setLaunchPath:@"/usr/bin/openssl"];

      //Sätt egenskaper för kommandot
    [genPriv setArguments:[NSArray arrayWithObjects:
          @"genrsa",
          @"-out",
          @"/Developer/server-private.pem",
           @"1024",
          nil]];

      //Kör kommandot
    [self setLogText: @"Start of private key generation"];
    [genPriv  launch];
    while([genPriv isRunning])
      {}
    [self setLogText:@"              Generation finished"];
    [genPriv release];

    //Vi kör OPENSSL för att generera våra nycklar
    [genPub setLaunchPath:@"/usr/bin/openssl"];

      //Sätt egenskaper för kommandot
    [genPub setArguments:[NSArray arrayWithObjects:
          @"rsa",
          @"-in",
          @"/Developer/server-private.pem",
           @"-out",
           @"/Developer/server-public.pem",
           @"-outform",
           @"PEM",
           @"-pubout",
          nil]];

      //Kör kommandot
    [self setLogText: @"Start of public key generation"];
    [genPub  launch];
    while([genPub isRunning])
      {}
    [self setLogText:@"              Generation finished"];
    [genPub release];

}

-(void)sendData: (NSData*)dataToSend
{

     #define dataSize 512

     NSString *start = @"STARTDATA";
     NSString *stop = @"STOPDATA";
     NSData *encryptedTextData = [crypto encrypt];

     //Skicka ett STARTDATA
     [crypto setClearTextWithString:start];
     encryptedTextData = [crypto encrypt];
     //Lägg på sändkö
     [toSend addObject:encryptedTextData];

     //Påbörja styckning av data
```

```
    NSData *test;
    NSData *temp;
    NSMutableArray *dataCont = [[NSMutableArray alloc] init];
    int timesInLoop = 0;

    while ([dataToSend length] > timesInLoop*(dataSize/8))
    {

            if ([dataToSend length]-timesInLoop*(dataSize/8) >= dataSize/
8)
            {
                test = malloc(dataSize/8);
                NSRange rangeCopy;
                rangeCopy.length = dataSize/8;
                rangeCopy.location = (dataSize/8)*timesInLoop;

                [dataToSend getBytes:test range:rangeCopy];
                temp = [[NSData alloc] initWithBytes:test
length:(dataSize/8)];
                [dataCont addObject:temp];
                free(test);
                test = 0;
            }
            else
            {
                test = malloc([dataToSend length]-timesInLoop*(dataSize/
8));
                NSRange rangeCopy;
                rangeCopy.length = [dataToSend length]-timesInLoop*(data-
Size/8);
                rangeCopy.location = (dataSize/8)*timesInLoop;

                [dataToSend getBytes:test range:rangeCopy];
                temp = [[NSData alloc] initWithBytes:test
length:[dataToSend length]-timesInLoop*(dataSize/8)];
                [dataCont addObject:temp];
                free(test);
                test = 0;
            }
            timesInLoop++;
    }

    while([dataCont count] > 0)
    {

        NSData *objectToEncryptAndSend;
        objectToEncryptAndSend = [dataCont objectAtIndex:0];

        [crypto setClearTextWithData:objectToEncryptAndSend];
        encryptedTextData = [crypto encrypt];
        //Lägg på sändkö
        [toSend addObject:encryptedTextData];

        [dataCont removeObjectAtIndex:0];

    }


    //Skicka ett STOPDATA
    [crypto setClearTextWithString:stop];
    encryptedTextData = [crypto encrypt];
    //Lägg på sändkö
    [toSend addObject:encryptedTextData];

    [dataCont release];
    [temp release];
```

```
    if(!isStarted)
        [self sendNextInQueue];

}

-(void) sendNextInQueue
{

    if ([toSend count] > 0)
    {
        if(!isStarted)
            [self setLogText:@"Start sending data"];

        [connectedClient writeData: [toSend objectAtIndex:0]];
        [toSend removeObjectAtIndex:0];
        [self setLogText:@"       Sending datachunk"];


        isStarted = YES;

    }
    else
    {
        isStarted = NO;
        [self setLogText:@"Finished sending data"];
    }

}


@end
```