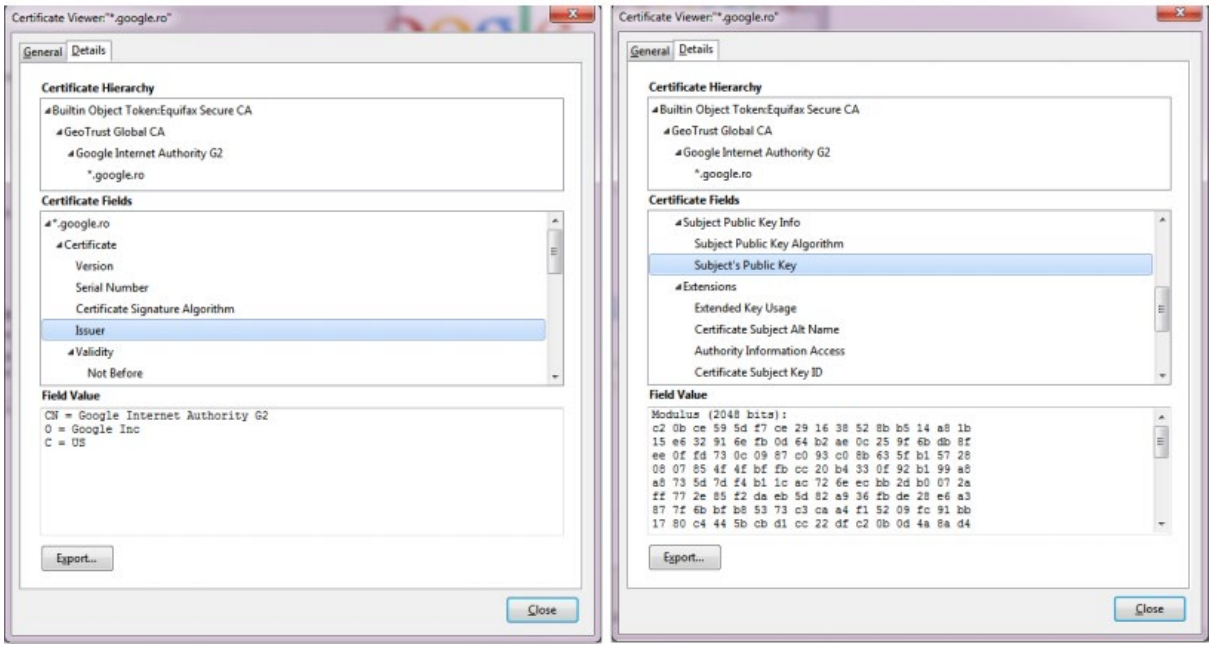
Asymmetrisk

Kryptering

Asymmetrisk kryptering i .NET med RSA

Denne øvelse omhandler RSA krypteringssystemet baseret på systemets udførelsesform i .NET frameworket. Navnet RSA stammer fra navnet på de tre opfindere: Rivest, Shamir og Adleman, der udgav krypteringssystemet i 1978. RSA kan bruges til at udføre offentlig/privat kryptering samt til funktionaliteten digitale signatur. Ofte anvendes asymmetriske krypteringer algoritmer til at kryptere nøgler der benyttes i symmetrisk kryptering (du kan bruge offentlige nøgler til at kryptere meddelelser eller filer, men dette ville være meget ineffektivt), mens digitale signaturer bruges til at bevise at et stykke data stammer fra en bestemt enhed. For eksempel benytter du Googles offentligt certifikat for at hente en offentlige nøgle og derefter kryptere med en mindre AES-nøgle for at oprette en krypteret tunnel mellem din e-mail-klient og Gmail's server. Årsagen til at kryptere en lille sessionnøgle med RSA i stedet for kryptering af meddelelser, der udveksles mellem parter, er enkel: effektivitet. RSA har fordelen ved ikke at kræve en hemmelig nøgle, der deles mellem parterne, men RSA er meget mindre effektive (tidsforbrug) end symmetriske algoritmer såsom AES. Derfor bruges RSA-kryptering generelt til udveksling af små hemmelige nøgler fra AES til DES (hybrid kryptering).

Nedenstående dataer fra Google, den offentlige nøgle skal underskrives/signed af en betroet part, anerkendt af din browser, for at sikre, at dataene faktisk er til Googles (for at undgå man-in-the-middle angreb). Figur 1 viser dele af et sådant certifikat



Figur 1: Del af et RSA-certifikat udstedt til Google, offentlig nøgle til højre

Kort teoretisk baggrundi

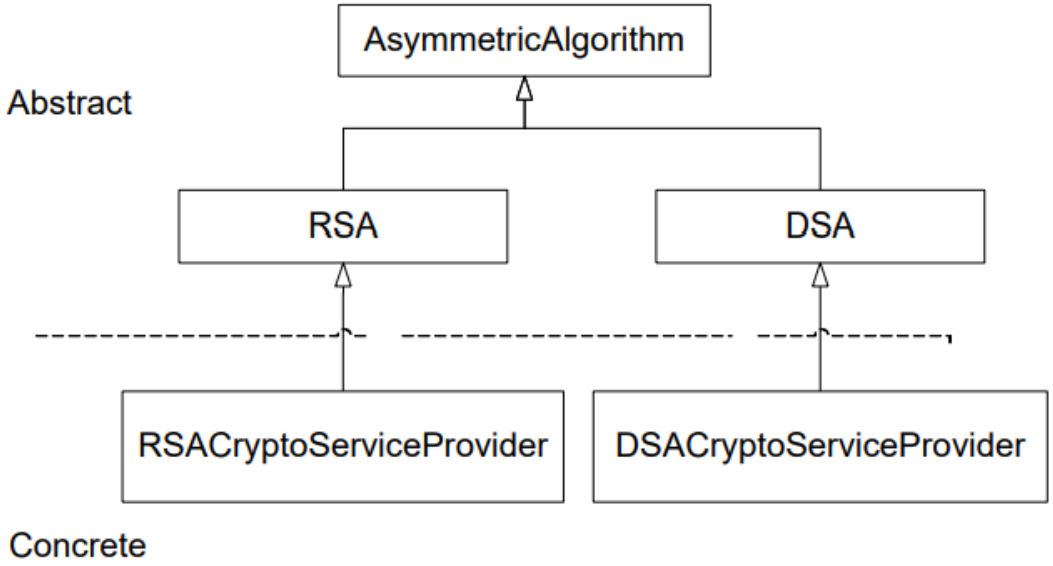
Systemet virker ved at vælge to (store) primtal p og q, hvorfra man beregner n = pq samt d og e, hvor de mod (p-1)(q-1) = 1, og e og (p-1)(q-1) ikke har fælles primtalsfaktorer. (n,e) er den offentlige nøgle, (n,d) er den private/hemmelige nøgle. Kryptering af tallet m foregår herefter ved at beregne c = me mod n. Dekryptering udføres ved at beregne m = cd mod n.

Sikkerheden i systemet hviler på antagelsen om at primtalsfaktorisering er et "svært" problem, dvs. at der ikke findes algoritmer der kan udføre opgaven i polynomiel tid. RSA kan dog brydes i polynomiel tid på en kvantecomputer med Shors algoritme

Det er vigtigt at i forstår er at sikkerheden i systemet hviler på antagelsen om at primtalsfaktorisering er et "svært" problem

Rsacryptoserviceprovider: egenskaber og metoder

RSA-implementeringen i .NET understøtter nøgler fra 384 til 16384 bit i 8 bit intervaller. Nøglestørrelsen kan specificeres via konstruktøren i `RSACryptoServiceProvider` klassen, der genererer en tilfældig RSA-nøgle. Konstruktøren tillader også initialisering med en eksisterende nøgle, der er angivet som `CspParameters` objekt. I det kommende afsnit gives der flere detaljer om nøglestrukturen, nu fokuseres der på egenskaber og metoder, der er i klassen `RSACryptoServiceProvider`, disse er opsummeret i tabel 1 og 2. Figur 2 viser klassehierarkiet for RSA og DSA i .NET



Figur 2: RSA og DSA klasser i .NET

	Get/Set	Type	Brief Description
PublicOnly	g	Boolean	Return true if the object contains just the public key
KeySize	g	Int	Key size in bits
LegalKeySizes	g	KeySizes[]	Key sizes in bits supported by the algorithm
SignatureAlgorithm	g	String	The name of the signature algorithm, in .NET signing is always performed as RSA with SHA1

Tabel 1: egenskaber fra RSACryptoServiceProvider

	Return type	Brief Description
`Decrypt (byte[] data, bool fOAEP)`	byte[]	Decrypts data given as byte and returns the decrypted value as byte. The Boolean indicates if the OAEP padding is used, if false, then PKCS# v.15 padding is used instead.
`Encrypt (byte[] data, bool fOAEP)`	byte[]	Encrypts data given as byte and returns the encrypted value as byte. The Boolean indicates if the OAEP padding is used, if false, then PKCS# v.15 padding is used instead.
`ExportParameters (bool includePrivateParameters)`	RSAParameters	Gets the RSA key as RSAParameters object. The Boolean specifies if the private part of the key is or not included.
`ImportParameters (RSAParameters parameters)`	void	Sets the RSA key from RSAParameters object
`ToXmlString (bool includePrivateParameters)`	string	Gets the RSA key as string in XML format. The Boolean specifies if the private part of the key is or not included.
`FromXmlString (bool includePrivateParameters)`	void	Sets the RSA key from a string in XML format.
`SignData (byte[] buffer, Object halg)`	byte[]	Signs the given array of bytes with the specified hash algorithm, returns the signature as array of bytes
`SignData(Stream inputStream, Object halg)`	byte[]	Same as previously, but this time the data is given as stream
`SignData(byte[] buffer, int offset, int count, Object halg)`	byte[]	Signs the byte array starting from offset for count bytes
`SignHash(byte[] hash, string str)`	byte[]	Signs the hash of the data, the string is the name of the algorithm that was used to hash the data
`SignHash(byte[] hash, string str`	bool	Verifies the signature given a hash algorithm as object, the signature and message as byte arrays
`VerifyHash (byte[] Hash, string str, byte[] Signature)`	bool	Verifies the signature given the hash of the message and the name of the hash algorithm

Tabel 2: metoder fra RSACryptoServiceProvider

Kryptering og signering med RSA i .NET. Kryptering og dekryptering med RSA er ligetil. Der er kun to trin, du skal følge:

1. Opret et RSA objek (letteste måde er ved at specificere størrelsen på nøglen)
2. Kald krypteringsmetode på de data, der er specificeret som byte-array og en bool, der angiver, om OAEP[ii][1] skal bruges (anbefales).

[1] https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding

```
RSACryptoServiceProvider myRSA = new RSACryptoServiceProvider(2048);
AesManaged myAES = new AesManaged();
byte[] RSAciphertext;
byte[] plaintext;
//generate an AES key
myAES.GenerateKey();
//encrypt an AES key with RSA
RSAciphertext = myRSA.Encrypt(myAES.Key, true);
//decrypt and recover the AES key
plaintext = myRSA.Decrypt(RSAciphertext, true);
```

Figur 1: eksempel på RSA-kryptering og dekryptering i .NET

```
SHA256Managed myHash = new SHA256Managed();
 string some_text = "this is an important message";
 //sign the message
 byte[] signature;
 signature =
 myRSA.SignData(System.Text.Encoding.ASCII.GetBytes(some_text), myHash);
 //verified a signature on a given message
 bool verified;

 verified =
 myRSA.VerifyData(System.Text.Encoding.ASCII.GetBytes(some_text), myHash, signature);
```

Figur 2: eksempel på RSA-signering og verifikation i .NET

Strukturen for den offentlige og private nøgle

Strukturen af RSA-nøglen i .NET følger PKCS#1 (Public Key Cryptography Standards) beskrivelse. RSA-skemaet inkluderer nøglens parametre. Struktureren i .NET er en smule anderledes end den generiske beskrivelse i afsnit *Kort teoretisk baggrund*. Følgende parametre er til stede i nøglen:

- Modulus — modulus, dvs. n,
- Exponent — den offentlige eksponent, dvs. e,
- P — den første primære faktor modulus, dvs. p,
- Q — modulens anden primære faktor, dvs. q,
- DP — modulens anden primære faktor, dvs. q,
- DQ — den private eksponentmodul q-1, dvs. d mod q - 1,
- InverseQ — InverseQ - det inverse af q modulo p, dvs. $q - 1$ modp,
- D — D - den private eksponent, dvs. d.

Eksport og import af nøgler som XML-strenge

Nøgler kan eksporteres til XML-strenge med metoderne `ToXMLString(bool includePrivateParameters)`, der tager et boolsk input, der angiver, om den private del af nøglen er eller ikke er inkluderet i det returnerede string. I figur 3 og 4 vises en RSA-nøgle, der er eksporteret fra .NET med og uden den privat del. En nøgle kan også importeres fra en sådan streng via `FromXmlString(streng xmlString)` metoden.

```
<RSAKeyValue>
<Modulus>
uPmqM3pzkazPZAVC0pCA+unlLorxuxcwZb/AwcOE64qAIUZuLjRCKc0HFyJSwp38qw
y2JWNm7vQQmsm9xVECcBTUqTVR17hviNwof6qJ1BlpFbNqS5IXPM1oj2spVKVvaiC
nE+RPegQ2AZACxEOkoGZBxQFupfbbuzuoMNEt3qs=
</Modulus>
<Exponent>
AQAB
</Exponent>
<P>
/BP+eh9ZiAw5PXjniNzEEZ8+5+q12lYQ5peCJDUHNkzA7yyhWo9ayg+ZRt2yJ7tFvgF4t
RLF0nCBrJDQvSWTUw==
</P>
<Q>
u9pn4Ph7MDEAgwSk61VrOe8mH3XW7f54157LwklcBc7tzzB3DHsQ6UEJUfmTTE4ed5
ogX52F7hPVcXW86w40SQ==
</Q>
<DP>
KVBJk9BRhyehtf57zAWKqOy1jaL9HQSgDnrkXHTIctDPiiOBams2UQmPcHrjOPnLa2G
oW9zwyRWhWxv86hMfew==
</DP>
<DQ>
PQoPvPMgnB0gDHKC373XtKB3o7tXlkecia/Ih53sr9p4PV2DIWQPr6s5SxCsgxvTHIvRP
yBhN2XscgyO0VXxOQ==
</DQ>
<InverseQ>
pr2OyNnCyceTOWWPGn3x9yCHyPaAYiHyP/dLFNKqGmgWLkShtBbuVO8t97dtNPNd
sgHeS8mxnpZV0hxoYVJodQ==
</InverseQ>
<D>
qYzv3c/YLydf0iagYbHjCBts34Ssnvlae2mQngtBw0VovRd51xA/tWEhpqrngUyfVYqJSy
waJd3BeqCBOmRO/ipZRd4SXr3HX4vU3qtTwtSOKHMJW8BEn2dwgW3B4xbQkWo+t
i7VJZIxLSMS02IowLs3FfwjXz2ATVx71LqywoE=
</D>
</RSAKeyValue>
```

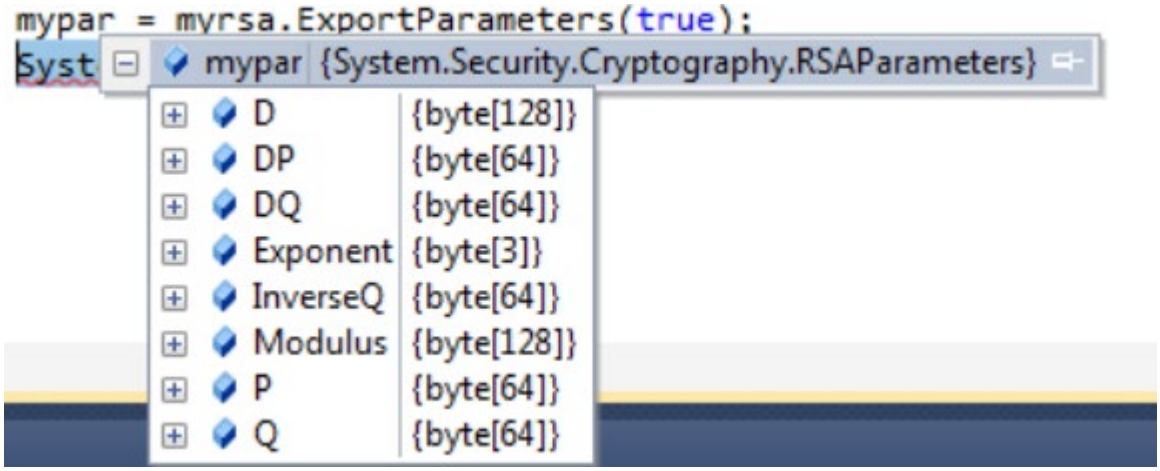
Figur 3:RSA-nøgle eksporteret som XML-streng med private parametre

```
<RSAKeyValue>
<Modulus>uPmqM3pzkazPZAVC0pCA+unlLorxuxcwZb/AwcOE64qAIUZuLjRCKc0HFy
JSwp38qwy2JWNm7vQQmsm9xVECcBTUqTVR17hviNwof6qJ1BlpFbNqS5IXPM1oj2s
pVKVvaiCnE+RPegQ2AZACxEOkoGZBxQFupfbbuzuoMNEt3qs=
</Modulus>
<Exponent>
AQAB
</Exponent>
</RSAKeyValue>
```

Figur 4: RSA-nøgle eksporteret som XML-streng uden private parametre

Eksport og import af nøgler som byte-arrays

Tilsvarende kan nøgler importeres og eksporteres som `System.Security.Cryptography.RSAParameters`, som er en struktur der indeholder et byte-array for hver af de tidligere beskrevne parametre. Denne import/eksport metode er nødvendig, når du vil importere/eksportere nøglen imellem forskellige platforme, f.eks. til en C++ eller Java-implementering. Figur 5 viser et "skærm capture" fra .NET-miljøet, hvor strukturen på en nøgle kan ses



Figur 5: felter med en RSAParameters struktur

Øvelser RSA

Skriv et program, der skal fungere som sender og modtager af/til RSA-kommunikation.

Modtagerprogrammet skal have disse funktioner:

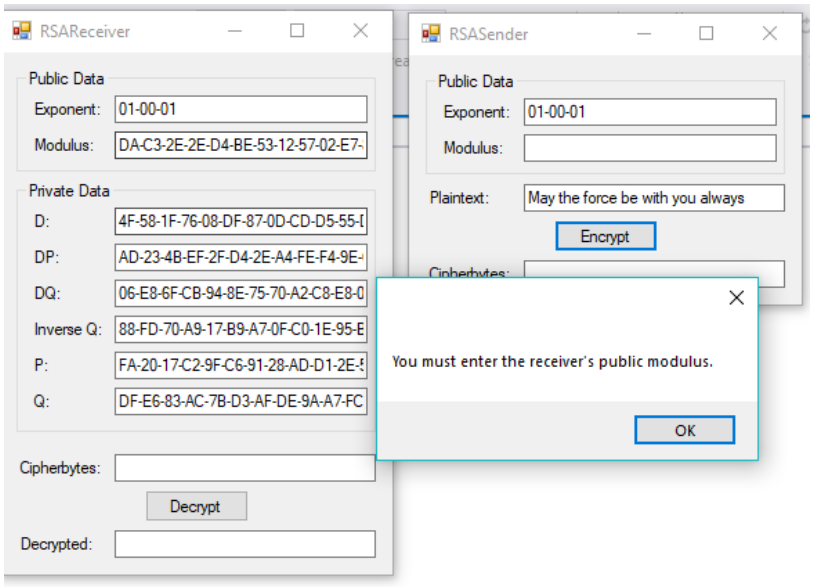
1. Når det starter, skal programmet oprette et `RSACryptoServiceProvide`r objekt med en tilknyttet en nøgle, der er gemt i en key container. (Hvis nøglen allerede findes, indlæses den ind i provider. Hvis nøglen ikke findes, opretter provider en nøgle og gemmer den i containeren).
2. Når der er oprettet provider, skal programmet vise de offentlige Exponent and Modulus i TextBoxes eller på konsolen. (Du kan også få din applikationen til at vise de private data, hvis du vil).
3. Brugeren skal indtaste krypterede data i det format, der bruges af `BitConverter.ToString` metode (som F9-9A-98-6F-BC-9F).
4. Når brugeren klikker på knappen Dekrypter, skal programmet konvertere det indtastede data til et `byte`-array og brug den private nøgledata til at dekryptere dem.

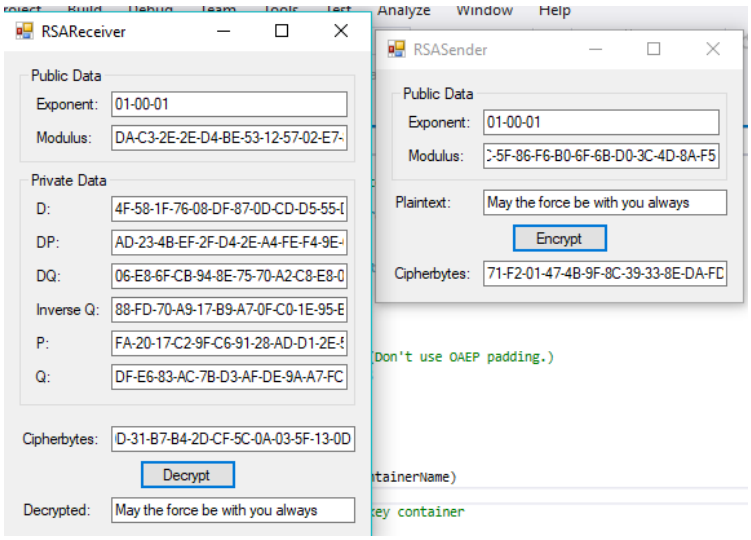
Afsenderprogrammet skal have disse funktioner:

1. Brugeren skal kopiere og indsætte de offentlige nøgledata fra modtagerprogrammet ind i `Textboxes` eller i konsolen i afsenderendelen. (afsenderen skal ikke kende de offentlige nøgledata).
2. Når brugeren indtaster en meddelelse og klikker på Krypter, skal programmet benytte data om privat nøgle for at kryptere meddelelsen og vise resultatet. (Husk at bruge en kort besked, fordi RSA- provider ikke kan krypterer lange.)

Sådan testes programmerne:

1. Afvikle begge programmer.
2. Kopier og indsæt de offentlige nøgledata fra modtageren i afsenderen.
3. Indtast en meddelelse i afsenderen, og krypter den.
4. Kopier og indsæt de krypterede data fra afsenderen til modtageren.
5. Brug modtageren til at dekryptere meddelelsen og kontrollere, at den er korrekt.





Øvelse tid

Evaluer beregningsomkostningerne i tid ved RSA-kryptosystem i .NET med hensyn til: nøglegenerering, kryptering, dekryptering, signering og verifikationstid. Resultaterne skal præsenteret i tabelform som vist nedenfor.

Tidomkostning ved generering af RSA-nøgler

1024 bit	2048 bit	3072 bit	4096 bit

Tidomkostning ved RSA-kryptering

1024 bit	2048 bit	3072 bit	4096 bit

Tidomkostning ved RSA-dekryptering

1024 bit	2048 bit	3072 bit	4096 bit

Tidomkostning ved RSA signing (frivillig)

1024 bit	2048 bit	3072 bit	4096 bit

[i] https://da.wikipedia.org/wiki/RSA