Kryptering

- **Kryptering** är att göra information svårläslig för alla som inte ska kunna läsa den.
- För att göra informationen läslig igen krävs **dekryptering**.
- Kryptering består av två delar, en *algoritm* och en *nyckel*. När man ska kryptera information använder man både algoritm och en nyckel. Nyckeln är hemlig och algoritmen är publik.
- Under dekryptering av informationen måste man ha tillgång till den hemliga nyckeln och algoritmen.

Källa: http://sv.wikipedia.org/wiki/Kryptering

Krypteringsmetoder

1. Symmetrisk kryptering
2. Asymmetrisk kryptering
3. Kryptografiska Hashfunktioner

Symmetrisk kryptering

- **Samma nyckel** används vid kryptering och dekryptering.
- Vid informationsutbyte mellan sändaren och mottagaren måste båda ha tillgång till samma nyckel.
- Fördelen är snabbheten.
- Nackdelen är att flera personer måste ha tillgång till samma nyckel.

Symmetriska krypteringsalgoritmer

- **Triple-DES (3DES)**
  - Säkrare version av krypteringsalgoritmen DES, "Data Encryption Standard".
  - Sändaren och mottagaren använder tre nycklar för att kryptera meddelandet.
  - Nyckelstorlek är 168 bitar.
- **Advanced Encryption Standard (AES)**
  - AES är tillräckligt effektiv i processor och ram för att det skall fungera i mobila telefoner.

Jämförelse mellan DES, 3DES och AES

	DES	3DES	AES
Nyckellängd (bitar)	56	168	128 192 256
Säkerhet	Dålig	Bra	Bra
Processoranvändning	Låg	Hög	Väldigt låg
RAM-minnesbehov	Lågt	Högt	Väldigt lågt

Källa: http://sv.wikipedia.org/wiki/Kryptering

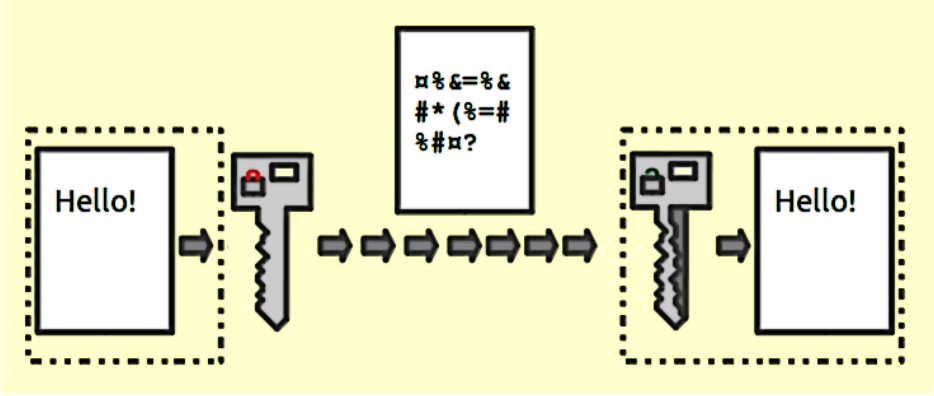
Asymmetrisk kryptering

- Asymmetrisk kryptering innebär att man använder två olika nycklar: en öppen nyckel ("**Public Key**") en egen personlig nyckel "**Private Key**".
- Den öppna nyckeln används för att kryptera informationen. För dekryptering används den personliga nyckeln.
- **RSA**-kryptering är den vanligaste algoritmen för asymmetrisk kryptering. Namnet kommer från upphovsmännen: Ron **R**ivest, Adi **S**hamir och Len **A**dleman.

Säkerhet

- En vanlig nyckellängd för symmetrisk kryptering är 128 bitar, vilket resulterar i att det totala antalet möjliga nycklar är 2^{128}.
- I genomsnitt så måste man pröva hälften av alla nycklar innan man kan hitta den rätta vilket ger 2^{127} nycklar som måste testas innan meddelandet kan läsas.
- Testar man 100 miljarder nycklar per sekund så tar det $5 * 10^{19}$ år innan man hittar den rätta nyckeln.
- I asymmetrisk kryptering använder man sig av nyckelstorlekar som är från 512 upp till 4096 bitar.

Illustration av hur ett digitalt dokument skickas med "Asymmetrisk kryptering"



Källa: http://sv.wikipedia.org/wiki/Kryptering

Kryptografiska Hashfunktioner

- En hashfunktion är en algoritm som gör om någon sorts data till ett heltal (oftast i hex-format) som kan fungera som index till en array.
- Kryptografiska hashfunktioner används exempelvis för att spara lösenord i databaser, då man inte önskar att användarnas lösenord syns i klartext utifall databasen skulle bli stulen.
- Några välkända kryptografiska hashfunktioner är MD5, SHA och PBKDF2 (RFC 2898).

MD5

- MD5 står för **Message-Digest algorithm** 5.
- MD5 är en kryptografisk hashfunktion som skapar ett 128 bitar stort hashvärde av valfri data.
- MD5 används bland annat som checksumma vid vissa typer av filöverföring, digital signering och för lösenordsverifiering.
- MD5 anses idag inte vara en kryptografisk säker funktion

SHA

- SHA står för **Secure Hash Algorithm.**
- SHA är en kryptografisk hash-funktion utvecklad av amerikanska National Security Agency (NSA).
- Det finns idag flera olika SHA-algoritmer SHA-0, SHA-1, SHA-256, SHA-512 osv.
- Läs mer här
  http://en.wikipedia.org/wiki/SHA-3

Att salta

- Ett salt är ett slumpmässigt tillägg till ett lösenord som gör det svårare för en lösenordstjuv att lista ut det riktiga lösenordet.
- En annan fördel är t.ex. om man väljer samma lösenord på två olika konton.
  Utan salt skulle detta lösenord lagras som samma hashsträng, och om man känner till en lösenordshashsträng så kommer man åt det andra kontot.
- Saltning av lösenord med två slumpmässiga tecken gör att, även om de båda kontona använder samma lösenord, ingen kan upptäcka detta genom att läsa hashsträngar.
- Tips: http://sv.wikipedia.org/wiki/Salt_(kryptografi)

PBKDF2 (RFC 2898)

- PBKDF2 står för
  Password-Based Key Derivation Function 2
- Specifikationen om denna krypteringsmetod är publicerad av Internet Engineering Task Force's (RFC 2898)
  http://tools.ietf.org/html/rfc2898
- Algoritmen använder 128-bitars salt och 256-bitars nyckel och 1000 iterationer.
- Exempel på system som använder PBKDF2
  WiFi (WPA och WPA2), Apple's iOS mobile (tex iPhone) och Mac OSX.

Implementera kryptering i ASP.NET med hjälp av klassen Crypto

- **Crypto** är en statisk klass som finns i namnrymden System.Web.Helpers
- Några användbara metoder

Hash()
Metoden hashar en sträng med hjälp av algoritmen SHA-256 (standard) SHA-1 eller MD5.

GenerateSalt()
Generar ett salt som kan läggas till ett lösenord inför hashing.

HashPassword()
Metoden använder algoritmen PBKDF2 (RFC 2898)

bool VerifyHashedPassword(hashedPassword, password)
Returnerar true om det krypterade lösenordet matchar lösenordet i klartext

Klassen Crypto – Exempel
Del 1 HTML

```
<body>
<form method="post">
    <input type="password" name="password" >
    <input type="submit" value="Submit">
</form>
<pre>
    Lösenord       @password
    Salt           @salt
    MD5            @md5
    SHA256         @sha256
    SHA1           @sha1
    HashedPassword @hashedPassword
    Verify         @verify.ToString()
</pre>
</body>
```

Spara i en cshtml-fil

Klassen Crypto – Exempel
Del 2 Razor C#

```
@{
    var password = "";
    var md5 = "";
    var sha256 = "";
    var sha1 = "";
    var salt = "";
    var hashedPassword = "";
    var verify = false;
    if (IsPost) {
       password = Request.Form["password"];
       salt     = Crypto.GenerateSalt();
       // password += salt; // Testa med och utan salt
       sha256   = Crypto.Hash(password);
       md5      = Crypto.Hash(password, "MD5");
       sha1     = Crypto.Hash(password, "SHA1");
       hashedPassword = Crypto.HashPassword(password);
       verify =
        Crypto.VerifyHashedPassword(hashedPassword, password);
    }
}
```

Once hashed, it is impossible to convert back!

