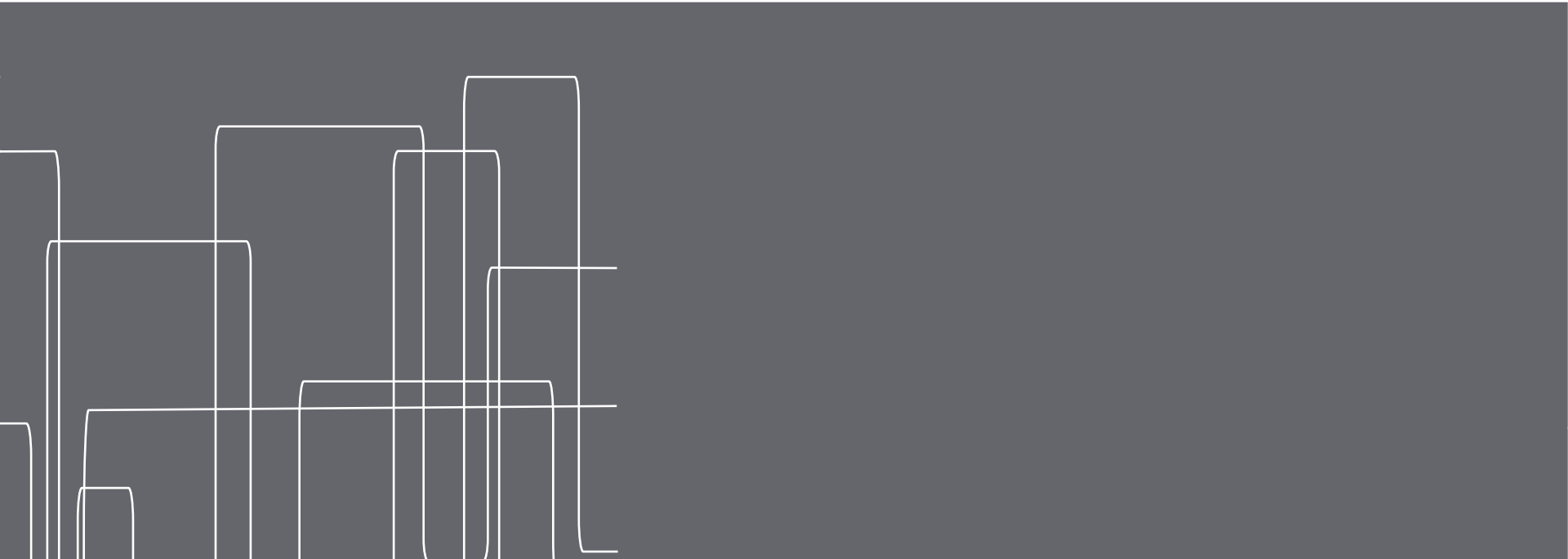
Objektorienterad Programkonstruktion

Föreläsning 16
6 feb 2017



Kryptering

- För ordentlig behandling rekommenderas kursen **DD2448**, *Kryptografins Grunder*
- Moderna krypton kan delas in i två sorter, baserat på nycklarna:
  - Symmetriska krypton - använder samma nyckel för att kryptera som för att dekryptera.
  - Assymetriska krypton - använder separata nycklar för kryptering och dekryptering.

Symmetriska krypton

- Använder samma nyckel för kryptering och dekryptering
- Fördelar
  - Snabba algoritmer
- Nackdelar
  - Kräver försiktig behandling av nycklar
- Exempel:
  - DES
  - AES
  - Blowfish
  - Twofish

Assymetriska krypton

- Använder ett nyckelpar, med en privat och en publik nyckel
- Bygger på att krypteringsprocessen är en envägsfunktion, dvs det går inte att på ett enkelt sätt köra krypteringen "baklänges" om man bara känner till krypteringsnyckeln
- Fördelar:
  - Lättare att hantera nycklar, man behöver inte hålla krypteringsnyckeln hemlig (kan dock vara svårt att autenticera)
- Nackdelar:
  - Kräver ofta mer beräkningskapacitet för att kryptera och dekryptera

Blockkrypton

- Man tar indata av en fördefinierad storlek och bearbetar dessa oberoende av varandra
- Blockstorleken behöver inte vara lika stor som nyckeln, men de är ofta av samma storleksordning
- Typiskt används samma nyckel till varje block

Strömkrypto

- Kryptot har ett internt tillstånd (state) som uppdateras efter varje krypteringssteg.
- Ofta krypteras väldigt små enheter i taget, t.ex en bit i taget.
- Nyckeln uppdateras för varje steg, t.ex genom att använda den gamla nyckeln som frö till en pseudoslumpfunktion
- Krypterad data kan bli omöjligt att dekryptera om en del av meddelandet går förlorad

AES (Advanced Encryption Standard)

- Amerikanska standardiseringsinstitutet utlyste en tävling för att hitta en ny krypteringsstandard, algoritmen (Rijndael) av Joan Daemen och Vincent Rijmen vann tävlingen och blev standarden
- Grundidén är att man tar ett block (128 bitar) och utför följande:
  - XOR med rundnyckel
  - ersätter alla bytes enligt en lookup-table
  - arrangerar om ordningen av bytes
  - arrangerar om ordningen av bitar
  - XOR med rundnyckel

  } n gånger

  - ersätter bytes enligt lookup table
  - arrangerar om ordningen av bytes
  - XOR med rundnyckel
- En ny rundnyckel genereras för varje block, baserat på huvudnyckeln (128, 192 eller 256 bitar)
- Kryptering och dekryptering går fort, runt 18 cykler per byte.

RSA

- Uppkallad efter Ron Rivest, Adi Shamir och Leonard Adleman
- Assymmetriskt krypto, bygger på envägsfunktioner
  - p och q är primtal, med n=pq
  - $\varphi = (p-1)(q-1)$
  - $d*e = 1 \pmod{\varphi}$
  - => $m^{ed} = m \pmod{n}$
  - $1<e<\varphi$, ofta är e=65537
- Kryptering: kryptot c skapas ur meddelandet m:

  $c = m^e \pmod{n}$
- Dekryptering: meddelandet m återskapas ur c:

  $m = c^d \pmod{n}$

Kryptering i Java

- Java tillhandahåller ett antal relevanta klasser för kryptering i ramverken **JCA** **J**ava **C**ryptography **A**rchitecture (java.security.*), och **JCE** - **J**ava **C**ryptography **E**xtension (javax.crypto.*)
- Klasserna i dessa bibliotek har gränsnitt för att skapa nycklar och kryptera antingen block eller strömmar
- Själva krypteringsalgoritmerna (inkl. Nyckelgeneratorer) finns i s.k providers som kan installeras separat.
- Per default kan man använda providers med stark men begränsad kryptering, vill man ha obegränsad kryptostyrka får man installera dessa providers specifikt.
- De är (i teorin) bara tillgängliga för användare I "tillförlitliga" länder

Kryptonycklar i Java

- Nycklar lagras i objekt som uppfyller gränssnittet `Key` eller `SecretKey`
- Nycklar kan antingen skapas utifrån givna värden, eller utifrån (relativt) säkra pseudoslumpmetoder
- Om man inte har en **bra** anledning att låta bli, bör man låta en betrodd nyckelalgoritm skapa ens nycklar
- De faktiska klasser som innehåller nycklarna är t.ex `SecretKeySpec`, `RSAPublicKeySpec`, mm

Kryptering i Java

- För att kryptera något i Java skapar man ett `Cipher`-objekt
- Detta skapas med en kryptoalgoritm och en nyckel
- Sedan väljer man om man ska kryptera eller dekryptera
- Slutligen anropar man `Cipher.doFinal(byte[])`, där bytearray:en är det meddelande man vill kryptera/avkryptera
- Eftersom `Cipher` arbetar med byte-arrayer måste man själv konvertera sitt meddelande, t.ex med hjälp av `String.getBytes()` och `new String(byte[])`

AES i Java

```
byte[] dataToEncrypt = "Hej".getBytes();
byte[] keyContent;

// Skapa nyckel
KeyGenerator AESgen = KeyGenerator.getInstance("AES");
AESgen.init(128);
SecretKeySpec AESkey = (SecretKeySpec)AESgen.generateKey();
keyContent = AESkey.getEncoded();

// Kryptera
Cipher AEScipher = Cipher.getInstance("AES");
AEScipher.init(Cipher.ENCRYPT_MODE, AESkey);
byte[] cipherData = AEScipher.doFinal(dataToEncrypt);

// Avkryptera
SecretKeySpec decodeKey = new SecretKeySpec(keyContent, "AES");
AEScipher.init(Cipher.DECRYPT_MODE, decodeKey);
byte[] decryptedData = AEScipher.doFinal(cipherData);
System.out.println("Decrypted: " + new String(decryptedData));
```

RSA i Java

```
byte[] dataToEncrypt = "Hej alla glada!".getBytes();
byte[] cipherData;
byte[] decryptedData;

// generera nycklar
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(2048);
KeyPair kp = kpg.genKeyPair();
Key publicKey = kp.getPublic();
Key privateKey = kp.getPrivate();

// kryptera
Cipher cipher = Cipher.getInstance("RSA");
cipher.init(Cipher.ENCRYPT_MODE, publicKey);
cipherData = cipher.doFinal(dataToEncrypt);

// dekryptera
cipher.init(Cipher.DECRYPT_MODE, privateKey);
decryptedData = cipher.doFinal(cipherData);
System.out.println("Decrypted: " + new String(decryptedData));
```

Experiment, 14-16 februari

rkth@kth.se