AES-IMPLEMENTATION (PROJEKT 11)

DA563A - INTRODUKTION TILL DATASÄKERHET, HÖGSKOLAN KRISTIANSTAD
ROBIN HOLM – 920605-2970
07-04-2020

—

ÖVERBLICK

Den här rapporten detaljerar utförandet av *projekt 11: AES implementation*. Instruktionerna var att skriva ett program som utför AES-kryptering och dekryptering med hjälp av ett valfritt programspråk. Som demonstration skall programmet kunna kryptera och dekryptera texten "*introduction to computer security*".

I rapporten kommer jag inledningsvis att presentera den teori och de kunskaper som ligger till grund för projektet. Därefter kommer jag att detaljera genomförandet och diskutera de problem eller intressanta saker jag stötte på. Efter det följer en ingående beskrivning och presentation av programmet, vilket inkluderar skärmdumpar på interaktionen med det grafiska gränssnittet, samt en titt "under huven" där jag visar och förklarar metoderna, variablerna, exekveringsflödet och logiken. Till sist följer en avslutande reflektion över projektet, en kort källförteckning, och den kompletta källkoden.

TEORETISKT UNDERLAG

Här följer en genomgång av teorin bakom de tekniker som implementeras i mitt program. Materialet är huvudsakligen taget ur kapitel 8 i kursboken Introduction to Computer Security (*Goodrich*, *Tamassia; 2014*) men även ur Computer Security: Principles and Practice *(Stalings*, *Brown*, *2014)*.

I sin enklaste form brukar man beskriva *kryptografi* som tekniker eller metoder för att etablera konfidentiell kommunikation över en osäker kanal. Detta sker genom att avsändaren *krypterar* ett meddelande, vilket betyder att hon med hjälp av en algoritm (en serie väldefinierade instruktioner eller steg) konverterar meddelandet till en oläsbar och intetsägande form kallad *ciphertext*. Mottagaren *dekrypterar* denna ciphertext genom att mata in den i samma algoritm "baklänges" så att texten konverteras tillbaka till sin ursprungliga form, s.k. *plaintext*. Kryptografiska algoritmer kallas ofta *ciphers*. Ciphers använder olika metoder för att kryptera text; man brukar skilja på *stream ciphers*, där ett meddelande krypteras en symbol i taget, och *block*

ciphers, där kryptering sker i block om exempelvis 128 bitar. När det gäller metoden för själva konverteringen används olika sorters tekniker, exempelvis *substitution*, där varje bit byts ut mot en annan, och *permutation*, där ordningen bitarna ligger i kastas om. Allt detta sker förstås på ett sådant sätt att bitarna faller tillbaka på sin plats när man kör algoritmen baklänges (dvs när man dekrypterar).

Det som förhindrar att varje individ med tillgång till en cipher kan dekryptera all text som skapades med den ciphern, är *nycklar*. En nyckel är ett numeriskt eller alfabetiskt värde som matas in i en cipher tillsammans med det meddelande som ska krypteras. När meddelandet sedan ska dekrypteras igen behövs samma nyckel för att algoritmen skall kunna återskapa den ursprungliga texten. Om en tjuvlyssnare snappar upp ett meddelande kan han inte dekryptera det om han inte besitter denna nyckel. Den här sortens upplägg kallas ett *symmetric* eller *shared-key cryptosystem*, med vilket alltså menas att både avsändaren och mottagaren besitter samma privata nyckel och att endast denna nyckel kan återskapa korrekt plaintext ur ciphertext.

Det finns ett alternativt upplägg som kallas *public-key cryptosystem*, men vi behöver inte gå in på det här, eftersom det här projektet intresserar sig för *AES-kryptering*. AES eller *Advanced Encryption Standard* är en familj av likartade algoritmer som utgör internationell standard. Familjen består av tre olika block ciphers som alla tar emot 128-bitarsblock och nycklar av längderna 128, 192 eller 256 bitar. Förenklat kan man säga att AES-kryptering går till som så att varje block av plaintext går genom ett visst antal rundor (vilket bestäms av längden på nyckeln) där varje runda består av en substitution, en permutation och andra likartade processer. Kryptering och dekryptering enligt AES är vad det här projektet skall implementera.

GENOMFÖRANDE

Jag har tidigare erfarenhet av Java och valde därför att implementera programmet i detta språk. Jag har i huvudsak använt mig av tre Java-standardbibliotek: *javax.swing* och *java.awt* för det grafiska gränssnittet och de händelsedrivna komponenterna, och *javax.crypto* för de kryptografiska aspekterna. Programmet skrevs med hjälp av den integrerade utvecklingsmiljön *Eclipse*.

Den största svårigheten var att verkligen få grepp om de teoretiska delarna, det vill säga vad kryptografi är och hur de kryptografiska processerna verkligen fungerar. Jag läste om ämnet och bildade en övergripande förståelse av teorin och trodde att detta skulle räcka för att kunna börja skriva programkod. Det är ju trots allt så att kryptografiska program är “enkla” i den mening att programlogiken inte är komplicerad och att relativt få rader kod räcker för att få till ett komplett program. Men jag märkte snabbt att jag inte förstod Javas kryptografiska bibliotek, inte förstod de olika exempel jag hittade på internet, och inte visste ens i vilken ände jag skulle

börja med mitt eget kodande. Det blev tydligt att jag måste få ett mycket mer omfattande och detaljerat grepp om ciphers, nycklar, symmetrisk och asymmetrisk kryptering, m.m., innan jag kunde påbörja det praktiska arbetet.

När mina teoretiska kunskaper satt i ryggmärgen påbörjades det praktiska genomförandet. På vägen märktes även att jag behövde förbättra vissa Java-relaterade kunskaper såsom hur bytes, arrays och ActionListener fungerar. Men det praktiska genomförandet gick mestadels snabbt och enkelt och programmet färdigställdes utan större bakslag.

RESULTAT

Utseende och användning

Användaren interagerar med programmet/applikationen på följande sätt: han skriver in en godtycklig text i ett fält (där “introduction to computer security” står som förinställt förslag), och trycker på knappen "Encrypt text". Då krypteras texten till ciphertext och den dyker upp i textfältet nedanför. Trycker man på knappen "Decrypt text" så dekrypteras texten tillbaka till plaintext. Diverse logik skyddar applikationen från att användas inkorrekt: du kan inte kryptera två gånger i rad, du kan inte kryptera ciphertext eller dekryptera plaintext, etc. Om någonting går fel såsom att nyckeln eller ciphern inte går att instansiera, finns exceptions som fångar upp exekveringsflödet så att programmet inte kraschar. Så här ser det ut att använda programmet:

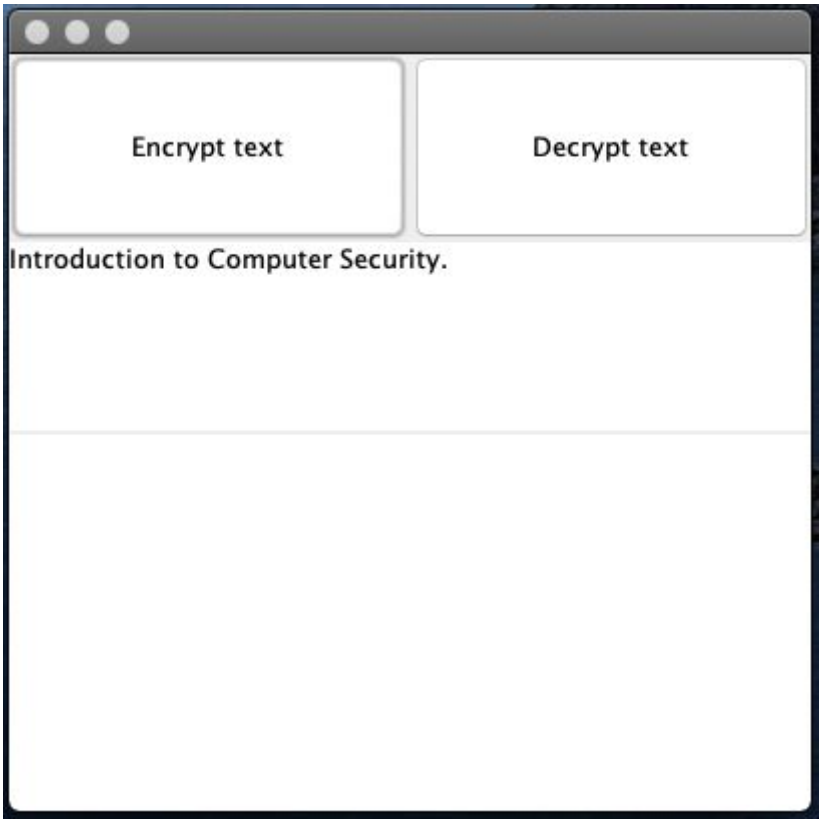

Bild 1: applikationens utseende vid start.

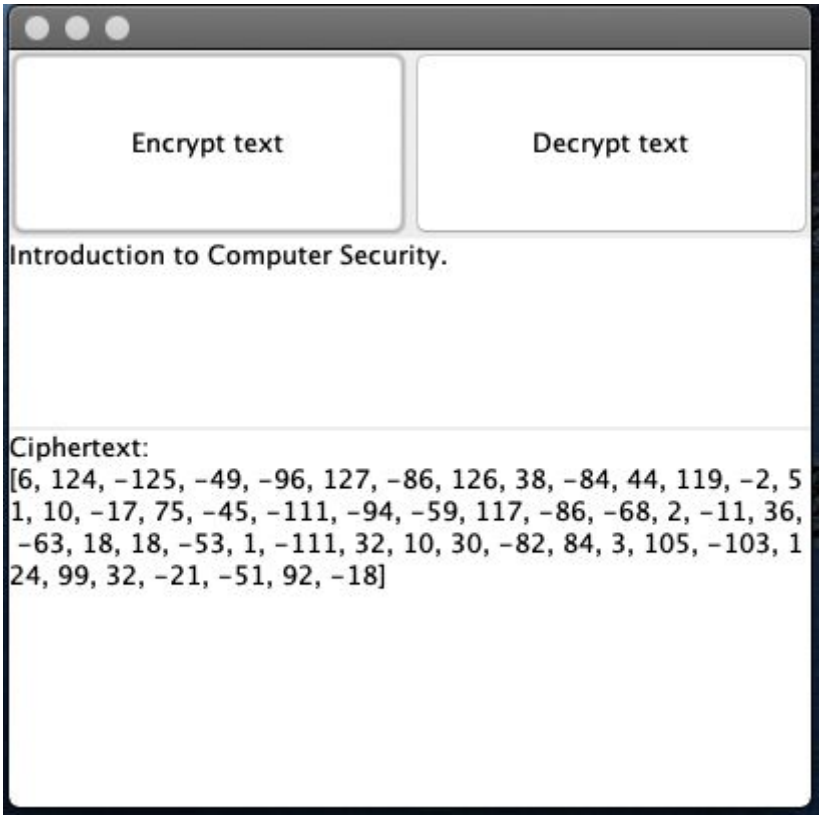

Bild 2: användaren klickar på Encrypt text och ciphertexten presenteras i det nedre textfältet.

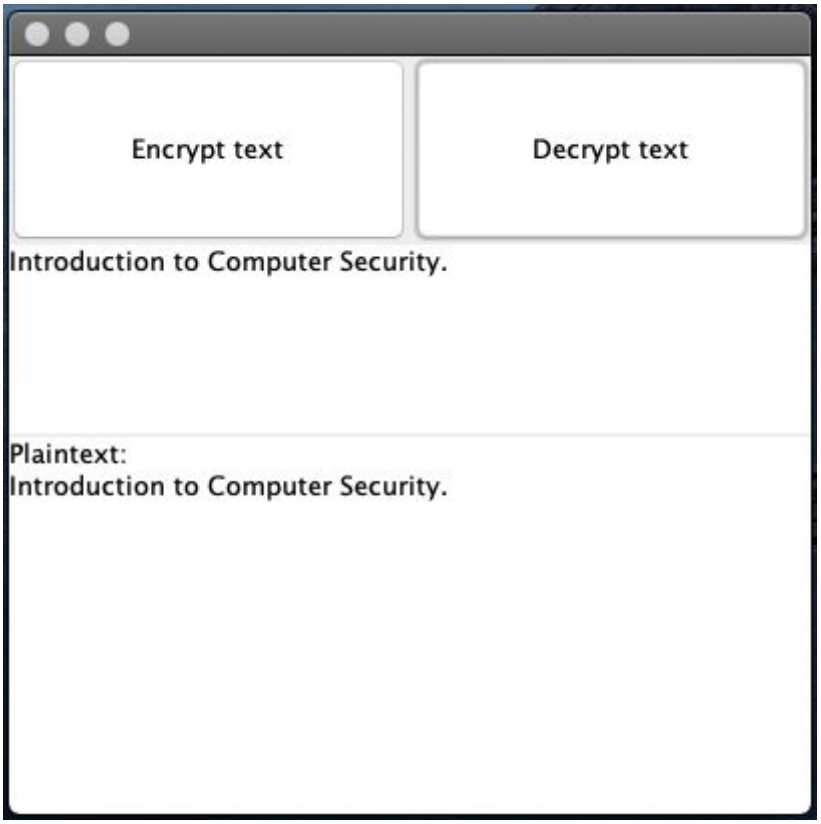

Bild 3: användaren klickar på Decrypt text och den nya plaintexten hamnar i samma fält.

Metoder och programflöde

Mitt program består av en Java-klass kallad *AESimplementation*, som ärver från swing-klassen *JFrame* och implementerar awt-interfacet *ActionListener*. Förutom main-metoden och konstruktorn innehåller klassen metoden *actionPerformed*(), som utför mestadelen av den kryptografiska logiken. Programflödet är händelsestyrt och är i korthet utformat så här:

1. Main-metoden exekveras. Den anropar konstruktorn.
2. Konstruktorn exekveras. Den genererar ett swing-baserat grafiskt gränssnitt och skapar därefter en AES-associerad privat nyckel. Efter detta är "setup-fasen" över och programflödet stannar i inväntan på användarens input.
3. Användaren klickar på någon av knapparna. Detta anropar ActionListener-metoden *actionPerformed*(), som exekverar följande:
    a. Skapar en AES-cipher.
    b. Om encrypt-knappen trycks, instansieras ciphern i krypteringsläge och verkställer krypteringen genom att anropa doFinal(). Resultatet placeras i variabeln ciphertext och presenteras i GUIt.
    c. Om decrypt-knappen trycks kommer istället ciphern initieras i dekrypteringsläge, krypteringen verkställs på ciphertexten och ny plaintext visas i GUIt.

Beskrivning av kryptografin

Här följer en mer genomgående förklaring av hur koden utför de specifikt kryptografiska processerna:

```
key = KeyGenerator.getInstance("AES").generateKey();
```

KeyGenerator-objektets metod *getInstance*() hämtar en instans av en nyckelgenerator som kan generera en AES-associerad nyckel. Metoden *generate.Key*() anropas på resultatet av den komputationen vilket gör att en AES-nyckel skapas. Den placeras i variabeln *key*, som är av objekttypen *SecretKey*.

```
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

En variabel av typen *Cipher* deklareras, och tilldelas resultatet av Cipher-objektets metod *getInstance*(). Den metoden hämtar en cipher-instans med följande egenskaper: den använder AES-standarden, den använder *Electronic Code Book*-läget, och den använder *PKCS5*-padding för att fylla ut änden av plaintexten för att garantera 128-bitarsblock.

```
cipher.init(Cipher.ENCRYPT_MODE, key);
ciphertext = cipher.doFinal(input.getText().getBytes());
```

Ciphern initieras i krypteringsläge och matas med den privata nyckeln. *DoFinal*() tar därefter användarens byte-omvandlade input och utför krypteringen. Resultatet hamnar i variabeln *ciphertext*.

SAMMANFATTNING OCH SLUTSATS

Programmet fungerar som det ska och utför sin funktion utan problem och i princip utan att något kan gå fel. Programmet är ur viss synpunkt "onödig/meningslös" eftersom den endast krypterar och dekrypterar inuti applikationens isolerade miljö. Syftet med en kryptografisk lösning är att erbjuda säker kommunikation mellan två parter över en osäker kanal, och i det här projektet finns så att säga ingen kanal och inte heller två parter som kommunicerar.

Men hur som helst har projektet varit mycket intressant att genomföra. I det abstrakta verkar kryptografi oerhört komplicerat och det är lätt att bli överväldigad av den stora mängden termer i stil med s*ubstitution-permutation network* eller *asymmetric key algorithm*. Jag föreställde mig att "riktig" kryptering krävde hundratals rader kod och en masterexamen i datavetenskap men det behövdes visst bara en handfull metoder och objekt ur ett redan färdigt klassbibliotek samt halvdugliga Java-kunskaper. Det är lärorikt, intressant och motiverande att själv utföra kryptering snarare än att bara läsa om det i en bok eller wikipedia-artikel.

REFERENSER

Goodrich, M., Tamassia, R. (2014). *Introduction to Computer Security*. Pearson New International Edition.

Stallings, W., Brown, L. (2014). *Computer Security: Principles and practice*. Third Edition. Harlow: Pearson Education.

KÄLLKOD

```java
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Arrays;
import javax.crypto.*;
import javax.swing.*;

public class AESimplementation extends JFrame implements ActionListener {

	JButton encryptBtn, decryptBtn;
	JTextArea output, input;
	SecretKey key;
	byte[] ciphertext;

	// Main-metoden. Instansierar konstruktorn.
	public static void main(String[] args) {
		new AESimplementation();
	}

	// Klassens konstruktor. Genererar GUIt och skapar en AES-nyckel.
	public AESimplementation() {
		JPanel upanel = new JPanel(new GridLayout(1, 2, 2, 2));
		JPanel dpanel = new JPanel(new GridLayout(2, 1));
		encryptBtn = new JButton("Encrypt text");
		decryptBtn = new JButton("Decrypt text");
		output = new JTextArea();
		input = new JTextArea("Introduction to Computer Security.");
		encryptBtn.addActionListener(this);
		decryptBtn.addActionListener(this);
		input.setLineWrap(true);
		output.setLineWrap(true);
		this.setLayout(new GridLayout(2, 1, 2, 2));
		upanel.add(encryptBtn);
		upanel.add(decryptBtn);
		dpanel.add(upanel);
		dpanel.add(input);
		this.add(dpanel);
		this.add(output);
		this.setSize(400, 400);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setVisible(true);
		try {key = KeyGenerator.getInstance("AES").generateKey();}
		catch (Exception e) {output.setText("Could not generate key!");}
	}

	// Metod som anropas när användaren trycker på en knapp. Den skapar först en AES-cipher.
	// Om encryptBtn trycks, initieras ciphern i krypteringsläge, användarinput krypteras via
	// doFinal() och placeras i variabeln ciphertext, som sen presenteras i GUIt. Om decrypt
	// trycks initieras ciphern i dekrypteringsläge, texten dekrypteras och presenteras i GUIt.
	public void actionPerformed(ActionEvent e) {
		try {
			Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
			if (e.getSource() == encryptBtn && ciphertext == null) {
				cipher.init(Cipher.ENCRYPT_MODE, key);
				ciphertext = cipher.doFinal(input.getText().getBytes());
				output.setText("Ciphertext:" +"\n"+ Arrays.toString(ciphertext));
			}
			else if (e.getSource() == decryptBtn && ciphertext != null) {
				cipher.init(Cipher.DECRYPT_MODE, key);
				output.setText("Plaintext:" + "\n"
				+ new String(cipher.doFinal(ciphertext)));
				ciphertext = null;
			}
		} catch (Exception e1){output.setText("Could not perform encryption/decryption!");}
	}
}
```