Coursework for CSC3621 Cryptography

This is the second of the three parts of the coursework for CSC 3621 Cryptography. It consists of the following three exercises with each exercise 10 marks.

Submission instruction

- Submit all three exercises together, as a zipped archive (including text files for the reports and Java source files for the programs)
- **Deadline:** Friday, 17:00, 21 Nov, 2013

Exercise II-1 (10 marks)

Aim: To understand basic number theory and the Extended Euclidian Algorithm and apply these methods to solve linear equations in Z_N.

As a side aim, you will become familiar with Java BigInteger.

Background information: The material in this coursework was covered in **Lecture 7**, **Number Theory I**. Victor Shoup's book Computational Introduction to Number Theory and Algebra contains further information. http://shoup.net/ntb/ntb-v2.pdf. Chapters 1, 2-2.5 on Z_N and $(Z_N)^*$. Chapter 4 on Euclid's Algorithm.

The coursework draws upon the Java class **BigInteger** to do modular computations.

Instruction: Computation of the Extended Euclidian Algorithm

In this part, you are to implement the Extended Euclidian Algorithm yourself, an important tool for computation in Z_N. **Hint 1**: The Extended Euclidian Algorithm is well documented in a series of sources. **Hint 2:** Try the algorithm first with pen&paper and a small example.

Question 1

Develop an algorithm in Java, under use of the **BigInteger** class to compute the Extended Euclidian Algorithm. In Part I, you are **not** allowed to use the **BigInteger.gcd**() function. The required function has the following form:

Inputs: **BigInteger** x and y

Outputs: three **BigInteger** d, s, t, such that $d =$ **gcd**(x, y) and $xs + yt = d$.

Compute the Extended Euclidian algorithm for the following inputs and provide (d, s, t) in decimal system format in a text file with one line per number.

Inputs: x = 1572855870797393 y = 630065648824575

Outputs: d = … s = …. t = …

Question 2

Explain the principle of the Extended Euclidian Algorithm in a **concise** paragraph. Why does it work?

Question 3

For which purposes can we use the Extended Euclidian Algorithm? Name three uses.

Submission Guideline

Write the answers to the Question 2 and 3 in a short report (**text file**). Attach the **Java source code** for Question 1 in the submission. Attach the **text files** with the computation results for Question 1.

The submission is on NESS.

Marking:

10 marks in total	
Question 1:	5 marks
Question 2:	3 marks
Question 3:	2 marks

Exercise II-2 (10 marks)

Aim: In this part, you will practice how to solve linear equations in Z_N. They are of the following form:

Given $a\,x + b = 0$ in Z_N

Solve for x

Background information: The material in this coursework is covered in **Lecture 7**, **Number Theory I** and **Lecture 9, Number Theory II**. Victor Shoup's book Computational Introduction to Number Theory and Algebra contains further information. http://shoup.net/ntb/ntb-v2.pdf. Chapters 1, 2-2.5 on Z_N and $(Z_N)^*$. Chapter 4 on Euclid's Algorithm.

The coursework draws upon the Java class **BigInteger** to do modular computations.

Instruction: Linear Equations

Question 1

Write a linear equation solver for Z_N. It has the following function interface:

Inputs: **BigIntegers** a and b, N

Output: **BigInteger** x, such that the linear equation is fulfilled, **null** if it's unsolvable.

Question 2a

Solve the following linear equations Z_N with your algorithm, with the parameters:
N = 643808006803554439230129854961492699151386107534013432918073439524138264842370630061369715394739134090922937332590384720397133335969549256322620979036686633213903952966175107096769180017646161851573147596390153

a = 34325464564574564564768795534569998743457687643234566579654234676796634378768434237897634345765879087764242354365767869780876543424

b = 45292384209127917243621242398573220935835723464332452353464376432246757234546765745246457656354765878442547568543334677652352657235

Question 2b

Solve the following linear equations Z_N with your algorithm, with the parameters:
N = 34248723532593458235023583785345602939423526832829428589598243238758257023423
84876259232895263823795235732659632932938392985950720935732042930927056234852738
93582930285732889238492377364284728834632342522323422

a = 34325464564574564564768795534569998743457687643234566579654234676796634378768
434237897634345765879087764242354365767869780876543424

b = 24243252873562935279236582385723952735639239823957923562835832582635283562852
2525252568829092859592384209402572952653298200353246 46

Question 3

Make observations: What special does the number N in Questions 2a and 2b have? What does this mean for the linear equations in Z_N? Give the reason for the result of Question 2b.

Compare how much time it takes to compute the step related to integer **a** with the BigInteger library method as well as the method we introduced in Number Theory II with Fermat's Theorem. Write a concise report on your findings.

Submission Guideline

Write the answers to the Question 3 in a short report (**text file**). Attach the **Java source code** in the submission. Attach the **text files** with the computation results for Question 2a and 2b.

The submission is on NESS.

Marking:

10 marks in total	
Question 1:	5 marks
Question 2:	2 marks
Question 3:	3 marks

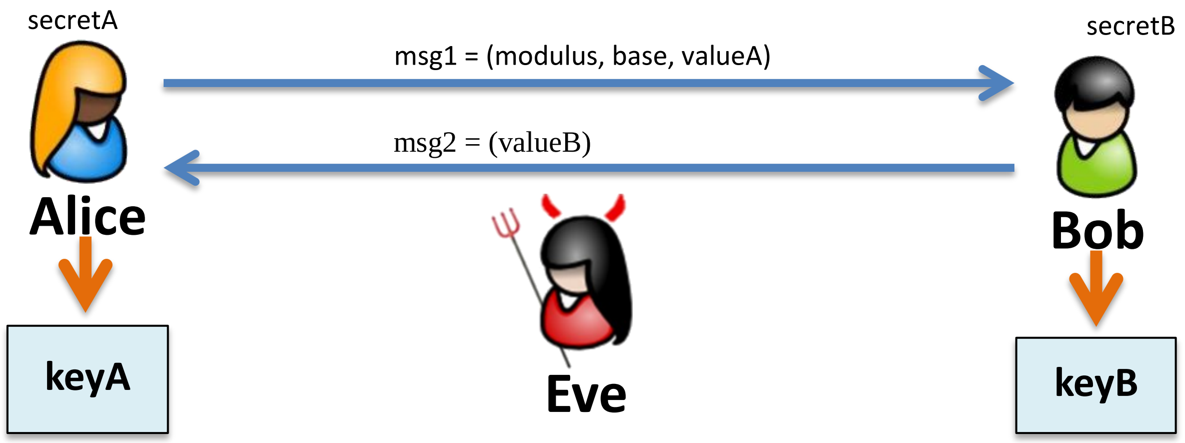
Exercise II-3 (10 marks)

Aim: To understand a major method of key exchange and how to attack it.

Background information: The material in this coursework is covered in **Lecture 8**, **Key Exchange**. There's a nice Youtube video http://www.youtube.com/watch?v=YEBfamv-_do on DH key exchange. Have a look at it! The coursework draws upon the Java class **BigInteger** to do modular computations.

Instruction: Establish Diffie-Hellman Key Exchange

In this part, you are to create the Diffie-Hellman Key Exchange with the following structure:



Question 1

Develop the Diffie-Hellman Key Exchange in Java, using the **BigInteger** class for the computations. The length of the modulus shall always be 1024 bits.

Observe: You do not need to create a strong setup for Diffie-Hellman, i.e., generating a generator *g* with large prime-order *q*. It is sufficient to solve the exercise conceptually and just choose a random generator *g* from {2, …, *p*-1}.

ComputeMessageAtoB (Store the state needed for **ComputeKeyA**):

Input: none

Output: *msg1* = (**BigInteger** *modulus*, **BigInteger** *base*, **BigInteger** *valueA*)

ComputeMessageBtoA (Store the state needed for **ComputeKeyB**):

Input: *msg1* (as produced by **ComputeMessageAtoB**)

Output: *msg2* = (**BigInteger** *valueB*)

ComputeKeyA (based on the state after **ComputeMessageAtoB**):

Input: *msg2* (**BigInteger** *valueB*) **Output:** **BigInteger** keyA.

ComputeKeyB (based on the state after **ComputeMessageBtoA**):

Input: *msg1* (as produced by **ComputeMessageAtoB**) **Output:** **BigInteger** keyB.

Question 2

Run a key exchange with a fellow CSC3621 student (or you playing both sides) and store a transcript of all the data in a text file, one line per datum:

```
secretA = …
secretB = …
msg1.modulus = …
msg1.base = …
msg1.valueA = …
msg2.valueB = …
keyA = …
keyB = …
```

Question 3

How can one use the resulting session keys *keyA* and *keyB* to create a secure channel for further communication between Alice and Bob? Explain what protection measures you need to set up and how to get the inputs for those.

Part II: Attacking Diffie-Hellman Key Exchange

Now it's time to play Eve's role and attack Diffie-Hellman as an active attacker. How can Eve break the key exchange once she's able to intercept messages between Alice and Bob and send messages of her own? **Hint:** Diffie-Hellman is an unauthenticated key-exchange.

Question 4a

Explain how an active attacker Eve can break the Diffie-Hellman key exchange in a concise paragraph.

Question 4b

Implement Eve's attack on the key exchange based on the message-exchange formats as shown in Part I (*msg1* and *msg2*) without having access to the secrets of Alice and Bob. Run the attack (either with other students or playing all three sides) and include a transcript of the completed attack output in the format analogous to the one specified in Question 2.

Submission Guideline

Write the answers to the Questions 3&4 in a short report (**text file**). Attach the **Java source code** for Questions 1 and 5 in the submission. Attach the **text files** with the computation results for Question 2 and 4.

The submission is on NESS.

Marking:

10 marks in total	
Question 1:	4 marks
Question 2:	1 marks
Question 3:	2 marks
Question 4:	3 marks

FAQ for Coursework II

1. Exercise II-1: Output Format for the Extended Euclidian Algorithm

The question asked to provide the BigInteger outputs in decimal system formatting. You need to output the BigInteger (and this can be in difference radices/bases). E.g., you could output the BigInteger as binary string or hexdecimal. What we ask you to do is to output it as a decimal number (i.e., a normal number with base 10).

Thus, if the BigInteger contains `124456216564`,

DO NOT output the hex-value `1CFA2B27F4` or the binary value
`1110011111010001010110010011111110100`

But output the decimal number:

```
124456216564
```

2. Exercise II-1: What to output

There was a question what you need to output when computing the Extended Euclidian Algorithm egcd(x,y).

The algorithm will not only give you the gcd d, but also auxiliary values s and t such that the following equation is fulfilled:

$d = xs + yt$

Output the decimal-system representation of the BigIntegers d, s, t.

The outputs can be copied and pasted. You do not need to write a dedicated file output routine.

2. Exercise II-2: Inputs for the Linear Equation Computation

Erratum: The inputs you receive include BigInteger N, that is, the modulus of the group.

Thus we have: Inputs: BigIntegers *a*,*b* and *N.*

3. Exercise II-3: How to generate a generator for (Z_p)*

(Creating a generator g for all $(Z_p)^*$ is not needed to do the exercise!)

In practice: Creating a generator for all $(\mathbf{Z}_p)^*$, that is, with group order *p*-1 is not an easy task and there is no efficient algorithm to do that well. Victor Shoup's book gives an algorithm how do to that smartly: http://shoup.net/ntb/ntb-v2.pdf, see Chapter 11.1. The basic principle is to generate g together with the prime number *p*.

In the coursework: For this coursework it is full sufficient to work with *any* generator for a subgroup of $(Z_p)^*$. This means you can pick a random element from $(Z_p)^*$ and treat it as a generator. This is not a secure way to do it, because in a real application you would want the generator g to have an order equal to a large prime *q*.

4. Exercise II-3: What should the size of the Diffie-Hellman exponent be?

In practice: If one executes Diffie-Hellman in practice one computes the group q order of the generator g used in the computations and just chooses an exponent uniformly at random from {1, …, q-1}, that is from $(\mathbf{Z}_q)^*$.

In the coursework: In the coursework we largely ignore the problem of the group order of the generator as a simplification. For your exercise it is fully sufficient to choose the exponent as random number from {1,…, p-1}.