

Course Project Descriptions

Xinxin Fan and Guang Gong

Student evaluations in E&CE 493 will be based in part on an individual based course project. The project involves a typed report on a topic of your choice, due on **Thursday, March 26, 2009**. You may choose one of the following two topics.

Topic 1: DSS-Based Mini-Certificate Generation [10 points]

You have already learned the Digital Signature Standard (DSS) from your lecture and played some toy example in your assignment. It is the time for us to do more! In this project, we will implement a simplified certification authority (CA) which is in charge of issuing mini-certificates for users based on the DSS. To work with DSS we first need to generate a set of system-wide parameters satisfying the following three criteria:

- a prime number p , which length is a multiple of 64 bits lying between 512 and 1024 bits
- a 160-bit prime factor q of $p - 1$
- an element $g \in \mathbb{F}_p$ with order q

For your convenience, you can work with the following set of system parameters:

```

p = 168199388701209853920129085113302407023173962717160229197318545484
    823101018386724351964316301278642143567435810448472465887143222934
    545154943005714265124445244247988777471773193847131514083030740407
    543233616696550197643519458134465700691569680905568000063025830089
    599260400096259430726498683087138415465107499 (1024 bits)
q = 959452661475451209325433595634941112150003865821 (160 bits)
g = 943891927763273985898453269803498145264338690934127823454309460592
    065688040051816008558259061429672718725483758777389498758125404332
    234449684613507894613850437750299639006381231834351335372621529733
    554984329953645051389125697558596236498663751353531793626707987717
    70711847430626954864269888988371113567502852 (1024 bits)
    
```

Moreover, you can use the following key pair (Sk_{CA}, Pk_{CA}) as the CA's private key and public key:

```

Sk_CA = 432398415306986194693973996870836079581453988813 (159 bits)
Pk_CA = 493360183248080935347335488404117524857260585278296306689674805688
    547564165674962162949190519101486861866227068697023216644650947032
    473686465068210152903024809904501302806169292269172462551470632923
    017242976806834012586361821855991241311700775484507542940837288850
    75516985144944984920010138492897272069257160 (1023 bits)
    
```

We assume that the system parameters and the CA's public key are in the public domain and therefore they are known by all users. We also assume that a user's real identity has been authenticated through an out-of-band channel (for example, a user might show his driver licence for registration with the CA). Our task is to ask the CA to generate a mini-certificate for a user using DSS when he/she submits him/her identity and public key, as shown in Figure 1.

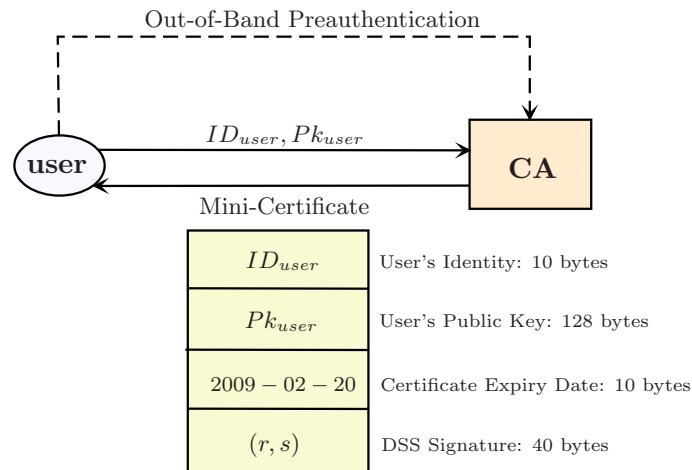


Figure 1: Mini-Certificate Request Procedure

In the above application scenario, the CA and the user act as a server and a client, respectively. The user first submits his identity (maximum 10 bytes) and self-generated public key (128 bytes) to the CA. Then the CA determines the expiry date (10 bytes) of the mini-certificate and signs the data field with DSS (see Figure 1). Finally, the CA returns the generated mini-certificate to the user and the user will check the validity of the mini-certificate by verifying the CA's signature. For this course project, you need to finish the following tasks:

1. Check that the system parameters p, q and g satisfy the aforementioned three criteria and the CA's key pair is valid. [1 point]
2. Implement a module which enables the CA to generate a mini-certificate (see Figure 1) for a user using DSS after it obtains the user's identity and public key. [4 points]
3. Implement a module which enables a user to verify the validity of the mini-certificate issued by the CA. [4 points]
4. Implement the communication between the CA and the user. More specifically, the CA starts out by listening for a connection on a designated port and the user starts out by talking to the CA and sending his/her identity and public key. After the mini-certificate is generated, the CA will send the certificate back to the user. [1 point]

You need to submit a course project report which explains each step of your implementation in detail. You also need to attach the source code and output results of your program.

[Some Hints]: For the first three tasks, you may use the BigInteger class¹ in java.math package to do all the big integer operations involving in DSS. For generating the message digest in DSS with the hash function SHA-1, you can directly use the MessageDigest class² in java.security package. For the last task, to get two computers to talk to each other, you may use the two classes from the java.net package: Socket and ServerSocket.

Here is a simple demo I have already implemented. You can refer to the following screenshots to get an idea of what the output results look like in the CA side and the user side. If you like doing programming very much, you may also implement a beautiful graphical user interface (GUI) for this project by yourself.

In CA side:

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

G:\Documents and Settings\x5fan>D:

D:\>cd eclipse

D:\eclipse>cd project

D:\eclipse\Project>CA
Wait for the request of a user.....
Enter the expiry date of the certificate (yyyy-mm-dd):
2009-02-20
The user Alice with IP address /127.0.0.1 holds the following mini-certificate:
Alice1432266378816859441183701362747448694211398454845464694633820430320666983250389476928
514298961928011974922307808572278038455397318683053960905375180644059834659089539639635671
288678051130063374947617873111485901846950025284307655324072279782303331057101299702648742
841356415322423194355629880268802037800985432009-02-20
168441743813513900316314396421420913119675558475
394476306072434593691245302277928872326001537986

D:\eclipse\Project>_
    
```

In user side:

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

G:\Documents and Settings\x5fan>D:

D:\>cd eclipse

D:\eclipse>cd project

D:\eclipse\Project>user
Enter the user's identity:
Alice
Enter the IP address of CA:
127.0.0.1
Accept the CA's signature and the mini-certificate is valid!
The mini-certificate issued by the CA is:
Alice1432266378816859441183701362747448694211398454845464694633820430320666983250389476928
514298961928011974922307808572278038455397318683053960905375180644059834659089539639635671
288678051130063374947617873111485901846950025284307655324072279782303331057101299702648742
841356415322423194355629880268802037800985432009-02-20
168441743813513900316314396421420913119675558475
394476306072434593691245302277928872326001537986

D:\eclipse\Project>_
    
```

¹see <http://java.sun.com/j2se/1.5.0/docs/api/java/math/BigInteger.html>

²see <http://java.sun.com/j2se/1.5.0/docs/api/java/security/MessageDigest.html>

Topic 2: Protocol Operations and Messages in IKE [10 points]

IKEv2, shortened as IKE below, performs mutual authentication and key establishment between two parties. Four main exchanges are defined for IKEv2 as follows.

1. IKE SA INIT: Negotiate parameters for IKE SAs;
2. IKE AUTH: Transmit identities and prove knowledge of authentication credits corresponding to the identities;
3. CREATE CHILD SA: Create CHILD SAs for ESP, AH, or both; and
4. INFORMATIONAL: Delete an SA, report error condition, or pass other housekeeping information.

The first two exchanges are called the *main mode* of IKE. In the following, we assume that both Initiator and Responder request the other party's public-key certificate, and we simplify the operations in the main mode of IKE as follows.

Stage 1. IKE SA INIT

<i>Initiator</i>	<i>Responder</i>
$SA_i, KE_i = g^i, N_i \longrightarrow$	$\longleftarrow SA_r, KE_r = g^r, N_r, CERTREQ$

where $SA_i = SA_r = \{(g, q, GF(p)), PRF, MAC, Enc\}$.

Both Initiator and Responder compute:

1. $SKEYSEED = PRF(g^{ir}, N_i || N_r)$.
2. $SK = KDF(SKEYSEED, N_i || N_r) = SK_d || SK_{ai} || SK_{ar} || SK_{ei} || SK_{er} || SK_{pi} || SK_{pr}$, where

$$KDF(K, a, b, c) = PRF(K, a, b, c)$$

Stage 2. IKE Auth

<i>Initiator</i>	<i>Responder</i>
$C_i =$ $Enc_{SK_{ei}}(ID_i, CERT_i, CERTREQ, AUTH_i, \rho_i)$	$C_r =$ $Enc_{SK_{er}}(ID_r, CERT_r, ID_i, AUTH_r, \rho_r)$

Table 1: Authentication Verifications

	Responder's Operations	Comments
1.	Decrypt C_i .	
2.	Verify $CERT_i = (pk_i, (r_{CA}, s_{CA}))$,	verify the digital signature of CA over pk_i and ID_i .
3.	Verify $Auth_i = (r_i, s_i)$,	verify the digital signature of Initiator over R_i .
4.	Verify ρ_i .	

where $CERT_i$, the certificate issued for Initiator by CA, includes the parameters of the Diffie-Hellman group which CA uses, the certified public-key, and the digital signature of the public-key, version number, expiring date, etc.. Assume that the certificate authority (CA) uses the DSS with the same parameters as the communication parties. Under this assumption,

$$CERT_i = (ID_i, pk_i, Sig_{CA}(ID_i, pk_i)).$$

Similarly,

$$CERT_r = (ID_r, pk_r, Sig_{CA}(ID_r, pk_r)).$$

Note that the private key of the Initiator, denoted as sk_i where sk_i with $1 < sk_i < p - 1$ and $\gcd(sk_i, p - 1) = 1$. The corresponding public-key is given by $pk_i = g^{sk_i}$. In the following, we only list the format of Initiator, while Responder has the same format as Initiator where the subscripts i and r are interchanged. Since the certificate is requested, $AUTH_i$ is given by

$$\begin{aligned} Auth_i &= Sig_i(R_i), \text{ where } R_i = (SA_i, g^i, g^r, N_i, N_r, PRF(SK_{pi}, ID_i)), \text{ and} \\ \rho_i &= MAC(SK_{ai}, R_i, Auth_i). \end{aligned}$$

Responder performs the operations listed in Table 1. Initiator will perform a similar process for decryption and verifications.

We assume the following four crypto primitives and parameters for executing IKE.

1. 8-bit DH group: $p = 239$, $q = 17$, $g = 132$ with order 17. The digital signature scheme is DSS. For convenience, the values of the exponentials g^i modulo p and the inverses of i 's modulo q are listed below.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$g^i \text{ mod } 239$	132	216	71	51	40	22	36	211	128	166	163	6	75	101	187	67

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$i^{-1} \text{ mod } 17$	1	9	6	13	7	3	5	15	2	12	14	10	4	11	8	16

2. **PRF** is implemented by a 5-stage LFSR with the characteristic polynomial $x^5 + x^2 + 1$, for example, $PRF(10000)$ is the output sequence of the LFSR starting at the 10th state with the initial state $(a_0, a_1, a_2, a_3, a_4) = (10000)$. In general, $PRF(a, b, c)$ is computed by first concatenating a, b , and c , secondly, dividing the resulting string as 5-bit words (if the last word is less than 5 bits, then padding 0's), the last, adding all the 5-bit words by bit-wise exclusive-or (i.e., bit-wise addition modulo 2). The key deviation function is defined as

$$KDF(a, b, c) = PRF(a, b, c).$$

3. **MAC** is the same one defined in Exercise 1 in Chapter 3, and Enc is the 4-bit AES defined in the same exercise.

4. The parameters that you may freely choose in the IKE main mode are listed in Tables 2 and 3.

Table 2: CA's Operations

CA	Private Key: $sk_{CA} = 3$
	Public Key: $pk_{CA} = g^{sk_{CA}} = 71$
	$CERT_i = (pk_i, Sig_{CA}(pk_i, ID_i))$
	$CERT_r = (pk_r, Sig_{CA}(pk_r, ID_r))$

Table 3: Parameters of Initiator and Responder

	Initiator	Responder
Identities	$ID_i = 10000$	$ID_r = 00001$
Long term keys certified by CA	Private Key: sk_i Public Key: $pk_i = g^{sk_i}$	Private Key sk_r Public Key: $pk_r = g^{sk_r}$
DH ephemeral keys	$KE_i = g^i$	$KE_r = g^r$
Nonce	N_i	N_r

Simulate IKE, i.e.,

1. Generate the messages exchanged in Stage 1 for both Initiator and Responder. [2 points]
2. Compute the shared keys SKEYSEED and SK from your parameters. [2 points]
3. Compute the messages exchanged in Stage 2, and operations that Responder (or Initiator) performed after this exchange. [4 points]
4. Explain how Responder (or Initiator) authenticates Initiator (or Responder). [2 points]
5. (Optional.) Do you think that the messages $Auth_i$ and ρ_i (or $Auth_r$ and ρ_r) have unnecessary components? If so, what are they? Justify your answer.