CSS 322 – ASSIGNMENT 2 ANSWERS

First name: _____ Last name: _____

ID: _____ Total Marks: _____
out of 80

Due Date: Wednesday 21 February 2007, 9am (you can hand in before the start of the lecture)

I certify that, unless otherwise acknowledged, all work carried out in this assignment is my own.

Sign Name: _____ Date: _____

Instructions

- This is an individual assignment. You are not allowed to work in groups on any part of the assignment. Plagiarism will be penalised. (You must sign the statement at the top of this cover sheet).
- The assignment must be handed in by the due date. Late assignments will receive 0 marks.
- The assignment should be neatly handwritten and/or computer generated (for example, Word).
- You must give your calculations, working out, discussion and design decisions. That is, if you only give a correct answer, but no calculations, then you will not receive full marks (in fact, you will receive very few, maybe 0 marks).
- The assignment should be on A4 sheets, with a single staple in the top-left corner. Please do not use plastic sleeves, folders etc.
- You must attach this Cover Sheet (including name, ID and signature) to the front of your assignment.
- Some of these questions, and the text explaining them are taken from the course textbook by Stallings.

Question 1 [10 marks + 10 bonus]

The toy *tetragraph hash* (*tth*) is a hash function similar in nature to SHA, but operates on letters instead of binary data. The function can be described as follows:

Given a message consisting of a sequence of letters, *tth* produces a hash value consisting of four letters. First, *tth* divides the message into blocks of 16 letters, ignoring spaces, punctuation and capitalisation. If the message length is not divisible by 16, it is padded out with nulls. A four-number running total is maintained that start out with the value (0,0,0,0); this is input to the compression function for processing the first block.

The compression function consists of two rounds:

1. Round 1: Get the next block of text and arrange it as a row-wise 4 x 4 block of text and convert it to numbers (A = 0, B = 1, etc.). For example, for the block ABCDEFGHIJKLMNOP, we have:

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

   Then, add each column mod 26 and add the result to the running total, mod 26. In this example, the running total is (24,2,6,10).

2. Round 2: Using the matrix from round 1, rotate the first row left by 1, second row left by 2, third row left by 3, and reverse the order of the fourth row. In our example:

B	C	D	A
G	H	E	F
L	I	J	K
P	O	N	M

1	2	3	0
6	7	4	5
11	8	9	10
15	14	13	12

Now, add each column mod 26 and add the result to the running total. The new running total is (5,7,9,11).

This running total is now the input into the first round of the compression function for the next block of text. After the final block is processed, convert the final running total to letters. For example, if the message is ABCDEFGHIJKLMNOP, then the hash is FHJL.

a) Calculate the hash function for the 48-letter message: "I leave twenty million dollars to my friendly cousin Bill." Hint: Use a Excel spreadsheet or similar to perform the calculation. [10 marks]

b) To demonstrate a weakness of *tth* (that it is easy to find collisions), find a 48-letter block that produces the same hash as that just derived. Hint: start with all A's and then look at the influence of changing the first 4 or 5 letters. [**Bonus 10 marks**]

Answers

Part (a)

The best approach is to use an Excel spreadsheet so that the spreadsheet will automatically calculate the sums and perform shifts etc. Below is the data from my spreadsheet. The final hash value is BFQG.

Block 1	I	L	E	A
	V	E	T	W
	E	N	T	Y
	M	I	L	L
Round 1	8	11	4	0
	21	4	19	22
	4	13	19	24
	12	8	11	11
Initial Total	0	0	0	0
Sum of rows and total	19	10	1	5
Shift left by 1	11	4	0	8
Shift left by 2	19	22	21	4
Shift left by 3	24	4	13	19
Reverse row	11	11	8	12
Running total	19	10	1	5
Sum of rows and total	6	25	17	22
Block 2	I	O	N	D
	O	L	L	A
	R	S	T	O
	M	Y	F	R
Round 1	8	14	13	3
	14	11	11	0
	17	18	19	14
	12	24	5	17

Running total	6	25	17	22
Sum of rows and total	5	14	13	4
Shift left by 1	14	13	3	8
Shift left by 2	11	0	14	11
Shift left by 3	14	17	18	19
Reverse row	17	5	24	12
Running total	5	14	13	4
Sum of rows and total	9	23	20	2
Block 3	I	E	N	D
	L	Y	C	O
	U	S	I	N
	B	I	L	L
Round 1	8	4	13	3
	11	24	2	14
	20	18	8	13
	1	8	11	11
Running total	9	23	20	2
Sum of rows and total	23	25	2	17
Shift left by 1	4	13	3	8
Shift left by 2	2	14	11	24
Shift left by 3	13	20	18	8
Reverse row	11	11	8	1
Running total	23	25	2	17
Sum of rows and total	1	5	16	6
Final Hash Value	B	F	Q	G

Part (b)

Start with all A's (0's) in your spreadsheet and manipulate the first few letters. It is easy to notice the patter. For example, the table below shows all zeros (A's) except for 4 letters:

0	24	7	6
3	0	0	0
0	0	0	0
0	0	0	0

We notice that 0 + 24 + must equal 1 (B) when mod by 26. And 24 + 7 mod 26 should equal 5 (F). And 6 +7 mod 26 should equal 16 (Q). And 6 is the correct value for G.

The following message produces the same hash value as the "I leave twenty …" message:

AYHG DAAA AAAA AAAA AAAA AAAA AAAA AAAA AAAA AAAA AAAA AAAA

Question 2 [25 marks]

A direct digital signature scheme involves only the source and destination users. The examples of signatures covered in lectures were using direct digital signatures. A problem with these schemes is if the source/sender later wants to deny signing a message by claiming their private key was lost or stolen.

The problems associated with direct digital signatures can be address by using an arbiter. Every signed message from sender X to receiver Y actually goes from sender X to arbiter A and then from arbiter A to receiver Y. That is, the arbiter A also signs the message and checks the origin and content. This can solve the problem of sender X disowning a message (e.g. saying their private key was stolen).

Three arbitrated digital signature techniques available are (when X wants to send a signed message to Y):

Option 1 – Conventional encryption is used; the arbiter sees the message

Message 1 from X to A: $M \,||\, E(K_{xa},[ID_X \,||\, H(M)])$

Message 2 from A to Y: $E(K_{ay}, [ID_X \,||\, M \,||\, E(K_{xa},[ID_X \,||\, H(M)]) \,||\, T])$

Option 2 – Conventional encryption is used; the arbiter does not see the message

Message 1 from X to A: $ID_X \,||\, E(K_{xy}, M) \,||\, E(K_{xa}, [ID_X \,||\, H(E(K_{xy}, M))])$

Message 2 from A to Y: $E(K_{ay}, [ID_X \,||\, E(K_{xy}, M)]) \,||\, E(K_{xa}, [ID_X \,||\, H(E(K_{xy}, M)) \,||\, T])$

Option 3 – Public key encryption; arbiter does not see the message

Message 1 from X to A: $ID_X \,||\, E(PR_X, [ID_X \,||\, E(PU_y, E(PR_x, M))])$

Message 2 from A to Y: $E(PR_a, [ID_X \,||\, E(PU_Y, E(PR_X, M)) \,||\, T])$

Notation:

X = sender	M = message	T = timestamp
Y = recipient	A = arbiter	K_{ab} = Shared secret key between a and b
PU_a = Public key of A		PR_a = Private key of A

In the following questions, it is very important that you clearly describe the procedures (e.g. when you refer to a key, you must say what type and possible whose key). It is recommended that you use the same notation as used above in your answers.

a) Explain how a man-in-the-middle attack on the message between X and A in Option 1 can be detected. State your assumptions about the key and the hash function H(). [5 marks]

b) Explain how the man-in-the-middle attack in part (a) can be successful (that is, undetected by A) if the same assumption hold about the key as in part (a), but the hash function used is that used in Question 1 (*tth* – make note of *tth*'s weakness in Q1(b), even if you do not solve Q1(b)). [5 marks]

c) If a malicious node intercepts the message from A to Y in Option 3, can the malicious node perform a replay attack. If yes, explain how the attack is performed. If no, explain how it is detected. [5 marks]

d) Modify the technique in Option 3 to avoid triple encryption of the entire message. [5 marks]

e) A limitation of direct digital signatures is that a node can "cheat" by saying someone has stolen the key. With the arbitrated schemes above, explain a way that two of the nodes could collude (work together) to cheat the system. Also explain why this is a very unlikely scenario. [5 marks]

Answers

Part (a)

The man-in-the-middle attack involves a malicious node (B) intercepting the message from X to A, modifying the message and then forwarding the modified message to A (with the intention that A will not detect the change).

If B receives M || E(K_{xa},[ID_X || H(M)]) and changes M to M' (but does not change $H(M)$) then when A receives the modified message, A will detect the change, since A will calculate $H(M')$ and discover it is different to $H(M)$.

This assumes that B does not know K_{xa}. If B did know K_{xa} then B could modify M and also send $H(M')$ signed with K_{xa}.

This also assumes that the hash function is weak collision resistant. That is, B cannot find a message M' such that $H(M) = H(M')$. If B could find such a message, then B could send M' and the original $H(M)$ and A could not detect any changes (since A would check $H(M')$ and find it the same as the received $H(M)$).

Part (b)

The hash function in tth is not weak collision resistant. That is, it is easy to find a message M' that has the same hash value as message M. That is, $H(M) = H(M')$. Therefore from the answer for part (a) above, B could modify M to M' and send it with the original signature E(K_{xa},[ID_X || H(M)]) and A could not detect the change. Note that this does not require B to know K_{xa}.

Part (c)

A sends the following to Y: E(PR_a, [ID_X || E(PU_Y, E(PR_X, M)) || T])

A replay attack involves a malicious node B capturing the message and sending it again at a later time. If B sent E(PR_a, [ID_X || E(PU_Y, E(PR_X, M)) || T]) exactly as is at a later time, then Y should detect it as a replay, since the timestamp T would be incorrect (and also the same as a previous message with the same timestamp).

Note that B cannot try to modify the timestamp, since it is encrypted with A's private key, PR_a.

Part (d)

Hash functions can be used, instead of encrypting the entire message. For example,

ID_X || E(PR_X, [ID_X || E(PU_y, E(PR_x, M))])

could become

ID_X || E(PR_X, [ID_X || E(PU_y, E(PR_x, H(M)) || M)])

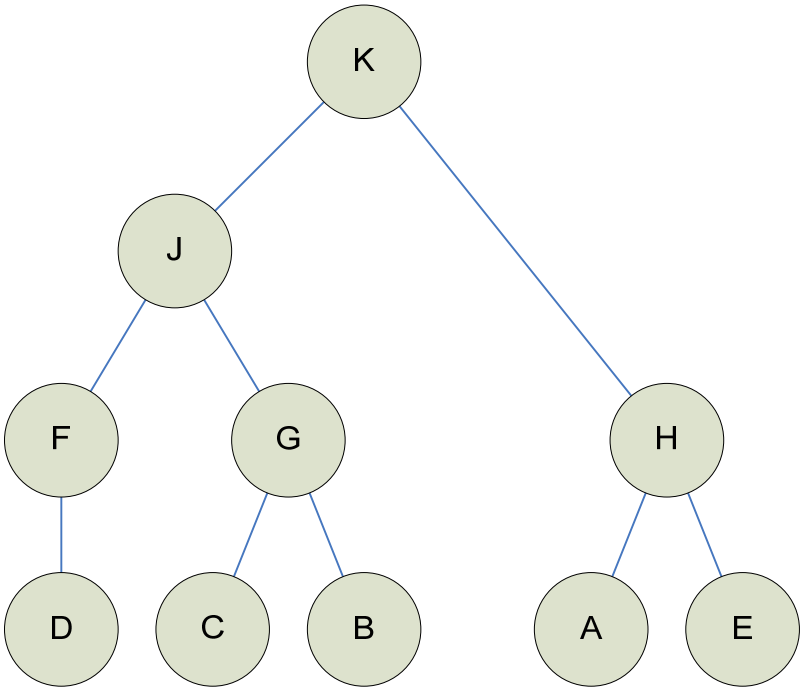
Here, X signs a hash of the message, instead of the entire message.

Part (e)

X and A, wanting to commit fraud, could disclose PR_x and PR_a, respectively, and claim that these were lost or stolen. The possibility of both private keys becoming public through accident or theft is so unlikely, however, that the sender and arbitrator's claims would have very little credibility.

Question 3 [30 marks]

The figure below shows a X.509 hierarchy of users and Certificate Authorities (CAs), where the users are leaf nodes and each parent node is a CA of its children nodes.



a) Which node issues A's certificate? [2 marks]

b) Explain the steps that must be taken for user A to verify the identity and public key of user B. In your steps you must include all necessary exchanges with CAs (e.g. users' obtaining certificates). State any assumptions you make. [10 marks]

c) If you followed your steps described in part (a), for each of the 10 nodes, list the public keys it now has. [5 marks]

d) List and describe an advantage and disadvantage of using such a certificate hierarchy. [5 marks]

e) An important aspect of public key distribution is determining who to trust. Design a scheme for establishing trust amongst users where there is no hierarchy (e.g. no CA's) and

different levels of trust are assigned to users. For example, one user may assign a trust level to another user of 0.8, meaning they "trust them 80%". You can assume there are some users (but not all) that a user will trust 100%.

In your design, consider ways to obtain and calculate the trust level, and when to know that you can trust (and hence use) information from a user or not. As much as possible, these methods should be automatic, that is, without any human intervention (although you may need some initial assignments of trust by human users).

Your design does not have to be long, but should be detailed enough so that I could use it to implement a simple application that could be used in a real computer network.

In your design make sure you state any assumptions and you identify and comment on any security or performance limitations of your scheme. [8 marks]

Answers

Part (a)

The CA H issues node A's certificate.

Part (b)

The first steps are issuing of certificates. Node K is the root CA, and therefore issues certificates to H and J. Then these CA's issue certificates to their children: H issues to A; and J issues to G. Then G issues a certificate to B.

Now, when A and B want to exchange data, they first exchange certificates. Assuming B sends its certificate to A, A wants to verify the certificate. But since B's certificate is signed by G, A does not yet trust it. So now A sends a request to its own CA, H, asking for G's public key (certificate).

Lets assume two things:

1. Each CA has the certificates of their immediate children that they issued certificates to. For example, K has the certificates of J and H.

2. All certificates contain hierarchical identifiers, that allows the system to identify who issued the certificate. For example, the certificate issued by J to G has an identifier such as IDK.J.G, and certificate issued by K to J has an IDK.J.

From the these assumptions when H receives a request from A for the certificate IDK.J.G, then H sends the request to K. K then sends the request to J, which should have the certificate of G. Hence J returns the certificate of G to A (via K and H).

Now A has (and trusts) the public key of G, and since B's certificate is signed by G's private key, A now has verified B.

Part (c)

Node	Has public keys of:
A	H, B, G
B	G
C	G
D	F
E	H
F	D, J
G	B, C, J
H	A, E, K, [G]
J	F, G, K
K	J, H, [G]

Part (d)

Advantages:

- Users can register and obtain certificates manually from local CAs (e.g CA in their country/region). This is much easier than if every single user in the world had to obtain a certificate from one CA
- Requests for certificates and signing is distributed among many CAs, therefore reducing the load on a single CA if no hierarchy is present

Disadvantages:

- CA's must trust each other (agreements should be in place)
- Takes time to verify the certificate of a node, if that certificate is not signed by your local CA

Part (e)

There are different possible answers, but consider this as a basic scheme:

1. When a network is established between two users (A and B), they each assign trust value of 1.0 to each other's public key. That is, A trusts B's public key 100% and B trusts A's public key 100%.

2. As new nodes join the network, they are assigned a trust value of 1.0 (although it could be less) with the user that introduces them. For example, node C is introduced through node B, and therefore they give each other a trust value of 1.0

3. When new nodes join the network, they also obtain the trust values of other nodes already in the network through the 'introducer'. For example, C was introduced by B. Since B also knows A's, trust value (1.0), it passes A's information to C. Normally C will not trust A with a value 1.0. Instead lets assume we have a function F() which reduces the trust. If for example we choose to half the trust. The end result (after C has joined the network):

- A trusts B: 1.0

- B trusts A: 1.0

- B trusts C: 1.0

- C trusts B: 1.0

- C trusts A: 0.5

4. The process continues as new nodes join the network. If D joins through C, then:

- C trusts D: 1.0

- D trusts C: 1.0

- D trusts B: 0.5

- D trusts A: 0.25

5. As well as exchanging trust information when a node joins a network, there may be periodic updates during network operation (e.g. every 10 seconds, nodes advertise who they trust). Or when two nodes exchange data, they may also exchange their trust information. As a result, most nodes will have trust values of most other nodes in the network.

6. We use this trust information to determine whether a public key signed by a node is trustworthy. Each node has its own threshold of trustworthy or not. For example, if D receives a public key of X signed by B, then:

a. If D's trust threshold is 60%, then D will NOT accept the signature of B (since D only trusts be with 0.5 or 50%)

b. If D's trust threshold is 40%, then D will accept the signature of B (and hence now have the public key of X)

7. With many nodes in the network, node A may form a trust value of another node based on trust values from multiple nodes. For example, if B trusts Y with value 0.4 and C trusts Y with value 0.6, then A (who trusts B 1.0 and trusts C with 0.5) may form a weighted sum to obtain the trust level of Y: (1.0x0.4 + 0.5x0.6)/2 = 0.15. There are many other variants (such as max, min, sum).

8. Some methods for calculating the trust value may be based not only on introductions, but also based on past experiences. For example, if B exchanges data with C and has a "good experience" (e.g. no packets are lost, throughput is good, C does not perform any malicious actions) then B may upgrade the trust of C. But if B has a "bad experience" (e.g. C takes a long time to respond to security challenges) then B may downgrade the trust of C. This is similar to "user rating" level that can be obtained on web sites like E-bay, Digg, ….

Advantages of this type of scheme is that it doesn't rely on CA's: all users are peers. This is a common method used in informal groups of friends. (You trust your best friend 100%, and will trust your best friends friend slightly less, …). Also there is freedom for individual nodes to assign trust threshold depending on their application. For example, some secure applications may require a high trust threshold (90%) while others may be acceptable with lower thresholds.

Problems with such a scheme include:

- Initiating the network requires assigning initial trust levels of to users. If a malicious node can go undetected in this initiation phase, then maybe they can get a higher level of trust than they should (and hence eventually perform malicious activities like pretending to be someone else).

- Nodes can collude with each other to give artificially high trust levels.

- There may be significant communication overheads in exchanging the trust information.

Question 4 [15 marks]

You must submit your answers in two forms:

1. Written (including this assignment sheet)
2. Electronic (via email to steve@siit.tu.ac.th)

The steps below outline what you must do to complete the electronic submission. You must use CrypTool to create the certificates and perform encryptions.

1. Generate your own certificate using RSA/1024-bit. This will automatically be signed by CrypTool, which acts as a Certificate Authority. You can use any PIN for PSE (I do not have to know it). Include a printout of your certificate data in your report (you can copy and paste it from CrypTool into MS Word).
2. Export your certificate and save it using the default file name (e.g. mine is similar to "[Gordon][Steven][RSA-1024][12345678].p12"). Note that when exporting you must enter a PKCS#12 PIN – this is different than the PSE PIN. Use the value 1234 for the PKCS#12 PIN, as I need to know this value.
3. Import my certificate into CrypTool (I also used the PKCS#12 PIN of 1234). An electronic copy (in the Personal Information Exchange format, which can be directly imported into CrypTool) is available at: http://it.siit.tu.ac.th/~sgordon/cs322/protected/GordonStevenRSA-1024-certificate.p12
4. Create a text file with your answers to question 1 only (not calculations) in the following format:

   ID=12345689

   Answer1a=value1

   Answer1b=value2

   where you replace 12345689 with your ID, and replace value1 with the value you calculated for Answer 1(a) and so on. If you did not calculate an answer, use 0 as the answer.
5. Sign your answer text file using your certificate and a SHA1 160-bit hash function.
6. Encrypt the signed answer text file using RSA and my certificate. Save the resulting file as a binary (.hex) file using your ID as the file name, e.g. 123456789.hex (if your ID was 123456789).
7. Send me via email your certificate file (.p12) and the encrypted/signed answer text file (.hex). Your subject of the email must be: CSS322 Assignment 2 ID (where your replace ID with your actual ID), and the two files should be attached. There is no need to archive the two files using ZIP or RAR – simply attach them as is.

When marking your assignments, I will perform the following steps to check:

1. Import your certificate into CrypTool
2. Decrypt the answers file using RSA and my certificate
3. Verify your signature.
4. Check the answers.

I will not be checking your certificate and answers before the assignment is due (in other words, please do not ask me if your certificate worked). However, I will try to reply to your email confirming its receipt as soon as I receive the email.

Answers:

Here is the export of my public key I created for user ID 123456789

```
Version:                 2 (X.509v3-1996)
SubjectName:             CN=Steven Gordon [1171857985], DC=cryptool, DC=org
IssuerName:              CN=CrypTool CA 2, DC=cryptool, DC=org
SerialNumber:            07
Validity  -  NotBefore:  Mon Feb 19 11:06:30 2007 (070219040630Z)
              NotAfter:  Tue Feb 19 11:06:30 2008 (080219040630Z)
Public Key Fingerprint:  32A8 535A A4A1 FCFB 13A5 D7DB B2BC 3D20
SubjectKey:              Algorithm rsa (OID 2.5.8.1.1), Keysize = 1024
             Public modulus (no. of bits = 1024):
               0  FEB53DB0 43DEB912  CF818617 8296B848
              10  5D57B9FE 23AC479D  F009CAF3 0516BFAF
              20  4EF6F2A9 4868324C  47F782CA 227A5414
              30  E0B7A460 3FABF12D  C210ECD1 68E2528C
              40  3469432E EDF1FCBA  6B3E7DCA 2155DFEF
              50  43A67BE5 7E50D275  2926FB7D 5AABBC49
              60  34780C4A C07BBDA8  A28F6CB7 52C907E9
              70  0B212A1A 942F853E  8B865D1B 9017EA5F
             Public exponent  (no. of bits = 17):
               0  010001
Certificate extensions:
Private extensions:
    OID 2.206.5.4.3.2:
        PrintableString:
                |[Gordon][Steven][RSA-1024][11718|
                |57985]                          |

SHA1 digest of DER code of ToBeSigned:
               0  9D255049 86409465  771371E8 852456D5
              10  98FDF14C
Signature:               Algorithm sha1WithRSASignature (OID 1.3.14.3.2.29), NULL
               0  E0BE83BD 377674F3  EABD49CA B685734C
              10  19F9B84B 69592962  AB02CDE5 E03FC786
              20  7505FA60 1CD5AE38  DF1EF70E 5A40925A
              30  62BBDA09 2A7B361F  E979A8F2 8C6B7324
              40  E5EA5E49 09AC1085  023877BD 688AC97B
              50  06E85B49 FE272B01  4A099764 F894B327
              60  BDA07127 9B2AA52D  E369EFB9 A993B885
              70  D6D56A2F 166CCB56  F9749FD1 FE8E15F0
              80  EA37DED6 71DCD251  E42CA491 C3F1B83F
              90  01F08B26 7E84B950  A64C15F2 4C69397E
              A0  89CD6B4F 0490743D  DEDCFC53 1840B4C1
              B0  7DF8CB0F EAFCC45D  85B3AC13 2F598F2E
              C0  D7303630 F47B6CEE  36BE6046 31D98940
              D0  D21E5141 5EFCDB86  0248B121 15855C15
```

```
               E0  B431BB98 B0AE4599  193D6176 04F806F7
               F0  8C418FA3 AA084B8B  005EB5B2 3883C366
Certificate Fingerprint (MD5):    F4:4D:BA:02:94:E5:D6:FE:9E:BF:83:1B:A1:47:B3:C2
Certificate Fingerprint (SHA-1):  AEC5 5E86 2FF8 39B1 48D7 AEDF D891 8158 439A 9499
```

Here is my answers text file:

```
ID=123456789
Answer1a=BFQG
Answer1b=AYHGDAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Signed and encrypted document can be found in 123456789.hex

User 123456789's public key is found in [Gordon][Steven][RSA-1024][1171857985].p12

My (the CA's) public certificate is found in GordonStevenRSA-1024-certificate.p12