Coursework for CSC3621 Cryptography

Part 3

Exercise 1

Question 1:

Smallest of e:

Using a small exponent e of 3 and a small message of size 8, it is unlikely that 8^3 (= 512) is larger than that of N and therefore you are not benefiting from the hardness of calculating the eth root because mod N has no effect. Instead you can just calculate the 3rd root over the integers.
Another problem with the smallness of e is that if a cipher text is sent to e or more recipients and all receivers of the message share the same exponent e. Then by using Chinese remainder theorem it is possible to decrypt the message.

Same RSA modulus:

If we are using the same Modulus for every employee then they all share the same group order p(N) and this can be calculated if you know any pair of 'e' and 'd' given by the equation e*d = 1 (mod p(N)). Once they have attained the group order for all employees they can compute any secret key d with the appropriate public key e thus allowing them to decrypt messages sent to that employee.

Question 2:

The cube root of the cipher text = 8769767668797869. This can be broken down into pairs which represent their ascii value. 87,69,76,76,68,79,78,69.
This gives the output 'WELLDONE'

Question 3:

To make RSA a fully CPA-secure encryption scheme, you have to add padding to the message. Once padding has been added 3 vital properties of the encrypted message will hold. Firstly padding will add randomness to the message. By adding a set of random padding, if a message is encrypted multiple times it will always produce different results. It will be non-deterministic. The second property that padding adds is structure. An example of structure via padding is adding 16 bits to the front of your message with the value equal to 2. The point of adding structure to the message is to detect if anyone has mangled the message. From the example if the message has been mangled, the receiver of the message would detect that the first 16 bits do not equal to 2 and the message can be discarded. The final property is that the message length will be padded so that the message has the length of the full modulus so that the message raised to the exponent will always be greater than that of the modulus.

Exercise 2

Question 1: Integrity Verification

GnuPG has two ways of verifying that the installation you have got is authentic and integral. The first way is to use an older version of GnuPG that has already been validated or is trusted. Within this old version you can use the command gpg –verify gnupg-version.extention.sig gnupg-version.extention. This checks if the signature matches the source file. If you don't already have a trusted version of GnuPG you can run a checksum calculation on the downloaded file and compare it to the checksums on the website. This is how you would 'bootstrap' an installation. To check the validity and integrity of the portable file downloaded from blackboard. You can import the demo key and verify the downloaded zip file against it using the command gpg2 –verify PortableGnuPG.zip.gpg

Validating recipient public keys is done by using a fingerprint that is sent with the key. This fingerprint needs to be validated with the owner of the public key by person to person communication or through another means so long as you know are talking to the keys owner. Once you have validated the key is correct for that user, you can sign the public key to say that you have authenticated it. GnuPG gives you a number of options to see a fingerprint using 'fpr' and sign the fingerprint using 'sign'. You can also check who else has signed the key by using the 'check' command. By signing the key you begin to build up a 'web of trust' model which can build up to allow quick validating of keys. Say you receive a public key from Alice that has already been signed by your friend, Bob. You completely trust Bob's signature that is found on the key and therefore do not need to check with Alice that the fingerprint is correct. Now you know that Alice's key is valid. Alice has also signed several other keys C, D and E. Because you trust Bob and Bob trusts Alice you can trust the keys from C, D and E.

GnuPG has an inbuilt 'trust' command. The trust command is used to justify how much you trust each person's keys. This can then expand out that if you trust Bob fully you have no problem trusting Alice. However if you 'trust marginally' you may want to double check with Alice that the fingerprint you have received is correct.

It is important to validate the installation and key for authenticity and integrity. Authenticity means that it comes from the source it says it comes from and as you trust that source won't be malicious. Integrity ensures you know that the message has not been tampered with. It may be that the message seems to becoming from someone you trust but it could have been intercepted on the way and now contain malicious data.

Question 2: Encryption of Large Files

GnuPG encrypts large files by compressing them first. It then encrypts this with a symmetric cipher due to its speed advantage. Finally it encrypts the key with an asymmetric cipher and attaches it to the encrypted compressed file.

To encrypt a file for multiple recipients you must stack the –recipients command when –encrypting the message. An example would look like:

gpg2 --encrypt --recipient alice@example.com --recipient bob@example.com doc.txt

GPG encrypts the file once with a symmetric cipher and then encrypts the key for each recipient asymmetrically. This ensures that the file size stay relatively low no matter how many recipients you have. If you want to be able to decrypt your encrypted files, you have to add your own name/email address to the recipients who are receiving the file. Thus it will be encrypted with your public key. There is also an encrypt-to command that can be used for the same results. The difference is that encrypt-to does not do any trust checking on the key being used.