SOLUTIONS *FOR CS4 EXTERNAL EXAMINER (date of this version: 1/3/2002)*

UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

COMPUTER SCIENCE FOUR

COMPUTER SECURITY

Tuesday 16 April 2002

??:00 to ??:30

Examiners:

Prof. K. Turner (External)

Prof. C. Stirling (Chairman)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. (a) **Authentication** is the establishment of the identity of a subject that makes a request to access some object [2 marks]. **Authorization** is the granting of a request, based on identity and perhaps other aspects of the security policy [2 marks]. (The traditional picture shows a *reference monitor* which guards the object and decides whether to authorize access).

   (b) The access control matrix should look like this:

	`type.exe`	`alice.msg`	`bob.plan`	`mail.log`
Alice	{exec}	{read, write}	{read}	{append}
Bob	{exec}	{}	{read, write}	{append}
Charlotte	{exec}	{}	{read}	{append}

   [4 marks for perfect table; scaling 3–1 for small–serious errors]

   (c) The new ACLs are

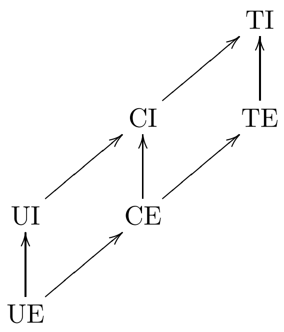
	`type2.exe`	`stolen.msgs`
Alice	{exec}	{append}
Bob	{exec}	{append}
Charlotte	{read, write}	{read, write}

   For `type2.exe`, everyone except Charlotte needs to be able to execute it. (If other users can inspect permissions, Charlotte should probably add execute permission for herself to allay suspicion). For `stolen.msgs`, the other users need at most append permission. For both objects, Charlotte must have read and write access since she is the owner.

   When Alice executes Charlotte's program, the process assumes her identity. Therefore, it can open `alice.msg` for reading. It can also open Charlotte's `stolen.msgs` for appending, and so append the contents of `alice.msg`. [2 marks for each ACL; 2 marks for explanations]

   (d) In the MLS, the discretionary access controls are as before. When Alice executes Charlotte's program, the program assumes Alice's identity as before, and so can open `alice.msg` for reading. However, when it opens Charlotte's `stolen.msgs` for writing at the same time, a security violation occurs, because the *no-write down* policy (part of the **-property* of Bell-LaPadula) would be broken. [3 marks for point of failure; 2 marks for property.]

   (e) Using the obvious abbreviations, the lattice is: [2 marks]



System high, TI, is for top-secret internal documents. System low, UE, is for unclassified externally available documents [2 marks]. Releasing a top-secret document, but also letting Alice see it means a level of CI $\wedge$ TE = CE. Also letting Charlotte see it means a level of system low, UE [2 marks].

2. (a) PGP, SSH, SSL, or any other similar application/protocol. They all use freshly generated shared session keys for bulk encryption (providing speed, forward and backward secrecy) [2 marks] and public keys for authentication [2 marks].

   (b) A rule expressing the closure of observations for messages encrypted under a public key K:

$$\frac{P \models \xrightarrow{K} P \qquad P \triangleleft \{X\}_K}{P \triangleleft X}$$

   That is, if a principal P sees a message encrypted under her public key K, she also sees the contents of the message.

   (c) i. T is a timestamp which specifies a lifetime for the certificate (or the key). Its use is to prevent a replay attack on the protocol with an outdated key.

   ii. For a public-private key-pair (k, k^{-1}), ordinary PK satisfies $D_{k^{-1}}(E_k(M)) = M$ (where E_k is the encryption operation, and $D_{k^{-1}}$ is the decryption operation). Reversible PK also satisfies $E_k(D_{k^{-1}}(M)) = M$ [2 marks for last fact]. The protocol supposes that A can read the contents of the certificate to retrieve K_b, but it was *encrypted* with the server's private key, which would only be used for decryption in ordinary PKC [2 marks for this observation].

   iii. The idealized form of the protocol discards all plain text, and formalizes the key transfer, binding K_b to B:

$$\text{Message 2'} \quad S \to A: \quad \{\xrightarrow{K_b} B,\, T\}_{K_s{}^{-1}}$$

   (d) **Assumptions** [1 mark each]:

   - $A \models \xrightarrow{K_s} S$ — Alice believes that K_s is the server's public key
   - $A \models \sharp(T)$ — A believes the timestamp is fresh
   - $A \models S \Rightarrow \xrightarrow{K} P$ — A believes S has jurisdiction over public keys
   - $S \models \xrightarrow{K_b} B$; $S \models \sharp(T)$ — S believes what he says at the outset (NB: correctness condition for protocol, not used in proof).

   Proof: start from $A \triangleleft (\xrightarrow{K_b} B, T)$ [1 mark], then (in outline):

   - Use the given message meaning rule and (a) to show $A \models S \mid\sim (\xrightarrow{K_b} B, T)$
   - Use the *nonce verification rule* and (b) to show $A \models S \models \xrightarrow{K_b} B$
   - Use the *jurisdiction* rule and (c) to show $A \models \xrightarrow{K_b} B$.

   [1 mark for each correct step; solution should show details]

3. (a) The Vinegar cipher performs these substitutions:

i	0	1	2	3	4	5	6	7
0	2	3	4	5	6	7	0	1
1	3	4	5	6	7	0	1	2
2	5	6	7	0	1	2	3	4

where i is the position in each block of 3 letters.

   i. `123 452 034 521` is mapped to `350 607 261 756`. [3 marks]

   ii. The cipher preserves frequency distributions of symbols in the plaintext. Over longer plaintexts, we can use this for cryptanalysis: determine the block size by finding repeated patterns in the ciphertext, then consider block positions independently, using frequency analysis. [3 marks]

   iii. The only way to make an "unbreakable" cipher is using a one-time pad: use a key (and block-size) as long as the message. The disadvantage is that the key must be random and never re-used. [3 marks]

(b) i. Vinegar revised maps `3506 0726 1756` to `0563 2760 5761`. [3 marks]

   ii. The revised cipher can be improved by *repetition* of the substitution and permutation steps, in several *rounds*. [2 marks]

   iii. The revised cipher, even if several rounds are added, is *deterministic*, which can make it vulnerable to known-plaintext attacks. Adding a random prefix or suffix (the *salt*) rectifies this. [2 marks]

(c) i. $s = m^{d_A} \bmod n_A = 59847900$ and
   $c = s^{e_B} \bmod n_B = 38842235$ [2 marks].

   ii. $s' = 38842235^{44360237} \bmod n_B = 4382681$ and
   $m' = 4382681^5 \bmod n_A = 54383568$ [2 marks].

   iii. Since $m \neq m'$, the signature verification has failed. The reason is that in the second step performed by A, $s > n_B$, so this step was an improper use of RSA [2 marks for this observation].

   iv. Assuming messages are randomly distributed with equal probabilities, the probability of this failure is related to the difference between the two moduli, specifically it is the proportion $(n_A - n_B)/n_A$ [1 mark]. Some methods of preventing this problem are:

   - encrypt first and then sign if $n_A > n_B$
   - use separate encryption and signing keys, and ensure that each user's signing key has a smaller modulus than any encryption key (e.g., encrypt modulo $t+1$-bit primes and sign modulo t-bit primes)
   - restrict the form of moduli to reduce the possibility of failure to an insignificant level (e.g., with a binary expansion beginning with 10^k)

   [2 marks for any two suggestions]