Exercice 1 (**8** pts)

a) Dans le chiffrement par bloc, le rôle du bourrage consiste à
1- servir de clé de chiffrement
2- servir de clé de déchiffrement
3- jouer le rôle du Vecteur d'Initialisation
4- **Autre**: Compléter la taille du message pour qu'il soit un multiple de bloc, et délimiter le message réel au déchiffrement **(0.75)**

b) Dans le chiffrement par flot/flux, la clé du chiffrement est utilisée
1- directement pour chiffrer le message via XOR
2- directement pour déchiffrer le message via XOR
3- pour chiffrer des blocs de taille fixe de 16 octets
4- **Autre**: pour derriver des clés intermédiaires qui elles seront XOR avec chaque octet du message en clair **(0.75)**

c) Dans l'authentication windows NTLM Hash, deux utilisateurs ayant choisi deux mots de passe différents, mais dont l'un est le préfixe de l'autre, auront deux hachés
- identiques
- différents vu l'utilisation d'une valeur de sel unique par mot de passe
- dont l'un est le préfixe de l'autre
- **Autre**: différents (vu l'utilisation d'une fonction de hachage pour la génération du haché) **(0.75)**

d) Dans l'authentication Linux basé sur DES
-
- le mot de passe peut être de taille quelconque, mais uniquement les 8 premiers caractères sont pris en considération **(0.5)**
-

e) CBC-MAC définit
- un algorithme de chiffrement par bloc
- un algorithme de chiffrement par flot
- une fonction de hachage
- **Autre**: un algorithme/fonction d'intégrité de donnée **(0.75)**

f) EMAC améliore CBC-MAC en
- supprimant le chainage entre les blocs clairs et chiffrés
- en utilisant un VI aléatoire pour génerer le code d'intégrité
- en incrémentant par 1 la clé utilisé pour chaque bloc successive
- **Autre**: en utilisant deux clés secrètes partagées, l'une chiffrant les blocs intérieurs l'autre le dernier bloc **(0.75)**

g) HMAC est
- une fonction de hachage
- une fonction d'intégrité basée sur un chiffrement par bloc
- basé uniquement sur la fonction de hachage SHA
- **Autre**: un algorithme d'intégrité de donnée générique basée sur une fonction de hachage (MD5, SHA, SHA1, …) **(0.75)**

h) L'attaque par rejeu (replay) consiste à
- déduire la clé de chiffrement à partir d'un couple <message clair, message chiffré>
- trouver à partir d'un haché *h* le message correspondant *M* telque h= H(M)
- usurper l'identité d'une entité
- **Autre**: rejouer, ou ré-envoyer un paquet qui a déjà transiter dans le réseau à un instant t en un instant ultérieur t'>t **(1)**
la réponse (3) est aussi correct

I) L'authentification type Challenge/Réponse consiste à
- Envoyé le mot de passe en clair
- Envoyé le haché du mot de passe
- Envoyé le mot de passe chiffré par une clé pré-partagée
- **Autre**: envoyer un challenge (question) au vérificateur afin de tester -suivant sa réponse- s'il possède bien le secret partagé et par conséquent l'authentifier avec succès **(1)**

J) Un logiciel malveillant de type Ransomware vise principalement à
- Voler les mots de passes enregistrés ou saisis par l'utilisateur
- Espioner l'utilisateur en allumant discrètement sa caméra/microphone
- Infecter la machine cible pour servir comme machine zombie (botnet)
- **Autre** : prendre en otage les données de la victime -en les cryptant par exemple- contre le paiment d'une rançon **(1)**

Exercice 2
On suppose que le Système d'exploitation Windows possède un fichier nommée Verifsys.exe, et que tous les fichiers systèmes sont protégées en intégrité grâche à un MAC, (y compris Verifsys.exe) et que Verifsys.exe possède la clé nécessaire pour le calcul/vérification des MACs.

Verifsys.exe possède les caractéristique suivantes
- le premier fichier système chargé en mémoire après le demarage de la machine
- il est responsable de charger en mémoire le reste des fichiers systèmes (.exe, .dll, etc.)
- il vérifit l'intégrité des fichiers systèmes à chaque démarage et chaque fois qu'un fichier système est chargé en mémoire (y compris lui même), si un fichier ne passe pas avec succès cette vérification il n'est pas chargé en mémoire et une alerte est affiché sur écran
- suite aux mises à jours systèmes, il calcule un nouveau MAC pour chaque fichier concerné

Questions:

1) Pourquoi a t-on besoin de protéger les fichiers systèmes, vu que leur utilisation se fait uniquement en local? Donnez deux exemples concrêt ou ceci s'avère utile (**1.25)**
R : les fichiers systèmes peuvent être altérés/modifiés soit d'une façon accidentel (ex : plantage système) ou de façon prémédité à travers une attaque (ex : logiciels malveillants)
Par exemple, un virus peut altérer un fichier système (en greffant son code dans le code du fichier) afin de pouvoir s'éxecuter et se propager dans la machine cible. De même, un rootkit peut modifier certaines commandes systèmes (lister fichiers processus, etc.), afin de cacher une activité malveillantes (ex, ne pas afficher les fichiers des logiciels malveillants qu'il cache, ni les processus malveillants en cours d'éxecution)

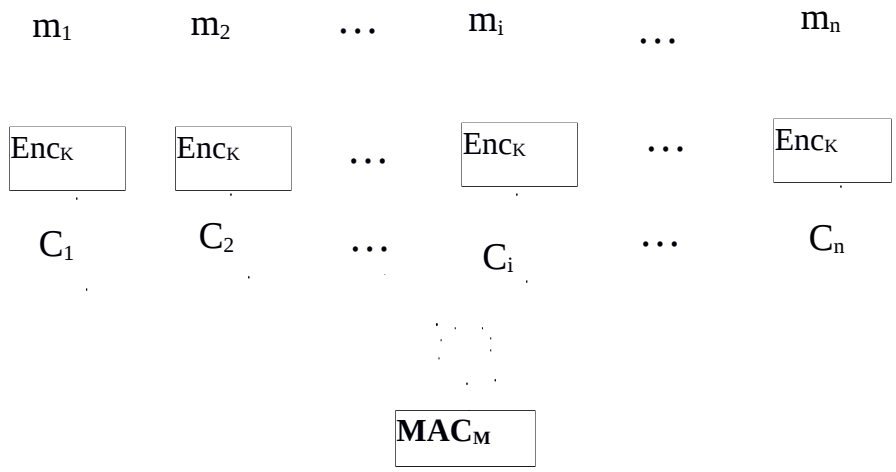
2) Quel est le but de l'utilisation des MACs au lieu d'une simple fonction de hachage ? Justifiez (**1.25**)
R : L'utilisation d'une simple fonction de hachage peut protège uniquement contre les modifications accidentels, alors que l'utilisation de MAC -donc un secret partagé utilisé lors du calcul/vérification MAC) protège en plus contre les modification intentionelles.
3) Un virus infectant uniquement les fichiers systèmes peut il passer inaperçu (on suppose que la machine ne dipose pas d'un Anti-virus) (**1**) ? Si la réponse est non que doit il faire pour que son attaque passe inaperçue dans les deux cas suivants :
1)Verifsys.exe est stocké dans une mémoire à lecture seule (**0.75**)
2)Verifsys.exe est stocké sur disque dure (**0.75**)

R :
- Théoriquement, si un virus modifie un fichier système , et qu'il ne possède pas la clé necessaire, sa modification sera détecté par Verifsys.exe , vu que le virus ne sera pas capable de calculer le nouveau MAC correspondant au fichier modifié
Pour que l'attaque passe inaperçu, c'est à dire la modification d'un fichier système par le virus ne soit pas détecté par Verifsys.exe, il faut l'une des deux choses : 1) soit trouver la clé secrète qu'utilise Verifsys.exe, et ainsi le virus est en mesure de calculer un MAC valide, soit modifier Verifsys.exe ou plus exactement la partie qui est chargé de vérifier l'intégrité des autres fichiers systèmes -bypasser/inhiber cette partie dans le code de Verifsys.exe –

1)Verifsys.exe est stocké dans une mémoire à lecture seule : dans ce cas la modification de Verifsys.exe n'est normalement pas possible, et vu qu'il est le 1er module système chargé après le démarage du système toute tentative de modification risque d'être détecté à moins que le virus parvient à trouver la clé → une autre solution, peu probable est d'attaquer le BIOS car c'est ce dernier qui est censé chargé Verifsys.exe, et dans ce cas le BIOS va pointer vers un programme mis au point par l'attaquant
2)Verifsys.exe est stocké sur disque dure → dans ce cas il suffit de désactiver le module charger de vérifier l'intégrite des fichiers systèmes pour que l'attaque passe inaperçue, lors des prochains démarrage chargement de fichiers systèmes modifiés

4) Un virus infectant n'importe quel fichier exécutable, peut il passer inaperçu (on suppose que la machine ne dipose pas d'un Anti-virus)
R : oui, s'il modifie les fichiers non systèmes (autres applications word.exe, viber.exe, etc.) car ces fichiers ne sont pas supposés protégés/vérifiés par **Verifsys.exe (1)**

Exercice 3 (6 pts)
On propose d'étudier l'algorithme d'intégrité de donnée XOR-MAC définit comme suis :
M : le message qu'on désire protéger l'intégrité, dont la taille en octets est un multiple de 16
m_i : i^{eme} bloc en clair du message M
Enc_K fonction de chiffrement en bloc, avec une taille de bloc de 16 octets, utilisant la clé secrète **K**
: Ou-Exclusive (XOR)



Questions (On suppose qu'un attaquant ne possède pas la clé **K**)

1) Que représente MAC_M ? Donnez son expression (formule)
R : représente le code d'intégrité où le code d'authentification calculé sur M. **(0.75)**
$\mathbf{MAC_M}$= Enc_K (m_1) XoR $Enc_K(m_2)$ XoR … XoR $Enc_K(m_i)$ XoR ... $Enc_K(m_n)$ **(0.75)**

2) Pour chacun, des cas suivants, indiquez si un attaquant interceptant <M, MAC_M> pourra modifier avec succès (sans être détecté) le message **M** en **M'** (Justifiez les réponses):

- L'attaquant supprime un seul bloc $\mathbf{m_i}$
  R : Non, l'attaque sera detecté car $MAC_{M'}$=Enc_K (m_1) XoR $Enc_K(m_2)$ XoR … XoR $Enc_K(m_{i-1})$ XoR $Enc_K(m_{i+1})$ XoR... $Enc_K(m_n)$, = MAC_M XoR $Enc_K(m_i)$, mais l'attaquant n'est pas en mesure de le calculer ou de le dériver de MAC_M car il n'est pas en mesure de calculer $Enc_K(m_i)$. Donc si l'attaquant modifie <M, MAC_M> en <M', MAC_M> le MAC calculée par le recepteur sera différent du MAC reçu **(0.75)**
- L'attaquant permute deux blocs $\mathbf{m_i}$ et $\mathbf{m_j}$
  R : L'attaque ne sera pas detécté, car si M= m_1 … $\mathbf{m_i}$..$\mathbf{m_j}$… m_n alors M' = m_1 … $\mathbf{m_j}$..$\mathbf{m_i}$… m_n et comme XoR est commutative on obtient MAC_M=Enc_K (m_1) XoR…XoR $\mathbf{Enc_K(m_i)}$ XoR … XoR $\mathbf{Enc_K(m_j)}$ XoR ... $Enc_K(m_n)$=Enc_K (m_1) XoR…XoR $\mathbf{Enc_K(m_j)}$ XoR … XoR $\mathbf{Enc_K(m_i)}$ XoR ... $Enc_K(m_n)$ **(1)**
- L'attaquant modifie un seul bloc $\mathbf{m_i}$
  R : L'attaque sera detectée car en modifiant $\mathbf{m_i}$ **en** $\mathbf{m'_i}$ on obtient forcément un nouveau message **M'** avec M≠M' et vu que $Enc_K(\mathbf{m_i}) \neq Enc_K(\mathbf{m'_i})$ on aura forcement $\mathbf{MAC_M \neq MAC_{M'}}$ **(** $\mathbf{MAC_{M'}}$=$\mathbf{MAC_M}$ XoR$Enc_K(\mathbf{m_i})$ XoR $Enc_K(\mathbf{m'_i})$ **)**, l'attaquant étant incapable de calculer $Enc_K(\mathbf{m'_i})$ et $Enc_K(\mathbf{m_i})$ **(0.75)**
- L'attaquant rajoute un seul bloc $\mathbf{m_{n+1}}$
  R : Dans ce cas $\mathbf{M'=Mm_{n+1}}$ par conséquent $\mathbf{MAC_{M'}=MAC_M}$ XoR $\mathbf{Enc_K(m_{n+1})}$, l'attaquant ne pouvant calculer $\mathbf{Enc_K(m_{n+1})}$, par conséquant ne peut calculer $\mathbf{MAC_{M'}}$ valide **(0.75)**

3) Cette algorithme possède une autre faiblesse permettant à un attaquant de générer $\mathbf{M' \neq M}$ avec $\mathbf{MAC_M}$= $\mathbf{MAC_{M'}}$, dans le cas où M (resp. M') contient un bloc dupliqué $\mathbf{m_i}$ expliquez l'attaque ?
R :
a) Supossant M= m_1, m_2, …, $\mathbf{m_i}$, …, $\mathbf{m_j}$, …, m_n avec $\mathbf{m_j=m_i}$
Dans ce cas MAC_M = Enc_K (m_1) XoR ...XoR $\mathbf{Enc_K(m_i)}$ XoR … XoR $\mathbf{Enc_K(m_j)}$ XoR ... $Enc_K(m_n)$, donc on aura $\mathbf{Enc_K(m_i)}$ qui va s'annuler avec $\mathbf{Enc_K(m_j)}$
L'attaquant calcul **M' = m1, m², …, mi-1, mi+1, …,mj-1, mj+1 (**enlève $\mathbf{m_i}$ **et** $\mathbf{m_j}$**)** et obtient par conséquent $\mathbf{MAC_{M'}=MAC_M}$, ainsi l'attaque ne sera pas détecté **(0.75)**

b) à la reception de <**M**, $\mathbf{MAC_M}$>, l'attaquant calcul **M'**=M$\mathbf{m_i m_i}$ (ajoute deux blocs identique $\mathbf{m_i}$ à M), le récepteur recevra <**M'**, $\mathbf{MAC_M}$>, calculera $\mathbf{MAC_{M'}}$ qui n'est rien d'autre que $\mathbf{MAC_M}$ XoR $\mathbf{Enc_K(m_i)}$ XoR $\mathbf{Enc_K(m_i)}$ = $\mathbf{MAC_M}$, ainsi l'attaque ne sera pas détecté. **(0.75)**