

555 Boulevard de l'Université,

Chicoutimi, QC G7H 2B1

(418) 545-5011

**Baccalauréat en informatique**

**Sécurité des Réseaux et du Web**

8INF135

**Rédacteurs**

Daniel Bisson

Alexandre Cox

**Professeur :** David Gagnon

**Observation du prof**:

**Note : Date : 02/11/2018**

**Particularités :**

**Instructions de création du projet (à partir du code source) :**

1. Créer avec visual studio une solution C++ projet vide.
2. Ajouter 3 projets à la solution et les nommer ainsi :
   1. Clement
   2. Agnesse
   3. Bernard
3. Inclure les fichiers main.cpp dans les projets correspondants:
   1. Ajouter dans Clément mainC.cpp.
   2. Ajouter dans Agnesse mainA.cpp.
   3. Ajouter dans Bernard mainB.cpp.
4. Inclure les fichiers .h et .cpp des dossiers suivant dans chacun des projets(le faire pour Clément, Agnesse et Bernard)
   1. Dossier « Encryption »
   2. Dossier « Connection »
5. Inclure à visual studio la librairie « wsock32.lib » dans la section éditeurs de liens.

**Marche à suivre, déroulement général du programme:**

1. Démarrer les projets dans l’ordre suivant : Clément, puis Agnesse ou Bernard par la suite.
2. Agnesse recevra une question de la part de Clément. Répondre « Bernard » et tapez sur la touche « enter ».
3. Clement distribue une clé de session pour le chiffrement et le MAC, ainsi que les informations nécessaires pour qu’Agnesse établisse une connexion avec Bernard. La communication s’établira alors automatiquement entre Agnesse et Bernard avec la clé de session. Clément terminera son exécution.
4. Les échanges entre Agnesse et Bernard peuvent débuter.
5. Tant que le mot clé « end » n’est pas saisi au clavier :
   1. Écrire le message d’Agnesse.
   2. Bernard recevra le message.
   3. Écrire le message de Bernard.
   4. Agnesse recevra le message.
6. Terminer l’exécution du programme en saisissant le mot clé « end » et en tapant la touche « enter ».

**Fonctionnement du programme par étapes (voir diagramme ci-dessous):**

1. Clément possède déjà deux clés pour chiffrement et deux pour la génération d’un MAC, il partage une paire avec Agnesse et une autre avec Bernard.
2. Clément attend une connexion de la part d’Agnesse et de Bernard.
3. À la connexion, il authentifie chacun d’eux avec la fonction « authenticate ». « Authenticate » retourne la valeur de vrai si pour le premier message, Agnesse ou Bernard, un après l’autre, envoie comme message « is Agnesse » et « is Bernard » et que le code MAC est vérifié positivement avec la fonction « verifyMAC ». Les codes MAC sont générés par une fonction HMAC qui se nomme « simpleHMCA ».
4. Une fois le client authentifié, Clément s’authentifie auprès d’Agnesse et de Bernard avec le message « is Clement » accompagné du code MAC généré avec la clé correspondante.
5. Clément met en marche la fonction « scriptedConvoTest ». Il génère une clé de session aléatoire et une clé mac aléatoire, pour distribution future. Pour les échanges qui suivent, tous les messages seront cryptés, et accompagnés d’un code MAC généré à partir du cryptogramme. À la réception, le message est vérifié pour le code MAC puis décrypté par la suite.
6. Clément demande à Agnesse à qui elle veut parler. Son message est crypté avec la clé privée d’Agnesse.
7. Pour notre scénario, Agnesse devra répondre « Bernard » à la question de Clément. Agnesse attend alors la réponse de Clément.
8. Si Clément identifie le nom de Bernard, il engage le processus de distribution de clés de session.
9. Clément commence par envoyer un message d’information à Agnesse et Bernard en utilisant leur clé respective. Il mentionne entre autres que Bernard est de confiance à Agnesse et qu’Agnesse est de confiance à Bernard.
10. Ensuite, Clément fait parvenir les clés de session (chiffrement et MAC) à Agnesse et à Bernard. Il donne également à Bernard un port sur lequel écouter et à Agnesse l’adresse de Bernard plus le port. Les informations sont dans cet ordre : clé de session de chiffrement, clé de session pour MAC, adresse et port.
11. Le travail du tiers de confiance se termine. Clément termine son programme.
12. Bernard effectue un « accept() » sur l’adresse IP et port mentionné par le tiers de confiance. Il s’attend à une connexion de la part d’Agnesse.
13. Agnesse, pour sa part, tente un « connect() » avec les informations reçues par le tiers de confiance vers le serveur de Bernard.
14. Lorsque la connexion est établie, on entre mutuellement dans notre fonction « runConversationAgnesseBernard ». Cette fonction permettra à Agnesse et à Bernard de s’échanger des messages jusqu’à ce qu’un d’eux mentionne le mot clé « end ». Ceci mettra fin à la communication. La communication se fait en utilisant la clé de session et la clé MAC de session. Chacun des échanges est crypté, joint d’un code MAC puis envoyé. À la réception, le message est authentifié avec la clé MAC de session, puis décrypté par la suite.

**Diagramme de fonctionnement :**

**Clément**

1-2-4-7-10

3

1

3

1

8-9

5

6

8-9

12

**Agnesse**

**Bernard**

13

11-13

13

13

**Description des fonctions principales:**

**Encryption.cpp :**

Fonctions d’encryption :

1. string encrypt(const string& message, const string& key);
   1. Fonction interface pour le chiffrement. Utilise la fonction cbcEncrypt.
2. string decrypt(const string& message, const string& key);
   1. Fonction interface pour le déchiffrement. Utilise la fonction cbcDecrypt.
3. string cbcEncrypt(const string& message, const string& key);
   1. Algorithme de chiffrement par bloc. Utilise le mode d'opération CBC

avec un réseau de Feistel. Son entrée est le message clair à encrypter et sa sortie est le cryptogramme. La méthode de « padding » PKCS5 est utilisé pour compléter le dernier bloc de donnée.

1. string cbcDecrypt(const string& message, const string& key);
   1. Implémente l’algorithme de déchiffrement pour le mode d’opération CBC. Retire les octets de « padding » ajouté à la fin du message lors du chiffrement.
2. void reseauFeistel(unsigned char\* blockInput, const string& key, unsigned turn\_count);
   1. Cette fonction implémente le réseau de Feistel utilisé pour l’encryption des données. La fonction F est un algorithme de chiffrement au choix prenant en entré un bloc de donnée, la clé de chiffrement et la taille du bloc. Le programme utilise le chiffrement de Vigenère.
3. void vigenere(unsigned char\* block, const unsigned char\* key, unsigned length);
   1. Cette fonction implémente le chiffrement des données en utilisant un modèle de Vigenère.
4. void xorSbox(unsigned char\* block, const unsigned char\* key, unsigned length);
   1. Fonction de chiffrement avec une S-box, mais non utilisée dans la version actuelle.
5. void useSBox(unsigned char\* msg, unsigned int len);
   1. Fonction implémentant le passage par une S-box, mais non utilisée dans la version actuelle.

Fonctions de hachage :

1. string simpleHash(const string& message);
2. Fonction qui implémente le hachage simple des données, utilisée par la fonction simpleHMCA. Elle traite successivement les données par la rotation des bits d'un bloc suivit d'un XOR avec le code de hashage.
3. string simpleHMCA(const string& message, const string& key);
   1. Cette fonction implémente l’algorithme HMAC pour la génération de code MAC. Elle est utilisée pour l’authentification des messages dans le programme. Elle prend en entrée un message et la clé de MAC, et donne en sortie un string correspondant au code HMAC généré.

Fonctions d’authentification :

1. bool authenticate(string msg, string name, string mac\_key)
   1. Cette fonction est le début du processus d’authentification. Elle s’attend à recevoir la bonne string en entrée puis appelle verifyMAC() pour authentifié le message. Elle donne en sortie « true » si l’authentification est réussie et « false » si en échec.
2. string generateMac(const string& message, const string& key)
   1. Fonction interface pour la génération de code MAC. Utilise la fonction simpleHMCA().
3. bool verifyMAC(const string& str, const string& key)
   1. Cette function fait la vérification du code MAC. Elle retourne « true » sur la comparaison du code MAC est vérifié positivement et « false » si le résultat est négatif.
4. string extractMsg(const string& str)
   1. Procède à l’extraction de la partie message du string correspondant.