

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. Clément
   2. Agnesse
   3. Bernard
3. Inclure les codes sources dans les projets suivant les instructions suivantes :
   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, l’ordre voulu.
2. Agnesse recevra une question de la part de Clément. Répondre « Bernard » et tapez sur la touche « enter ».
3. 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. Faire tant que le mot clé « end » soit 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 shéma ci-dessous):**

1. Clément ouvre un serveur et attend une connexion de la part d’Agnesse et de Bernard.
2. 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 ». Le MAC utilisé par notre programme est une fonction HMAC qui se nomme « simpleHMCA ».
3. Une fois le client authentifié, Clément s’authentifie auprès d’Agnesse et de Bernard avec le message crypté « is Clement », et la clé MAC correspondant pour chacun d’eux.
4. Tour à tour, Agnesse et Bernard authentifie Clement avec la fonction « verifyMAC. ».
5. Clément met en marche la fonction « scriptedConvoTest ». Il a à sa disposition la clé secrète d’Agnesse et celle de Bernard, ainsi qu’une clé MAC secrète pour chacun d’eux. Il génère également une clé de session aléatoire et une clé mac aléatoire, pour utilisation future.
6. Pour la suite, chacun des échanges est crypté, et un code MAC est généré à partir du cryptogramme. À la réception, le message est vérifié pour le code MAC, et décrypté par la suite.
7. Clément demande à Agnesse à qui elle veut parler. Son message est crypté avec la clé privée d’Agnesse.
8. Pour notre scénario, Agnesse devra répondre « Bernard » à la question de Clément. Agnesse attend alors la réponse de Clément.
9. Si Clément identifie le nom de Bernard, il engage le processus de distribution de clés de session.
10. Clément commence par envoyer un message d’information à Agnesse avec la clé privée d’Agnesse et à Bernard avec la clé privée de Bernard. Il mentionne entre autres que Bernard est de confiance à Agnesse et que Agnesse est de confiance à Bernard.
11. Ensuite, Clément fait parvenir une clé de session à Agnesse et à Bernard. Il leur fait aussi parvenir une clé MAC de session. Il donne également à chacun d’eux l’addresse IP et le port de communication pour la communication de la session.
12. Le travail du tiers de confiance se termine. Clément termine son programme.
13. Agnesse et Bernard reçoivent les informations de la part de Clément. Le premier message est la clé de session. Le 2e est la clé MAC de session. Le 3e est l’adresse IP et le port de communication. Bernard fait de même.
14. Bernard démarre alors un serveur qui fera 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.
15. Agnesse, pour sa part, tente un « connect() » aces les informations reçu par le tiers de partie vers le serveur de Bernard.
16. 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.
17. À partir de ce moment, 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.
18. Lorsque le mot clé « end » est rencontré, la communication se termine.

**Shéma de fonctionnement :**

**Clément**

1-2-5-6-9-12

3

1

3

1

10-11

7

8

10-11

15

**Agnesse**

**Bernard**

4-6-13-16-18

4-6-13-14-16-18

17

17

**Description des fonctions principales:**

**Encryption.cpp :**

Fonctions d’encryption :

1. string cbcEncrypt(const string& message, const string& key);
   1. Procède à l’encryption des données. Elle utilise le mode d’opération CBC crypte les données avec un réseau de Feistel. Son entrée est le message clair à encrypter et sa sortie est le cryptogramme.
2. string cbcDecrypt(const string& message, const string& key);
   1. Procède à la désencryption des données. Elle utilise le mode d’opération CBC décrypte les données avec un réseau de Feistel. Son entrée est le cryptogramme à décrypter et sa sortie est le message clair.
3. void reseauFeistel(unsigned char\* blockInput, const string& key, unsigned turn\_count);
   1. Cette fonction implémente le réseau de Feistel utilise pour l’encryption des données via la fonction vigenere.
4. void vigenere(unsigned char\*, const unsigned char\*, unsigned);
   1. Cette fonction implémente le cryptage des données en utilisant un modèle de Vigenère.
5. void xorSbox(unsigned char\*, const unsigned char\*, unsigned);
   1. Fonction implémentée mais non utilisée dans la version actuelle.
6. void useSBox(unsigned char\* msg, unsigned int len);
   1. Fonction implémentée 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 successivemnt 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 un code HMAC pour l’authentification des messages dans le programme. Elle prend en entrée un message avec 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. Cette function génère le code MAC en appellant la function simpleHMCA().
3. bool verifyMAC(const string& str, const string& key)
   1. Cette function fait la verification 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.

**Autres informations dignes de mention:**