Théorie des codes - TP 2

ZZ3 F5 - Réseaux et Sécurité
Cryptographie Classique : Chiffrement de Vigenère

15 octobre 2024

1 Introduction

Blaise de Vigenère, né le 5 avril 1523 à Saint-Pourçain sur Sioule dans l'Allier, fut un savant et cryptographe. Dans son traité de 1586, il introduit une méthode de chiffrement qui a résisté aux tentatives de décryptage pendant plus de trois siècles. Ce rapport présente le chiffrement de Vigenère, son fonctionnement, et les méthodes de décryptage ainsi que la cryptanalyse associée.

2 Chiffrement de Vigenère

2.1 Principe

Le chiffrement de Vigenère utilise une clé sous forme de mot ou de phrase. Pour chaque lettre du texte clair, une lettre de la clé est utilisée pour effectuer la substitution. La formule du chiffrement est donnée par :

$$C = (T + K) \mod 26$$

Pour le déchiffrement, l'opération inverse est appliquée :

$$T = (C - K) \mod 26$$

2.2 Exemple

Considérons le texte clair T = pythagore et la clé K = algorythme. En utilisant le carré de Vigenère, nous obtenons :

$$C = \text{pjzvrehyq}$$

3 Fonctions de chiffrement et de déchiffrement

3.1 Implémentation

Nous avons développé des fonctions pour le chiffrement et le déchiffrement en C++. Voici un extrait du code :

```
class Vigenere {
public:
    string key;

    Vigenere(string key) {
        // Traitement de la clé
        this->key = key;
        // Optionnel: Normalisation de la clé (ex. supprimer les espaces, rendre tout en majuscules)
        this->key.erase(remove_if(this->key.begin(), this->key.end(), ::isspace), this->key.end());
        transform(this->key.begin(), this->key.end(), this->key.begin(), ::toupper);
    }
```

```
    string encrypt(string text) {
        string result;
        int keyIndex = 0;
        for (char c : text) {
            if (isalpha(c)) { // Vérifie si c est une lettre
                char shift = toupper(key[keyIndex % key.length()]) - 'A'; // Calcule le décalage
                char encryptedChar = (toupper(c) - 'A' + shift) % 26 + 'A'; // Chiffrement
                result += encryptedChar;
                keyIndex++; // On incrémente l'index de la clé seulement pour les lettres
            } else {
                result += c; // Ajoute les caractères non-alphabétiques sans changement
            }
        }
        return result;
    }

    string decrypt(string text) {
        string result;
        int keyIndex = 0;
        for (char c : text) {
            if (isalpha(c)) { // Vérifie si c est une lettre
                char shift = toupper(key[keyIndex % key.length()]) - 'A'; // Calcule le décalage
                char decryptedChar = (toupper(c) - 'A' - shift + 26) % 26 + 'A'; // Déchiffrement
                result += decryptedChar;
                keyIndex++; // On incrémente l'index de la clé seulement pour les lettres
            } else {
                result += c; // Ajoute les caractères non-alphabétiques sans changement
            }
        }
        return result;
    }
};
```

3.2 Analyse

L'algorithme de Vigenère est une méthode de chiffrement par substitution polyalphabétique qui utilise une clé pour déterminer le décalage à appliquer à chaque caractère du texte en clair. Voici une analyse détaillée des différentes parties de l'implémentation :

3.2.1 1. Traitement de la clé

Dans le constructeur `Vigenere`, nous traitons la clé en supprimant les espaces et en la convertissant en majuscules. Cela assure que la clé est toujours dans un format cohérent et simplifie le chiffrement et le déchiffrement.

— **Normalisation de la clé** : Cela empêche des variations non intentionnelles dues à des majuscules/minuscules ou à des espaces dans la clé.

3.2.2 2. Fonction de chiffrement

La méthode `encrypt` prend une chaîne de caractères comme entrée et produit une chaîne chiffrée. Les étapes principales sont :

— **Boucle sur chaque caractère du texte** : On parcourt chaque caractère du texte. Si le caractère est une lettre, nous calculons le décalage correspondant à la clé.

— **Calcul du décalage** : Le décalage est obtenu en convertissant le caractère de la clé en une valeur numérique (de 0 à 25) en soustrayant 'A'.

— **Chiffrement** : On applique la formule pour chiffrer le caractère. Nous ajoutons le décalage et utilisons l'opération modulo 26 pour garantir que le résultat reste dans les limites des lettres de l'alphabet.

— **Gestion des caractères non-alphabétiques** : Les caractères non-alphabétiques sont ajoutés au résultat sans modification, ce qui préserve la structure du texte d'origine.

3.2.3 3. Fonction de déchiffrement

La méthode `decrypt` fonctionne de manière similaire à `encrypt`, mais en appliquant le décalage de manière inverse :

- **Calcul du décalage inverse** : Au lieu d'ajouter le décalage, nous le soustrayons. Nous ajoutons également 26 avant d'appliquer le modulo pour gérer les cas où la soustraction donnerait un résultat négatif.
- **Traitement identique pour les caractères non-alphabétiques** : Comme pour la fonction de chiffrement, les caractères non-alphabétiques sont ajoutés tels quels.

3.3 Conclusion

L'implémentation du chiffrement et du déchiffrement de Vigenère est efficace et respecte les principes de base de l'algorithme. Les fonctions sont conçues pour gérer à la fois les caractères alphabétiques et non-alphabétiques, garantissant ainsi la préservation du format original du texte tout en offrant une sécurité raisonnable pour des applications non critiques. Des améliorations pourraient inclure une gestion plus robuste des clés et des options pour d'autres alphabets ou jeux de caractères.

4 Cryptanalyse du chiffrement de Vigenère

4.1 Méthodes de cryptanalyse

La cryptanalyse du chiffrement de Vigenère inclut deux étapes principales :

1. Identifier la période de la clé.
2. Trouver la clé spécifique.

Pour identifier la période, nous utilisons l'Indice de Coïncidence (IC). La formule de l'IC est donnée par :

$$IC = \frac{\sum_{q=A}^{Z} n_q(n_q - 1)}{n(n-1)}$$

où n_q est le nombre de fois que la lettre q apparaît, et n est le nombre total de lettres dans le texte chiffré.

4.2 Application de la méthode

Nous avons testé notre méthode sur un message chiffré donné et obtenu les valeurs moyennes d'IC pour différentes longueurs de clé. Cela a permis d'estimer la longueur de la clé et de déduire le texte original.

4.3 Analyse du code de cryptanalyse

Le code suivant permet d'automatiser la cryptanalyse du chiffrement de Vigenère en trois étapes principales : filtrer le texte, estimer la longueur de la clé à l'aide de l'Indice de Coïncidence (IC), et trouver la clé à partir des fréquences des lettres.

```
string cryptanalyse(string ciphertext)
{
    string filtered_text;
    for (char c : ciphertext)
    {
        if (c >= 'A' && c <= 'Z')
            filtered_text += c;
        else if (c >= 'a' && c <= 'z')
            filtered_text += c + 'A' - 'a';
    }

    int keyLength = findKeyLength(filtered_text);
    string estimatedKey = findKey(filtered_text, keyLength);

    cout << "Longueur estimée de la clé : " << keyLength << endl;
    cout << "Clé estimée : " << estimatedKey << endl;
```

```
    Vigenere tempCipher(estimatedKey);
    string decryptedText = tempCipher.decrypt(ciphertext);

    return decryptedText;
}
```

4.3.1 1. Filtrage du texte

La première partie du code consiste à filtrer le texte chiffré pour ne conserver que les lettres alphabétiques et à convertir toutes les lettres en majuscules. Cela est essentiel pour normaliser le texte et simplifier l'analyse.

— `filtered_text += c + 'A' - 'a';` : Cette ligne convertit les lettres minuscules en majuscules.
— Tous les autres caractères non alphabétiques sont ignorés.

4.3.2 2. Estimation de la longueur de la clé

La fonction `findKeyLength` calcule l'Indice de Coïncidence pour différentes longueurs de clé possibles, de 1 à 20. La longueur de clé qui maximise l'IC moyen est considérée comme la plus probable.

```
int findKeyLength(const string &text)
{
    int probableKeyLength = 1;
    double bestIC = 0.0;

    for (int keyLength = 1; keyLength <= 20; ++keyLength)
    {
        double averageIC = 0.0;
        for (int i = 0; i < keyLength; ++i)
        {
            string subText;
            for (unsigned int j = i; j < text.length(); j += keyLength)
            {
                subText += text[j];
            }

            averageIC += calculateIC(subText);
        }
        averageIC /= keyLength;

        if (averageIC > bestIC)
        {
            bestIC = averageIC;
            probableKeyLength = keyLength;
        }
    }

    return probableKeyLength;
}
```

Cette fonction segmente le texte en plusieurs sous-textes, chacun correspondant à une position spécifique dans le texte chiffré modulo la longueur de la clé. Elle calcule ensuite l'IC pour chaque sous-texte et prend la moyenne de ces valeurs. La longueur de clé avec le meilleur IC est retournée.

4.3.3 3. Trouver la clé

La fonction `findKey` utilise les fréquences des lettres dans la langue anglaise pour déterminer la clé. Elle applique un test du χ^2 pour chaque position afin de trouver le décalage qui minimise cette statistique par rapport aux fréquences des lettres en anglais.

```
string findKey(const string &text, int keyLength)
{
    string key;
    vector<double> englishFrequencies = {
        8.167, 1.492, 2.782, 4.253, 12.702, 2.228, 2.015, 6.094, 6.966, 0.153,
        0.772, 4.025, 2.406, 6.749, 7.507, 1.929, 0.095, 5.987, 6.327, 9.056,
        2.758, 0.978, 2.360, 0.150, 1.974, 0.074};

    for (int i = 0; i < keyLength; ++i)
    {
        string subText;
        for (unsigned int j = i; j < text.length(); j += keyLength)
        {
            subText += text[j];
        }

        double minChiSquared = numeric_limits<double>::max();
        char bestShift = 'A';

        for (int shift = 0; shift < 26; ++shift)
        {
            int counts[26] = {0};
            for (char c : subText)
            {
                int shifted = (c - 'A' - shift + 26) % 26;
                counts[shifted]++;
            }

            double chiSquared = 0.0;
            int total = subText.length();
            for (int k = 0; k < 26; ++k)
            {
                double expected = englishFrequencies[k] * total / 100.0;
                chiSquared += pow(counts[k] - expected, 2) / expected;
            }

            if (chiSquared < minChiSquared)
            {
                minChiSquared = chiSquared;
                bestShift = 'A' + shift;
            }
        }

        key += bestShift;
    }

    return key;
}
```

Chaque sous-texte est décalé par un nombre de positions possible (0 à 25), et un test du χ^2 est appliqué pour comparer les fréquences des lettres obtenues avec les fréquences standard de la langue anglaise. Le décalage avec le χ^2 minimal est considéré comme le bon décalage, et ce décalage est ajouté à la clé estimée.

4.4 Conclusion

Le chiffrement de Vigenère reste une technique classique de cryptographie qui, bien que résistant à certaines méthodes de décryptage, peut être vulnérable si des clés courtes sont utilisées. La cryptanalyse basée sur l'IC et le test du χ^2 sont des méthodes efficaces pour récupérer des clés courtes, comme illustré dans l'analyse de notre implémentation en C++.

A Annexes

A.1 Fréquences des lettres

Les fréquences des lettres en anglais et en français sont essentielles pour la cryptanalyse. Par exemple, la lettre A apparaît 8.55% du temps en anglais.

Appendix Letter Frequencies

English letter frequencies (%):

A :	8.55	K :	0.81	U :	2.68
B :	1.60	L :	4.21	V :	1.06
C :	3.16	M :	2.53	W :	1.83
D :	3.87	N :	7.17	X :	0.19
E :	12.10	O :	7.47	Y :	1.72
F :	2.18	P :	2.07	Z :	0.11
G :	2.09	Q :	0.10		
H :	4.96	R :	6.33		
I :	7.33	S :	6.73		
J :	0.22	T :	8.94		

French letter frequencies (%):

A :	7.60	G :	1.18	Q :	0.85
À :	0.43	H :	0.93	R :	6.86
Â :	0.05	I :	7.21	S :	7.98
Æ :	0.00	Î :	0.04	T :	7.11
B :	0.96	Ï :	0.01	U :	5.55
C :	3.39	J :	0.30	Ù :	0.02
Ç :	0.05	K :	0.16	Û :	0.02
D :	4.08	L :	5.86	Ü :	0.00
E :	14.47	M :	2.78	V :	1.29
É :	2.43	N :	7.32	W :	0.08
È :	0.42	O :	5.39	X :	0.43
Ê :	0.13	Ô :	0.05	Y :	0.34
Ë :	0.00	OE:	0.02	Z :	0.10
F :	1.12	P :	2.98		

FIGURE 1 – Exemple de fréquence des lettres.

Références