ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

PROJET VHDL : Le chiffre de Vigenère

Sous la supervision de : Mr. Soufiane EL MOUMNI

REALISÉ PAR :

- FARHI OUSSAMA
- YOUSSEF GUARNAOUI
- SARA MOUHSEN

SOMMAIRE

Partie théorique

1 Introduction au Chiffrement :

Le chiffrement est une technique de cryptographie qui consiste `à transformer des informations lisibles en un format code pour assurer leur confidentialité. Seules les personnes possédant la l'E de d´chiffrement peuvent accéder au message original, ce qui en fait un outil essentiel pour la protection des données dans notre monde numérique.

2 Le chiffre de Vigenère :

Une M´méthode poly Alphabétique Le chiffre de Vigenère est une méthode de cryptage poly Alphabétique qui utilise une clé pour chiffrer un message. Il s'agit d'une amélioration du chiffre de César, qui utilise un seul décalage pour tout le message. Le chiffrement de Vigenère utilise une série de décalages différents, rendant ainsi le cryptage plus complexe et plus sur.

3 Définition et Histoire

Le chiffre de Vigenère est un système de cryptage par substitution poly alphabétique, inventé au XVIe siècle. Il porte le nom de Blaise de Vigenère, un diplomate français qui l'a popularisé, bien que la méthode ait été initialement développée par Giovan Battista Bellaso1. Ce chiffre utilise une clé de cryptage pour effectuer des substitutions multiples basées sur les lettres d'un mot-clé, rendant chaque lettre du message clair susceptible d'être remplacée par plusieurs lettres chiffrées selon sa position dans le message1. Contrairement aux systèmes monoalphabetiques comme le chiffre de César, le chiffre de Vigenère résiste à l'analyse de fréquences grâce à l'utilisation de plusieurs alphabets de substitution. Cependant, il a été finalement décodé par le major prussien Friedrich Kasiski en 1863, ce qui a révèle ses vulnérabilités et a conduit `à la recherche de méthodes de chiffrement plus s´ecurisees1.

Cette méthode historique de cryptage reste un sujet d'étude important pour comprendre l'évolution de la cryptographie et les fondements de la sécurité de l'information moderne.

4 Principe du Chiffre de Vigenère :

Le principe fondamental du chiffre de Vigenère est d'associer `à chaque lettre du message une valeur numérique correspondant `à son rang dans l'alphabet, `à partir de 0 (A=0, B=1, . . ., Z=25). Le chiffrement s'effectue en additionnant la clé au texte clair, lettre par lettre, et le résultat est exprimé modulo 26. Formule de Chiffrement : on chiffre un message, on utilise la formule suivante pour chaque lettre du texte :

$$E_i = (P_i + K_i) mod 26$$

Formule de Déhiffrement:

$$D_k(C_i) = (C_i - K_i + 26) mod 26$$

5 Exemple Détaillé de Chiffrement et Déchiffrement avec le Chiffre de Vigenère :

Prenons l'exemple avec le mot-clé "CLE" et le texte "BONJOUR". Voici les étapes détaillées :

5.1 Conversion des lettres en nombres:

Texte: B O N J O U R
1 14 13 9 14 20 17
Mot-clé: C L E C L E C
2 11 4 2 11 4 2

5.2 Répétition du mot-clé:

Le mot-clé "CLE" est répété pour correspondre à la longueur du texte "BONJOUR".

5.3 Application de la formule de chiffrement:

Pour chaque lettre du texte, on applique la formule :

$$E_i = (P_i + K_i) mod 26$$

.

5.4 Conversion des nombres en lettres:

Les nombres obtenus sont reconverties en lettres pour former le texte chiffré "DQYLYXT".

5.5 Déchiffrement du message:

Pour déchiffrer le message "DQYLYXT", on applique la formule de déchiffrement en utilisant la même clé "CLE".

5.6 Démonstration détaillée:

Texte: B O N J O U R
1 14 13 9 14 20 17
Mot-clé: C L E C L E C
2 11 4 2 11 4 2
Chiffré: D Q Y L Y X T
(1+2) (14+11) (13+4) (9+2) (14+11) (20+4) (17+2)
3 25 17 11 25 24 19

6 Conclusion :

Le code de Vigenère constitue un chapitre fondamental de l'histoire de la cryptographie. Malgré sa vulnérabilité face aux techniques d'attaque contemporaines, il demeure un instrument ´éducatif essentiel pour saisir les concepts élémentaires de cryptage et de protection des données. L'analyse de ce système de chiffrement souligne la nécessite d'une adaptation continue des stratégies cryptographiques afin de prévenir les risques émergents.

Partie pratique

Explication du code VHDL

Le code VHDL implémente un chiffrement de Vigenère. Il prend en entrée un message et une clé, et peut soit chiffrer soit déchiffrer le message selon le signal de commande. Voici une explication détaillée de chaque composant et du fonctionnement global du code :

1- Entity chiffre_de_vigenere

```
5  entity chiffre_de_vigenere is
6      port (
7          message_entree : in  std_logic_vector(6 downto 0);
8          key            : in  std_logic_vector(6 downto 0);
9          reset          : in  std_logic;
10         encry          : in  std_logic;
11         decry          : in  std_logic;
12         clock          : in  std_logic;
13         message_sortie : out std_logic_vector(6 downto 0)
```

L'entité définit l'interface du module avec les ports d'entrée et de sortie :

message_entree : entrée, vecteur de bits représentant le message à chiffrer ou déchiffrer.

key : entrée, vecteur de bits représentant la clé de chiffrement.

reset : entrée, signal de réinitialisation.

encry : entrée, signal pour activer le chiffrement.

decry : entrée, signal pour activer le déchiffrement.

clock : entrée, signal d'horloge.

message_sortie : sortie, vecteur de bits représentant le message chiffré ou déchiffré.

2- Architecture cs of chiffre_de_vigenere

```
architecture cs of chiffre_de_vigenere is

    component stockage_key is
        port (
            key_entree : in  std_logic_vector(6 downto 0);
            clk        : in  std_logic;
            reset      : in  std_logic;
            R_sortie   : out std_logic_vector(6 downto 0) array (0 to 9);
        );
    end component;
```

L'architecture décrit la structure interne et les composants utilisés dans le module.

Composants :

stockage_key : Composant pour stocker la clé.

key_entree : entrée, vecteur de bits de la clé.

clk : entrée, signal d'horloge.

reset : entrée, signal de réinitialisation.

R_sortie : sortie, tableau de vecteurs de bits stockant la clé.

```
28      component incre_mux is
29          port (
30              presence_on : in  std_logic;
31              reset       : in  std_logic;
32              position    : out std_logic_vector(3 downto 0)
33          );
34      end component;
```

incre_mux : Composant pour incrémenter et multiplexer la position.

presence_on : entrée, signal de présence de message.

reset : entrée, signal de réinitialisation.

position : sortie, vecteur de bits représentant la position actuelle**.**

```
36      component mux_10_1 is
37          port (
38              key      : out std_logic_vector(6 downto 0);
39              position : in  std_logic_vector(3 downto 0);
40              R        : in  std_logic_vector(6 downto 0) array (0 to 9);
41          );
42      end component;
```

mux_10_1 : Composant multiplexeur pour sélectionner la clé.

key : sortie, vecteur de bits de la clé sélectionnée.

position : entrée, vecteur de bits représentant la position actuelle.

R : entrée, tableau de vecteurs de bits représentant les clés stockées.

```
44      component add is
45          port (
46              encry : in  std_logic;
47              decry : in  std_logic;
48              x     : in  std_logic_vector(6 downto 0);
49              key   : in  std_logic_vector(6 downto 0);
50              R     : out std_logic_vector(6 downto 0)
51          );
52      end component;
```

add : Composant pour ajouter ou soustraire les bits du message et de la clé.

encry : entrée, signal pour activer le chiffrement.

decry : entrée, signal pour activer le déchiffrement.

x : entrée, vecteur de bits de la clé sélectionnée.

key : entrée, vecteur de bits du message.

R : sortie, vecteur de bits représentant le résultat chiffré ou déchiffré.

```
54      component presence_test is
55          port (
56              presence_on    : out std_logic;
57              message_entree : in  std_logic_vector(6 downto 0)
58          );
59      end component;
```

presence_test : Composant pour tester la présence du message.

presence_on : sortie, signal indiquant la présence du message.

message_entree : entrée, vecteur de bits du message.

```
60
61      signal presence_message : std_logic;
62      signal position         : std_logic_vector(3 downto 0);
63      signal R                : std_logic_vector(6 downto 0) array (0 to 9);
64      signal x                : std_logic_vector(6 downto 0);
65
```

Signaux internes :

presence_message : signal indiquant la présence du message.

position : signal représentant la position actuelle.

R : tableau de vecteurs de bits pour stocker les clés.

x : vecteur de bits représentant la clé sélectionnée.

3- Instanciations des composants

a- Incrémentation et multiplexage de la position

```
65
66  begin
67      incrementeur: incre_mux
68          port map (
69              presence_on => presence_message,
70              reset       => reset,
71              position    => position
72          );
73
```

Ce composant gère l'incrémentation de la position basée sur la présence du message et le signal de réinitialisation.

b- Test de présence de message

```
73
74      test: presence_test
75          port map (
76              presence_on    => presence_message,
77              message_entree => message_entree
78          );
```

Ce composant vérifie si un message est présent et met à jour le signal presence_message.

c- Stockage de la clé

```
80      ram: stockage_key
81          port map (
82              key_entree => key,
83              clk        => clock,
84              reset      => reset,
85              R_sortie   => R
86          );
```

Ce composant stocke la clé fournie à chaque cycle d'horloge et la réinitialise si nécessaire.

d- Multiplexage de la clé

```
88      mux: mux_10_1
89          port map (
90              key      => x,
91              position => position,
92              R        => R
93          );
```

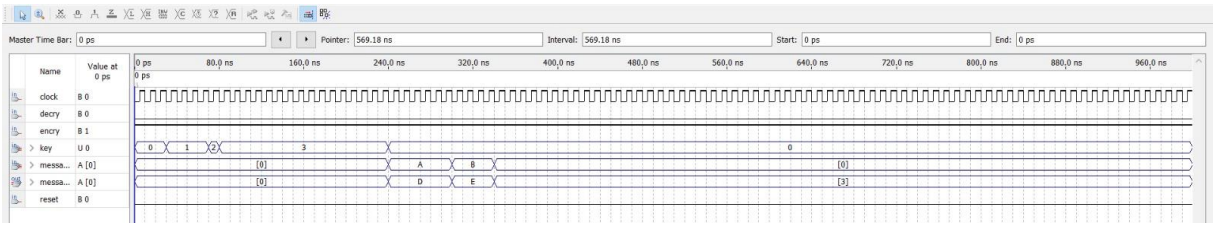
Ce composant sélectionne la clé appropriée à utiliser pour le chiffrement ou le déchiffrement basé sur la position actuelle.

e- Chiffrement ou déchiffrement

```
95      enry_decry: add
96          port map (
97              encry  => encry,
98              decry  => decry,
99              x      => x,
100             key    => message_entree,
101             R      => message_sortie
102         );
```

Ce composant effectue l'opération de chiffrement ou de déchiffrement en ajoutant ou soustrayant la clé sélectionnée et le message d'entrée, et fournit le résultat en sortie.

Simulation du code VHDL :



Nous avons pu crypter le message en utilisant la clé que nous avons sélectionnée. Cela confirme que notre implémentation du chiffrement de Vigenère fonctionne correctement.

Remerciement

Notre groupe souhaite exprimer une sincère reconnaissance envers notre professeur Mr Soufiane El Moumni pour son accompagnement tout au long du module VHDL. Ses cours magistraux ont été non seulement instructifs, mais également captivants, grâce à ses explications claires et à sa passion pour le sujet. Durant les travaux dirigés (TD), sa patience et sa disponibilité ont été d'une grande aide pour résoudre les difficultés et approfondir notre compréhension. De même, lors des travaux pratiques (TP), ses conseils avisés et ses retours constructifs ont été précieux pour développer nos compétences pratiques en VHDL. Son dévouement constant envers notre réussite académique a été une source d'inspiration pour nous tous. Nous lui sommes profondément reconnaissants pour son engagement exceptionnel dans notre formation."