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

Chiffrement et déchiffrement César : Conception et Implémentation en VHDL

Réalisé par:

JBALI Majda
FAJRI Zahra
IBERGIGUI Ibtihal

Encadré par: Mr. El Moumni

Sommaire

01

Introduction

Le chiffrement par décalage, connu sous le nom de code de César ou chiffre de César, est l'une des méthodes de cryptage les plus anciennes et les plus simples. Nommé d'après Jules César, qui l'utilisait pour ses communications secrètes, cet algorithme consiste à décaler chaque lettre d'un message d'un nombre fixe de positions dans l'alphabet. Par exemple, avec un décalage de trois positions, la lettre A devient D, B devient E, etc.

La simplicité du chiffre de César le rend facile à comprendre et à implémenter, mais cette même simplicité constitue aussi sa principale faiblesse. Un attaquant peut facilement casser le code en essayant les 25 décalages possibles. Pour renforcer la sécurité, une variante plus robuste utilise une clé. Dans cette version, chaque lettre du message est décalée selon une valeur déterminée par une clé, rendant le cryptage plus difficile à casser sans connaître cette clé.

Ce projet vise à implémenter un déchiffreur pour le code de César en utilisant le langage de description matériel VHDL. Le choix de VHDL permet de traduire cet algorithme simple en une architecture matérielle efficace, démontrant comment une solution matérielle peut optimiser le traitement de données chiffrées.

02

Principe de base

Le chiffrement de César classique est une méthode de cryptage qui décale chaque lettre d'un message d'un certain nombre de positions dans l'alphabet. Ce nombre de positions est appelé le "décalage". Par exemple, avec un décalage de 2, la lettre 'A' devient 'C', 'B' devient 'D', et ainsi de suite, jusqu'à 'Z' qui devient 'B'. Cette transformation simple repose sur le décalage circulaire des lettres de l'alphabet, c'est-à-dire que l'on revient au début de l'alphabet une fois la dernière lettre atteinte.

Le processus de chiffrement de César peut être formalisé mathématiquement. Si chaque lettre est représentée par un nombre (A = 0, B = 1 ,..., Z = 25), alors le chiffrement d'une lettre x avec un décalage k peut être exprimé par la formule : $E(x)=(x+k) \mod 26$ où mod26 représente l'opération de modulo 26, assurant le retour au début de l'alphabet après 'Z'.

Pour renforcer cette sécurité, une variante appelée Keyed Caesar Cipher ou chiffrement de César avec clé utilise une clé pour déterminer le décalage. Plutôt que d'utiliser un décalage fixe pour toutes les lettres, chaque lettre du message est décalée en fonction d'une clé, qui est un mot ou une phrase. La clé est répétée ou tronquée pour correspondre à la longueur du message. Chaque lettre de la clé est alors convertie en un décalage numérique (A = 0, B = 1, ..., Z = 25), et ce décalage est appliqué à la lettre correspondante du message.

Par exemple, si la clé est "KEY" et le message est "HELLO", la clé est répétée pour correspondre à la longueur du message : "KEYKE". Chaque lettre du message est ensuite décalée selon la valeur de la lettre correspondante dans la clé.

Le chiffrement de César avec clé ajoute une couche de sécurité en rendant le décalage non uniforme et dépendant de la clé, ce qui complique significativement le décryptage sans connaître cette clé. Ainsi, le même message chiffré avec des clés différentes produira des résultats différents, rendant les attaques par force brute beaucoup moins efficaces. Ce principe illustre une des premières améliorations vers des méthodes de cryptage plus robustes, posant les bases des systèmes cryptographiques modernes.

03

Étapes de l'Algorithme

Pour implémenter le chiffrement de César avec clé (Keyed Caesar Cipher) en VHDL, suivez les étapes suivantes :

01 Préparation du message et de la clé

- Convertir le message et la clé en lettres majuscules.
- Répéter ou tronquer la clé pour qu'elle corresponde à la longueur du message

02 Conversion des caractères en indices numériques

- Pour chaque lettre du message, convertir les lettres en indices numériques (A = 0, B = 1, ..., Z = 25).
- Convertir chaque lettre de la clé en indices numériques.

03 Chiffrement des lettres

Pour chaque lettre du message, appliquer le décalage correspondant à la lettre de la clé en utilisant la formule suivante :

$$Ci = (Mi + Ki) \bmod 26$$

où Ci est le caractère chiffré, Mi est le caractère du message, et Ki est le caractère de la clé.

04

Conversion des indices chiffrés en caractères

Convertir les indices chiffrés obtenus en lettres majuscules.

05

Assemblage du message chiffré

Combiner les lettres chiffrées pour former le message final chiffré.

04

Exemple de Chiffrement avec Clé

1. Préparation :
   - Message : "HELLO"
   - Clé : "KEY"
   - Répéter la clé pour qu'elle corresponde à la longueur du message : "KEYKE"

2. Conversion en indices numériques :
   - Message : H = 7, E = 4, L = 11, L = 11, O = 14
   - Clé : K = 10, E = 4, Y = 24, K = 10, E = 4

3. Chiffrement des lettres :
   - H (7) + K (10) = 17 (R)
   - E (4) + E (4) = 8 (I)
   - L (11) + Y (24) = 35 mod 26 = 9 (J)
   - L (11) + K (10) = 21 (V)
   - O (14) + E (4) = 18 (S)

4. Conversion en caractères chiffrés :
   - Indices chiffrés : 17 = R, 8 = I, 9 = J, 21 = V, 18 = S

5. Assemblage du message chiffré :
   - Message chiffré : "RIJVS"

05

Implémentation en VHDL

L'algorithme de chiffrement de César avec clé, dans le contexte du langage VHDL, implique des opérations sur des vecteurs de bits pour réaliser le chiffrement.

1. Déclaration des librairies et de l'entité: On utilise les librairies IEEE standard pour la logique numérique et les opérations arithmétiques.

2. Entité: On définit l'entité `KeyedCaesarCipher` avec des ports pour le signal d'horloge (`clk`), le signal de réinitialisation (`reset`), le mot-clé (`keyword`), le message à chiffrer (`message`), et le message chiffré en sortie (`ciphered_message`). Un paramètre générique `N` est utilisé pour définir la taille en bits des mots-clés et des messages.

3. Architecture comportementale: L'architecture Behavioral` est utilisée pour décrire le fonctionnement interne de l'entité.

1. ALGORITHME DE CHIFFREMENT

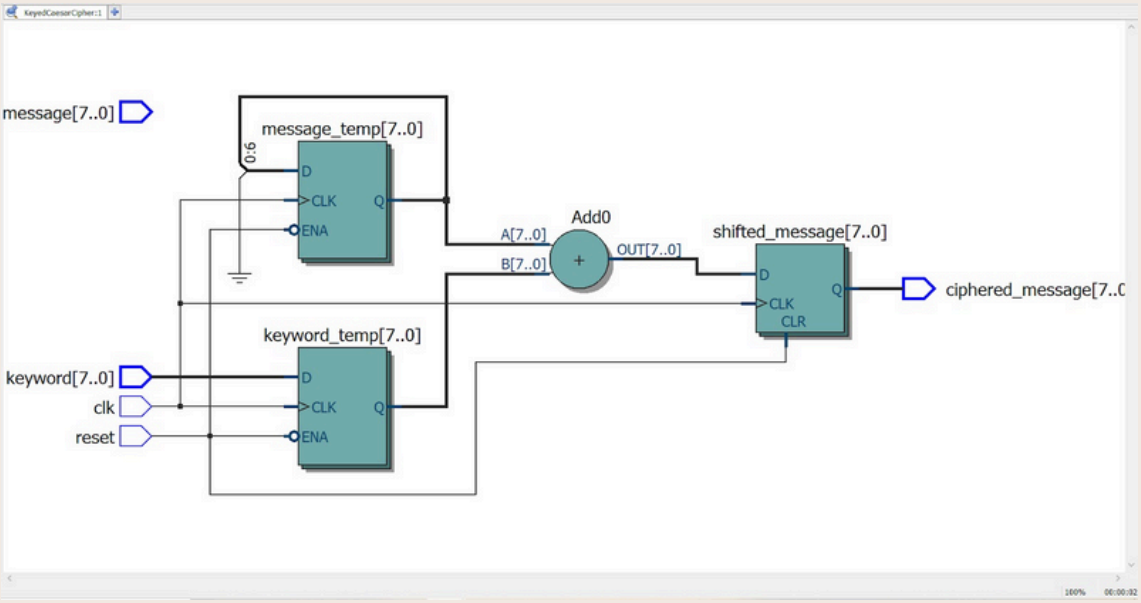
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity KeyedCaesarCipher is
    generic (
        N : integer := 8   -- Taille des mots-clés et des messages (en bits)
    );
    port (
        clk : in std_logic;
        reset : in std_logic;
        keyword : in std_logic_vector(N-1 downto 0);
        message : in std_logic_vector(N-1 downto 0);
        ciphered_message : out std_logic_vector(N-1 downto 0)
    );
end entity KeyedCaesarCipher;

architecture Behavioral of KeyedCaesarCipher is

    signal keyword_temp, message_temp, shifted_message : unsigned(N-1 downto 0);

begin

    process(clk, reset)
        variable sum_temp : unsigned(N downto 0);
    begin
        if reset = '1' then
```

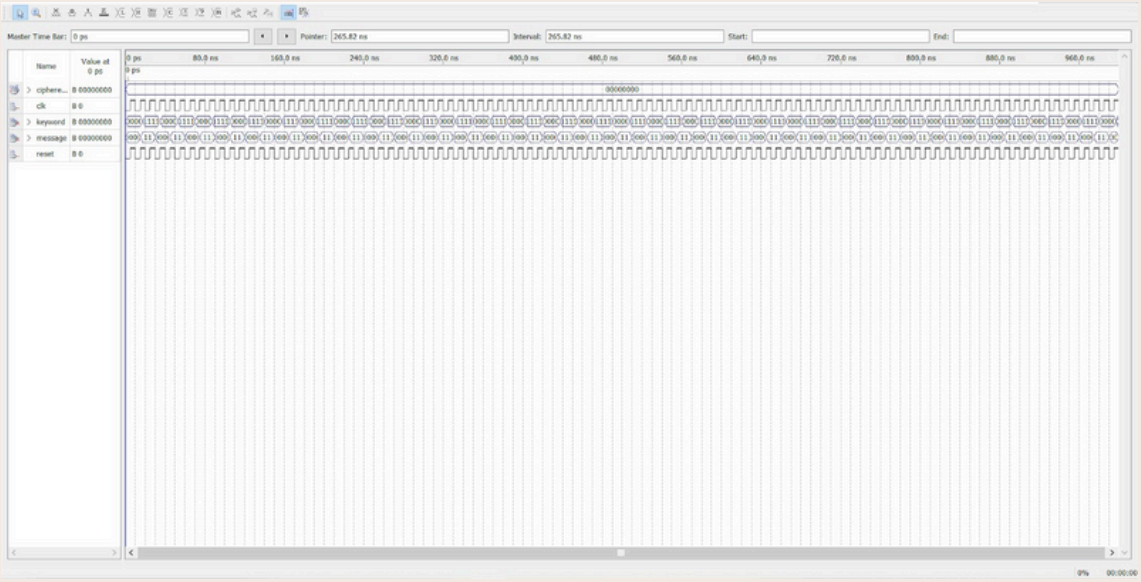
```vhdl
        if reset = '1' then
            -- Initialisation
            shifted_message <= (others => '0');
        elsif rising_edge(clk) then
            -- Traitement du message
            keyword_temp <= unsigned(keyword);
            message_temp <= unsigned(message);

            -- Ajout du mot-clé au message bit à bit
            for i in 0 to N-1 loop
                sum_temp := ('0' & message_temp) + ('0' & keyword_temp);  -- Ajout du bit de retenu
                shifted_message(i) <= sum_temp(N-1);
                message_temp <= message_temp(N-2 downto 0) & '0';  -- Décalage du message à gauche
            end loop;
        end if;
    end process;

    -- Sortie du message chiffré
    ciphered_message <= std_logic_vector(shifted_message);

end architecture Behavioral;
```

Explication du code:

Le code VHDL fourni met en œuvre un chiffrement de César avec clé, un processus permettant de crypter un message en utilisant un mot-clé spécifié. L'entité `KeyedCaesarCipher` est définie avec des ports pour l'horloge, le signal de réinitialisation, le mot-clé de chiffrement, le message à chiffrer, et le message chiffré en sortie. L'architecture `Behavioral` décrit le comportement du chiffrement de César. Dans le processus principal, le mot-clé et le message sont convertis en types de données `unsigned` pour permettre les opérations arithmétiques. Ensuite, chaque bit du message est parcouru, et pour chaque bit, le mot-clé est ajouté au message bit à bit. Cette addition est effectuée en prenant en compte les zéros de retenue. Le résultat partiel est stocké dans `shifted_message`, qui représente le message chiffré. Finalement, ce message chiffré est converti en `std_logic_vector` et assigné à la sortie `ciphered_message`. En résumé, ce code permet de chiffrer un message en utilisant un mot-clé spécifique selon l'algorithme du chiffrement de César avec clé en VHDL.

Résultats


message[7..0]
message_temp[7..0]
0:6
D
CLK
ENA
Q
Add0
A[7..0]
B[7..0]
+
OUT[7..0]
shifted_message[7..0]
D
CLK
CLR
Q
ciphered_message[7..0]
keyword_temp[7..0]
keyword[7..0]
clk
reset



2. ALGORITHME DE DECHIFFREMENT

Pour déchiffrer le message, l'algorithme inverse est appliqué comme suit :

1. Recréer l'Alphabet Modifié :
   – Utiliser le mot-clé pour recréer l'alphabet modifié, tel qu'il a été utilisé lors du chiffrement.

2. Trouver la Position dans l'Alphabet Modifié :
   – Pour chaque lettre du message chiffré, trouver sa position dans l'alphabet modifié.

3. Remplacer par la Lettre d'Origine:
   – Pour chaque position trouvée, remplacer la lettre chiffrée par la lettre correspondante dans l'alphabet d'origine.

Fonctionnement du Déchiffrement

Ce processus inverse permet de retrouver le message original à partir du message chiffré, en annulant l'effet du chiffrement appliqué avec le mot-clé.

Entrées :
– `keyword` : Un mot-clé de N bits, identique à celui utilisé pour le chiffrement.
– `ciphered_message` : Un message chiffré de N bits.

Processus :
1. Convertir `keyword` et `ciphered_message` en vecteurs de type unsigned.
2. Soustraire `keyword` de `ciphered_message`.

Sortie :
– Le message déchiffré est obtenu après la soustraction, et il est renvoyé comme résultat du processus de déchiffrement.

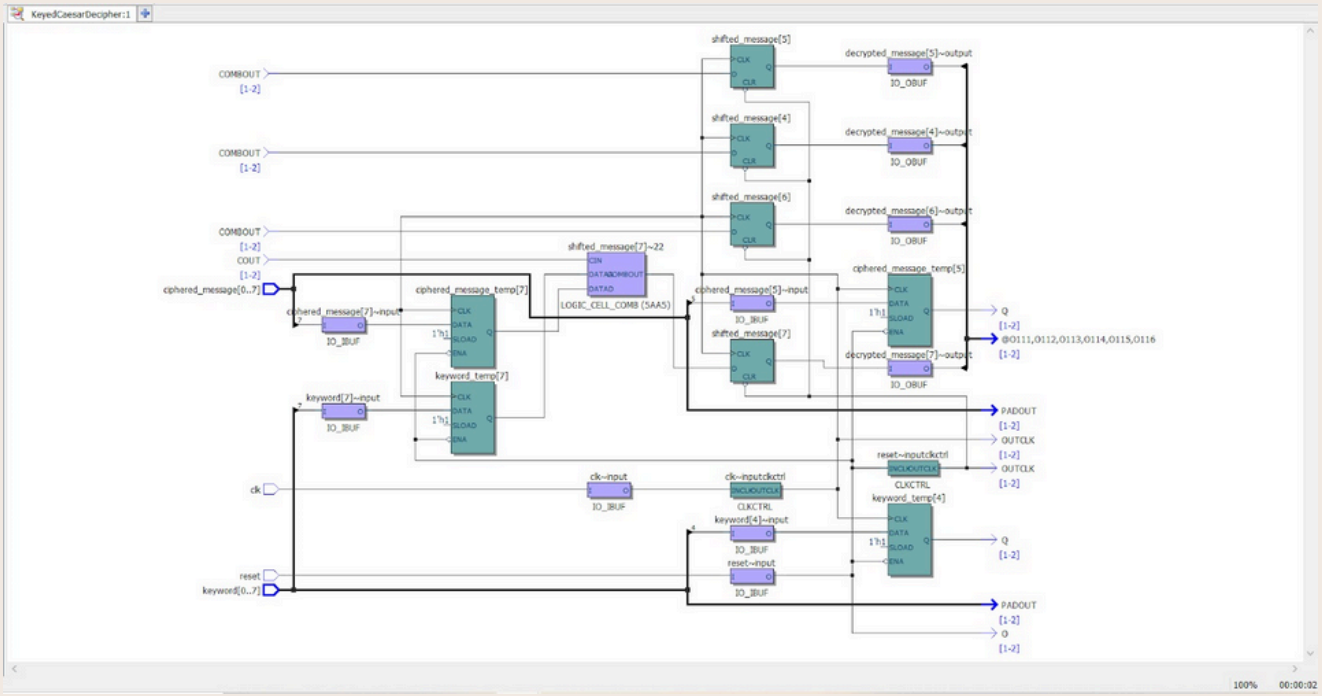
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity KeyedCaesarDecipher is
    generic (
        N : integer := 8   -- Taille des mots-clés et des messages (en bits)
    );
    port (
        clk : in std_logic;
        reset : in std_logic;
        keyword : in std_logic_vector(N-1 downto 0);
        ciphered_message : in std_logic_vector(N-1 downto 0);
        decrypted_message : out std_logic_vector(N-1 downto 0)
    );
end entity KeyedCaesarDecipher;

architecture Behavioral of KeyedCaesarDecipher is

    signal keyword_temp, ciphered_message_temp, shifted_message : unsigned(N-1 downto 0);

begin

    process(clk, reset)
    begin
        if reset = '1' then
            -- Initialisation
```

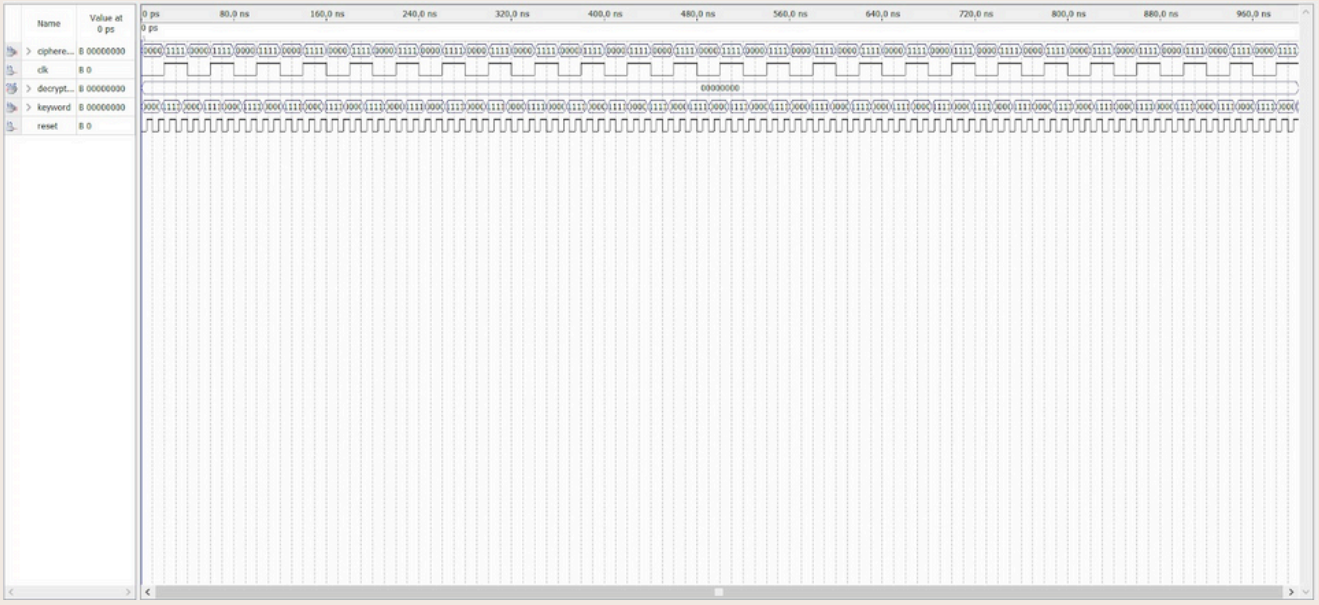
```vhdl
        if reset = '1' then
            -- Initialisation
            shifted_message <= (others => '0');
        elsif rising_edge(clk) then
            -- Convertir les mots-clés et messages chiffrés en type unsigned
            keyword_temp <= unsigned(keyword);
            ciphered_message_temp <= unsigned(ciphered_message);

            -- Soustraction du mot-clé du message chiffré
            shifted_message <= ciphered_message_temp - keyword_temp;
        end if;
    end process;

    -- Sortie du message déchiffré
    decrypted_message <= std_logic_vector(shifted_message);

end architecture Behavioral;
```

Explication du code:

Le code VHDL fourni implémente le processus de déchiffrement du chiffrement de César avec clé. L'entité `KeyedCaesarDecipher` est définie avec des ports pour l'horloge, le signal de réinitialisation, le mot-clé utilisé pour le chiffrement, le message chiffré et le message déchiffré en sortie. Dans l'architecture `Behavioral`, un processus est déclenché par une montée d'horloge ou un signal de réinitialisation. À chaque front montant de l'horloge, le mot-clé et le message chiffré sont convertis en types de données `unsigned`. Ensuite, le mot-clé est soustrait du message chiffré pour obtenir le message déchiffré. Le résultat est stocké dans `shifted_message`, qui représente le message déchiffré. Finalement, ce message déchiffré est converti en `std_logic_vector` et assigné à la sortie `decrypted_message`. En résumé, ce code permet de déchiffrer un message chiffré en utilisant le même mot-clé qui a été utilisé pour le chiffrement, en suivant le principe de l'opération inverse de l'algorithme de chiffrement de César avec clé.

Résultats


KeyedCaesarDecipher:1
COMBOUT
[1-2]
COUT
ciphered_message[0..7]
shifted_message[5]
shifted_message[4]
shifted_message[6]
shifted_message[7]~22
LOGIC_CELL_COMB (SAA5)
ciphered_message_temp[7]
ciphered_message[7]~input
keyword_temp[7]
keyword[7]~input
IO_IBUF
ciphered_message[5]~input
shifted_message[7]
decrypted_message[5]~output
decrypted_message[4]~output
decrypted_message[6]~output
decrypted_message[7]~output
IO_OBUF
ciphered_message_temp[5]
@O111,0112,0113,0114,0115,0116
PADOUT
OUTCLK
reset~inputclkctrl
CLKCTRL
clk
clk~input
clk~inputclkctrl
keyword_temp[4]
keyword[4]~input
reset
reset~input
keyword[0..7]
100%
00:00:02


Name
Value at 0 ps
0 ps
80,0 ns
160,0 ns
240,0 ns
320,0 ns
400,0 ns
480,0 ns
560,0 ns
640,0 ns
720,0 ns
800,0 ns
880,0 ns
960,0 ns
ciphere...
clk
decrypt...
keyword
reset
B 00000000
B 0
00000000

06

Conclusion

En conclusion, le chiffrement de César avec clé représente une évolution significative de la méthode de cryptage classique. Inspiré par le chiffre de César historique, ce système introduit une clé variable pour déterminer le décalage de chaque lettre, renforçant ainsi la sécurité des communications chiffrées.

Cette approche offre une meilleure protection contre les attaques par force brute et les analyses de fréquence, tout en conservant la simplicité et la compréhensibilité de l'algorithme d'origine. En utilisant une clé plus longue et complexe, il est possible d'améliorer encore la robustesse du chiffrement.

L'implémentation en VHDL de l'algorithme de chiffrement et de déchiffrement illustre de manière pratique la manière dont ces concepts peuvent être appliqués dans un environnement de logique numérique. Cette étude fournit une base solide pour comprendre les principes fondamentaux de la cryptographie et développer des systèmes de cryptage plus avancés et sécurisés dans le monde numérique d'aujourd'hui.