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

L'implémentation d'un générateur de nombres aléatoires cryptographiquement sécurisé en VHDL

Encadrée par :

Prof MOUMNI

Réaliser par :

Oumaima El Amoud

Wafae Es-Somid

Meryem Afandi

Introduction au générateur de nombres aléatoires (RNG)

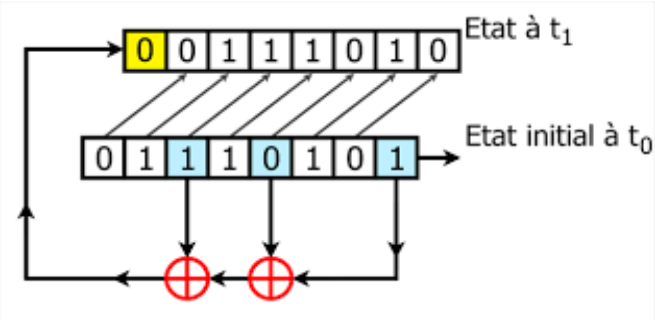
Dans un monde de plus en plus connecté, la sécurité des données et la cryptographie jouent un rôle crucial dans la protection des informations sensibles. Les générateurs de nombres aléatoires (RNG) sont des composants fondamentaux de nombreux systèmes cryptographiques. Ils sont utilisés pour générer des clés cryptographiques, des vecteurs d'initialisation, des salages pour les mots de passe, et bien plus encore. La qualité et la sécurité des nombres aléatoires produits déterminent en grande partie la robustesse des systèmes de cryptage.

Un générateur de nombres aléatoires cryptographiquement sécurisé (CSPRNG) doit satisfaire à plusieurs critères rigoureux : il doit être imprévisible, statistiquement robuste et résistant aux attaques cryptographiques. Contrairement aux générateurs de nombres pseudo-aléatoires classiques, les CSPRNG utilisent des méthodes avancées pour assurer que les séquences de nombres produites ne peuvent pas être facilement devinées ou reproduites.

Dans ce projet, nous nous concentrons sur l'implémentation d'un CSPRNG en utilisant le langage VHDL (VHSIC Hardware Description Langua ge). Le VHDL est un langage de description de matériel utilisé pour modéliser et simuler des circuits numériques. Il est largement utilisé dans le domaine de la conception de circuits intégrés et des systèmes embarqués.Conception du RNG en VHDL

Choix de l'Algorithme

Pour garantir la sécurité cryptographique, nous avons choisi d'implémenter un générateur de nombres aléatoires basé sur un algorithme de rétroaction linéaire (LFSR)



Un algorithme de rétroaction linéaire (LFSR, Linear Feedback Shift Register) est un type de registre à décalage utilisé en cryptographie pour générer des séquences pseudo-aléatoires. Il est constitué d'un ensemble de registres (ou de bits) qui sont décalés à chaque étape, avec certains bits utilisés comme entrée pour une fonction de rétroaction qui détermine le bit à insérer à la position de décalage. Les LFSR sont largement utilisés dans les applications où une séquence pseudo-aléatoire est nécessaire, comme le chiffrement ou la génération de codes

.

Architecture du CSPRNG

L'architecture proposée pour le CSPRNG en VHDL comprend les composants suivants :

LFSR (Linear Feedback Shift Register) : Génère des séquences pseudo-aléatoires.

Post-traitement cryptographique : Applique des opérations cryptographiques pour renforcer la sécurité des nombres générés.

Contrôle et Synchronisation : Gère les signaux de contrôle pour le bon fonctionnement du générateur.

LFSR (Linear Feedback Shift Register)

L'élément central de notre générateur est le LFSR. Il s'agit d'un registre à décalage où les nouveaux bits sont générés en appliquant une fonction de rétroaction sur les bits existants. La fonction de rétroaction est généralement une combinaison XOR des bits du registre.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LFSR is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           random_out : out STD_LOGIC_VECTOR (7 downto 0));
end LFSR;

architecture Behavioral of LFSR is
    signal lfsr_reg : STD_LOGIC_VECTOR(7 downto 0) := "10101010"; -- Initial seed
    signal feedback : STD_LOGIC;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            lfsr_reg <= "10101010"; -- Reset seed value
        elsif rising_edge(clk) then
            feedback <= lfsr_reg(7) xor lfsr_reg(5) xor lfsr_reg(4) xor lfsr_reg(3); -- Feedback polynomial
            lfsr_reg <= feedback & lfsr_reg(7 downto 1); -- Shift and insert feedback
        end if;
    end process;
    random_out <= lfsr_reg;
end Behavioral;
```

Dans cette implémentation, un LFSR à rétroaction XOR est utilisé pour générer un flux de nombres pseudo-aléatoires. Chaque fois que le signal d'horloge `clk` monte, le registre à décalage est décalé d'une position vers la gauche, et la nouvelle valeur du bit de droite est calculée en effectuant une opération XOR sur certains des bits du registre, selon le polynôme de rétroaction chois

Post-traitement Cryptographique

Pour améliorer la sécurité des nombres générés, nous ajoutons une étape de post-traitement. Cette étape peut inclure des techniques telles que le mélange des bits générés par plusieurs LFSR ou l'utilisation d'une fonction de hachage cryptographique.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity RNG_Control is
5       Port ( clk : in STD_LOGIC;
6              reset : in STD_LOGIC;
7              lfsr_output : out STD_LOGIC_VECTOR(7 downto 0);
8              key : in STD_LOGIC_VECTOR(7 downto 0);
9              random_secure : out STD_LOGIC_VECTOR(7 downto 0));
10  end RNG_Control;
11
12  architecture Behavioral of RNG_Control is
13      component LFSR is
14          Port ( clk : in STD_LOGIC;
15                 reset : in STD_LOGIC;
16                 random_out : out STD_LOGIC_VECTOR (7 downto 0));
17      end component;
18
19  component PostProcessing is
20          Port ( lfsr_output : in STD_LOGIC_VECTOR(7 downto 0);
21                 key : in STD_LOGIC_VECTOR(7 downto 0);
22                 random_secure : out STD_LOGIC_VECTOR(7 downto 0));
23  end component;
24
25  signal lfsr_out : STD_LOGIC_VECTOR(7 downto 0);
26
27  begin
28      U1: LFSR port map (clk => clk, reset => reset, random_out => lfsr_out);
29      U2: PostProcessing port map (lfsr_output => lfsr_out, key => key, random_secure => random_secure);
30  end Behavioral;
```

Contrôle et Synchronisation

Un module de contrôle et de synchronisation est nécessaire pour orchestrer les opérations du LFSR et du post-traitement. Ce module génère les signaux d'horloge et de réinitialisation nécessaires pour synchroniser les différentes parties du CSPRNG.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RNG_Control is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           lfsr_output : out STD_LOGIC_VECTOR(7 downto 0);
           key : in STD_LOGIC_VECTOR(7 downto 0);
           random_secure : out STD_LOGIC_VECTOR(7 downto 0));
end RNG_Control;

architecture Behavioral of RNG_Control is
    component LFSR is
        Port ( clk : in STD_LOGIC;
               reset : in STD_LOGIC;
               random_out : out STD_LOGIC_VECTOR (7 downto 0));
    end component;

component PostProcessing is
        Port ( lfsr_output : in STD_LOGIC_VECTOR(7 downto 0);
               key : in STD_LOGIC_VECTOR(7 downto 0);
               random_secure : out STD_LOGIC_VECTOR(7 downto 0));
end component;

signal lfsr_out : STD_LOGIC_VECTOR(7 downto 0);

begin
    U1: LFSR port map (clk => clk, reset => reset, random_out => lfsr_out);
    U2: PostProcessing port map (lfsr_output => lfsr_out, key => key, random_secure => random_secure);
end Behavioral;
```

```
component PostProcessing is
        Port (
            lfsr_output : in STD_LOGIC_VECTOR(7 downto 0);
            key : in STD_LOGIC_VECTOR(7 downto 0);
            random_secure : out STD_LOGIC_VECTOR(7 downto 0)
        );
end component;

signal lfsr_out : STD_LOGIC_VECTOR(7 downto 0);
begin
    -- Instantiate LFSR component
    U1: LFSR
        port map (
            clk => clk,
            reset => reset,
            random_out => lfsr_out
        );

    -- Instantiate PostProcessing component
    U2: PostProcessing
        port map (
            lfsr_output => lfsr_out,
            key => key,
            random_secure => random_secure
        );
    -- Map the internal LFSR output to the entity's output port
    lfsr_output <= lfsr_out;
end Behavioral;
```