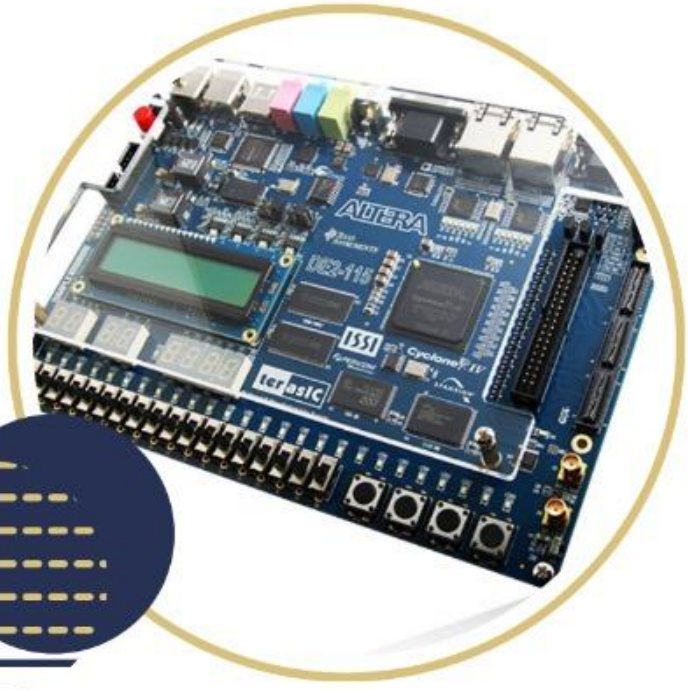
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA

MINI PROJET

MODULE DE CRYPTAGE RSA SUR UN CIRCUIT DE TYPE FPGA



Preparé par :
Amine Amhache
Yahya Elhousni
Safouan Lagram

Preparé pour:
Pr. S.EL MOUMNI

Introduction générale

Le RSA est un algorithme de cryptage asymétrique, considéré actuellement comme l'un des moyens les plus sûrs pour sécuriser les transferts de données. Le RSA est un protocole garantissant une sureté proportionnelle à la taille des clés utilisées.

D'un point de vue technique, l'implémentation de ce protocole sur les systèmes embarqués actuels pose un défi majeur, en raison des limitations de mémoire et de puissance de calcul de ces systèmes, qui ne sont pas suffisantes pour répondre aux exigences de sécurité de cet algorithme.

Pour surmonter ces contraintes, notre démarche a été de proposer une solution en adaptant l'algorithme RSA à partir de la technique de multiplication modulaire de Montgomery. Cette approche permet une multiplication modulaire efficace en diminuant la taille du résultat produit par deux grands nombres, ce qui pourrait potentiellement réduire la taille du résultat d'une opération de puissance.

Dans la perspective d'aboutir à un protocole RSA réduit par la multiplication de Montgomery, on doit se baser sur l'algorithme Squart and Multiply. Ce dernier nous permet la réduction de la taille d'une puissance par la multiplication modulaire de Montgomery.

Le but de la réduction de la taille du RSA est d'avoir un protocole qui pourra s'implémenter sur des circuits FPGA et de les utiliser dans des systèmes embarqués afin de décharger le processeur de cette tâche qui nécessite une grande puissance de calcul.

Cryptographie et étude algorithmique

I) Introduction :

Aujourd'hui, la cryptographie est largement utilisée dans les secteurs des télécommunications et des systèmes embarqués pour sécuriser les données et les informations. Cette sécurisation est souvent réalisée à l'aide d'ordinateurs via diverses méthodes de cryptage. Cependant, un défi majeur consiste à réduire la complexité de ces méthodes en les combinant avec d'autres approches mathématiques, afin de les adapter aux systèmes embarqués qui disposent de ressources limitées.

Dans cette section, nous explorerons la cryptographie dans son ensemble, en mettant l'accent sur le cryptage asymétrique RSA, l'algorithme de Montgomery pour la réduction modulaire, et l'algorithme Squart and Multiply.

II) La cryptographie :

La cryptographie est l'ensemble des techniques (algorithmes, matériels, logiciels) permettant de protéger une communication au moyen d'un code secret. L'information à communiquer est modifiée par l'émetteur de façon à la rendre compréhensible uniquement par le destinataire.

De nos jours, on distingue deux types de cryptographie : la cryptographie symétrique et la cryptographie asymétrique.

a) Cryptographie symétrique :

C’est la cryptographie la plus ancienne également dite à clé secrète. Elle est basée sur l’échange préalable d’une clé secrète commune. Cette clé a une taille variable, elle peut être sous la forme d’un chiffre,

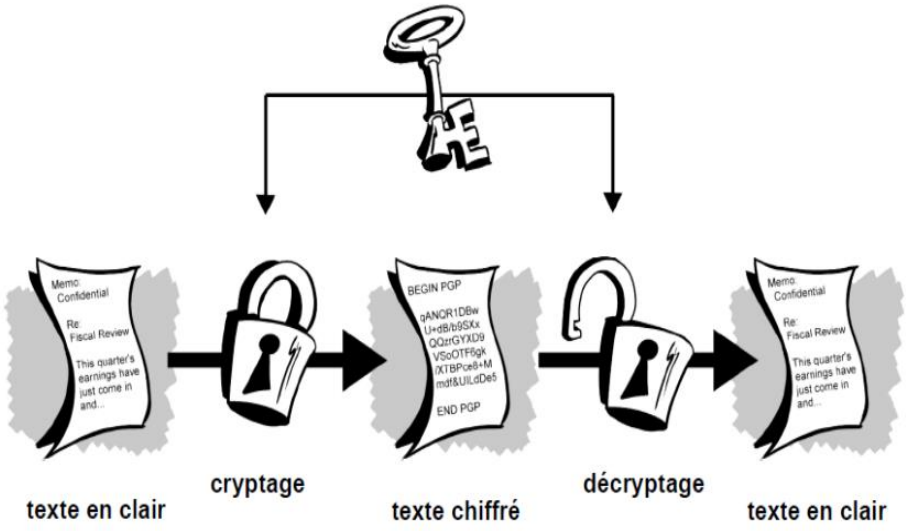


Figure (II.1) principe de la cryptographie symétrique

d’une lettre, d’un livre de code . . . Avec cette clé commune, les communicants peuvent crypter ou décrypter un message. Aujourd’hui, de nombreux algorithmes de cryptage à clé secrète sont utilisés tels que : le DES, le triple DES, le AES. . . . Ces algorithmes sont basés sur des permutations et des transformations de blocs et n’utilisent pas d’arithmétique modulaire. La figure (II.1) illustre le principe de la cryptographie symétrique.

b) Cryptographie asymétrique :

C’est une cryptographie moderne qui a vu le jour dans les années soixante-dix. Au lieu d’une seule clé, il y a deux clés différentes. Une clé publique (qui sera diffusée) pour crypter et une clé privée (gardée secrète) pour décrypter. Chaque récepteur peut à l’aide de la clé publique du destinataire crypter un message et seule le destinataire en possession de

la clé privée peut le décrypter. Le principe de la cryptographie à clé publique, est illustré dans la figure (II.2).

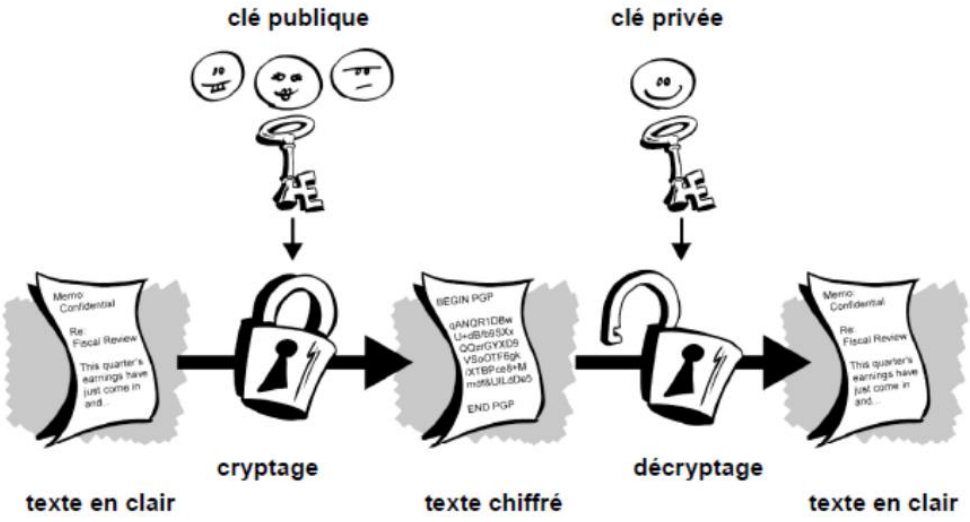


Figure II.2 principe de la cryptographie asymétrique

III)Le cryptage asymétrique 'RSA' :

RSA est sûrement le plus connu des algorithmes cryptographiques à clé publique. En proposant ce protocole en 1978, Rivest Shamir et Adelman ouvrent un nouveau pan de la cryptographie (la cryptographie asymétrique). La cryptographie asymétrique se nomme ainsi car elle est à sens unique. La cryptographie asymétrique utilise une clé (publique) pour crypter et une autre clé (secrète) pour décrypter.

L'idée est qu'une personne ''A'' crée deux clés qui lui sont associées. Une clé publique est utilisée par toute personne qui désire lui envoyer un message. Cette clé est associée à un protocole et, puisqu'elle est publique, toute personne peut crypter un message pour l'envoyer à ''A''. Une seconde clé, secrète et connue uniquement par ''A'', permet à ''A'' d'utiliser une fonction trappe pour décrypter les messages qu'il a reçus. En fait, après avoir crypté un message avec la clé publique, l'émetteur n'est pas capable de décrypter le message qu'il a crypté. C'est ce qui rend sûr la

cryptographie asymétrique et c'est ce qui permet à ''A'' de publier sa clé publique, elle ne permet pas de décrypter le message.

Rivest, Shamir et Adelman ont proposé un protocole basé sur ce principe de cryptographie à clé publique. Ils utilisent comme fonction trappe la factorisation, il est "facile" de faire la multiplication de deux nombres p et q mais "difficile" de factoriser un produit p*q pour récupérer les entiers p et q.

a) Création des clés RSA :

Algorithme :

- La personne ''A'' choisit deux grands nombres premiers p et q.
- Calculer n = p*q et Ø(n) = (p-1)*(q-1).
- Choisir un entier e premier avec Ø(n).
- Calculer b = (inverse de e) mod Ø(n).

La paire (n, e) est la clé publique du RSA.

La paire (n, b) est la clé privée du RSA.

Maintenant la personne ''A'' peut proposer sa clé publique afin de recevoir des messages sécurisés.

b) Protocole RSA :

Le protocole RSA va permettre à n'importe quelle personne qui connait la clé publique, de crypter un message afin de l'envoyer à la personne ''A'', et à son tour la personne ''A'' peut décrypter le message.

c) Le chiffrement avec le RSA :

La fonction de chiffrement RSA a pour entrées la clé publique du RSA (n, e) et un texte en clair M. Le texte va être découpé en blocs de K bits, tel que $2^k \leq n$. (M = m1, m2, …. ml) et chaque bloc est traité comme un entier m.

La sortie de la fonction de chiffrement est un message crypté C (C = c1, c2, …… cl), avec ci = mi^e mod n.

d) Le déchiffrement avec le RSA :

La fonction de déchiffrement a pour entrées la clé privée (n, b) et le texte chiffré C (C = c1, c2, ……, cl). Donc pour retrouver le message M. (M = m1, m2, …. ml) en clair on calcule mi = ci^{b} b mod n.

IV) Algorithme de Montgomery pour la multiplication modulaire :

En 1985, Peter Montgomery franchit un cap dans la réduction modulaire. Il transforme la réduction modulo n en une division par une puissance de la base. En pratique, une division par une puissance de la base revient à un décalage, qui ne coûte rien.

✓ Présentation du problème :

Etant donnés trois nombres entiers a, b, n, le problème est de calculer le plus rapidement possible a*b mod n en un minimum d'espace.

V) Transformation de Montgomery :

- **Définition de la transformation :**

Soit n le modulo intervenant dans l'opération. On supposera désormais que n est un nombre impair. Le nombre de bits de n est l'entier k tel que : 2^{k-1} ≤ n < 2^{k} . On appellera r le nombre 2^{k} . Comme n est impair, r est premier avec n et donc inversible modulo n. On notera r-1 l'inverse de r modulo n.

Soit mont (transformation de Montgomery) donc :

mont(a) = a*r mod n.

Cette application mont (multiplication par r modulo n) est une bijection puisque r est inversible modulo n et on peut écrire :

a = mont(a)* r^{-1}mod n.

- **Comment calculer la transformation d'un produit ?**

On calcule une fois pour toute r^{-1}, tel que :

r r^{-1}= 1 mod n.

Et on calculera aussi v tel que :

r r^{-1}– v n = 1.

Une fois ces coefficients calculés on suit les étapes suivantes :

- ✓ s = mont(a) * mont(b).
- ✓ t = (s*v) mod r.
- ✓ res_mont = (s + t*n) / r.

Les résultats de la multiplication obtenus sont dans le domaine de Montgomery. Pour avoir le résultat dans le domaine naturel on fait la transformation suivante :

res = res_mont *r^{-1} mod n.

VI) Algorithme Square and Multiply

La première façon de calculer une puissance n e , est de multiplier n par lui même e fois. Cependant, il existe des méthodes bien plus efficaces, où le nombre d'opérations nécessaires n'est plus de l'ordre de e.

L'algorithme square and multiply est l'une des méthodes qui calculent une puissance S = X^{-E} mod n, en un minimum d'opérations.

a) Description de l'algorithme

On initialise les paramètres d'entrée et de sortie :

S (0) = 1 ; (élément neutre).

Z (0) = X.

E = $\sum_{i=0}^{n-1} e_i 2^i$

On fait une boucle allant de 0 jusqu'à (n-1) et on calcule :

Z (i+1) =$Z_i^{\,2}$ mod n.

On fait un test sur chaque bit de e et si e (i) = 1 on calcule :

S (i+1) = S (i) *Z (i) mod n.

Si non on calcule :

S (i+1) = S (i).

VII) Conclusion :

En cryptographie asymétrique, de grands nombres sont utilisés (plusieurs centaines de bits pour le RSA). Dans certains systèmes embarqués ou l'espace est limité, le calcul du produit de deux grand nombres est problématique. Dans ce contexte nous allons proposer un nouveau protocole RSA qui permet de crypter un message tout en réduisant la taille de ce dernier afin de pouvoir l'implémenter sur un circuit FPGA.

Description en VHDL :

VIII) Introduction :

Dans le chapitre précèdent nous avons décrit la multiplication modulaire de Montgomery, le protocole de cryptage RSA et l'algorithme Squart and Multiply. En se basant sur ces derniers on va d'abord présenter notre protocole RSA, auquel nous avons abouti en se basant sur la multiplication modulaire de Montgomery et l'algorithme Squart and Multiply dans la perspective de réduire la taille afin de l'implémenter sur un FPGA. Par la suite on va programmer notre algorithme RSA en utilisant le langage VHDL et faire quelques simulations sur Quartus.

IX) Présentation de notre protocole RSA

Notre protocole consiste à chiffrer un message M avec le protocole RSA qui est un protocole simple, fiable ayant fait ses preuves dans la cryptographie moderne ; surtout qu'il est facile à programmer sur un ordinateur. Mais ce qui pose problème c'est la taille mémoire nécessaire à son implémentation. Afin d'essayer de remédier à ça, on va utiliser la multiplication modulaire de Montgomery et l'algorithme Squart and Multiply qu'on a présentés dans le chapitre I.

a) Description de notre protocole RSA

On veut crypter un message m, en utilisant le chiffrement RSA. On va avoir à la sortie un message crypté c, tel que c = me mod n. Ce type de chiffrement peut nécessiter un grand espace mémoire et de calcul. Pour essayer de réduire cet espace, on va opter pour la multiplication modulaire de Montgomery. Le protocole que nous proposons est basé sur l'algorithme square and multiply et la multiplication de Montgomery, qui va calculer C = $M^{\wedge e}$ mod n et qui va donner un résultat dans le domaine de Montgomery pour aboutir à une puissance qu'on appellera la puissance de Montgomery qui va nous donner le cryptage RSA en réduisant l'espace utilisé.

∧ : puissance de Montgomery.

b) Algorithme de notre protocole RSA

On commence par calculer les paramètres de l'algorithme de Montgomery comme on l'a expliqué dans le chapitre I, dont on aura besoin aussi dans notre algorithme de puissance.

- On transforme m vers le domaine de Montgomery :

  M = m*r mod n. (m = mont (m)).

- On écrit e sous forme binaire :

  E = écriture binaire de e.

- On initialise les paramètres d'entrée et de sortie :

  C (0) = r-n.

  z (0) = M.

- On fait une boucle allant de 0 jusqu'à (n-1) et on calcule :

  z (i+1) = z (i) ⋇ z (i) mod n.

- On fait un test sur chaque bit de e et si e (i) = 1 on calcule :

  C (i+1) = C (i) ⋇ z (i) mod n.

- Sinon on calcule :

  C (i+1) = c (i).

La sortie de notre module est un résultat C qui est dans le domaine de Montgomery, donc on doit le remettre dans le domaine naturel comme suit :

⋇ : multiplication de Montgomery.

c = C * r-1 mod n.

I. Implémentation

Dans ce qui suit nous allons présenter l'implémentation de notre protocole matériel décrit avec le langage de description du matériel VHDL en utilisant Le navigateur de projet Quartus.

Nous commençons ainsi par une brève description du langage VHDL, ainsi que le navigateur de projet Quartus. Par la suite, nous présenterons l'algorithme de notre protocole RSA sous VHDL, et testeront sa fonctionnalité par des simulations.

a. Langage de description VHDL

VHDL (VHSIC Hardware Description Language) est un langage de description pour les circuits numériques. Il a été développé dans le cadre du projet VHSIC (Very high speed integrated circuits) commandité par le département de la défense américaine DoD au début des années quatre-vingts. C'est un standard IEEE depuis 1987 largement utilisé en Europe. Il autorise plusieurs méthodologies de conception (comportemental, flot de données, structurel).

Il est indépendant de la technologie utilisée FPGA, CPLD, ASIC, etc., et permet d'aller d'un niveau d'abstraction très élevé par une description algorithmique jusqu'à un niveau proche du matériel, où l'on décrit le système par un ensemble de portes logiques et d'interconnexions. Entre les deux se trouve le niveau RTL (Register Transfer Level) qui permet de définir le système par une architecture de type machine de Moore ou Mealy. Cette abstraction permet d'ailleurs de simuler sur ordinateur avant de programmer le moindre circuit.

b. Environnement de développement Quartus

Quartus Prime est l'outil de choix pour les concepteurs de circuits numériques, offrant une interface utilisateur intuitive et une suite complète d'outils pour la conception, la simulation et la programmation de FPGA et de CPLD. Avec sa prise en charge des langages de description matérielle tels que VHDL et Verilog, il permet aux utilisateurs de décrire le comportement et la structure de leurs conceptions avec précision. De la spécification initiale à la programmation du dispositif final, Quartus Prime simplifie chaque étape du processus de développement, offrant une intégration transparente avec les FPGA Intel et une documentation exhaustive pour guider les concepteurs à travers chaque étape.

Outre ses fonctionnalités robustes, Quartus Prime est soutenu par une équipe dévouée de support technique et une communauté active d'utilisateurs. Cette combinaison assure aux concepteurs une assistance de qualité tout au long de leurs projets, garantissant une expérience de développement fluide et efficace. En résumé, Quartus Prime est bien plus qu'un simple outil logiciel : il est un partenaire de confiance pour transformer les idées en circuits numériques fonctionnels et innovants.

c. simulation de notre protocole

Avant de passer à la simulation de notre protocole, on doit d'abord définir les paramètres du RSA comme suit :

- n=p×q=13×17=221.
- $\phi(n)$=(p−1)×(q−1)=12×16=192
- *e* est choisi comme un nombre premier avec $\phi(n)$. Supposons e=11.
- Calculons *b*, l'inverse modulaire de *e* modulo $\phi(n)$. Pour e=7 et $\phi(n)$=192, nous trouvons b=35.

Après avoir défini les paramètres du RSA, on peut définir les paramètres de notre protocole :

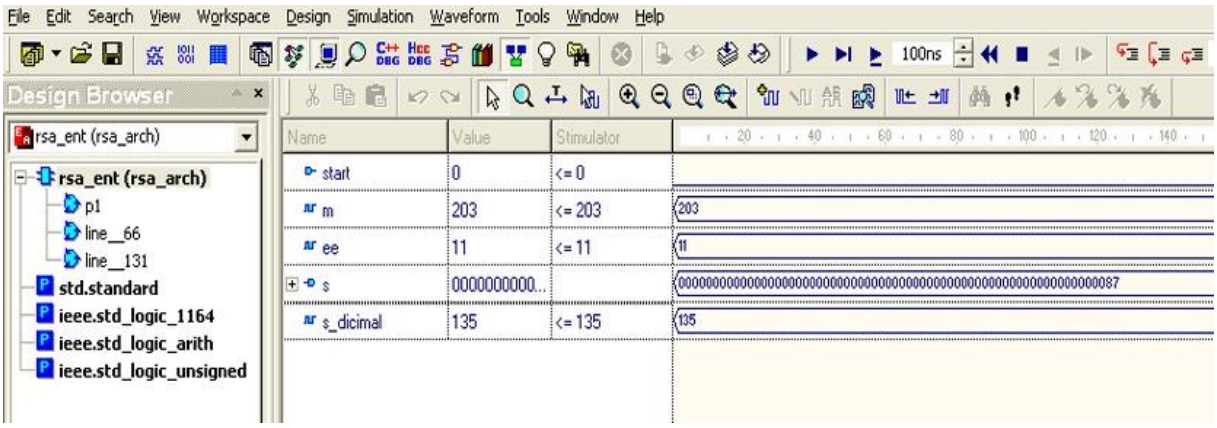
- *R* est la puissance supérieure de n. Pour n=221, R=256.
- Calculons R−1, l'inverse modulaire de *R* modulo n. Pour R=256 et n=221, nous trouvons R−1=120.
- Définissons *v* tel que R×R−1−v×n=1. Dans cet exemple, supposons v=139.
- Choisissons un message *m* à crypter : prenons m=203.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity RSA is
port(
start : in STD_LOGIC;
s : out STD_LOGIC_vector(63 downto 0)
 );
end RSA;
architecture RSA_arch of RSA is
signal res_aff : std_logic_vector(63 downto 0);
signal e : std_logic_vector(63 downto 0);
signal calc : std_logic;
begin
  p1:process (start)
  constant e1: integer := 7;
  begin
    if start = '0' then
       e <= conv_std_logic_vector (e1,64);
       calc <='1';
           else
       calc <= '0';
    end if;
   end process p1 ;
   process(calc)
```

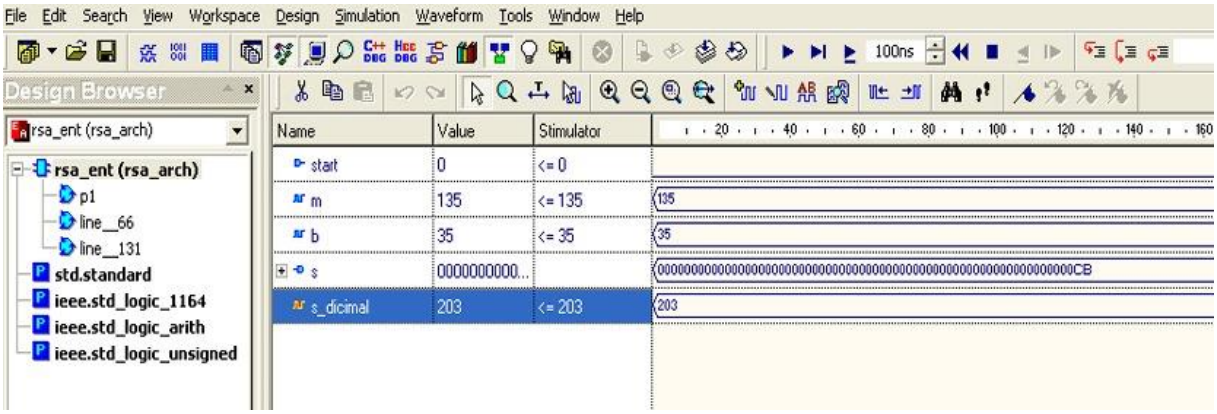
```
        variable p,resul,y,m,z,res : integer;
        constant m1: integer := 203;
constant n: integer := 221 ;
constant r: integer := 256;
constant j: integer := 120 ;
constant v: integer := 139 ;
        begin
          if calc = '1' then
                    m := m1*r mod n;
                          p:= r-n ;
     z:= m ;
                          for i in e'reverse_range loop
                            if e(i) = '1' then
                                    p:= p*z;
                                          y:= p*v mod r;
                                          resul:= (p+y*n)/r;
                                          p:= resul mod n;
                                else
                                   p:= p;
                                end if;
                                z:= z*z;
                                y:= z*v mod r;
                                resul := (z+y*n)/r;
                                z := resul mod n;
                          end loop;
           res := p*j mod n;
```

```
            res_aff <= conv_std_logic_vector (res,64);
            end if;
   end process;
s <= res_aff;
end RSA_arch;
```

Chiffrement avec notre protocole



Déchiffrement avec notre protocole



CONCLUSION

En conclusion, notre travail illustre la recherche de solutions pour adapter l'algorithme RSA aux contraintes des systèmes embarqués, où la disponibilité limitée de mémoire et de puissance de calcul pose des défis significatifs. En explorant des techniques telles que la multiplication modulaire de Montgomery et l'algorithme Square and Multiply, nous avons proposé un protocole RSA optimisé pour ces environnements restreints.

L'implémentation en VHDL de notre protocole RSA a démontré sa faisabilité et son efficacité, offrant ainsi une solution pratique pour sécuriser les communications dans les systèmes embarqués. Cette adaptation de l'algorithme RSA ouvre de nouvelles perspectives pour son utilisation dans un large éventail d'applications, offrant un équilibre entre sécurité et efficacité dans des environnements où les ressources sont limitées.

Nous tenons à exprimer notre profonde gratitude à notre professeur **EL MOUMNI** pour son enseignement dévoué et sa guidance tout au long de ce semestre.