Baptiste MONDION EI23

Conception d'un Système Numérique ASCON128

Introduction

La sécurité des données est un élément clé pour garantir la confidentialité des conversations dans les systèmes de communication numériques. Dans le cadre de ce projet, nous allons utiliser l'algorithme de chiffrement symétrique Ascon128 pour sécuriser efficacement des données textuelles. Développé en 2014 par des chercheurs spécialisés en cryptographie, tels que Christoph Dobraunig, Maria Eichlseder, Florian Mendel et Martin Schläffer, Ascon128 est basé sur une architecture de permutation et de substitution. Il utilise une clé de 128 bits pour crypter les données, offrant ainsi une robustesse optimale en matière de sécurité.

Les performances élevées d'Ascon128 en font un choix idéal pour les applications telles que les communications sécurisées, le stockage de données et les dispositifs IoT. De plus, sa conception légère et flexible lui permet de s'adapter facilement à une grande variété de plateformes, y compris les systèmes embarqués et les appareils IoT.

En somme, Ascon128 est un algorithme de chiffrement symétrique efficace et fiable pour sécuriser les données textuelles dans les systèmes de communication numériques.

Principe

Lorsqu'il est appliqué à du texte en clair et à des données associées, l'algorithme ASCON génère un texte chiffré ainsi qu'un Tag servant de vérification. Lors du processus de déchiffrement, il prend en compte le Tag, le texte chiffré et les données associées pour restituer le texte en clair d'origine. Il réalise des permutations et des opérations XOR lors du chiffrement.

Structure

L'algorithme de chiffrement Ascon128 suit un processus structuré pour chiffrer les données en toute sécurité. Tout d'abord, les données subissent une permutation en trois phases, à savoir l'addition de la constante de ronde, la substitution et la diffusion linéaire. Ensuite, des opérations XOR sont effectuées avec des variables spécifiques pour ajouter une couche supplémentaire de complexité au chiffrement. Les valeurs résultantes sont stockées dans des registres pour enregistrer le chiffrement final et la valeur du Tag.

Une machine à états finis (FSM) est utilisée pour coordonner le processus de chiffrement en sélectionnant et configurant les variables de contrôle pour chaque état. Cela permet une gestion du processus.

Enfin, des compteurs sont utilisés pour suivre le nombre de rondes et de blocs de données traités, garantissant ainsi la cohérence du processus de chiffrement. Ces compteurs sont essentiels pour assurer la synchronisation des différents modules de l'algorithme dans la machine à états finis.

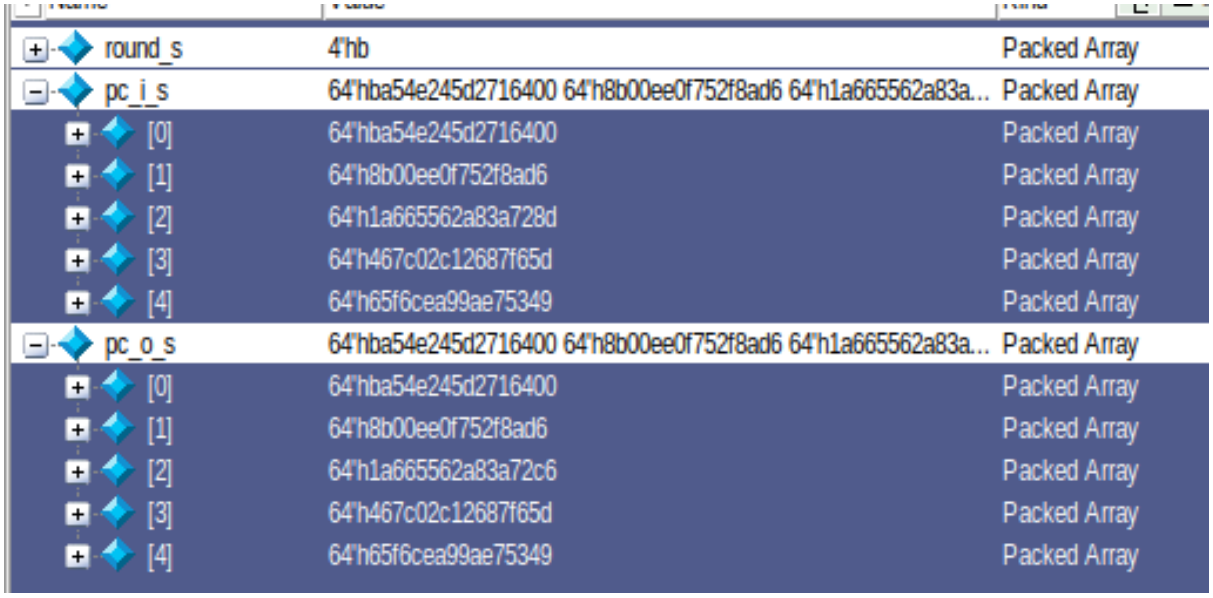
Résultat du projet

Les élements de Ascon

Les Permutations

L'addition de constante Pc

L'addition constante implique l'ajout d'une constante qu'on nomme constante de ronde au second registre de l'état S, qui dépend du nombre de tours déjà effectués par S.



Testbench de Pc

L'addition de constante sera réalisée à l'aide d'un tableau de constantes défini dans le package ascon_pack. Ce tableau, appelé round_constant, contient 12 constantes de 8 bits chacune, définies comme suit :

Chaque constante est ajoutée à l'état de manière cyclique à chaque tour de l'algorithme, en utilisant l'indice du tour pour sélectionner la constante appropriée dans le tableau.

Ronde r de p^{12}	Ronde r de p^6	Constante c_r
0		000000000000000f0
1		000000000000000e1
2		000000000000000d2
3		000000000000000c3
4		000000000000000b4
5		000000000000000a5
6	0	0000000000000000096
7	1	0000000000000000087
8	2	0000000000000000078
9	3	0000000000000000069
10	4	000000000000000005a
11	5	000000000000000004b

Schéma Couche Pc

La couche de substitution Ps

Lors de l'étape de substitution, les 320 bits de l'état S sont modifiés. Les ièmes bits de chacun des 5 registres de S sont combinés pour former un nombre de 5 bits, qui est ensuite remplacé par la valeur correspondante dans le tableau de substitution.

Testbench de Ps

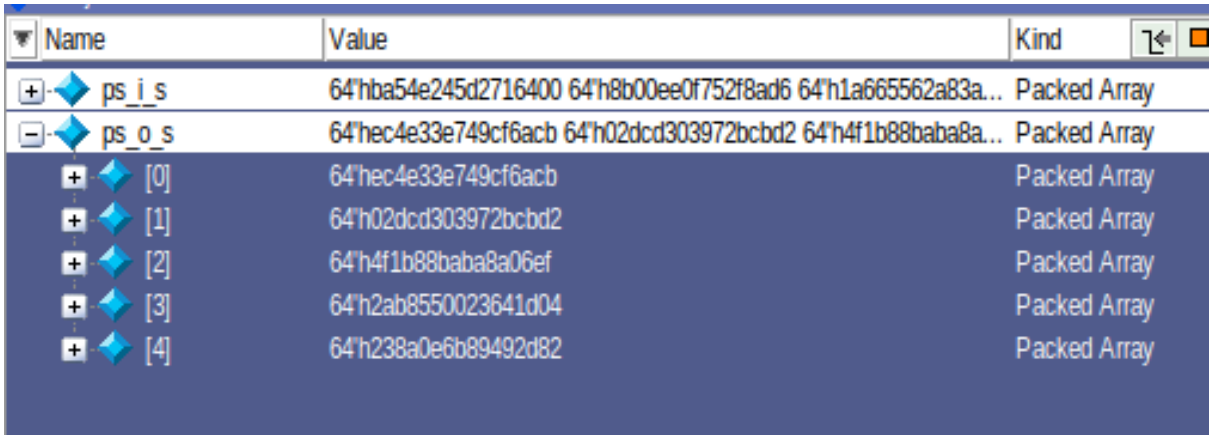
La sbox (substitution box) est un module qui renvoie la valeur correspondante aux 5 bits d'entrée en utilisant le tableau de substitution fourni dans le sujet. Pour construire la sbox, un switch-case est utilisé.

x	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
$S(x)$	04	0B	1F	14	1A	15	09	02	1B	05	08	12	1D	03	06	1C	1E	13	07	0E	00	0D	11	18	10	0C	01	19	16	0A	0F	17

Schéma Couche Ps

La couche de diffusion linéaire Pl

La dernière phase de la permutation dans l'algorithme Ascon est la phase de diffusion linéaire, notée Pl. Cette phase combine un décalage de bits et une opération XOR pour chaque registre de l'état S. Lors du décalage, les bits qui sont décalés vers la droite hors du registre sont réinsérés à gauche du registre (bit de poids fort). Ensuite, une opération XOR est effectuée entre les bits décalés et les bits d'origine dans chaque registre.



Testbench de Pl

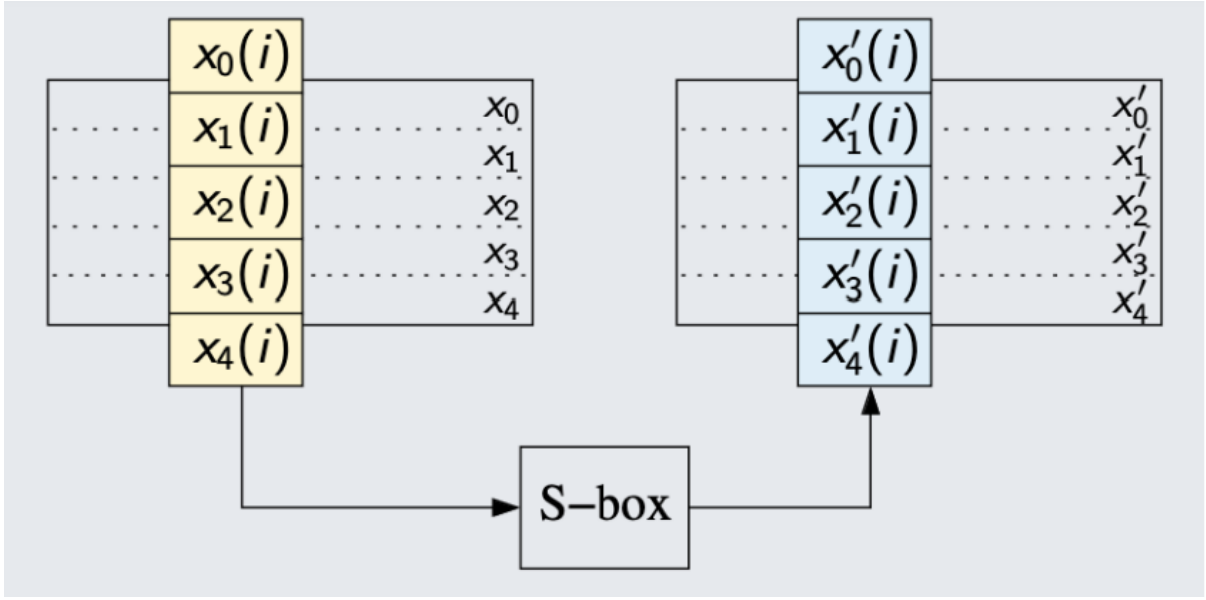
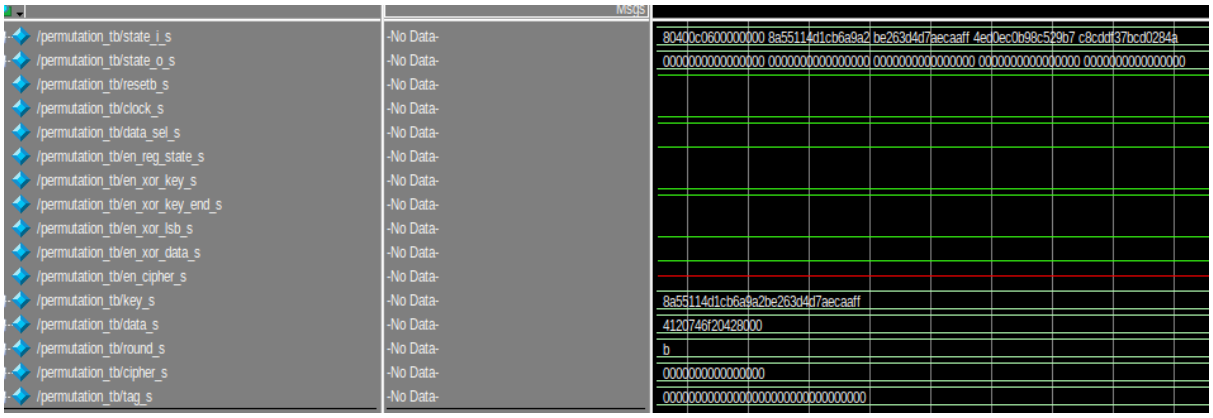


Schéma Couche Pl

La permutation complète

P est donc l'itération de Pc, Ps et Pl.



Testbench de la permutation finale

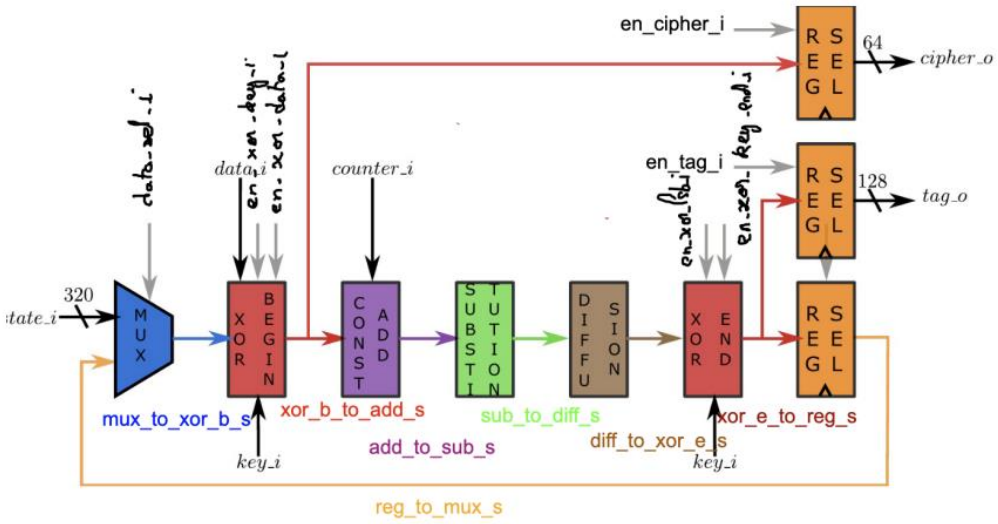
Je n'ai pas réussi à faire marché le testbench pourtant je pense que mon code est juste car c'est simplement envoyer la sortie d'une permutation sur l'entrée de la seconde.

Le multiplexeur et de la D-flip-flop

La permutation nécessite deux éléments essentiels : un multiplexeur et une bascule D. Le multiplexeur est utilisé pour sélectionner l'entrée appropriée pour chaque tour de permutation, en fonction de l'état de sortie du tour précédent ou de l'état initial lors du premier tour. La bascule D, quant à elle, stocke les valeurs de sortie à chaque tour de permutation.

Les XOR

L'étape suivante est l'ajout des XOR à la permutation comme on peut le voir sur le schéma suivant :



Permutation et XOR

Xor_begin

L'opération Xor_begin est utilisée pour absorber le texte clair en tranches de 64 bits (P1, P2, P3) et l'insérer dans la source interne du système. Elle est également utilisée pour récupérer les données associées.

Les différentes opérations XOR initiales sont contrôlées par les deux variables en_xor_data_i et en_data_key_i.

Name	Value	Kind
en_xor_data_s	1'h1	Register
en_xor_key_s	1'h0	Register
state_i_s	64'h9de1e2d04b50bd86 64'h13141c888a702ce4 64'h55aa22c50d2...	Packed Array
state_o_s	64'hdcc196bf6b123d86 64'h13141c888a702ce4 64'h55aa22c50d2...	Packed Array
[0]	64'hdcc196bf6b123d86	Packed Array
[1]	64'h13141c888a702ce4	Packed Array
[2]	64'h55aa22c50d252d83	Packed Array
[3]	64'hef65ff04555a7c13	Packed Array
[4]	64'hac2f83ace5a6d021	Packed Array
data_s	64'h4120746f20428000	Packed Array
key_s	128'h8a55114d1cb6a9a2be263d4d7aecaaff	Packed Array

Testbench de Xor_begin

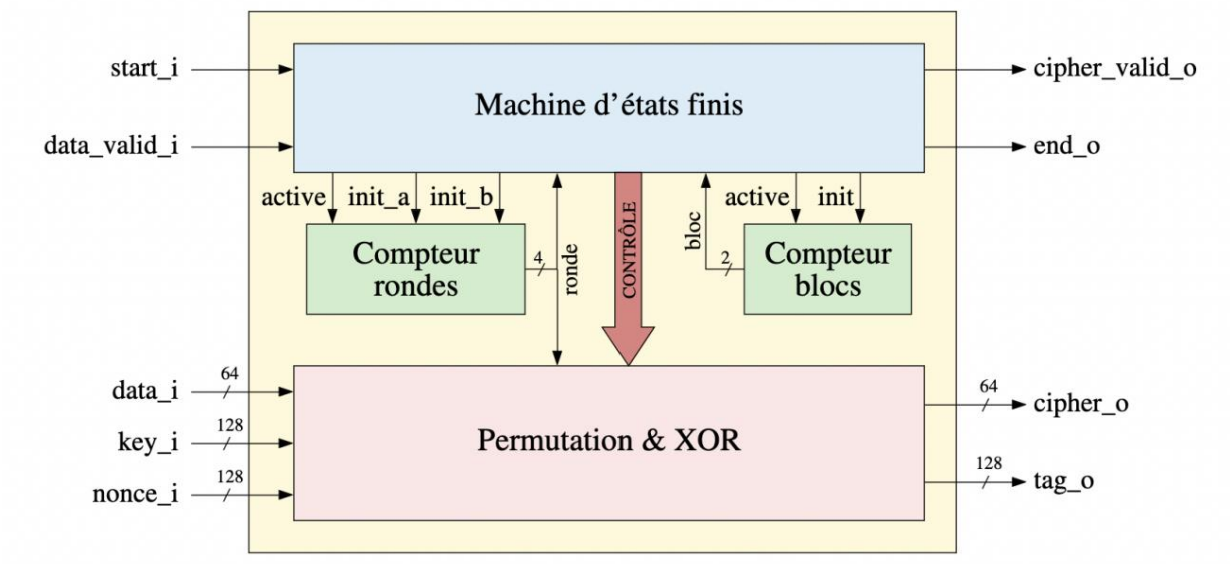
Xor_end

L'opération Xor_end est utilisée pour trois opérations différentes sur la source interne Sc, similaires à celles de Xor_begin. Elle permet d'injecter la clé K sur l'état externe après le bloc de permutation lors de la phase d'initialisation, d'être utilisée dans la phase de donnée associée et de constituer la dernière étape de chiffrement avant la publication du tag lors de la phase de finalisation.

Les différentes opérations XOR finales sont contrôlées par les deux variables en_xor_lsb_i et en_xor_key_end_i.

Machine d'état

Il nous reste maintenant l'étape de la Machine d'état :



Architecture Globale d'Ascon 128

Compteur de tours et compteur de blocs de permutation

La machine à états finis de l'algorithme ASCON-128 nécessite deux compteurs pour assurer la synchronisation des différents modules. Le premier compteur, le compteur de tours, est utilisé pour déterminer la position actuelle dans le bloc de permutation et pour déterminer la valeur de

la constante d'addition. Selon le type de bloc de permutation (p6 ou p12), deux variables (init_a et init_b). Le compteur de tours est codé sur 4 bits car il peut y avoir au maximum 12 tours.

Le deuxième compteur, le compteur de blocs, permet de se situer dans la phase de chiffrement. Il y a 3 blocs de permutation de type p6 nécessaires pour injecter les 3 plaintext P1 à P3 de 64 bits, c'est pourquoi le compteur est programmé sur 2 bits.

La FSM

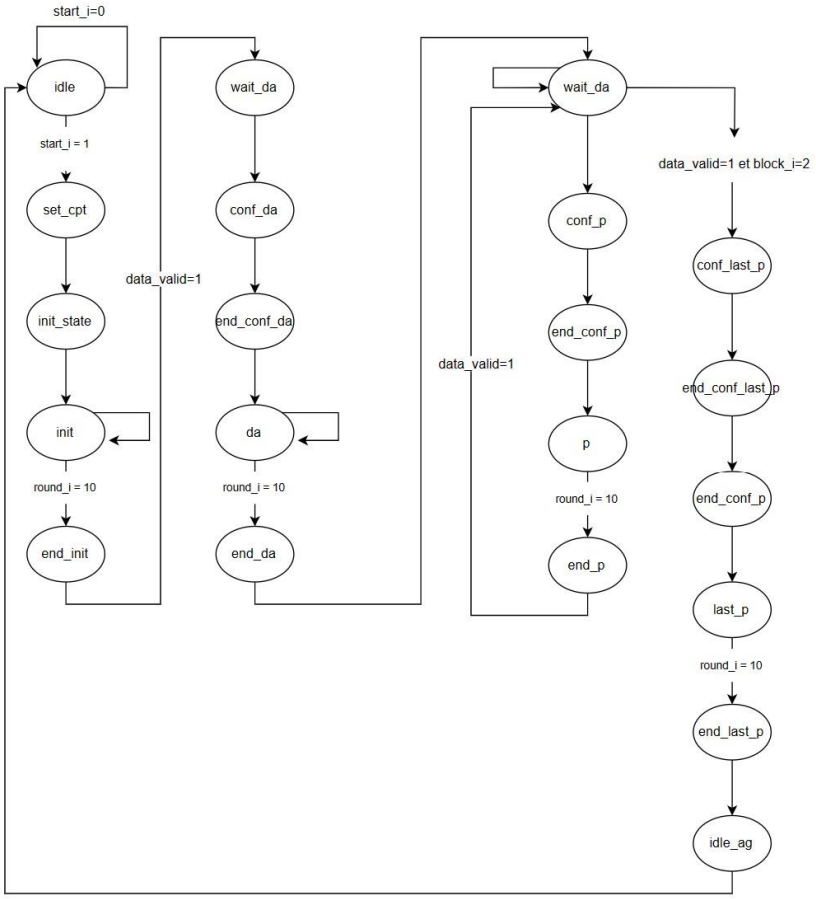


Schéma sur Draw.io de la FSM

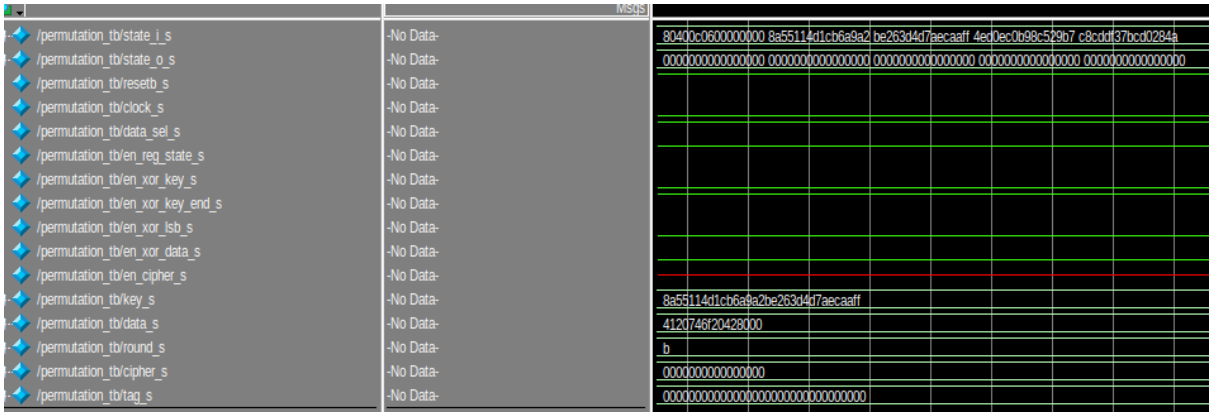
La machine à états finis (FSM) doit être en mesure de sélectionner et de configurer les variables de contrôle pour chaque état. Elle peut passer à un nouvel état si la variable data_valid_i est vraie. Si data_valid_i n'est pas activée, l'algorithme reste dans le même état sans avancer.

La FSM se divise en quatre étapes : l'initialisation, le traitement des données associées, les permutations 1 et 2, et enfin, la dernière permutation.

J’ai directement codé ascon_top pour débugger la FSM. Helas il doit y avoir une erreur car je ne parviens pas à avoir un cipher et un Tag.

Ascon_top

Le module ascon_top regroupe tous les modules nécessaires pour la permutation avec les XOR , la FSM et les compteurs.



Testbench de ascon_top

On peut observer que le tag et le cipher ne sont pas générés à la fin du processus de chiffrement malheureusement.

Conclusion

Malgré plusieurs tentatives en modifiant les valeurs des variables en_xor_key_end_i et en_xor_lsb_i, le problème persiste. Il semble que le dysfonctionnement soit lié à ces variables, notamment aux XORs de début et de fin. J'ai examiné différentes valeurs pour ces variables, mais je n'ai pas réussi à résoudre le problème.

Ascon 128 m'a offert l'opportunité de me familiariser avec le chiffrement de données. Bien que le projet paraisse initialement complexe et abstrait, la représentation schématique des principes et la visualisation des résultats m'ont permis de saisir plus facilement le fonctionnement global du protocole de cryptage.

Notation

J’ai basé la construction de l’algorithme sur la structure suivante qui m’a été donnée :

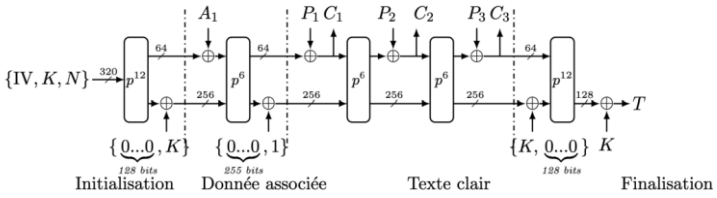


Schéma de Ascon 128

On observe les éléments suivants :

• IV : vecteur d’initialisation de 64 bits

• K : clé de 128 bits

• N : nombre arbitraire de 128 bits

• A : données associées de 48 bits

• P : texte clair de 184 bits

Le triplet (IV, K, N) concaténé fait bien 320 bits, on peut donc le découper en 5 états de 64 bits x0, x1, x2, x3, x4.

Le quadruplet (A1, P1, P2, P3) s’obtient ensuite à partir de A et P avec A1 = A + {10……00} indice 16 pour la concaténation, P1 = P0…..63, P2 = P64…..127 et P3 = P128,183 + {100……000}

Lexique Ascon_Top

• data_s : Ensemble de données à traiter pour le chiffrement/déchiffrement (64 bits)

• key_s : Clé utilisée pour le chiffrement/déchiffrement des données (128 bits)

• clock_s : Signal d'horloge qui synchronise les opérations

• en_cipher_s : Signal qui active la sortie des données chiffrées/déchiffrées

• cipher_s : Résultat du chiffrement/déchiffrement des données (64 bits)

• tag_s : Tag de vérification généré à la fin du processus de chiffrement/déchiffrement (128 bits)

• round_s : Compteur utilisé pour suivre les tours de permutation effectués (4 bits)

- data_sel_s : Sélecteur utilisé pour choisir les données à traiter dans le bloc de permutation
- en_xor_key_s : Signal qui active l'opérateur XOR pour la clé
- en_xor_lsb_s : Signal qui active l'opérateur XOR pour le bit de poids faible
- en_xor_data_s : Signal qui active l'opérateur XOR pour les données
- en_xor_key_end_s : Signal qui active l'opérateur XOR pour la clé à la fin du traitement