Ensta Bretagne
2, rue Francois Verny
29806 Brest cedex
France
Tel +33 (0)2 98 34 88 00
www.ensta-bretagne.fr

Rapport

CI2019

30 mai 2018

Projet Sécurité des systèmes
Cryptograhie légère

Paul Bouquet
Cédric Delaunay
Alexandre Degurse

Table des matières

Première partie

Introduction à la cryptographie légère

L'objectif de ce projet est d'expliquer le fonctionnement des algorithmes de crypto légère (lightweight cryptography). Il est demandé de choisir deux algorithmes de cryptographie légère, de les expliquer, de les implémenter et de donner des valeurs de performance aussi bien pour des solutions académiques qu'industrielles. Il est également demandé de détailler les domaines d'utilisation de ce type d'algorithme.

1 Généralités

La cryptographie légère est une branche de la cryptographie apparue assez récemment, et s'étant peu à peu démocratisée au cours de ces vingt dernières années. Sa création est née d'un besoin de sécuriser des appareils informatiques de plus en plus petits et de plus en plus nombreux. Aujourd'hui, en 2018, on peut dénombrer plus de quinze milliards d'objets connectés [5], et ces produits nécessitent d'être sécurisés. Si les ordinateurs et mêmes les smartphones de dernière génération offrent une puissance de calcul permettant d'implémenter facilement les algorithmes de cryptographie standard, comme par exemple celui du standard AES (algorithme Rinjdael [7]) ou encore RSA, ce n'est pas le cas pour les objets les moins puissants et les plus petits comme par exemple les puces RFID, certains systèmes embarqués ou les réseaux de capteurs (aussi connus sous leur nom anglais « Sensors Networks ») [6]. Cette impossibilité d'utiliser des standards de chiffrement « lourds » provient en effet d'un manque de puissance de calcul mais aussi dans la plupart de cas d'un manque d'espace, aussi bien physique que mémoire.

On comprend alors l'objet et les enjeux de la cryptographie légère : sécuriser, au moyen d'algorithmes moins coûteux, que ce soit en termes de temps d'exécution ou d'espace mémoire, l'ensemble des systèmes informatiques actuels qui ne peuvent être couverts par les standards de la cryptographie « traditionnelle ».

2 Les différences entre cryptographie « traditionnelle » et cryptographie légèrele

Avant de commencer la présentation des algorithmes de cryptographie légère que nous avons choisi d'implémenter, nous pensons qu'il est de bon ton de décrire les éléments qui permettent de différencier cryptographie légère et « traditionnelle ».

Commençons par les algorithmes de chiffrement par bloc, dont AES fait partie, au même titre que PRESENT qui sera évoqué dans la partie 2 mais qui lui fait partie des algorithmes de cryptographie légère. Ces derniers se démarquent des algorithmes de cryptographie standard d'abord par des tailles de bloc plus petites. Là où AES utilise des blocs de 128 bits [7], le chiffrement par blocs en cryptographie légère fait le plus souvent appel à des blocs de 64 ou 80 bits [6]. De même, les clés utilisées sont elles aussi plus courtes : 128, 192 ou 256 bits pour AES [7] tandis que l'algorithme PRESENT par exemple utilise des clés

de 80 bits [2]. De plus, les rondes de ces algorithmes sont simplifiées, avec notamment des « S-Box » de quatre bits au lieu de huit dans la plupart des algorithmes légers. Cela se traduit notamment par un espace physique requis plus faible. En effet, la S-Box de l'AES nécessite 395 GE [8] (« gate equivalents ») alors que la S-Box utilisée par PRESENT n'en nécessite que 28. Pour rappel, le « gate equivalent » est une unité de mesure utilisée en électronique qui permet de spécifier la complexité des circuits électroniques en indiquant un nombre de portes logiques nécessaires à la réalisation d'une fonction. Le constructeur pourra alors avec ce GE savoir l'espace physique nécessaire pour réaliser ladite fonction [9]. Pour vous donner une idée plus précise des ordres de grandeur mis en jeu, on peut noter qu'une puce RFID possède une surface généralement comprise entre 1000 et 10000 GE, et seulement 200 à 2000 sont dédiées à la sécurité [6]. Enfin, les algorithmes de chiffrement par blocs en cryptographie légère implémentes des opérations sur les clés de chiffrement beaucoup plus simples et beaucoup moins coûteux qu'en cryptographie standard.

D'autre part, intéressons-nous cette fois aux algorithmes de hachage. Tout d'abord, la principale différence réside dans la taille des états internes et des hachés produits par l'algorithme. Par exemple, SHA-2 sort des hachés de 256 bits tandis que SPONGENT (algorithme présenté dans la deuxième partie) retourne des hachés qui peuvent descendre jusqu'à 88 bits [1]. De plus, une autre différence entre ces algorithmes de hachage se situe dans la taille des messages en entrée des algorithmes. Là où des algorithmes standards peuvent travailler sur des messages très grands (264 bits), les algorithmes de hachage de cryptographie légère travaillent sur des données d'entrée beaucoup plus faibles, plutôt de l'ordre de 2^8 bits.

Enfin, il faut tout de même noter que la cryptographie légère n'a pas que des avantages face à la cryptographie traditionnelle. En effet, réduire les tailles des blocs, des clés, a un coût en termes de sécurité. Par exemple, l'utilisation de blocs de 64 bits dans certains algorithmes réduit considérablement le nombre de sorties possibles d'un algorithme, et dans ces cas-là un chiffré peut-être différencié d'une séquence aléatoire en utilisant seulement 232 blocs. Il en résulte alors que certains algorithmes sont vulnérables aux attaques de type « plaintext recovery » ou « key recovery » avec des probabilités non négligeables.

Deuxième partie

Présentation de deux algorithmes de cryptographie légère

3 SPONGENT

Cette partie a pour objectif de présenter l'algorithme de cryptographie légère SPONGENT. Il découle directement de l'algorithme PRESENT, développé en 2007 par « Orange Labs », la « Ruhr University Bochum » et la « Technical University of Denmark ». Ce dernier est un algorithme par blocs (de 64 bits) dont la clé fait 80 ou 128 bits [2], et il a la particularité d'avoir été inclus au dernier standard international des méthodes de cryptographie légère par l' « International Electrotechnical Commission » [3].

Cette partie a pour objectif de présenter l'algorithme de cryptographie légère SPONGENT. Il découle directement de l'algorithme PRESENT, développé en 2007 par « Orange Labs »,

la « Ruhr University Bochum » et la « Technical University of Denmark ». Ce dernier est un algorithme par blocs (de 64 bits) dont la clé fait 80 ou 128 bits [2], et il a la particularité d'avoir été inclus au dernier standard international des méthodes de cryptographie légère par l' « International Electrotechnical Commission » [3].

3.1 Fonctionnement général de SPONGENT

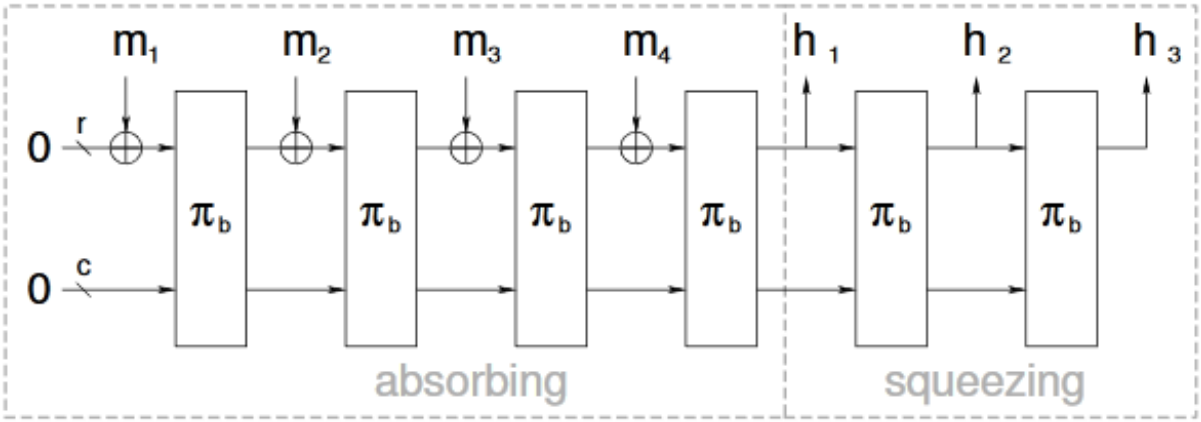


FIGURE 1 – Schéma illustrant le fonctionnement de SPONGENT, tiré de [1]

Comme énoncé précédemment, SPONGENT s'appuie sur le fonctionnement de l'algorithme PRESENT. Pour un nombre fini de bits en entrée, SPONGENT va produire un hach de taille n (fixé). La figure 1 schématise le fonctionnement de l'algorithme, qui s'organise en trois phases distinctes, que sont l'initialisation, l'absorption (absorbing) et le relâchement (squeezing).

L'initialisation consiste à ajouter un 1 binaire suivi de x 0 afin que la taille du message en entrée soit un multiple de la taille des blocs, appelé r (rate en anglais). Le message est ensuite découpé en blocs de r bits. Ensuite vient la phase d'absorption. Durant cette phase, un bloc mi subit un XOR avec les r premiers bits de l'état courant et passe ensuite dans la fonction de permutation. Cette permutation génère un état de b bits, où $b = r + c$, et où c est appelé la capacité de l'état et b est appelé la largeur de l'état. Enfin, la phase de relâchement consiste à retourner les r premiers bits de l'état, puis de passer l'état dans la fonction de permutation, et de répéter ces deux opérations jusqu'à ce que le hach retourné ait une longueur de n bits.

Les valeurs de n, b, c et r ne sont pas prises au hasard. En effet, il existe treize versions différentes de SPONGENT et elles sont définies grâce au tableau représenté à la figure 2. Ces treize versions s'appuient sur des hachés de tailles différentes, mais les valeurs des trois autres constantes citées ci-avant diffèrent elles-aussi selon la version utilisée. Dans le cas de notre projet, nous avons décidé d'implémenter SPONGENT 160/160/80.

3.2 Fonctionnement détaillé de SPONGENT 160 / 160 / 80

Cette sous-partie va permettre d'expliquer le fonctionnement détaillé de la permutation notée π_b. Cette permutation est définie par l'algorithme 1:

	n (bit)	b (bit)	c (bit)	r (bit)	R number of rounds	security(bit) pre.	2nd pre.	col.
SPONGENT-88/80/8	88	88	80	8	45	80	40	40
SPONGENT-88/176/88	88	264	176	88	135	88	88	44
SPONGENT-128/128/8	128	136	128	8	70	120	64	64
SPONGENT-128/256/128	128	384	256	128	195	128	128	64
SPONGENT-160/160/16	160	176	160	16	90	144	80	80
SPONGENT-160/160/80	160	240	160	80	120	80	80	80
SPONGENT-160/320/160	160	480	320	160	240	160	160	80
SPONGENT-224/224/16	224	240	224	16	120	208	112	112
SPONGENT-224/224/112	224	336	224	112	170	112	112	112
SPONGENT-224/448/224	224	672	448	224	340	224	224	112
SPONGENT-256/256/16	256	272	256	16	140	240	128	128
SPONGENT-256/256/128	256	384	256	128	195	128	128	128
SPONGENT-256/512/256	256	768	512	256	385	256	256	128

FIGURE 2 – Tableau définissant les 13 variantes de SPONGENT, tiré de [1]

Algorithm 1 Algorithme de permutation de SPONGENT

for i = 1 to R **do**
  $STATE \leftarrow$ ɹɐʇɐɯɹoↃlC$(i) \oplus STATE \oplus lCounter_b(i)$
  $STATE \leftarrow sBoxLayer_b(STATE)$
  $STATE \leftarrow pLayer_b(STATE)$
end for

Tout d'abord, intéressons-nous aux variables de cet algorithme. La variable R est le nombre de rondes de l'algorithme. Il varie en fonction de la variante de SPONGENT utilisée, et dans notre cas R vaut 80. Ensuite, la variable STATE représente l'état courant.

La première opération appliquée à STATE est un XOR avec d'une part lCounterb(i) et d'autre part la valeur « retournée » de lCounterb(i), c'est-à-dire que si lCounterb(i) = 1000 sa valeur « retournée » est 0001. lCounterb(i) est en réalité un LFSR, ou Registre à décalage à rétroaction linéaire, soit un générateur pseudo-aléatoire qui dépend lui aussi de la variante de l'algorithme utilisée. La figure 3 illustre les polynômes générateurs du LFSR, et la figure 4 indique les valeurs initiales que prend le LFSR.

LFSR size (bit)	Primitive Polynomial
6	$\zeta^6 + \zeta^5 + 1$
7	$\zeta^7 + \zeta + 1$
8	$\zeta^8 + \zeta^4 + \zeta^3 + \zeta^2 + 1$
9	$\zeta^9 + \zeta^4 + 1$

FIGURE 3 – Polynômes primitifs du LFSR

	LFSR size (bit)	Initial Value (hex)
SPONGENT-88/80/8	6	05
SPONGENT-88/176/88	8	D2
SPONGENT-128/128/8	7	7A
SPONGENT-128/256/128	8	FB
SPONGENT-160/160/16	7	45
SPONGENT-160/160/80	7	01
SPONGENT-160/320/160	8	A7
SPONGENT-224/224/16	7	01
SPONGENT-224/224/112	8	52
SPONGENT-224/448/224	9	105
SPONGENT-256/256/16	8	9E
SPONGENT-256/256/128	8	FB
SPONGENT-256/512/256	9	015

FIGURE 4 – Valeurs initiales du LFSR

La deuxième opération de la ronde est de passer STATE dans une S-Box. La S-Box est la même pour toutes les variantes de l'algorithme SPONGENT et elle est définie par la figure 5 :

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

FIGURE 5 – S-Box de l'algorithme SPONGENT

Enfin, la troisième et dernière opération est de passer STATE dans $pLayer_b$. $pLayer_b$ est une fonction de permutation définie comme l'inverse de la permutation de bits de l'algorithme PRESENT. Ainsi, chaque bit j est permuté avec le bit de position $P_b(j)$ où :

$$P_b(j) = \begin{cases} j \cdot \frac{b}{4} \bmod b-1, if j \in \{0,\ldots,b-2\} \\ b-1, if j = b-1. \end{cases} \quad (1)$$

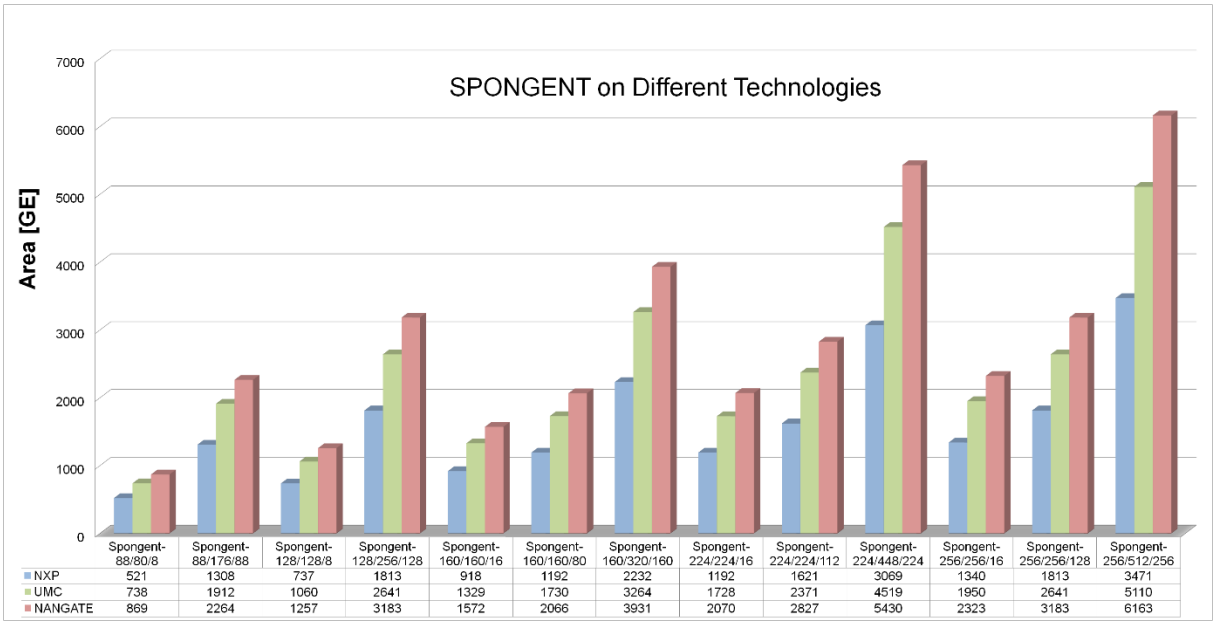
FIGURE 6 – Formule de permutation de pLayer

Performances et sécurité Les variantes de SPONGENT peuvent être implémentées en utilisant entre 738 GE (pour SPONGENT 80/80/8) et 5100 GE (pour SPONGENT 256/512/256). Ces nombres ne semblent peut-être pas très explicites, mais ils dénotent une très grande compacité de l'implémentation physique de l'algorithme. Pour plus de détails, la figure 7 présente pour différentes variantes de l'algorithme l'espace physique nécessaire en fonction de la technologie (NXP, UMC et NANGATE)[11] :

Pour ce qui est du débit de traitement, il est donné comme variant entre 360 Mbps et 2 Gbps (en fonction de la variante de SPONGENT utilisée). En comparaison, l'algorithme de hachage SHA-1 possède un débit de l'ordre d' 1 Gbps [10].

En termes de sécurité, nous pouvons voir sur la figure 8 les différentes valeurs de complexité pour l'ensemble des variantes de SPONGENT. Pour les variantes 88/80/8 et 128/128/8, il est intéressant de noter que nous sommes dans des capacités de calcul atteignables, et que de ce fait ces algorithmes ne sont pas « sûrs ». Pour le reste, nous pouvons observer des complexités relatives au minimum inatteignable (2^{80}) voire raisonnablement inatteignable (2^{128}).

Enfin, la figure 9 présente les résultats de sécurité face aux attaques par pré-image, par $2^{\text{ème}}$ pré-image et par collision [11]. L'attaque par pré-image consiste, à partir d'un haché y, à retrouver un x tel que $h(x) = y$. L'attaque par $2^{\text{ème}}$ pré-image consiste quant à elle, à partir d'un clair x, à trouver x' tel que $h(x) = h(x')$. Enfin, l'attaque par collision consiste à essayer de trouver deux clairs différents produisant le même chiffré. Elle ressemble à l'attaque par $2^{\text{ème}}$ pré-image à ceci près que le clair n'est pas spécifié.



	Spongent-88/80/8	Spongent-88/176/88	Spongent-128/128/8	Spongent-128/256/128	Spongent-160/160/16	Spongent-160/160/80	Spongent-160/320/160	Spongent-224/224/16	Spongent-224/224/112	Spongent-224/448/224	Spongent-256/256/16	Spongent-256/256/128	Spongent-256/512/256
NXP	521	1308	737	1813	918	1192	2232	1192	1621	3069	1340	1813	3471
UMC	738	1912	1060	2641	1329	1730	3264	1728	2371	4519	1950	2641	5110
NANGATE	869	2264	1257	3183	1572	2066	3931	2070	2827	5430	2323	3183	6163

FIGURE 7 – Espace physique nécessaire pour différentes variantes de SPONGENT

4 Speck

Speck est un algorithme de chiffrement par bloc léger crée par la *National Security Agency* (NSA) et rendu publique en 2013. C'est un algorithme spécialement conçu pour avoir des performances élevées afin d'offrir un algorithme de chiffrement utilisable dans le cadre de "l'Internet of Things".

4.1 Algorithmes ARX

Cet algorithme fait parti des algorithmes dits ARX, Add-Rotate-Xor. C'est une famille d'algorithmes qui n'utilisent que les opérations d'additions, rotations et ou exclusif dans l'espace GF_{2^n}. Il y a plusieurs avantages à se limiter à ces opérations:

- Rapidité: ces opérations sont des opérations logiques. Ainsi, elles sont des primitives de tout micro-controlleur et donc sont effectué en un seul cycle d'horloge.
- Sécurité matérielle: le fait que toutes les opérations soient des opérations logiques atomiques permet à ces algorithmes de fonctionner en temps constant. Prévenant les attaques par canaux cachés basés sur les mesures de temps.
- Implémentation: ces algorithmes sont souvent très simples. Leur implémentation qu'elle soit logicielle ou matérielle est très simple. Par conséquent, le temps de développement et le coût de leur implémentation est très faible.

4.2 Fonctionnement de Speck

Avant de détailler comment fonctionne Speck, considérons les notations suivantes:

- Le ou-exclusif bit à bit, noté xor
- L'addiction modulo 2^n, noté $\oplus$

	k	Time Complexity $max(2^{n-r}, 2^{c/2})$	Memory Complexity $(2^{c/2})$
SPONGENT-88/80/8	5	2^{80}	2^{40}
SPONGENT-88/176/88	1	2^{88}	2^{88}
SPONGENT-128/128/8	8	2^{120}	2^{64}
SPONGENT-128/256/128	1	2^{128}	2^{128}
SPONGENT-160/160/16	5	2^{144}	2^{80}
SPONGENT-160/160/80	1	2^{80}	2^{80}
SPONGENT-160/320/160	1	2^{160}	2^{160}
SPONGENT-224/224/16	7	2^{208}	2^{112}
SPONGENT-224/224/112	1	2^{112}	2^{112}
SPONGENT-224/448/224	1	2^{224}	2^{224}
SPONGENT-256/256/16	8	2^{240}	2^{128}
SPONGENT-256/256/128	1	2^{128}	2^{128}
SPONGENT-256/512/256	1	2^{256}	2^{256}

FIGURE 8 – Complexités en temps et en mémoire de SPONGENT

	preimage security (bit)	2nd preimage security (bit)	collision security (bit)
SPONGENT-88/80/8	80	40	40
SPONGENT-88/176/88	88	88	44
SPONGENT-128/128/8	120	64	64
SPONGENT-128/256/128	128	128	64
SPONGENT-160/160/16	144	80	80
SPONGENT-160/160/80	80	80	80
SPONGENT-160/320/160	160	160	80
SPONGENT-224/224/16	208	112	112
SPONGENT-224/224/112	112	112	112
SPONGENT-224/448/224	224	224	112
SPONGENT-256/256/16	240	128	128
SPONGENT-256/256/128	128	128	128
SPONGENT-256/256/256	256	256	128

FIGURE 9 – Attaque par pré-image et collision

- Les rotations circulaires à gauche et à droite respectivement notées, S^i et S^{-i} pour des rotations de i-bits.

Un tour de chiffrement de l'algorithme Speck est définit de la façon suivante.
Pour $k \in GF(2^n)$ une clée, le tour de chiffrement est définit par la fonction suivante:

$$\begin{array}{rccc} R_k & : & GF(2^n)xGF(2^n) & \rightarrow & GF(2^n)xGF(2^n) \\ & & (x,y) & \mapsto & ((S^{-\alpha}(x)+y)\oplus k, S^{\beta}(y)\oplus(S^{-\alpha}+y)\oplus k) \end{array}$$

avec $\alpha = 7$ et $\beta = 2$ si n = 16, α= 8 et $\beta = 3$ sinon.

La fonction de déchiffrement est définie par:

$$\begin{array}{rcccc} R_k^{-1} & : & GF(2^n)xGF(2^n) & \rightarrow & GF(2^n)xGF(2^n) \\ & & (x,y) & \mapsto & (S^{\alpha}((x \oplus k) - S^{-\beta}(x \oplus y)), S^{-\beta}(x \oplus y)) \end{array}$$

Conclusion

La conlusion

Table des figures

Références

[1] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers. *Simon and Speck : Block Ciphers for the Internet of Things*. Rapp. tech. National Security Agency, 9 juin 2015. 15 p.

[2] *Notes on the design and analysis of Simon and Speck*. Rapp. tech. International Association for Cryptologic Research, 13 jan. 2018. 23 p.

[3] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers. *The Simon and Speck families of lightweight block ciphers*. Rapp. tech. National Security Agency, 19 juin 2013. 45 p.

[4] A. Bogdanov et al. « SPONGENT : The Design Space of Lightweight Cryptographic Hashing ». In : *IEEE Transactions on Computers* 62.10 (oct. 2013), p. 2041-2053. issn : 0018-9340. doi : 10.1109/TC.2012.196.

[5] Andrey Bogdanov et al. « PRESENT : an ultra-lightweight block cipher ». In : 4727 (sept. 2007), p. 450-466.

[6] « Report on Lightweight Cryptography ». In : (28 mar. 2017), p. 27.

[7] Martin Feldhofer, Sandra Dominikus et Johannes Wolkerstorfer. « Strong Authentication for RFID Systems Using the AES Algorithm ». In : *Cryptographic Hardware and Embedded Systems - CHES 2004*. Sous la dir. de Marc Joye et Jean-Jacques Quisquater. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004, p. 357-370. isbn : 978-3-540-28632-5.

[8] Il-Hwan Park Eun-Hee Lee Je-Hoon Lee et Kyoung-Rok Cho. « Implementation of high-speed SHA-1 architecture ». In : *IEEE Transactions on Computers* (25 août 2009), p. 2041-2053.

[9] *Announcing the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf. National Institute of Standards and Technology. 2001.

[10] Wikipedia contributors. *Gate equivalent*. [En ligne ; consulté le 28 Avril 2018]. 2004. url : https://en.wikipedia.org/wiki/Gate_equivalent.

[11] Renaud. *Le développement des objets connectés : les nouveaux chiffres de 2018*. fr. Jan. 2017. URL : https://www.objetconnecte.net/objets-connectes-chiffres-etudes-2401/.