Prédictibilité des générateurs de nombres pseudo-aléatoires

Junia ISEN | Lille.

Etat de l’art

Adam OUALI Michelle Martin HAMED ABOELNAGA Ulysse Dahiez CHARLES Chaudron

2022 - 2023

# Sommaire

[I. Sommaire 1](#_Toc119443178)

[II. Introduction 2](#_Toc119443179)

[III. Description du sujet : 3](#_Toc119443180)

[IV. Aléatoire 3](#_Toc119443181)

[V. Pseudo-aléatoire 4](#_Toc119443182)

[VI. Les différents types de générateurs 4](#_Toc119443183)

[A. Générateurs de nombres pseudo-aléatoires 5](#_Toc119443184)

[1. PRNGs 5](#_Toc119443185)

[2. CSPRNG 5](#_Toc119443186)

[B. Générateurs de nombres aléatoires 7](#_Toc119443187)

[3. TRNGs / HRNG 7](#_Toc119443188)

[4. QRNG 8](#_Toc119443189)

[VII. Importance de la non-prédictibilité et domaines d’utilisation des PRNGs 8](#_Toc119443190)

[VIII. Différentes familles de PRNG / Liste PRNG 8](#_Toc119443191)

[C. Middle-square (carré médian) 8](#_Toc119443192)

[D. LCGs 9](#_Toc119443193)

[E. LFSR 10](#_Toc119443194)

[F. Mersenne Twister 10](#_Toc119443195)

[G. Blum Blum Shub 11](#_Toc119443196)

[IX. Les prédicteurs des principaux PRNGs 12](#_Toc119443197)

[X. Objectifs : 13](#_Toc119443198)

[XI. Conclusion 13](#_Toc119443199)

# Introduction

Lors de cette première période de notre année d’AP4, nous avons eu l’occasion à travers un projet d’étude de nous former et nous renseigner sur le sujet des nombres pseudo-aléatoires et la prédictibilité de leurs générateurs. Ce rapport est un état de l’art des phases de recherches préliminaire à l’implémentation de ce projet. Vous pourrez découvrir les premiers termes et les bases de la détermination d’un nombre pseudo-aléatoire dans l’informatique.

Durant de cette phase de recherche, nous avons pu travailler en équipe afin de mettre en commun les connaissances acquises.

Dans un premier temps, nous présenterons plus formellement le sujet sur lequel nous avons travaillé, à travers la problématique qui nous a été donnée.

Nous aurons ensuite l’occasion de vous parler des différents types de générateurs de nombres aléatoires et pseudo-aléatoires.

Pour vous présenter l’utilité de ce projet et l’utilité des nombres pseudo-aléatoires, nous parlerons de l’importance de ces nombres et leurs utilités.

Ensuite, nous parlerons des différents algorithmes que nous avons pu étudier et les différents types de prédicteur ainsi que leurs fiabilités.

Après cela, nous pourrons étudier plus précisément comment prédire la sortie d’un type de générateur.

Enfin, avant de conclure, nous pourrons parler des objectifs que nous nous sommes fixés pour la prochaine phase de ce projet recherche.

# Description du sujet :

Les générateurs de nombres pseudo-aléatoires (PRNG en anglais) sont des outils importants de plusieurs systèmes. Ces derniers génèrent, de façon déterministe et à partir d’une graine (seed), une suite de valeur qui présente toutes les caractéristiques d’une suite de nombres aléatoires.

Leur usage dans un contexte de sécurité informatique nécessite que les valeurs générées soient imprévisibles, c’est-à-dire qu’il soit ardu pour un attaquant de deviner avec un taux de succès non négligeable quelles seront les prochaines valeurs générées. Or il s’agit d’une caractéristique difficile à obtenir : les générateurs de nombres pseudo-aléatoires présents de base dans la plupart des langages ne l’ont pas, ce qui mène parfois à des fiascos monumentaux (par exemple : un gestionnaire de mots de passe de Kaspersky qui proposait des mots de passes prévisibles en 2021).

Les humains, en particulier, sont particulièrement mauvais en tant que générateurs de nombres pseudo-aléatoires et a priori faciles à détecter et prévoir. Le but de ce projet est, dans un premier temps, de faire un survol de l’état de l’art sur le sujet du point de vue de la reconnaissance de la « signature » de différents PRNG ainsi que de leur prédictibilité.

Dans un second temps, une application de démonstration sera développée (par exemple pour tâcher de prévoir quelles seront les prochaines valeurs émises par un utilisateur humain à qui on demande de générer des valeurs « au hasard »).

# Aléatoire

Par définition, un évènement aléatoire est dit “incertain” dans l’avenir avec l’intervention du hasard ou d’une perturbation extérieure qui vient rendre ce phénomène aléatoire. Mathématiquement parlant, il s’agit d’un événement qui dépend d’une loi de probabilité : si l’on prend la pièce de monnaie, on a une notion d’aléatoire avec 50% de chance d’obtenir pile, et 50 % de chance d’obtenir face[[1]](#footnote-2). Le pile ou face est définit comme aléatoire à cause du nombre d’éléments perturbateurs extérieur qui viennent interférer avec le lancer (courant d’air, puissance du lancer, frottement de l’air lors de la rotation, position initiale...).

Plusieurs sources environnementales d’aléatoire existent. On peut notamment parler du bruit issu des radiofréquences[[2]](#footnote-3), basé sur la désintégration radioactive[[3]](#footnote-4), du temps entre plusieurs appuis sur des touches de clavier[[4]](#footnote-5), du bruit issu des semi-conducteurs[[5]](#footnote-6)…

# Pseudo-aléatoire

Les nombres pseudo-aléatoires étant générés non naturellement, ils sont par définition différents des nombres réellement aléatoires mais présentent certaines propriétés des aléatoires.

Sur une suite de nombres générés, on retrouve par exemple une répartition statistique proche de celle de l’aléatoire. En effet tous les nombres sont censés avoir la même chance d’être tirés.

Ensuite, une vraie génération aléatoire n’a aucune chance de retomber sur les mêmes suites de nombres de manière cyclique. On peut donc déterminer pour les nombres pseudo-aléatoires une taille de cycle qui peut être plus ou moins importante selon la méthode de génération.

Enfin, l’avantage du pseudo aléatoire est qu’il est facilement et rapidement créable contrairement au vrai aléatoire. On peut déterminer pour chaque méthode de génération une efficacité et une rapidité.

# Les différents types de générateurs

Quant à eux, les nombres pseudo-aléatoires n’existent pas « naturellement ». Ces nombres pseudo-aléatoires, ou suite de nombres pseudo-aléatoires, sont issus d’algorithmes mathématiques. Ils sont utilisés dans de nombreux domaines que nous aborderons plus tard dans ce document.

## Générateurs de nombres pseudo-aléatoires

### PRNGs

Les PRNGs (Pseudo Random Numbers Generator), sont des algorithmes qui génèrent des séquences de nombres présentant certaines propriétés du hasard. Les nombres pseudo-aléatoires sont donc désignés par une suite de nombres qui s’approche d’un aléa statistiquement parfait. Le pseudo-aléatoire n’étant pas du réel aléatoire, est dit prédictible. Il suit certaines règles de l’aléatoire mais possède tout de même une forme déterministe.

En effet, les propriétés qui permettent de qualifier un PRNG s’approche des propriétés idéales des suites aléatoires. Les PRNG peuvent produire de nombreux nombres en peu de temps, ce qui est avantageux pour les applications nécessitant beaucoup de nombres. De plus, le fait qu’ils soient déterministes permet d’obtenir une séquence de nombres données qui peut être reproduite ultérieurement si le point de départ de la séquence est connu et s’il l’on a besoin de rejouer la même séquence. Les PRNG sont également périodique, ce qui signifie que la séquence finira par se répéter, alors que la périodicité n’est presque jamais une caractéristique souhaitable, les PRNG modernes ont une période si longue qu’elle peut être ignorée dans la plupart des pratiques.

### CSPRNG

Un générateur de nombres pseudo-aléatoires cryptographiquement sécurisé ou Cryptographically Secure PRNG (CSPRNG) doit respecter certaines conditions pour être considéré comme tel.

En premier lieu le générateur doit disposer de toutes les caractéristiques d’un PRNG (répartition statistique par exemple).

Ensuite, le CSPRNG doit être capable de valider le “test du bit suivant” : aucun algorithme ne doit pouvoir trouver avec une probabilité de réussite supérieure à 50% le bit n+1 d’une séquence donnée de n bits.[[6]](#footnote-7)

Le test du bit suivant est déterminé comme étant le test le plus restrictif. Un générateur capable de passer le test du bit suivant est capable de passer tous les autres tests (prouvé par Andrew Yao en 1982).

Enfin, le générateur doit résister au fait que son état interne soit connu. Si l’état est connu, il est impératif qu’il soit impossible de reconstruire la suite de nombres randoms. Connaître les conditions initiales du générateur de permet pas non plus de déterminer le prochain état du générateur. Pour être totalement imprédictible, il est tout de même préférable que le CSPRNG soit initialisé avec une graine purement aléatoire.[[7]](#footnote-8)

Seuls certains PRNGs peuvent être considérés comme assez « sûrs » et pour être utilisés comme des CSPRNGs. Parmi ces algorithmes, nous verrons historiquement ChaCha20, ainsi que ARC4.

ChaCha20 est un CSPRNG issu de l’algorithme Salsa20. Il a été inventé pour améliorer les performances de son prédécesseur. Couplé à l’algorithme Poly1305, ils sont entièrement décrits dans le RFC 8439 et sont parfois utilisés dans le protocole TLS utile au chiffrement des données sur un réseau informatique. Comme tous les algorithmes basés sur AXR, il s’appuie autour de 3 différentes actions arithmétiques : addition modulaire, OU exclusifs (XOR), et rotations.[[8]](#footnote-9)

Quant à lui, ARC4, aussi appelé RC4, est un ancien CSPRNG, désormais considéré comme obsolète pour diverses raisons. Il était utilisé dans différents domaines d’applications : SSL/TLS et WEP. Ses vulnérabilités ont été trouvées en 2003 et 2013.

C’est le CMVP (Cryptographic Module Validation Program) qui recommande et conseille ou non l’utilisation d’un algorithme dans le domaine d’utilisation cryptographique. Cette organisation est encadrée par le NIST (National Institute of Standards and Technology under the Department of Commerce), ainsi que le centre de cybersécurité Canadien.[[9]](#footnote-10)

On remarque donc que tous les CSPRNGs sont des PRNGs, mais tous les PRNGs ne sont pas assez sûrs pour être définis comme CSPRNGs.

#### L’entropie de nos ordinateurs

Quand on parle de « générateur de vrais nombres aléatoires » ou TRNG (True Random Numbers Generator) on parle d'un processus capable de générer des séquences de nombres qui ne sont pas en lien entre eux ou entre ses prédécesseurs (système non déterministe) et qui n’existe pas par conséquent un calcul pour prédire cette séquence. Ces séquences sont indépendantes de sorte qu'il soit impossible de prédire les valeurs futures sur la base des valeurs passées ou présentes.

La fonction CryptGenRandom sous Windows, ou le répertoire /dev/random sous les systèmes Unix, permettent l’utilisation d’un « pool d’entropie ». Ce pool d’entropie définit la capacité du système à fournir du « réel » aléatoire. Les données aléatoires de ce pool sont créées en mélangeant plusieurs phénomènes particulièrement hasardeux et qui n’ont aucune corrélation entre eux et qui peuvent être issus par exemple : d’une entrée de minutage d’un périphérique qui est ensuite combinée avec l’heure d’amorçage stockée, les données système et les données utilisateur, telles que l’ID des processus, l’horloge système, l’heure système, le compteur système, l’état de mémoire, ... Cette entropie permet de répondre de façon temporaire à la problématique de nécessité de nombre aléatoire dans le contexte d’utilisations de commandes comme GPG ou SSH par exmple. On parle de résolution « temporaire » car le pool d’entropie n'est pas infini. Lorsqu’il est vide, les lectures depuis le périphérique /dev/random seront bloquantes jusqu'à l'obtention de suffisamment de bruit en provenance de l'environnement.[[10]](#footnote-11)

## Générateurs de nombres aléatoires

### TRNGs / HRNG

Les TRNGs (True Random Numbers Generator) sont souvent sous forme matérielle. Ils sont même parfois appelés les HRNG (Hardware Random Numbers Generator). Ils utilisent des sources d’entropie basées sur des phénomènes physiques dit imprédictibles et non-déterministes. Ces dispositifs recueillent en continu le caractère aléatoire du bruit des composants matériels pour générer la prochaine sortie aléatoire.

### QRNG

Les QRNG tirent leurs nombres aléatoires de processus quantiques intrinsèquement indéterministes. L'impossibilité de prédire les nombres n'est pas seulement basée sur la complexité, mais il est en principe impossible de prédire les nombres aléatoires produits par les QRNG. On pourrait dire que même la nature ne connaît pas ces nombres aléatoires avant qu'ils ne soient produits. Ils sont donc réellement aléatoires.

# Importance de la non-prédictibilité et domaines d’utilisation des PRNGs

La non-prédictibilité des nombres pseudo-aléatoires porte principalement son importance dans les domaines sensibles, tels que la vie privée, le chiffrement des données, pour que des données non-publiques le restent. La génération des nombres pseudo-aléatoire sont à la base de la cryptographie. Si ces derniers sont devinés ou prédictibles, les données risquent d’être interceptées et utilisées à d’autres fins.

L’utilisation de ces nombres est présente dans de nombreux domaines tels que le chiffrement de données ou la génération de mots de passe (cryptographie) par exemple. Mais ils peuvent également être utilisés dans d’autres domaines tels que la simulation, la prédiction de phénomènes naturels comme les séismes ou les trajectoires de tsunami, dans les machines des casinos, et bien d’autres domaines d’applications.

# Différentes familles de PRNG / Liste PRNG

## Middle-square (carré médian)

La théorie de middle-square repose sur le principe de produire une séquence de 4 chiffres pseudo-aléatoires. La valeur initiale à 4 chiffres permet de créer une valeur à 8 chiffres. Si l’on obtient une valeur inférieure à 8 chiffres, on renvoie 0 pour la compléter. Pour le générateur de nombres à n chiffres, le processus de rotation n’est plus 8n si la valeur en résultat est de 4 chiffres alors la valeur de 0 peut être ajoutée.

La méthode est la suivante :

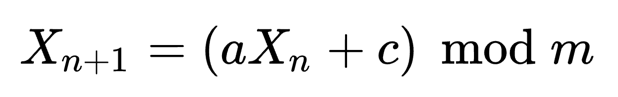
* On choisit des entiers positifs comme graines (Z0).
* On met le nombre au carré pour qu'il puisse prendre la forme d’un chiffre faisant 2 fois le nombre de chiffres Z0. Sinon, on ajoute la valeur 0 devant le nombre pour que sa taille devienne un multiple de 8.
* On prend ensuite le nombre de chiffres Z0 au milieu pour passer à Z1.
* On ajoute des chiffres décimaux devant Z1.
* On recommence les étapes jusqu’à l’étape n.[[11]](#footnote-12)

Une image contenant texte

Description générée automatiquement

## LCGs

Les LCGs sont des types de générateurs linéaires congruentiels (linéaires signifiant l’utilisation d’une fonction et congruentiels signifiant la relation entre chaque nombre généré). Ils sont définis par la formule suivante :



Le premier terme de la suite X0 est appelé « seed », servant de référence pour initialiser les prochaines valeurs de sortie Xn+1. Chaque terme Xn de la suite fait partie de la séquence pseudo-aléatoire.

Ils disposent de 3 constantes : a, c, et m. a étant la constante multiplicative, c étant la constante d’incrémentation, et m étant la constante de module.

Voici un exemple plus concret de la méthode :

## LFSR

Un LFSR (Linear Feedback Shift Register) ou registre à décalage à rétroaction linéaire est un dispositif physique (électronique) ou logiciel dont le bit d'entrée est une fonction de son état antérieur. La fonction la plus couramment utilisé est la fonction XOR. Le bit d'entrée d'état suivant est déterminé comme une combinaison de certains bits sélectionnés dans l'état actuel, représentés par un polynôme. Le choix des bits d'état courant à combiner joue un rôle majeur à la fois sur la longueur du cycle de répétition et sur la variété des valeurs en sortie.

Ainsi, un LFSR est souvent un registre à décalage dont le bit d'entrée est piloté par le ou exclusif (XOR) de certains bits choisis dans le registre à décalage total. La valeur initiale du LFSR est appelée la graine, comme pour certains autres algorithmes.

Sachant que le registre a un nombre défini d’états possibles, il va potentiellement entrer dans un cycle répétitif. Cependant, un LFSR avec une fonction de rétroaction bien choisie peut produire une séquence de bits qui semble aléatoire et qui a un cycle de répétition très long. Les applications des LFSR incluent la génération nombres pseudo-aléatoires.[[12]](#footnote-13)

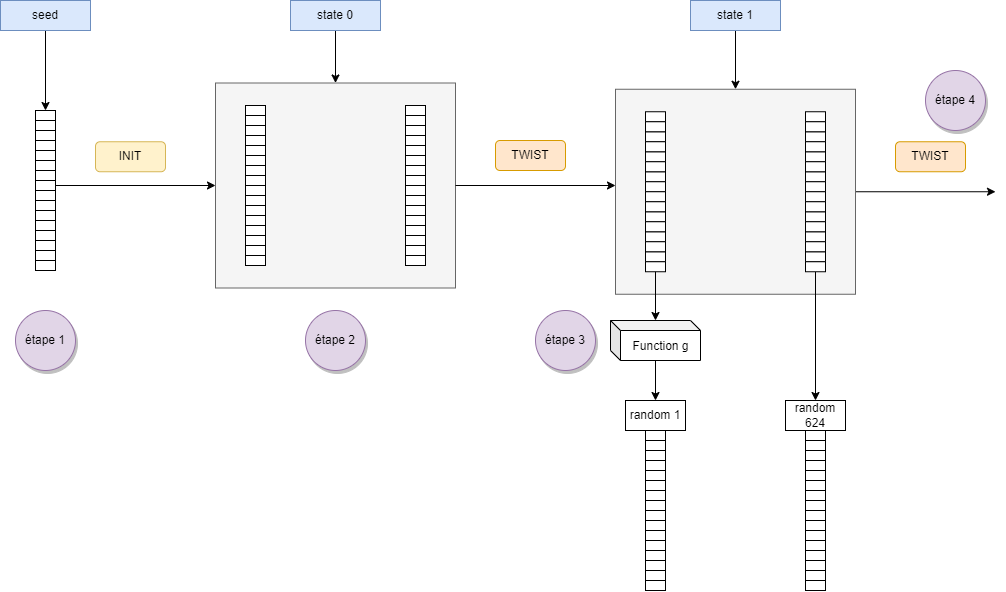
## Mersenne Twister

Cette méthode est la plus utilisée, notamment par les langages de programmation tels que Python, C, C++, R…

Comme beaucoup de méthode elle est basée sur une graine puis une méthode de calcul utilisant la récursivité.

Une image contenant texte

Description générée automatiquement



Dans Mersenne Twister, nous initialisons le processus par une seed (étape 1), puis nous la transformons dans en un premier état 0 (étape 2). De ces états nous pouvons tirer 624 entiers différents grâce à une fonction (étape 3). Tous les 624 nombres générés, il faudra donc effectuer un « Twist » (étape 4) pour en obtenir à nouveau.[[13]](#footnote-14)

## Blum Blum Shub

Blum Blum Shub est un autre algorithme générateur de nombres pseudo-aléatoires, et sa sortie est calculée grâce à cette formule mathématique :



M représente le reste d’une division sans reste entre deux nombres premiers p et q qui doivent respecter certaines contraintes telles que le fait d’être congru à 3 modulo 4 (le nombre mois 3 doit être divisible par 4).

Une des contraintes de cet algorithme est qu’il se base sur une graine qui doit être elle-même aléatoire. Également, le calcul du module de M est extrêmement long du fait du grand nombres d’itérations du calcul à effectuer avant de trouver les bons nombres premiers p et q.

Ces contraintes font que son utilisation est surtout importante en cryptographie où la rapidité est moins cruciale que lors de simulations de phénomènes naturels par exemple.

Cet algorithme a déjà été combiné avec un TRNG qui lui servait de générateur de graines réellement aléatoires.[[14]](#footnote-15)

# Les prédicteurs des principaux PRNGs

Malgré nos recherches en profondeur, nous avons constaté que les ouvrages poussés à propos des prédicteurs de PRNGs ne sont pas légion. En effet le but d’un PRNG dont on ne peut prédire la sortie (un CSPRNG donc) est utilisé tant qu’il est sûr. Dès lors qu’une méthode est trouvée pour les prédire, ces générateurs ne sont plus utilisés à des fins semblables mais ils servent alors à générer des nombres pseudo-aléatoires dont il n’est pas réellement important que l’on puisse les trouver. Il n’est donc pas intéressant de créer des prédicteurs pouvant prédire à grande échelle.

Nous sommes tout de même parvenus à trouver certains travaux. Nous avons découvert par exemple un prédicteur du PRGN MT19937[[15]](#footnote-16), une version de Mersenne Twister et qui est utilisé par le langage de programmation Python. Le prédicateur récolte 624 nombres générés par le PRNG pour recréer un état du Mersenne Twister, il est ensuite capable d’effectuer un twist, donc de générer l’état suivant et par conséquent de trouver les prochains nombres générés.

Une autre méthode est utilisée pour prédire un PRNG est le Machine Learning. Le but est ici d’entrainer une intelligence artificielle/réseau de neurones avec énormément de suites de nombres générés par un PRNG. Par la suite elle sera capable de prédire les nombres générés par la suite. Un prédicteur de ce type s’est servi du Machine Learning pour prédire la génération de l’algorithme « xorshift128 ».[[16]](#footnote-17)

# Objectifs :

Comme vu précédemment, durant notre phase de recherche, nous nous sommes heurtés au manque d’informations concernant les prédicteurs de PRNG. Notre fil rouge quant à notre projet à réaliser lors de notre prochaine période sera donc de tenter de réaliser un algorithme qui permet d’identifier une forme de PRNG et ensuite de prédire ses sorties futures. Nous commencerons par créer un ou plusieurs prédicteurs de PRNG et ensuite seulement un identificateur en fonction des difficultés que nous rencontrons.

# Conclusion

Nous pouvons en conclure que beaucoup de types de générateurs de nombres pseudo aléatoire existent et que dans la majorité des cas, certains sont plus facile à exploiter que d’autre, plus leur complexité est élevée, plus leur implémentation l’est. Nous avons donc pu étudier de manière large quelques-uns d’entre eux. Nous pourrons donc essayer d’implémenter différents types de prédicteurs, pour ensuite en créer un programme qui regroupe ces prédicteurs, afin d’en déduire à la fois la méthode de génération de nombre et le nombre à prédire. Cet état de l’art a donc été l’occasion pour nous de se documenter sur les principes de la génération de nombre pseudo aléatoire, les différents types entre PRNG, CSPRNG et autres types de générateurs de nombre aléatoire.

La suite logique de ce rapport sera donc l’implémentation les objectifs cités ci-dessus.

1. Batanero, C. (2016, March 1). Understanding randomness: Challenges for research and teaching. Retrieved from https://hal.archives-ouvertes.fr/hal-01280506/document [↑](#footnote-ref-2)
2. Haahr, M. (1998, October). True random number service. RANDOM.ORG - The History of RANDOM.ORG. Retrieved from https://www.random.org/history/ [↑](#footnote-ref-3)
3. Walker, J. (n.d.). Hotbits: Genuine random numbers, generated by radioactive decay. HotBits: Genuine Random Numbers. Retrieved from https://www.fourmilab.ch/hotbits/ [↑](#footnote-ref-4)
4. Sidhpurwala, H. (2019, June 5). Understanding random number generators, and their limitations, in linux. Red Hat. Retrieved from https://www.redhat.com/en/blog/understanding-random-number-generators-and-their-limitations-linux [↑](#footnote-ref-5)
5. Ewert, M. (n.d.). A random number generator based on electronic noise and the xorshift ... Retrieved from https://d-nb.info/1203628439/34 [↑](#footnote-ref-6)
6. Soucarros, M. (n.d.). Analyse des générateurs de nombres aléatoires dans des conditions anormales d’utilisation. Retrieved from https://tel.archives-ouvertes.fr/tel-00759976/file/SOUCARROS\_2012\_diffusion.pdf [↑](#footnote-ref-7)
7. KIETZMANN, P., &amp; SCHMIDT, T. (n.d.). A Guideline on Pseudorandom Number Generation (PRNG) in the IoT. Retrieved from https://arxiv.org/pdf/2007.11839.pdf [↑](#footnote-ref-8)
8. McKay, K. A., &amp; Vora, P. L. (n.d.). Analysis of ARX Functions: Pseudo-linear Methods for Approximation, Differentials, and Evaluating Diffusion. Retrieved from https://eprint.iacr.org/2014/895.pdf [↑](#footnote-ref-9)
9. Cryptographic module validation program: CSRC. csrc.nist.gov. (n.d.). Retrieved from https://csrc.nist.gov/projects/cryptographic-module-validation-program [↑](#footnote-ref-10)
10. Cryptgenrandom, fonction (wincrypt.h) - win32 apps. Win32 apps | Microsoft Learn. (n.d.). Retrieved from https://learn.microsoft.com/fr-fr/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom [↑](#footnote-ref-11)
11. Budiman, A., Bulolo, E., &amp; Saputra, I. (n.d.). Middle Square method analysis of number pseudorandom process. Retrieved from https://www.researchgate.net/publication/348456683\_Middle\_Square\_Method\_Analysis\_of\_Number\_Pseudorandom\_Process [↑](#footnote-ref-12)
12. What’s an LFSR? - texas instruments. (n.d.). Retrieved from https://www.ti.com/lit/an/scta036a/scta036a.pdf [↑](#footnote-ref-13)
13. DUTANG, C., MARCHAL, N’GUYEN, T. (n.d.). Mersenne Twister et Translation irrationnelle du tore. ressources-actuarielles.net. Retrieved from http://www.ressources-actuarielles.net/EXT/ISFA/fp-isfa.nsf/a5ee867f10dc2d79c1256f130065f05f/6ceb5b474cd3b563c125700c0040d8e3/$FILE/GT2008-01.pdf [↑](#footnote-ref-14)
14. William, J. (n.d.). Blum Blum Shub. Asecuritysite.com. Retrieved from https://asecuritysite.com/encryption/blum [↑](#footnote-ref-15)
15. Kmyk. (n.d.). KMYK/Mersenne-twister-predictor: Predict MT19937 PRNG. Retrieved from https://github.com/kmyk/mersenne-twister-predictor [↑](#footnote-ref-16)
16. Hassan, M. (n.d.). Cracking random number generators using machine learning – part 1: Xorshift128. NCC Group Research. Retrieved from https://research.nccgroup.com/2021/10/15/cracking-random-number-generators-using-machine-learning-part-1-xorshift128/ [↑](#footnote-ref-17)