Implémentation des Générateurs de Nombres Aléatoires et Analyse de la Qualité

January 28, 2026

1 Contexte et motivation

Le hasard et la capacité à produire des nombres aléatoires sont au cœur des protocoles de sécurité informatique. Les nombres aléatoires servent à générer des clés secrètes, des nonces, des vecteurs d'initialisation (IV) ou des jetons CSRF. Une génération non sécurisée permettrait à un attaquant d'inférer ou de prédire des clés ou des jetons ultérieurs, compromettant ainsi la sécurité du système. Même si des constructions théoriques robustes existent, elles sont parfois mal mises en œuvre et lorsque la qualité de l'aléa est insuffisante, l'ensemble du système devient vulnérable (voir, par exemple, La Recherche — L'aléatoire, clé de voûte de la sécurité informatique [1]).

2 Objectifs du projet

Les générateurs de nombres aléatoires (RNG, *Random Number Generators*) se répartissent en deux grandes catégories :

- **Générateurs de nombres véritablement aléatoires (TRNG)** (algorithmes non déterministes).
- **Générateurs de nombres pseudo-aléatoires (PRNG)** (algorithmes déterministes). Dans cette catégorie, on distingue :
  - **PRNG cryptographiquement sécurisés** (CSPRNG ou CS-PRNG) (Cryptographically Secure Pseudorandom Number Generators) (algorithmes déterministes offrant de fortes garanties quant à la difficulté de prédire les sorties futures).
  - **PRNG non cryptographiquement sécurisés**.

L'objectif de ce projet est d'implémenter, en Python, plusieurs méthodes de génération de nombres pseudo-aléatoires (PRNG), d'évaluer leur qualité, et de formuler des recommandations pratiques pour un usage sécurisé.

3 Objectifs pédagogiques

- Implémenter différents générateurs (PRNG et CSPRNG) en Python.
- Mettre en œuvre une suite de tests statistiques mesurant qualité et uniformité.
- Réaliser une démonstration d'attaque pédagogique exploitant la faiblesse d'un PRNG.
- Comparer la robustesse des générateurs étudiés.
- Produire un code commenté, et un rapport écrit.

4 Organisation du projet

- La durée du projet est fixée à **7 séances**.
- Le projet est réalisé en **groupes de trois élèves maximum**.
- Chaque élève du groupe doit être capable de :
  - comprendre l'ensemble des parties du projet ;
  - répondre aux questions relatives à toutes les étapes et implémentations réalisées.
- L'utilisation de l'IA de manière limitée est autorisée, mais toute utilisation doit être correctement citée.

5 Livrables attendus

- Rapport écrit: résumé, méthodes, résultats et analyses critiques, recommandations et bibliographie.
- Dépôt Git : code Python, notebooks, résultats et figures.
- démonstration (10 minutes).

6 Evaluation

L'évaluation du projet sera répartie selon les critères suivants :

- **Implémentation, réalisation des tests, qualité du code et démonstration** — 80%
- **Rapport écrit** — 20%

7 Structure suggérée du rapport

- Résumé (abstract) et mots-clés ; note de responsabilité précisant que les attaques réalisées sont pédagogiques et interdites sur des systèmes réels.
- Introduction et motivation.
- Description des algorithmes et des implémentations.
- Méthodologie des tests statistiques.
- Attaques (protocoles, résultats).
- Discussion et recommandations pratiques.
- Conclusion.
- Références (numérotées).
- Annexes : listings de code, figures.

8 Générateurs à étudier et implémenter

- **PRNG non cryptographiques**
  - **Linear Congruential Generator (LCG)** : Générateur pseudo-aléatoire simple et rapide, basé sur une relation linéaire récursive. Le LCG souffre de faibles propriétés statistiques et d'une absence totale de sécurité cryptographique.
  - **Mersenne Twister (MT19937)** : Générateur très répandu offrant d'excellentes propriétés statistiques, mais inadapté aux usages cryptographiques.
- **PRNG à distribution gaussienne (normale)**
  - **Transformée de Box–Muller** : Algorithme permettant de transformer deux variables aléatoires uniformes indépendantes en variables suivant une loi normale.
- **PRNG cryptographiquement sécurisés (CSPRNG)**
  - **NIST SP 800-90A DRBG (Deterministic Random Bit Generators)** : Famille de générateurs pseudo-aléatoires définis par le standard NIST SP 800-90A. Ces générateurs sont conçus pour des applications cryptographiques et offrent des garanties formelles contre la prédiction et le retour en arrière.
  - **Blum–Blum–Shub (BBS)** : Générateur fondé sur des hypothèses de théorie des nombres, réputé pour sa sécurité théorique.

  - **Générateur système (`os.urandom`)** : Interface fournie par le système d'exploitation, s'appuyant sur des sources d'entropie matérielles et logicielles. Il est couramment utilisé comme générateur cryptographiquement sûr en pratique.

- **Générateurs non déterministes et hybrides**
  - **Construction XOR NRBG (Non-Random Bit Generator)** : Générateur hybride combinant plusieurs générateurs (ou sources) de bits via une opération XOR bit à bit. Cette construction vise à améliorer la robustesse face à la défaillance partielle d'une source d'aléa.

Remarque : pour des raisons de temps d'exécution, certains algorithmes lourds devront être paramétrés avec des valeurs modestes pour les démonstrations.

9 Méthodes de test (statistiques & cryptographiques)

9.1 Tests statistiques à implémenter

- **Estimation d'entropie (Shannon) par octet.**
- **Test du χ^2 (chi-carré) pour l'uniformité des octets.**
- **Autocorrélation (lags 1, 8, ...).**
- **Test de Kolmogorov–Smirnov (KS)**.

9.2 Expériences / attaques

Réaliser au minimum deux des expériences/attaques listées ci-dessous. Pour chaque expérience/attaque menée, le rapport doit impérativement documenter les points suivants : Modèle de menace, hypothèses, algorithme, conditions de succès, métriques et résultats.

- **Récupération de la graine LCG.** Démonstration par known-plaintext / keystream XOR : récupération de la graine via résolution linéaire ou recherche exhaustive sur petits espaces de graine.
- **Reconstruction d'état MT19937.** Récupération de l'état interne à partir de 624 sorties 32-bits et prédiction des sorties futures (démonstration pédagogique).
- **Réutilisation de nonce en AES-CTR.** Mise en évidence de la fuite de l'XOR des messages lorsque le même nonce/IV est réutilisé.
- **IV prévisible en AES-CBC.** Exemple montrant comment un IV prévisible ou déterministe peut entraîner des fuites d'information ou faciliter la détection d'égalité de messages.

References

[1] Vergnaud, Damien. *L'aléatoire, clé de voûte de la sécurité informatique.* *La Recherche*, n°549, juillet–août 2019, pp. 46–49. Disponible en ligne via : `https://www.larecherche.fr`, consulté le 28 janvier 2026.

[2] Johnston, David. *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers.* De—G Press, 2018. ISBN: 978-1501506062. Disponible en ligne : `https://www.degruyterbrill.com/document/doi/10.1515/9781501506062/html`, consulté le 28 janvier 2026.

[3] Bureaud, Thierry. *Cybersécurité et qualité des générateurs informatiques de nombres aléatoires.* Projet E3, ESIEE Paris, 2019–2020. Disponible en ligne : `https://perso.esiee.fr/~bureaud/Unites/Pr302i/1920/ProjetsE3/e320vr01.pdf`, consulté le 28 janvier 2026.