A Custom Function-Based Seed-Driven Framework for Cryptographic Encryption

Author: Amer Alaa Eldin Attia Gomaa

Date: May 2025

Contact: ameralaah99@gmail.com

Abstract

This paper proposes a novel cryptographic architecture based on user-defined mathematical functions and multiple private seed values. Unlike traditional public-key systems such as RSA, which depend on well-known algebraic structures and are vulnerable to quantum computation, the proposed approach enables dynamic and unpredictable encryption functions that vary from user to user. We introduce the theoretical foundations, operational model, and a practical polynomial-based implementation to illustrate the concept. The proposed design is well-suited for lightweight systems, customized protocols, and experimental post-quantum cryptography.

1. Introduction

Conventional public-key cryptography (e.g., RSA) relies on the difficulty of factoring large integers and the predictable structure of mathematical expressions like $m = p \cdot q$. While historically secure, such systems face increasing risks from advancements in quantum algorithms and brute-force hardware.

In contrast, this work introduces a flexible, seed-driven encryption model where users define their own mathematical function $F(x)$ along with a set of secret seeds. Encryption and decryption are only possible when both the correct function and seed values are shared and known.

2. Motivation

- Unpredictability: No fixed structure or public formula to reverse-engineer.
- Seed-Level Security: Multi-seed architectures introduce layered entropy.
- Quantum Resistance Potential: Avoids reliance on algebraic hardness assumptions.
- Function Diversity: Allows each user to maintain a private, custom encryption logic.

3. Classical vs. Seed-Driven Encryption

Traditional Public-Key Encryption (e.g., RSA):
- Formula: c = m^e mod n, where n = p × q.
- Function is public and predictable.
- Security depends on factoring n.

Proposed Seed-Based Architecture:
- Formula: c = F(m, s1, s2, ..., sn)
- Function F is custom-defined.
- Seeds s1, ..., sn are private.
- Security depends on both function secrecy and seed entropy.

4. Architecture and Implementation

Each user defines:
- A function architecture: polynomial, modular, recursive, etc.
- A set of secret seeds: integers, vectors, or function-based.
- An encryption rule: how to apply F to each data unit.

Example Polynomial Function:
F(x) = s1 * x + s2 * x^2 + s3 * x^3 + s4
Where x is the data unit (e.g., ASCII code) and s1–s4 are secret seeds.

Encryption: c = F(x, s1, s2, s3, s4)
Decryption: Requires exact reconstruction of F and original seeds.

5. Illustrative Example

Plaintext: "A"
ASCII Value: x = 65

Function:
F(x) = 7x + 3x^2 + 2x^3 + 9

Encryption Result:
F(65) = 7·65 + 3·65^2 + 2·65^3 + 9 = encrypted_value

Decryption: Not feasible without full knowledge of function structure and seed values.

6. Security Advantages

- No standardized function to attack.
- Resistance to brute-force attacks without access to seeds and structure.
- Scalable design using more seeds or complex transformations.
- Lightweight for embedded systems or resource-constrained environments.
- Flexible encryption logic: per session, per file, per device.

7. Practical Considerations

Production Readiness: YES, with the following precautions:
- Seed Exchange: Requires secure seed delivery mechanisms.
- Function Obfuscation: Ideally stored in compiled form.
- Entropy Management: Use strong, unpredictable seeds.
- Formal Auditing: Recommended for secure production environments.

Ideal Use Cases:
- Device-to-device encryption
- Per-client encryption in APIs
- Post-quantum cryptography research
- Experimental protocol design

8. Security Analysis and Entropy Considerations

The strength of seed-driven cryptographic functions depends on two factors:

Entropy Estimation:
Let F(x) = Σ s_i * x^i + c. If each s_i is a random n-bit seed, entropy H(F) = k * n bits. This increases linearly with the number of seeds and their bit-length.

Collision Probability:
Assuming inputs x in [0, M], and F is injective or pseudorandom, then collisions are negligible.

Resistance to Known-Plaintext Attacks:
Attacker must guess both structure F and seeds s_i. This is infeasible for strong entropy H(F).

9. Generating Functions Automatically (Function Generator Model)

To support automated encryption, a function generator may accept:
- Security level (e.g., 128-bit entropy)
- Function type (polynomial, modular, hybrid)
- Number of seeds k

Output: Random function $F(x) = \Sigma\ s_i\ x^i + c$ where $s_i \sim U(Z_q)$

Steps:
1. Generate k random seeds using a cryptographic RNG.
2. Choose function structure.
3. Store function and seeds securely.

10. Conclusion

This work proposes a flexible cryptographic framework built on the foundations of user-defined mathematical functions and private seed values. By departing from fixed algebraic systems, it enables a wide range of secure, dynamic, and potentially post-quantum encryption strategies suitable for future-proof and embedded applications.