Digital Enciphering Machine Using Dual Permuted State Transition Networks

Enrique Flores

May 6, 2025

Background

In the mid-20th century, mathematician John F. Nash Jr. proposed theoretical methods for constructing highly secure ciphers based on state machines and minimal information leakage. Some of Nash's correspondence outlining these ideas was declassified and made public by the United States National Security Agency (NSA) in 2012.

While Nash described the core theoretical underpinnings, no detailed digital implementation suitable for modern cryptographic hardware was provided. The present invention builds upon these foundational concepts to realize a fully specified, hardware-efficient, high-speed cryptographic cipher compatible with current embedded and mobile computing platforms.

Conventional encryption systems, such as AES, RSA, and ECDSA, rely on complex mathematical structures such as finite field algebra, prime factorization, or elliptic curves. While effective, these systems require significant computational resources and depend on long key lengths and repeated rounds of transformations to achieve security. Furthermore, many existing ciphers are optimized for software execution rather than hardware minimalism, leading to power inefficiencies and reduced applicability for embedded or low-power devices.

There exists a need for an encryption system that is:

- Extremely lightweight in hardware.
- Based on simple deterministic digital structures.
- Capable of achieving high levels of security via structural complexity rather than algorithmic depth.
- Easily scalable with key material randomness.

Summary of the Invention

This invention provides a system and method for:

- Encrypting and decrypting data using a hardware-optimized state machine with two parallel permuted networks.
- Utilizing a 128-state architecture (or variable size) to align with digital system standards.
- Employing a random wiring and random state flip logic that can be seeded by a high-entropy generator such as MKRAND.
- Enabling efficient, high-speed, low-power encryption operations suitable for embedded, IoT, and mobile platforms.

The encryption engine operates as follows:

1. Two independent permuter structures ("Red" and "Blue") are generated, each representing a full permutation of the state space.

2. Each transition from a source state to a destination state is labeled with a control bit:
   1 = Invert the message bit; **0** = Pass the message bit unchanged.

3. An input message bit is used to select whether the "Red" or "Blue" permuter is used to process the next state.

4. As the system traverses through the states, bits are flipped or preserved according to the wiring.

5. After processing through the entire sequence, the final message bits are collected to form the cipher text.

Decryption is symmetric:
Using the same wiring and flip definitions, the inverse permutation paths are used to reconstruct the original message from the cipher text.

This structure provides cryptographic security by relying on the unpredictability of the wiring combined with the compounding effect of random flips through many small, deterministic steps.

Detailed Description

Pseudo-VHDL Specification for Nash Cipher

```
-- Pseudo VHDL for NashCipher

entity NashCipher is
    Port (
        clk : in std_logic;
        reset : in std_logic;
        input_bit : in std_logic;
        decider_bit : in std_logic;
        output_bit : out std_logic
    );
end entity;

architecture Behavioral of NashCipher is
    type state_type is range 0 to 127;
    signal current_state : state_type := 0;
    signal red_permuter : array (0 to 127) of state_type;
    signal blue_permuter : array (0 to 127) of state_type;
    signal flip_matrix : array (0 to 127) of std_logic;
begin
    process(clk, reset)
    begin
        if reset = '1' then
            current_state <= 0;
        elsif rising_edge(clk) then
            if decider_bit = '0' then
                current_state <= red_permuter(current_state);
            else
                current_state <= blue_permuter(current_state);
            end if;

            if flip_matrix(current_state) = '1' then
                output_bit <= not input_bit;
            else
                output_bit <= input_bit;
```

```
            end if;
        end if;
    end process;
end architecture;
```

Explanation of Nash Cipher Operation

The Nash Cipher operates as a deterministic state machine defined by:

- A **Red Permuter** and **Blue Permuter**, each mapping 128 possible states to 128 possible next states.
- A **Flip Matrix** of 128 bits, indicating whether the message bit should be flipped (inverted) at a given state.

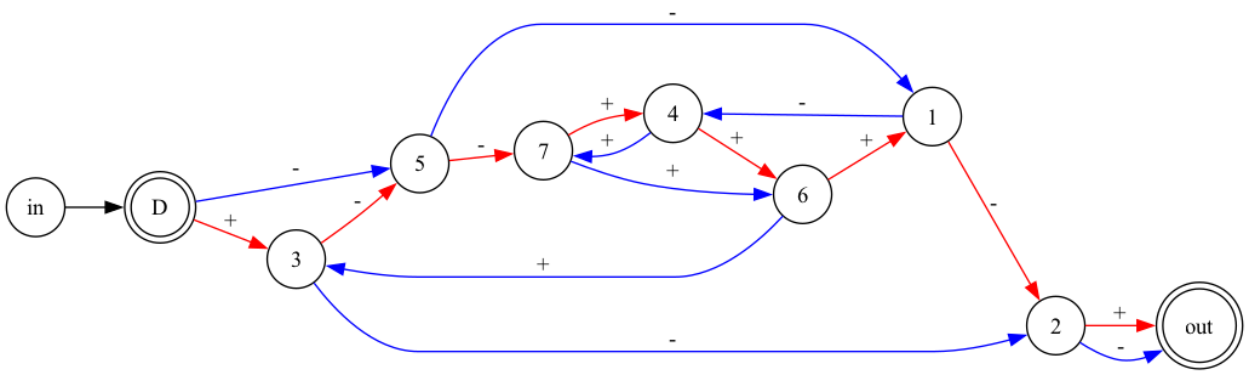

Figure 1: Illustration of Permuted State Transition Network

Figure 1 illustrates a small example of the dual permuter structure used in the Nash Cipher. Each node represents a cipher state, and the colored arrows represent transitions between states based on the decider bit: red arrows for a decider bit of 0, blue arrows for a decider bit of 1. The system dynamically selects a path depending on the value of an incoming message bit, applying flip operations as specified by the internal flip matrix.

The process is:

1. **Input Bit Arrival**: An incoming `input_bit` enters the system.
2. **Decision Based on Future Bit**: A `decider_bit`, drawn from a future position in the message stream (i.e., "one bit to the left"), determines which permuter is used:
   - If `decider_bit = 0`, the Red Permuter is used.
   - If `decider_bit = 1`, the Blue Permuter is used.
3. **State Transition**: Based on the selected permuter, the system advances to a new internal `current_state`.
4. **Bit Flip or Pass-Through**: The system checks the `flip_matrix` at the new `current_state`:
   - If the flip bit is `1`, the `input_bit` is inverted (`NOT input_bit`).
   - If the flip bit is `0`, the `input_bit` passes through unchanged.
5. **Output**: The resulting `output_bit` is emitted and the system awaits the next `input_bit`.

Key Generation and Randomization

- The **Red** and **Blue Permuters** and the **Flip Matrix** are generated from a random seed, optionally produced by the MKRAND generator described previously.
- To generate the cipher state:
  - Start with an ordered list of numbers from 0 to 127.
  - Apply a pseudo-random shuffle to this list to create the Red Permuter.
  - Repeat independently to create the Blue Permuter.
  - Generate 128 random bits to populate the Flip Matrix.
- These random structures define a unique cipher instance that can only be decrypted by reconstructing the same state machine.

Decryption follows the exact same steps, provided that both the sender and receiver have identical Red/Blue permuters and Flip Matrix structures. Reversibility is guaranteed by deterministic state evolution.

The cipher's strength derives from the combined complexity of:

- **Permuter Randomness**: Randomized state transitions.
- **Flip Randomness**: Randomized bit inversions.
- **Dynamic Traversal**: The traversal path is not static but determined dynamically by message input bits.

The system can be scaled to larger or smaller state sizes as required by application constraints.

Full 128-State Permuter Visualization

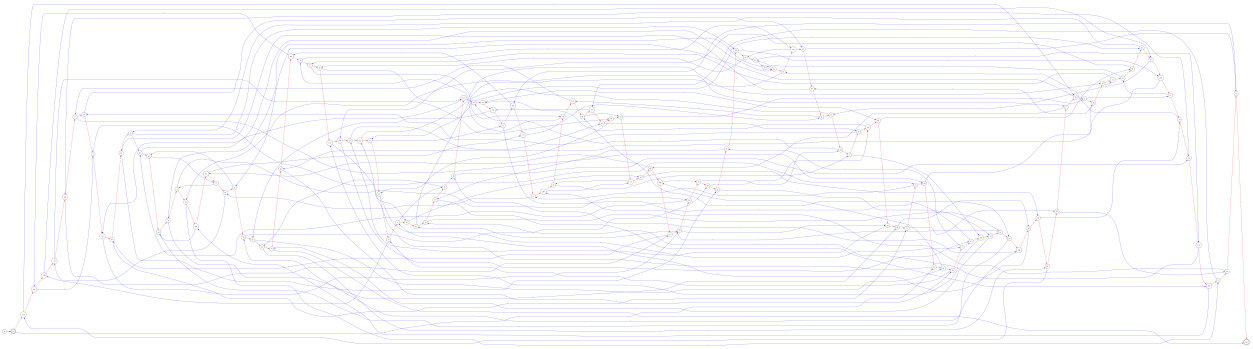

Figure 2: The above visualization depicts the full 128-state dual-permuter finite state machine (FSM) utilized by the Nash Cipher. Each node represents a unique internal state, and each directed edge represents a deterministic state transition triggered by an incoming bit, with color coding (red and blue) indicating the permuter path chosen by the "decider". Although the structure appears overwhelmingly complex at first glance, the cipher operates as a simple digital state machine — requiring only basic memory lookups and minimal logic per bit processed. Despite its apparent complexity, the implementation is vastly more computationally efficient than traditional math-based cryptosystems (e.g., AES, RSA, ECC), achieving encryption with orders of magnitude less energy and structural overhead.

Although the full 128-state dual permuter structure may appear complex visually, the underlying mechanism is fundamentally simple. It consists solely of deterministic state transitions and conditional bit inversions, requiring minimal computational effort per bit processed.

This architecture highlights that cryptographic strength can be achieved through structural randomness and dynamic traversal without reliance on complex mathematical operations. The Nash Cipher operates using basic digital logic, resulting in significantly lower energy consumption, faster throughput, and reduced implementation complexity compared to traditional math-intensive ciphers such as AES, RSA, and ECC.

By aligning encryption mechanisms more closely with native digital hardware behavior, the Nash Cipher demonstrates a practical and scalable path forward for cryptographic systems, particularly for resource-constrained or high-performance environments.

Comparison to Existing Cryptographic Systems

This section presents a comparative analysis between the Nash Cipher and widely used cryptographic algorithms, focusing on energy efficiency and computational simplicity.

Nash Cipher Energy Estimation

Each bit processed by the Nash Cipher requires:

- One memory lookup to select the next state via the Red or Blue Permuter.
- One memory lookup to retrieve the flip bit from the Flip Matrix.
- One NOT operation if the flip bit is set.

Thus, each bit requires approximately:

$$2 \times 0.1\,\text{pJ (SRAM reads)} + 0.1\,\text{pJ (NOT gate)} = 0.3\,\text{pJ per bit}$$

Processing 128 bits per clock cycle:

$$128 \times 0.3\,\text{pJ} = 38.4\,\text{pJ per cycle}$$

At a clock frequency of 200 MHz:

$$38.4 \times 10^{-12} \times 200 \times 10^{6} = 7.68\,\text{mW}$$

Thus, Nash Cipher hardware would consume approximately **7.7 milliwatts** during full-speed encryption.

Comparative Table

Crypto System	Type	Approx. Energy	Speed
AES-128	Symmetric Block Cipher	Moderate (30–50mW ASIC)	Fast
RSA-2048	Asymmetric Public Key	Very High (CPU intensive)	Very Slow
ECC (P-256)	Asymmetric Key Exchange	High (50–100mW)	Moderate
Nash Cipher	Symmetric Stream Cipher	Ultra-Low (∼7.7mW)	Ultra-Fast

Projected Performance Comparison Across Platforms

The Nash Cipher's simple and deterministic structure allows it to achieve extremely high speeds across a wide range of hardware implementations, from general-purpose CPUs to dedicated FPGA and ASIC designs.

Below is a comparative projection:

Technology	Platform	Throughput (Mbps)	Relative Complexity
Nash Cipher	FPGA (mid-range)	∼1,000	Very Low
Nash Cipher	ASIC (optimized)	>10,000	Extremely Low
AES-128	FPGA (mid-range)	∼500	Moderate
AES-128	ASIC (optimized)	∼2,000–5,000	Moderate
RSA-2048	CPU (server-grade)	∼0.1	Extremely High
ECC (P-256)	CPU (server-grade)	∼0.5	Very High

Notes

- **Nash Cipher FPGA**: Even on inexpensive FPGAs, Nash Cipher can process streams at over 1 Gbps due to its simple memory lookups and bit flips.
- **Nash Cipher ASIC**: A full-custom ASIC implementation could easily exceed 10 Gbps throughput with minimal power consumption.
- **AES**: Requires multiple rounds of substitution and permutation, increasing logic complexity and limiting clock frequency.
- **RSA/ECC**: Inherently slow due to reliance on modular exponentiation and elliptic curve operations, unsuited for high-speed data streams.

Implication

The Nash Cipher is uniquely positioned to provide high-throughput encryption for bandwidth-intensive, latency-sensitive, and energy-constrained applications — a role traditionally difficult for conventional cryptographic algorithms to fulfill.

Summary of the Invention

This invention provides a system and method for:

- Encrypting and decrypting data using a hardware-optimized state machine composed of dual parallel permuted networks.
- Employing a 128-state (or variable-sized) architecture aligned with digital system standards for simplicity and scalability.
- Generating the cipher structure dynamically from high-entropy sources, such as the MKRAND generator, to ensure cryptographic uniqueness.
- Enabling efficient, high-speed, low-power encryption operations suitable for embedded devices, mobile platforms, and offline-capable systems.

The encryption engine operates as follows:

1. Two independent permutation networks, referred to as the "Red" and "Blue" permuters, are constructed. Each is a complete and unique mapping of the system's states.
2. Each transition between states is annotated with a control bit indicating whether the message bit should be inverted (**1**) or passed unchanged (**0**).
3. Incoming message bits are directed through either the Red or Blue permuter based on a decider bit, derived from a future position in the message stream.
4. As the message traverses the state machine, its bits are selectively flipped or transmitted unchanged, based on the control bits encountered along the path.

5. After traversal, the output bits are reassembled into the final ciphertext.

Decryption uses the same structure and processes, relying on the deterministic reversibility of the system. Security is derived from the combined complexity of the random permutations, flip operations, and dynamic traversal paths, yielding a cipher that is simple to implement yet extremely resistant to cryptanalytic attacks.

Alternative Embodiments

While the core implementation of the Nash Cipher described herein utilizes a 128-state binary structure, the invention is not limited to binary systems nor to this specific number of states.

Possible alternative embodiments include:

- **Variable State Sizes**: The architecture can be scaled to smaller or larger state counts (e.g., 64, 256, or 512 states) depending on performance and security requirements.
- **Ternary and Multivalued Logic**: Instead of binary bits, the cipher can be adapted to operate on ternary (three-valued) or higher-order digital logic systems, with corresponding changes to the state transition rules and flip mechanisms.
- **Hardware Specialization**: Implementations can be optimized for specific platforms including FPGAs, custom ASICs, or even neuromorphic and memristor-based architectures.

The general principles of dual-permuter traversal, dynamic state-dependent behavior, and structural randomness remain applicable across all these variations.

Example Applications

The Nash Cipher's properties of low power consumption, minimal computational complexity, and high throughput make it particularly suited for a wide range of applications, including but not limited to:

- Embedded cryptographic modules for Internet-of-Things (IoT) devices.
- Secure communications for mobile and wearable platforms.
- Onboard encryption for small satellite and CubeSat systems where power and computational resources are highly constrained.
- High-speed encryptors for real-time video, sensor, or telemetry streams.
- Hardware-resident security modules for autonomous vehicles, drones, and robotic systems.

Conclusion

The Nash Cipher represents a breakthrough in cryptographic system design, translating elegant mathematical theory into a practical, hardware-friendly implementation. By leveraging simple deterministic structures, randomized permutation wiring, and dynamic traversal based on incoming data, the cipher achieves:

- High security without dependence on complex algebraic operations or computationally expensive primitives.
- Ultra-low power consumption, enabling practical deployment in embedded, mobile, and IoT environments.
- Exceptional speed and scalability, with a fixed and minimal hardware footprint.
- Natural resistance to fault injection and simple side-channel attacks due to absence of large modular multipliers or complex arithmetic.

Unlike conventional encryption systems that rely on iterative complexity and large key sizes, the Nash Cipher achieves security through structural randomness and dynamic behavior at the digital logic level. This approach creates an encryption mechanism that is resistant to side-channel analysis, efficient for hardware realization, and naturally suited to the evolving needs of a connected world.

By combining modern digital techniques with foundational cryptographic insights, the Nash Cipher offers a new direction for the future of secure communications.