# Hash-Based Encryption with Digest Selection: A Novel Pre-Image Security Primitive

Cris, DOSAYGO

December 27, 2024

## Abstract

This note introduces a hash-based encryption scheme leveraging pre-image resistance and digest indexing for constructing plaintext blocks. Designed for flexibility, the scheme supports multiple digest selection modes (prefix, sequence, scatter) and optional plaintext compression. It offers strong resistance to brute-force and timing attacks, though challenges include ciphertext expansion and mining speed. A proof-of-concept implementation is provided for further exploration and analysis by the cryptographic community.

## Introduction

This note introduces a novel hash-based encryption primitive, leveraging the pre-image resistance of cryptographic hash functions. The scheme encrypts plaintext by mapping blocks into keyed hash digests, where the plaintext is constructed by selecting specific parts of the digest via a mapping function. By design, this approach ensures robustness against brute-force and timing attacks.

Unlike a regular block cipher, the security of this system is not affected by the plaintext block size, but rather by the pre-image size mined to align with it.

The encryption is flexible, operating in multiple modes (prefix, sequence, scatter) and supports optional compression for redundancy reduction. This document formalizes the construction, analyzes its tradeoffs, and outlines future directions.

**Disclaimer:** This document introduces an experimental concept intended for research and analysis by cryptographers. The scheme presented here has not been formally proven secure or subjected to rigorous cryptanalysis.

## Encryption: Core Equation

The encryption process solves the following equation:

$$\text{Map}(H(K||N)) = P$$

where:

- $H$: A cryptographic hash function (e.g., SHA-256, BLAKE3).

- $K$: A secret key with sufficient entropy (e.g., 128 bits).
- $N$: A per-block random nonce (e.g., 64 bits).
- $P$: The plaintext block (e.g., 3 bytes, but any length is permitted. Longer blocks increase enciphering (mining) time).
- Map: A mapping function that selects specific parts of the digest $H(K||N)$ to construct $P$.

The plaintext block $P$ is derived from the digest as:

$$P = H(K||N) \otimes \text{Map}$$

where $\otimes$ denotes the selection operation defined by Map, which determines which parts of the digest contribute to $P$.

### Encryption Modes

The mapping Map can operate in several modes:

- **Prefix Mode:** $P$ matches the first $n$-bytes of the digest.
- **Sequence Mode:** $P$ matches a contiguous substring of the digest.
- **Scatter Mode:** $P$ matches bytes at arbitrary indices in the digest, with no duplicates.

Scatter mode ensures that each index is used only once, preventing patterns in the plaintext $P$ from being exposed.

### Ciphertext Encoding

The ciphertext for each block $P$ is:

$$C = (N, \text{indices})$$

where:

- $N$: The nonce used to generate the digest.
- indices: The mapping indices required to reconstruct $P$ from $H(K||N)$.

## Decryption

Decryption reconstructs the plaintext block $P$ as:

$$P = H(K||N) \otimes \text{indices}$$

The hash digest is computed with the provided $N$ and $K$, and the mapping defined by indices extracts the plaintext $P$.

## Security Analysis

### Pre-Image Security

The security of the scheme relies on the pre-image resistance of $H$. Specifically, given $P$, $N$, and indices, an attacker must solve:

$$H(K||N) = P$$

The effective brute-force effort is:

$$2^{|K|+|\text{nonce}|}$$

where $|K|$ and $|\text{nonce}|$ are the bit lengths of the key and nonce, respectively.

### Preventing Redundancy Patterns

Scatter mode ensures that each index is unique, preventing repeated plaintext bytes from being mapped to the same digest index. This avoids revealing plaintext patterns and ensures uniform utilization of the digest.

### Ciphertext Expansion

Ciphertext expansion is determined by:

$$\text{Overhead} = |\text{nonce}| + |\text{indices}|$$

Scatter mode incurs higher overhead due to the need to store an index for each byte of $P$. Optional compression reduces redundancy in the plaintext before encryption, offsetting this overhead.

### Timing Security

Mining the nonce $N$ involves a probabilistic search for a valid mapping $\text{Map}(H(K||N)) = P$. The stochastic runtime behavior ensures that timing attacks are mitigated, as the encryption time varies unpredictably.

## Tradeoffs

- **Efficiency vs. Expansion:** Prefix and sequence modes minimize ciphertext size but take longer to mine than the looser matching of Scatter mode. Scatter mode is faster to mine but increases expansion.
- **Compression:** Reduces plaintext redundancy, often resulting in smaller effective ciphertext size despite normal expansion overhead.
- **Mining Speed:** Smaller blocks are faster to mine encrypt, but larger blocks reduce the total number of blocks and nonces.

## Tradeoff Summary

| Aspect | Advantage | Limitation |
|---|---|---|
| Block size | Smaller blocks speed up mining | Larger ciphertext due to nonce overhea |
| Ciphertext expansion | Mitigated by compression | Compression adds preprocessing compl |
| Mining modes | Scatter mode provides faster mining | Requires storing index for every plainte |
| Timing security | Stochastic runtime eliminates fixed patterns | Mining is probabilistic and not constan |

## Proof of Concept

The repository contains a working proof-of-concept toy implementation demonstrating the feasibility of this hash-based encryption scheme. This toy is not attacked or analyzed and should not be used by 3rd-parties (ie, you!) for securing valuables. This toy cipher also does not use established cryptographic hash functions but instead uses the hashes defined in this repository (which nevertheless are high quality and fast hash functions that pass SMHasher3). The code in the toy supports all defined digest search modes (prefix, sequence, series/scatter) and uses a constant key per session, without a key schedule. It also incorporates compression for improved storage efficiency.

The repository can be found at: https://github.com/DOSAYGO-Research/rain.

To build and run the example:

```
# Clone the repository
git clone https://github.com/DOSAYGO-Research/rain
cd rain

# Build using make
make

# or if encountering problems, try
./scripts/build.sh

# Example usage
# Encrypt a file with sequence mode
./rain/bin/rainsum -m enc --search-mode sequence input_file.txt

# Decrypt the file
./rain/bin/rainsum -m dec encrypted_file.rc

# Verify the decrypted file matches the original
diff input_file.txt decrypted_file.txt
```

The repository includes a suite of test scripts to validate the correctness of the implementation across various configurations, ensuring decrypted contents match the originals. These tests systematically evaluate different combinations of hash functions, digest sizes, nonce sizes, block sizes, and input files. The test suite can be executed as follows:

```
./scripts/test_cipher.sh
```

## Future Work

Future enhancements include:

- Introducing a key schedule to vary $K$ across blocks, improving resistance against related-key attacks. Most likely using the hash function in an XOF mode.
- Optimizing for parallelized mining to accelerate encryption.
- Extending compatibility with alternative hash functions for performance tuning.
- Formalizing security proofs under cryptographic assumptions.

## Conclusion

This hash-based encryption primitive provides a secure and flexible mechanism for protecting data. By leveraging cryptographic hash functions and indexed digest selection, it ensures strong security against brute-force and timing attacks. While current limitations include ciphertext expansion and mining speed, future improvements aim to address these tradeoffs and expand its applicability.