Network Working Group
Internet-Draft
Intended status: Informational
Expires: 7 March 2026

F. Denis
Fastly Inc.

3 September 2025

Methods for IP Address Encryption and Obfuscation
draft-denis-ipcrypt-08

Abstract

IP addresses are personally identifiable information that requires protection, yet common techniques such as truncation destroy data irreversibly while providing inconsistent privacy guarantees, and ad-hoc encryption schemes often lack interoperability and security analysis.

This document specifies secure, efficient methods for encrypting IP addresses for privacy-preserving storage, logging, and analytics. The methods enable data analysis while protecting user privacy from third parties without key access, addressing data minimization concerns raised in [RFC6973].

Three concrete instantiations are defined: ipcrypt-deterministic provides deterministic, format-preserving encryption, while ipcrypt-nd and ipcrypt-ndx introduce randomness to prevent correlation. All methods are reversible with the encryption key and designed for high-performance processing at network speeds.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/jedisct1/draft-denis-ipcrypt.

Status of This Memo

This Internet-Draft will expire on 7 March 2026.

Copyright Notice

Table of Contents

1. Introduction

IP addresses are personally identifiable information requiring protection, yet common anonymization approaches have fundamental limitations. Truncation (zeroing parts of addresses) irreversibly destroys data while providing unpredictable privacy levels - a /24 mask may hide one user or thousands depending on network allocation. Hashing produces non-reversible outputs unsuitable for operational tasks such as abuse investigation. Ad-hoc encryption schemes often lack rigorous security analysis and cannot interoperate between systems.

This document addresses these deficiencies by specifying secure, efficient, and interoperable methods for IP address encryption and obfuscation. The objective is to enable network operators, researchers, and privacy advocates to share or analyze data while protecting sensitive address information through cryptographically sound techniques.

This work directly addresses concerns raised in [RFC7624] regarding confidentiality when sharing data with third parties. Unlike existing practices that merely obscure addresses, these methods provide mathematically provable security properties, discussed throughout this document and summarized in Section 7.

1.1. Use Cases and Motivations

Organizations handling IP addresses must protect user privacy while maintaining operational capabilities. Generic encryption systems, though secure, are poorly suited for IP addresses - they expand data unpredictably, break compatibility with network tools, and operate too slowly for high-volume processing. The specialized methods in this specification resolve these conflicts through purpose-built cryptographic techniques:

* Efficiency and Compactness: All variants operate on exactly 128 bits, achieving single-block encryption speed critical for network-rate processing. Non-deterministic variants add only 8-16 bytes of tweak overhead versus potentially hundreds of bytes with generic encryption. This difference enables processing addresses in real-time rather than requiring expensive batch operations.

* High Usage Limits: Non-deterministic variants safely handle massive volumes - approximately 4 billion operations for ipcrypt-nd and 18 quintillion for ipcrypt-ndx per key - without degrading security. Generic encryption often requires operationally complex key rotation schemes at much lower thresholds.

* Format Preservation (Deterministic): The ipcrypt-deterministic variant produces valid IP addresses, not arbitrary ciphertext. This enables encrypted addresses to flow through existing network infrastructure, monitoring tools, and databases without modification (see Section 5.2).

* Interoperability: This specification ensures that encrypted IP addresses can be exchanged between different systems, vendors, and programming languages. All conforming implementations produce identical results, enabling seamless data exchange and avoiding vendor lock-in.

These specialized encryption methods unlock several critical use cases:

* Privacy Protection: They prevent the exposure of sensitive user information to third parties in logs, analytics data, and network measurements ([RFC6973]). Note that protection is specifically against parties without key access; the key holder retains full decryption capability.

* Correlation Attack Resistance: While deterministic encryption can reveal repeated inputs, the non-deterministic variants leverage random tweaks to hide patterns and enhance confidentiality (see Section 6).

* Privacy-Preserving Analytics: Encrypted IP addresses can be used directly for operations such as counting unique clients, rate limiting, or deduplication—without needing to reveal the original values to third-party processors. This approach addresses the anonymization requirements for DNS query data sharing outlined in [RSSAC040], enabling research while protecting source IP privacy. Since network hierarchy and geographic relationships are not preserved by encryption, organizations requiring such metadata SHOULD extract and store it separately (e.g., country, ASN) rather than relying on the flawed practice of IP address truncation, which provides inconsistent privacy protection and irreversibly destroys information.

* Seamless Third-Party Integration: Encrypted IPs can act as privacy-preserving identifiers when interacting with untrusted services, cloud providers, or external platforms.

For implementation guidelines and practical examples, see Section 8.

1.2. Relationship to IETF Work

This section is to be removed before publishing as an RFC.

This document does not conflict with any active IETF working group efforts. While the IETF has produced several RFCs related to privacy ([RFC6973], [RFC7258], [RFC7624]), there is no current standardization effort for IP address encryption methods. This specification complements existing IETF privacy guidance by providing concrete implementation methods.

The cryptographic primitives used (AES, format-preserving encryption) align with IETF cryptographic recommendations, and the document follows IETF formatting and terminology conventions where applicable.

2. Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Throughout this document, the following terms and conventions apply:

* IP Address: An IPv4 or IPv6 address as defined in [RFC4291].

* 16-Byte Representation: A fixed-length representation used for both IPv4 (via IPv4-mapped IPv6) and IPv6 addresses.

* Tweak: A non-secret, additional input to a tweakable block cipher that further randomizes the output.

* Deterministic Encryption: Encryption that always produces the same ciphertext for a given input and key.

* Non-Deterministic Encryption: Encryption that produces different ciphertexts for the same input due to the inclusion of a randomly sampled tweak.

* (Input, Tweak) Collision: A scenario where the same input is encrypted with the same tweak. This reveals that the input was repeated but not the input’s value.

3. IP Address Conversion

This section describes the conversion of IP addresses to and from a 16-byte representation. This conversion is necessary to operate a 128-bit cipher on both IPv4 and IPv6 addresses.

3.1. Converting to a 16-Byte Representation

3.1.1. IPv6 Addresses

IPv6 addresses are natively 128 bits and are converted directly using network byte order (big-endian) as specified in [RFC4291].

Example:

```
IPv6 Address:    2001:0db8:85a3:0000:0000:8a2e:0370:7334
16-Byte Representation: [20 01 0d b8 85 a3 00 00 00 00 8a 2e 03 70 73 34]
```

3.1.2. IPv4 Addresses

IPv4 addresses (32 bits) are mapped using the IPv4-mapped IPv6 format as specified in [RFC4291]:

```
IPv4 Address:    192.0.2.1
16-Byte Representation: [00 00 00 00 00 00 00 00 00 00 FF FF C0 00 02 01]
```

3.2. Converting from a 16-Byte Representation to an IP Address

The conversion algorithm is as follows:

1. Examine the first 12 bytes of the 16-byte representation

2. If they match the IPv4-mapped prefix (10 bytes of 0x00 followed by 0xFF, 0xFF):

    * Interpret the last 4 bytes as an IPv4 address in dotted-decimal notation

3. Otherwise:

    * Interpret the 16 bytes as an IPv6 address in colon-hexadecimal notation

4. Generic Constructions

This specification defines two generic cryptographic constructions:

1. 128-bit Block Cipher Construction:

    * Used in deterministic encryption (see Section 5)

    * Operates on a single 16-byte block

    * Example: AES-128 treated as a permutation

2. 128-bit Tweakable Block Cipher (TBC) Construction:

    * Used in non-deterministic encryption (see Section 6)

    * Accepts a key, a tweak, and a message

    * The tweak must be uniformly random when generated

    * Reuse of the same tweak on different inputs does not compromise confidentiality

Valid options for implementing a tweakable block cipher include, but are not limited to:

* SKINNY (see [SKINNY])
* DEOXYS-BC (see [DEOXYS-BC])
* KIASU-BC (see Section 8.8 for implementation details)
* AES-XTS (see Section 6.4.2 for usage)

Implementers MUST choose a cipher that meets the required security properties and provides robust resistance against related-tweak and other cryptographic attacks.

5. Deterministic Encryption

Deterministic encryption applies a 128-bit block cipher directly to the 16-byte representation of an IP address. The defining characteristic is that the same IP address always encrypts to the same ciphertext when using the same key - this predictability is both its strength and limitation.

Choose deterministic encryption when:

* Duplicate IP addresses need to be detected in encrypted form (e.g., for rate limiting)
* Storage space is critical (produces only 16 bytes output)
* Format preservation is required (output remains a valid IP address)
* Correlation of the same address across records is acceptable

All instantiations documented in this specification (ipcrypt-deterministic, ipcrypt-nd, and ipcrypt-ndx) are invertible - encrypted IP addresses can be decrypted back to their original values using the same key. For non-deterministic modes, the tweak must be preserved along with the ciphertext to enable decryption.

For implementation details, see Section 8.

5.1. ipcrypt-deterministic

The ipcrypt-deterministic instantiation employs AES-128 in a single-block operation. The key MUST be exactly 16 bytes (128 bits) in length. Since AES-128 is a permutation, every distinct 16-byte input maps to a unique 16-byte ciphertext, preserving the IP address format.

For test vectors, see Appendix A.1.

```
+---------------------+
|      IP Address     |
|    (IPv4 or IPv6)   |
+---------------------+
           |
           v
+---------------------+
| Convert to 16 Bytes |
+---------------------+
           |
           v
+---------------------+
|   AES128 Encrypt    |
|   (Single Block)    |
+---------------------+
           |
           v
+---------------------+
|    16-Byte Output   |
+---------------------+
           |
           v
+---------------------+
| Convert to IP Format|
+---------------------+
```

5.2. Format Preservation and Limitations

5.2.1. Network Hierarchy Preservation

The encryption methods described in this specification do not preserve network hierarchy or prefix relationships.

* IPv4 and IPv6 prefixes are completely scrambled in the encrypted output

* Addresses from the same subnet will not appear related after encryption

* Geographic or topological proximity cannot be inferred from encrypted addresses

5.2.2. Format Preservation for IPv4

The methods specified in this document typically result in IPv4 addresses being encrypted as IPv6 addresses.

IPv4 format preservation (maintaining IPv4 addresses as IPv4 rather than mapping them to IPv6) is not specified in this document and is generally discouraged due to the limited 32-bit address space, which significantly reduces encryption security.

If IPv4 format preservation is absolutely required despite the security limitations, implementers SHOULD implement a Format-Preserving Encryption (FPE) mode such as the FF1 algorithm specified in [NIST-SP-800-38G] or FAST [FAST].

5.2.3. Preserving Metadata for Analytics

Organizations requiring network metadata for analytics SHOULD extract and store geographic location, ASN, or network classification separately before encryption, rather than using IP address truncation. Truncation (e.g., storing only /24 or /48 prefixes) is a fundamentally flawed privacy mechanism that provides inconsistent protection and irreversibly destroys data.

Recommended approach: 1. Extract metadata (geographic location, ASN, network type) from the original IP address 2. Store this information as separate fields alongside the encrypted IP address 3. Apply appropriate privacy-preserving aggregation to the metadata itself

Example storage schema: ~~~ { “encrypted_ip” : “bde9:6789:d353:824c:d7c6:f58a:6bd2:26eb” , “country” : “US” , “asn” : 15169, “network_type” : “cloud_provider” } ~~~

This approach ensures consistent privacy protection through proper encryption while preserving analytical capabilities in a controlled manner.

6. Non-Deterministic Encryption

Non-deterministic encryption enhances privacy by ensuring that the same IP address produces different ciphertexts each time it is encrypted, preventing correlation attacks that plague deterministic schemes. This is achieved through tweakable block ciphers that incorporate random values called tweaks.

Choose non-deterministic encryption when: - Preventing correlation of the same IP address across records is critical - Storage can accommodate the additional tweak data (8-16 bytes) - Stronger privacy guarantees than deterministic encryption provides are required - Processing the same address multiple times without revealing repetition patterns

For implementation details, see Section 8.

6.1. Encryption Process

The encryption process for non-deterministic modes consists of the following steps:

1. Generate a random tweak using a cryptographically secure random number generator

2. Convert the IP address to its 16-byte representation

3. Encrypt the 16-byte representation using the key and the tweak

4. Concatenate the tweak with the encrypted output to form the final ciphertext

The tweak is not considered secret and is included in the ciphertext. This allows the same tweak to be used for decryption.

6.2. Decryption Process

The decryption process consists of the following steps:

1. Split the ciphertext into the tweak and the encrypted IP

2. Decrypt the encrypted IP using the key and the tweak

3. Convert the resulting 16-byte representation back to an IP address

Although the tweak is generated uniformly at random, occasional collisions may occur according to birthday bounds. Such collisions are benign when they occur with different inputs. An (input, tweak) collision reveals that the same input was encrypted with the same tweak but does not disclose the input’s value. The usage limits discussed below apply per cryptographic key; rotating keys can extend secure usage beyond these bounds.

6.3. Output Format and Encoding

The output of non-deterministic encryption is binary data. For applications that require text representation (e.g., logging, JSON encoding, or text-based protocols), the binary output MUST be encoded. Common encoding options include hexadecimal and Base64. The choice of encoding is application-specific and outside the scope of this specification. However, implementations SHOULD document their chosen encoding method clearly.

6.4. Concrete Instantiations

This document defines two concrete instantiations:

* ipcrypt-nd: Uses the KIASU-BC tweakable block cipher with an 8-byte (64-bit) tweak. See [KIASU-BC] for details.

* ipcrypt-ndx: Uses the AES-XTS tweakable block cipher with a 16-byte (128-bit) tweak. See [XTS-AES] for background.

In both cases, if a tweak is generated randomly, it MUST be uniformly random. Reusing the same randomly generated tweak on different inputs is acceptable from a confidentiality standpoint.

For test vectors, see Appendix A.2 and Appendix A.3.

6.4.1. ipcrypt-nd (KIASU-BC)

The ipcrypt-nd instantiation uses the KIASU-BC tweakable block cipher with an 8-byte (64-bit) tweak. For implementation details, see Section 8.8. The output is 24 bytes total, consisting of an 8-byte tweak concatenated with a 16-byte ciphertext.

Random sampling of an 8-byte tweak yields an expected collision for a specific tweak value after about 2^(64/2) = 2^32 operations (approximately 4 billion operations). If an (input, tweak) collision occurs, it indicates that the same input was processed with that tweak without revealing the input’s value.

These collision bounds apply per cryptographic key. By rotating keys regularly, secure usage can be extended well beyond these bounds. The effective security is determined by the underlying block cipher’s strength.

For test vectors, see Appendix A.2.

6.4.2. ipcrypt-ndx (AES-XTS)

The ipcrypt-ndx instantiation uses the AES-XTS tweakable block cipher with a 16-byte (128-bit) tweak. The output is 32 bytes total, consisting of a 16-byte tweak concatenated with a 16-byte ciphertext.

For AES-XTS encryption of a single block, the computation avoids the sequential tweak calculations required in full XTS mode. Independent sampling of a 16-byte tweak results in an expected collision after about 2^(128/2) = 2^64 operations (approximately 18 quintillion operations).

As with ipcrypt-nd, an (input, tweak) collision reveals repetition without compromising the input value. These limits are per key, and regular key rotation further extends secure usage. The effective security is governed by the strength of AES-128 (approximately 2^128 operations).

6.4.3. Comparison of Modes

Choosing the right mode depends on specific privacy requirements and operational constraints:

* Deterministic (ipcrypt-deterministic):

  - Output size: 16 bytes (most compact)

  - Privacy: Same IP always produces same ciphertext (allows correlation)

  - Use case: When duplicate identification is needed or when format preservation is critical

  - Performance: Fastest (single AES operation)

* Non-Deterministic ipcrypt-nd (KIASU-BC):

  - Output size: 24 bytes (16-byte ciphertext + 8-byte tweak)

  - Privacy: Same IP produces different ciphertexts (prevents most correlation)

  - Use case: General privacy protection with reasonable storage overhead

  - Collision resistance: Approximately 4 billion operations per key

* Non-Deterministic ipcrypt-ndx (AES-XTS):

  - Output size: 32 bytes (16-byte ciphertext + 16-byte tweak)

  - Privacy: Same IP produces different ciphertexts (prevents correlation)

  - Use case: Maximum privacy protection when storage permits

  - Collision resistance: Approximately 18 quintillion operations per key

6.5. Alternatives to Random Tweaks

While this specification recommends the use of uniformly random tweaks for non-deterministic encryption, implementers may consider alternative approaches:

* Monotonic Counter: A counter could be used as a tweak, but this is difficult to maintain in distributed systems. If the counter is not encrypted and the tweakable block cipher is not secure against related-tweak attacks, this could enable correlation attacks.

* UUIDs: UUIDs (such as UUIDv6 or UUIDv7) could be used as tweaks; however, these would reveal the original timestamp of the logged IP addresses, which may not be desirable from a privacy perspective.

Although the birthday bound is a concern with random tweaks, the use of random tweaks remains the recommended and most practical approach, offering the best tradeoffs for most real-world use cases.

7. Security Considerations

The methods specified in this document provide strong confidentiality guarantees but explicitly do not provide integrity protection. Understanding this distinction is critical for secure deployment:

What these methods protect against:

* Unauthorized parties learning the original IP addresses (without the key)

* Statistical analysis revealing patterns in network traffic (non-deterministic modes)

* Brute-force attacks on the address space (128-bit security level)

What these methods do NOT protect against:

* Active attackers modifying, reordering, or removing encrypted addresses

* Authorized key holders decrypting addresses (by design)

* Traffic analysis based on volume and timing (metadata)

Applications requiring integrity protection must additionally employ authentication mechanisms such as HMAC, authenticated encryption modes, or digital signatures over the encrypted data. While outside this specification’s scope, implementers should carefully evaluate whether their threat model requires such additional protections.

7.1. Deterministic Mode Security

A permutation ensures distinct inputs yield distinct outputs. However, repeated inputs result in identical ciphertexts, thereby revealing repetition.

This property makes deterministic encryption suitable for applications where format preservation is required, but linkability of repeated inputs is acceptable.

7.2. Non-Deterministic Mode Security

The inclusion of a random tweak ensures that encrypting the same input generally produces different outputs. In cases where an (input, tweak) collision occurs, an attacker learns only that the same input was processed with that tweak, not the value of the input itself.

Security is determined by the underlying block cipher (2^128 for AES-128) on a per-key basis. Key rotation is recommended to extend secure usage beyond the per-key collision bounds.

7.3. Implementation Security

Implementations MUST ensure that:

1. Keys are generated using a cryptographically secure random number generator

2. Tweak values are uniformly random for non-deterministic modes

3. Side-channel attacks are mitigated through constant-time operations

4. Error handling does not leak sensitive information

7.4. Key Management Considerations

This specification focuses on the cryptographic transformations and does not mandate specific key management practices. However, implementers MUST ensure:

1. Keys are generated using cryptographically secure random number generators (see [RFC4086])

2. Keys are stored securely and access-controlled appropriately for the deployment environment

3. Key rotation policies are established based on usage volume and security requirements

4. Key compromise procedures are defined and tested

For high-volume deployments processing billions of IP addresses, regular key rotation (e.g., monthly or quarterly) is RECOMMENDED to stay well within the security bounds discussed in this document.

8. Implementation Details

This section provides detailed pseudocode and implementation guidance for the key operations described in this document.

8.1. Visual Diagrams

The following diagrams illustrate the key processes described in this specification.

8.1.1. IPv4 Address Conversion Diagram

```
       IPv4: 192.0.2.1
                |
                v
  Octets:  C0  00  02  01
                |
                v
   16-Byte Array:
[00 00 00 00 00 00 00 00 00 00 | FF FF | C0 00 02 01]
```

8.1.2. Deterministic Encryption Flow

```
               IP Address
                   |
                   v
          [Convert to 16 Bytes]
                   |
                   v
       [AES-128 Single-Block Encrypt]
                   |
                   v
          16-Byte Ciphertext
                   |
                   v
        [Convert to IP Format]
                   |
                   v
          Encrypted IP Address
```

8.1.3. Non-Deterministic Encryption Flow (ipcrypt-nd)

```
                 IP Address
                     |
                     v
          [Convert to 16 Bytes]
                     |
                     v
       [Generate Random 8-Byte Tweak]
                     |
                     v
          [KIASU-BC Tweakable Encrypt]
                     |
                     v
             16-Byte Ciphertext
                     |
                     v
       [Concatenate Tweak || Ciphertext]
                     |
                     v
          24-Byte Output (ipcrypt-nd)
```

8.1.4. Non-Deterministic Encryption Flow (ipcrypt-ndx)

```
                 IP Address
                     |
                     v
          [Convert to 16 Bytes]
                     |
                     v
       [Generate Random 16-Byte Tweak]
                     |
                     v
          [AES-XTS Tweakable Encrypt]
                     |
                     v
             16-Byte Ciphertext
                     |
                     v
       [Concatenate Tweak || Ciphertext]
                     |
                     v
          32-Byte Output (ipcrypt-ndx)
```

8.2. IPv4 Address Conversion

For a diagram of this conversion process, see Section 8.1.1.

```
function IPv4To16Bytes(ipv4_address):
    // Split the IPv4 address into its octets
    parts = ipv4_address.split(".")
    if length(parts) != 4:
         raise Error("Invalid IPv4 address")
    // Create a 16-byte array with the IPv4-mapped prefix
    bytes16 = [0x00] * 10         // 10 bytes of 0x00
    bytes16.append(0xFF)          // 11th byte: 0xFF
    bytes16.append(0xFF)          // 12th byte: 0xFF
    // Append each octet (converted to an 8-bit integer)
    for part in parts:
         bytes16.append(int(part))
    return bytes16
```

Example: For "192.0.2.1", the function returns

[00, 00, 00, 00, 00, 00, 00, 00, 00, 00, FF, FF, C0, 00, 02, 01]

8.3. IPv6 Address Conversion

```
function IPv6To16Bytes(ipv6_address):
    // Parse the IPv6 address into eight 16-bit words.
    words = parseIPv6(ipv6_address)  // Expands shorthand notation and returns 8 words
    bytes16 = []
    for word in words:
         high_byte = (word >> 8) & 0xFF
         low_byte = word & 0xFF
         bytes16.append(high_byte)
         bytes16.append(low_byte)
    return bytes16
```

Example: For "2001:0db8:85a3:0000:0000:8a2e:0370:7334", the output is the corresponding 16-byte sequence.

8.4. Conversion from a 16-Byte Array to an IP Address

```
function Bytes16ToIP(bytes16):
    if length(bytes16) != 16:
         raise Error("Invalid byte array")

    // Check for the IPv4-mapped prefix
    if bytes16[0:10] == [0x00]*10 and bytes16[10] == 0xFF and bytes16[11] == 0xFF:
         ipv4_parts = []
         for i from 12 to 15:
             ipv4_parts.append(str(bytes16[i]))
         ipv4_address = join(ipv4_parts, ".")
         return ipv4_address
    else:
         words = []
         for i from 0 to 15 step 2:
             word = (bytes16[i] << 8) | bytes16[i+1]
             words.append(format(word, "x"))
         ipv6_address = join(words, ":")
         return ipv6_address
```

8.5. Deterministic Encryption (ipcrypt-deterministic)

```
function ipcrypt_deterministic_encrypt(ip_address, key):
    // The key MUST be exactly 16 bytes (128 bits) in length
    if length(key) != 16:
        raise Error("Key must be 16 bytes")

    bytes16 = convertTo16Bytes(ip_address)
    ciphertext = AES128_encrypt(key, bytes16)
    encrypted_ip = Bytes16ToIP(ciphertext)
    return encrypted_ip

function ipcrypt_deterministic_decrypt(encrypted_ip, key):
    if length(key) != 16:
        raise Error("Key must be 16 bytes")

    bytes16 = convertTo16Bytes(encrypted_ip)
    plaintext = AES128_decrypt(key, bytes16)
    original_ip = Bytes16ToIP(plaintext)
    return original_ip
```

8.6. Non-Deterministic Encryption using KIASU-BC (ipcrypt-nd)

```
function ipcrypt_nd_encrypt(ip_address, key):
    if length(key) != 16:
        raise Error("Key must be 16 bytes")

    // Step 1: Generate random tweak (8 bytes)
    tweak = random_bytes(8)  // MUST be uniformly random

    // Step 2: Convert IP to 16-byte representation
    bytes16 = convertTo16Bytes(ip_address)

    // Step 3: Encrypt using key and tweak
    ciphertext = KIASU_BC_encrypt(key, tweak, bytes16)

    // Step 4: Concatenate tweak and ciphertext
    result = concatenate(tweak, ciphertext)  // 8 bytes || 16 bytes = 24 bytes total
    return result

function ipcrypt_nd_decrypt(ciphertext, key):
    // Step 1: Split ciphertext into tweak and encrypted IP
    tweak = ciphertext[0:8]  // First 8 bytes
    encrypted_ip = ciphertext[8:24]  // Remaining 16 bytes

    // Step 2: Decrypt using key and tweak
    bytes16 = KIASU_BC_decrypt(key, tweak, encrypted_ip)

    // Step 3: Convert back to IP address
    ip_address = Bytes16ToIP(bytes16)
    return ip_address
```

8.7. Non-Deterministic Encryption using AES-XTS (ipcrypt-ndx)

```
function AES_XTS_encrypt(key, tweak, block):
    // Split the key into two halves
    K1, K2 = split_key(key)

    // Encrypt the tweak with the second half of the key
    ET = AES128_encrypt(K2, tweak)

    // Encrypt the block: AES128(block  ET, K1)  ET
    return AES128_encrypt(K1, block  ET)  ET
```

```
function ipcrypt_ndx_encrypt(ip_address, key):
    if length(key) != 32:
        raise Error("Key must be 32 bytes (two AES-128 keys)")

    // Step 1: Generate random tweak (16 bytes)
    tweak = random_bytes(16)  // MUST be uniformly random

    // Step 2: Convert IP to 16-byte representation
    bytes16 = convertTo16Bytes(ip_address)

    // Step 3: Encrypt using key and tweak
    ciphertext = AES_XTS_encrypt(key, tweak, bytes16)

    // Step 4: Concatenate tweak and ciphertext
    result = concatenate(tweak, ciphertext)  // 16 bytes || 16 bytes = 32 bytes total
    return result

function ipcrypt_ndx_decrypt(ciphertext, key):
    // Step 1: Split ciphertext into tweak and encrypted IP
    tweak = ciphertext[0:16]  // First 16 bytes
    encrypted_ip = ciphertext[16:32]  // Remaining 16 bytes

    // Step 2: Decrypt using key and tweak
    bytes16 = AES_XTS_decrypt(key, tweak, encrypted_ip)

    // Step 3: Convert back to IP address
    ip_address = Bytes16ToIP(bytes16)
    return ip_address
```

8.8. KIASU-BC Implementation Guide

This section provides a detailed guide for implementing the KIASU-BC tweakable block cipher used in ipcrypt-nd. KIASU-BC is based on AES-128 with modifications to incorporate a tweak.

8.8.1. Overview

KIASU-BC extends AES-128 by incorporating an 8-byte tweak into each round. The tweak is padded to 16 bytes and XORed with the round key at each round of the cipher. This construction is used in the ipcrypt-nd instantiation.

8.8.2. Tweak Padding

The 8-byte tweak is padded to 16 bytes using the following method:

1. Split the 8-byte tweak into four 2-byte pairs

2. Place each 2-byte pair at the start of each 4-byte group

3. Fill the remaining 2 bytes of each group with zeros

Example:

```
8-byte tweak:    [T0 T1 T2 T3 T4 T5 T6 T7]
16-byte padded:  [T0 T1 00 00 T2 T3 00 00 T4 T5 00 00 T6 T7 00 00]
```

8.8.3. Round Structure

Each round of KIASU-BC consists of the following standard AES operations:

1. SubBytes: Apply the AES S-box to each byte of the state

2. ShiftRows: Rotate each row of the state matrix

3. MixColumns: Mix the columns of the state matrix (except in the final round)

4. AddRoundKey: XOR the state with the round key and padded tweak

For details about these operations, see [FIPS-197].

8.8.4. Key Schedule

The key schedule follows the standard AES-128 key expansion:

1. The initial key is expanded into 11 round keys

2. Each round key is XORed with the padded tweak before use

3. The first round key is used in the initial AddRoundKey operation

8.8.5. Implementation Steps

1. Key Expansion:

   * Expand the 16-byte key into 11 round keys using the standard AES key schedule

   * Each round key is 16 bytes

2. Tweak Processing:

   * Pad the 8-byte tweak to 16 bytes as described above

   * XOR the padded tweak with each round key before use

3. Encryption Process:

   * Perform initial AddRoundKey with the first tweaked round key

   * For rounds 1-9:

     - SubBytes

     - ShiftRows

     - MixColumns

     - AddRoundKey (with tweaked round key)

   * For round 10 (final round):

     - SubBytes

     - ShiftRows

     - AddRoundKey (with tweaked round key)

8.8.6. Example Implementation

The following pseudocode illustrates the core operations of KIASU-BC:

```
function pad_tweak(tweak):
    // Input: 8-byte tweak
    // Output: 16-byte padded tweak
    padded = [0] * 16
    for i in range(0, 8, 2):
        padded[i*2] = tweak[i]
        padded[i*2+1] = tweak[i+1]
    return padded

function kiasu_bc_encrypt(key, tweak, plaintext):
    // Input: 16-byte key, 8-byte tweak, 16-byte plaintext
    // Output: 16-byte ciphertext

    // Expand key and pad tweak
    round_keys = expand_key(key)
    padded_tweak = pad_tweak(tweak)

    // Initial round
    state = plaintext
    state = add_round_key(state, round_keys[0] ^ padded_tweak)

    // Main rounds
    for round in range(1, 10):
        state = sub_bytes(state)
        state = shift_rows(state)
        state = mix_columns(state)
        state = add_round_key(state, round_keys[round] ^ padded_tweak)

    // Final round
    state = sub_bytes(state)
    state = shift_rows(state)
    state = add_round_key(state, round_keys[10] ^ padded_tweak)

    return state
```

Key and tweak sizes for each variant:

* ipcrypt-deterministic: Key: 16 bytes (128 bits), no tweak

* ipcrypt-nd: Key: 16 bytes (128 bits), Tweak: 8 bytes (64 bits)

* ipcrypt-ndx: Key: 32 bytes (256 bits, two AES-128 keys), Tweak: 16 bytes (128 bits)

9. Implementation Status

This section is to be removed before publishing as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist draft document reviewers in judging whether the specification is suitable for publication.

Please note that the listing of any individual implementation here does not imply endorsement. Furthermore, no effort has been spent to verify the information presented here that was supplied by contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features.

Multiple independent, interoperable implementations of the schemes described in this document have been developed:

* C implementation
* D implementation
* Dart implementation (pub.dev package)
* Elixir implementation (hex package)
* Go implementation
* Java implementation (maven package)
* JavaScript/TypeScript implementation (npm package)
* PHP implementation (Composer package)
* Python reference implementation
* Ruby implementation (rubygems package)
* Rust implementation (cargo package)
* Swift implementation
* Zig implementation

A comprehensive list of implementations and their test results can be found at: https://ipcrypt-std.github.io/implementations/

All implementations pass the common test vectors specified in this document, demonstrating interoperability across programming languages.

10. Licensing

This section is to be removed before publishing as an RFC.

Implementations of the ipcrypt methods are freely available under permissive open source licenses (MIT, BSD, or Apache 2.0) at the repository listed in the Implementation Status section.

There are no known patent claims on these methods.

11. References

11.1. Normative References

[FAST] Doh, Y., Ha, J., and J. Kim, "FAST: Format-Preserving Encryption via Shortened AES Tweakable Block Cipher", Cryptology ePrint Archive Report 2021/1171, 12 September 2021, <https://eprint.iacr.org/2021/1171>.

[FIPS-197] NIST, "Advanced Encryption Standard (AES)", FIPS PUB 197, 26 November 2001, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.

[IEEE-P1619] IEEE, "IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices", IEEE 1619-2007, 18 December 2007, <https://standards.ieee.org/ieee/1619/2041/>.

[NIST-SP-800-38G] NIST, "Recommendation for Block Cipher Modes of Operation: Methods for Format-Preserving Encryption", NIST SP 800-38G, March 2016, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38G.pdf>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/rfc/rfc4291>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/rfc/rfc7258>.

[RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <https://www.rfc-editor.org/rfc/rfc7624>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

11.2. Informative References

[BRW2005] Bellare, M., Rogaway, P., and D. Wagner, "Format-Preserving Encryption", CRYPTO 2005, 2005, <https://www.cs.ucdavis.edu/~rogaway/papers/subset.pdf>.

[DEOXYS-BC] Jean, J., Nikoli, I., and T. Peyrin, "Deoxys-BC: A Highly Secure Tweakable Block Cipher", Cryptology ePrint Archive Paper 2014/427, 2014, <https://eprint.iacr.org/2014/427>.

[IPCRYPT2] Denis, F., "ipcrypt2: IP address encryption/obfuscation tool", 2025, <https://github.com/jedisct1/ipcrypt2>.

[KIASU-BC] Jean, J., Nikoli, I., and T. Peyrin, "Tweaks and Keys for Block Ciphers: the TWEAKEY Framework", Cryptology ePrint Archive Paper 2014/831, 2014, <https://eprint.iacr.org/2014/831>.

[LRW2002] Liskov, M., Rivest, R., and D. Wagner, "Tweakable Block Ciphers", Fast Software Encryption 2002, 2002, <https://www.cs.berkeley.edu/~daw/papers/tweak-crypto02.pdf>.

[RSSAC040] ICANN RSSAC, "RSSAC040: Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis", 9 March 2021, <https://www.icann.org/en/system/files/files/rssac-040-09mar21-en.pdf>.

[SKINNY] Beierle, C., Biryukov, A., Perrin, L., Udovenko, A., Velichkov, V., and Q. Wang, "The SKINNY Family of Block Ciphers and its Low-Latency Variant MANTIS", CRYPTO 2016, 2016, <https://eprint.iacr.org/2016/660>.

[XTS-AES] Black, J., Dawson, E., Gueron, S., and P. Rogaway, "The XTS-AES Mode for Disk Encryption", IEEE 1619-2007, 2010.

Appendix A. Test Vectors

This appendix provides test vectors for all three variants of ipcrypt. Each test vector includes the key, input IP address, and encrypted output. For non-deterministic variants (ipcrypt-nd and ipcrypt-ndx), the tweak value is also included.

Implementations MUST verify their correctness against these test vectors before deployment.

A.1. ipcrypt-deterministic Test Vectors

```
# Test vector 1
Key:          0123456789abcdeffedcba9876543210
Input IP:     0.0.0.0
Encrypted IP: bde9:6789:d353:824c:d7c6:f58a:6bd2:26eb

# Test vector 2
Key:          1032547698badcfeefcdab8967452301
Input IP:     255.255.255.255
Encrypted IP: aed2:92f6:ea23:58c3:48fd:8b8:74e8:45d8

# Test vector 3
Key:          2b7e151628aed2a6abf7158809cf4f3c
Input IP:     192.0.2.1
Encrypted IP: 1dbd:c1b9:fff1:7586:7d0b:67b4:e76e:4777
```

A.2. ipcrypt-nd Test Vectors

```
# Test vector 1
Key:          0123456789abcdeffedcba9876543210
Input IP:     0.0.0.0
Tweak:        08e0c289bff23b7c
Output:       08e0c289bff23b7cb349aadfe3bcef56221c384c7c217b16

# Test vector 2
Key:          1032547698badcfeefcdab8967452301
Input IP:     192.0.2.1
Tweak:        21bd1834bc088cd2
Output:       21bd1834bc088cd2e5e1fe55f95876e639faae2594a0caad

# Test vector 3
Key:          2b7e151628aed2a6abf7158809cf4f3c
Input IP:     2001:db8::1
Tweak:        b4ecbe30b70898d7
Output:       b4ecbe30b70898d7553ac8974d1b4250eafc4b0aa1f80c96
```

A.3. ipcrypt-ndx Test Vectors

```
# Test vector 1
Key:          0123456789abcdeffedcba98765432101032547698badcfeefcdab8967452301
Input IP:     0.0.0.0
Tweak:        21bd1834bc088cd2b4ecbe30b70898d7
Output:       21bd1834bc088cd2b4ecbe30b70898d782db0d4125fdace61db35b8339f20ee5

# Test vector 2
Key:          1032547698badcfeefcdab89674523010123456789abcdeffedcba9876543210
Input IP:     192.0.2.1
Tweak:        08e0c289bff23b7cb4ecbe30b70898d7
Output:       08e0c289bff23b7cb4ecbe30b70898d7766a533392a69edf1ad0d3ce362ba98a

# Test vector 3
Key:          2b7e151628aed2a6abf7158809cf4f3c3c4fcf098815f7aba6d2ae2816157e2b
Input IP:     2001:db8::1
Tweak:        21bd1834bc088cd2b4ecbe30b70898d7
Output:       21bd1834bc088cd2b4ecbe30b70898d76089c7e05ae30c2d10ca149870a263e4
```

For non-deterministic variants (ipcrypt-nd and ipcrypt-ndx), the tweak values shown are examples. In practice, tweaks MUST be uniformly random for each encryption operation.

IANA Considerations

This document does not require any IANA actions.

Acknowledgments

The author gratefully acknowledges the contributions and insightful comments from members of the IETF and the broader cryptographic community that have helped shape this specification. Special thanks to Tobias Fiebig for his thorough review of this draft.

Author's Address

Frank Denis
Fastly Inc.
Email: fde@00f.net