HCTR2 모드에서의 국산 블록 암호 적용 및 분석

https://youtu.be/rQMCMh2qcQU

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

1. HCTR2

- 워크숍과 보고서에서 확인된 문제들을 해결하기 위해서 두 개의 방향으로 표준화를 진행
  - (SP 800-197)Wider Variant of AES – 고정된 **큰 블록 크기**로 생일 경계 문제를 해결
  - (SP 800-197A)Accordion mode – **가변 길이 입력**으로 유연성을 제공(패딩 문제).
    Tweak 값을 통한 키 재사용 문제 해결

- NIST에서는 세 가지 범용 아코디언을 개발할 예정
  - Acc128 – 기존의 128비트 블록 크기의 블록 암호를 지원
    - 생일 역설 문제로 인해서 단일 키로 연산되는 데이터를 2^48비트로 제한될 것으로 예상
  - Acc256 – 높은 데이터 한계를 지원하는 256비트 블록 크기의 블록 암호를 지원
    - 더 큰 블록 크기로 단일 키로 생일 역설 문제를 뛰어넘는 데이터 양을 지원
    - SP 800-197(Wider Variant of AES)에서 개발되고 있는 256비트 블록 크기의 블록 암호가 적용
  - BBBAcc – 기존의 128비트 블록 크기의 블록 암호에서 생일 문제를 해결한 모드
    - Acc128과 동일한 128비트 블록 크기를 사용하지만, Acc128에서는 생일 문제로 인해서 단일키로 사용되는 데이터를 제한할 예정
    - BBBAcc는 이러한 문제를 해결하여 단일 키로도 Acc256과 같은 높은 데이터 연산을 지원하는 것을 목표로 함.

- Acc128 would support an underlying block cipher with 128-bit blocks (i.e., AES). Due to the birthday bound, NIST expects to limit the total data processed under a single key to 2^{48} bits.
- Acc256 would support very high usage bounds for an underlying block cipher with 256-bit blocks.
- BBBAcc would support extended usage beyond the birthday bound with AES. NIST expects the analogous limit on the total data processed under a single key to be at least 2^{64} bits.

1. HCTR2

- 아코디언 모드와 Rijndael-256(Wider Vaariant of AES)에서 언급되는 생일 문제는 CTR 모드에서의 한계로 인한 문제
  - 현재 많은 시스템에서 GCM 모드를 활용하는데 GCM은 CTR이 활용됨

- CTR 모드에서 Nonce+Counter 값을 암호화하여 평문과 XOR 함.
  - Counter 값이 초과하여 0부터 다시 시작한다면 같은 키로 같은 Nonce+Counter 값을 사용하게 됨.
  - 예를 들어 A평문을 알고 있을 때,
    A+E(nonce+1)과 B+E(nonce+1)는 같은 키와 같은 암호화 값을 사용하여 XOR 하였기 때문에 두 결과 값을 XOR하면 A+B를 얻을 수 있고, A평문을 알기 때문에 B평문도 알 수 있게됨

- 서로 다른 두 개의 평문 블록이 같은 카운터 블록을 사용하게 되는 상황
  - 블록 암호의 블록 크기가 N일때,. 암호화된 블록 값이 $2^{N/2}$개를 넘으면 중복 발생 확률이 50%
  - 따라서, 단일 키로 암호화할 수 있는 데이터의 양을 블록 크기의 절반보다 낮은 수준으로 제한

1. HCTR2

- Accordion을 위해서 HCTR2 기술의 변형을 개발할 것을 제안
  - 가변 입력 길이 강력 의사 난수 순열(Variable-Input-Length-Strong Pseudorandom Permutation, VIL-SPRP)

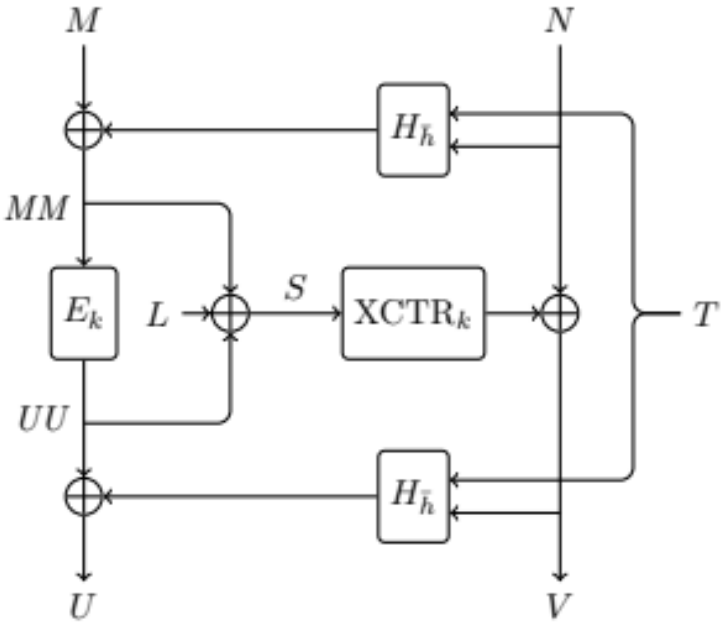


Figure 1: HCTR2

$$\text{XCTR}_k(S) = E_k(S \oplus \text{bin}(1)) \| E_k(S \oplus \text{bin}(2)) \| E_k(S \oplus \text{bin}(3)) \| \cdots$$

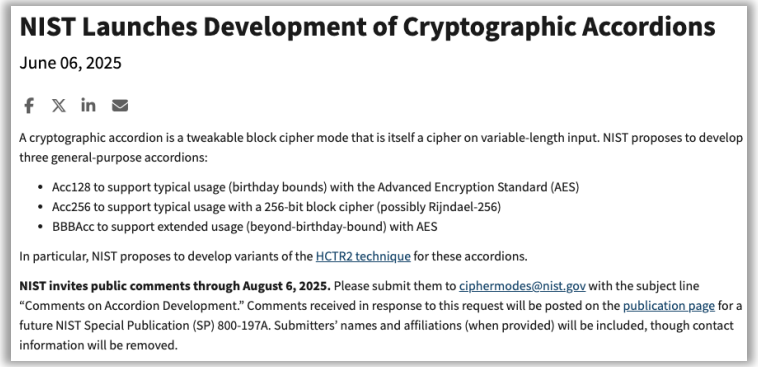
2.4 HCTR2

HCTR2 encryption, defined in Figure 2, takes a tweak and a plaintext, and returns a ciphertext of the same length as the plaintext. HCTR2 decryption (Figure 3) recovers the plaintext given the same tweak and the ciphertext, ie for $k \in \mathcal{K}$, $T \in \mathcal{T}$ and $P \in \mathcal{M}$, $\text{DECRYPT}(k, T, \text{ENCRYPT}(k, T, P)) = P$.

- HCTR2는 데이터를 고정 크기 M, 가변 크기 N으로 나누어 두 데이터가 암호화 과정에 영향을 주도록 설계됨
- N은 XCTR 모드를 통해 길이에 딱 맞는 키 스트림을 생성하여 암호화되므로, 원본 데이터와 길이가 항상 동일하게 유지
- M은 암호화 N에 의존하여, 최종 암호화된 U는 암호화된 V의 정보와 다시 섞여 데이터 전체의 무결성을 보장

1. HCTR2

- Accordion mode에서 목표로 하는 세 가지 개발 중에서 **Acc256**
  - Acc256은 256비트 블록을 가진 기본 블록 암호를 지원하는데, 이는 Wider Variant of AES와 연관됨.
  - 따라서, acc256은 Wider Variant of AES와 Accordion mode의 장점을 결합한 하이브리드 접근법으로 보여짐



NIST Launches Development of Cryptographic Accordions

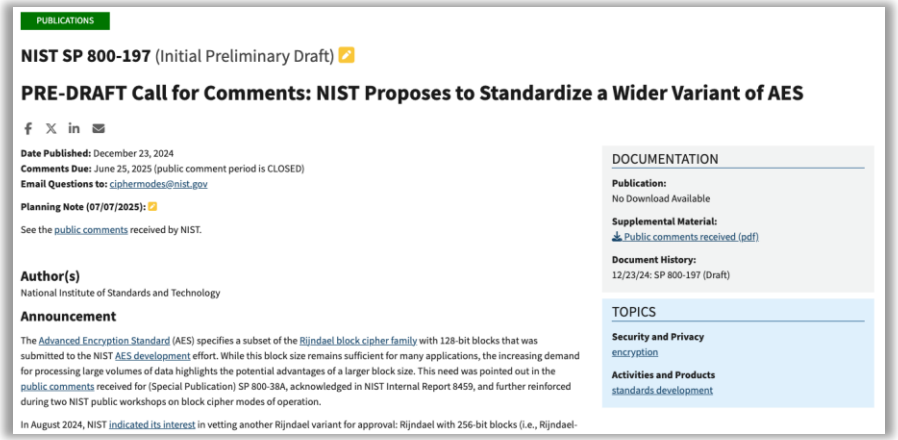
June 06, 2025

A cryptographic accordion is a tweakable block cipher mode that is itself a cipher on variable-length input. NIST proposes to develop three general-purpose accordions:

- Acc128 to support typical usage (birthday bounds) with the Advanced Encryption Standard (AES)
- Acc256 to support typical usage with a 256-bit block cipher (possibly Rijndael-256)
- BBBAcc to support extended usage (beyond-birthday-bound) with AES

In particular, NIST proposes to develop variants of the HCTR2 technique for these accordions.

NIST invites public comments through August 6, 2025. Please submit them to ciphermodes@nist.gov with the subject line "Comments on Accordion Development." Comments received in response to this request will be posted on the publication page for a future NIST Special Publication (SP) 800-197A. Submitters' names and affiliations (when provided) will be included, though contact information will be removed.



PUBLICATIONS

NIST SP 800-197 (Initial Preliminary Draft)

PRE-DRAFT Call for Comments: NIST Proposes to Standardize a Wider Variant of AES

Date Published: December 23, 2024
Comments Due: June 25, 2025 (public comment period is CLOSED)
Email Questions to: ciphermodes@nist.gov

Planning Note (07/07/2025):

See the public comments received by NIST.

Author(s)
National Institute of Standards and Technology

Announcement

The Advanced Encryption Standard (AES) specifies a subset of the Rijndael block cipher family with 128-bit blocks that was submitted to the NIST AES development effort. While this block size remains sufficient for many applications, the increasing demand for processing large volumes of data highlights the potential advantages of a larger block size. This need was pointed out in the public comments received for (Special Publication) SP 800-38A, acknowledged in NIST Internal Report 8459, and further reinforced during two NIST public workshops on block cipher modes of operation.

In August 2024, NIST indicated its interest in vetting another Rijndael variant for approval: Rijndael with 256-bit blocks (i.e., Rijndael-

DOCUMENTATION

Publication:
No Download Available

Supplemental Material:
Public comments received (pdf)

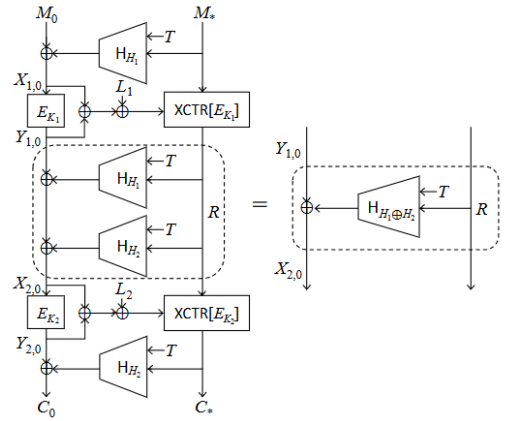
Document History:
12/23/24: SP 800-197 (Draft)

TOPICS

Security and Privacy
encryption

Activities and Products
standards development

- 위의 설명은 예시이며, 아직 의견을 수집해 표준화를 진행하고 있음.
- BBBAcc도 정확한 방법은 제공되지 않지만, 공개 Comment에서 Yusuke Naito가 CHCTR을 제안함
  - CHCTR은 HCTR2를 연접하여 두 번 연산하는 구조



2. HCTR2 코드 분석

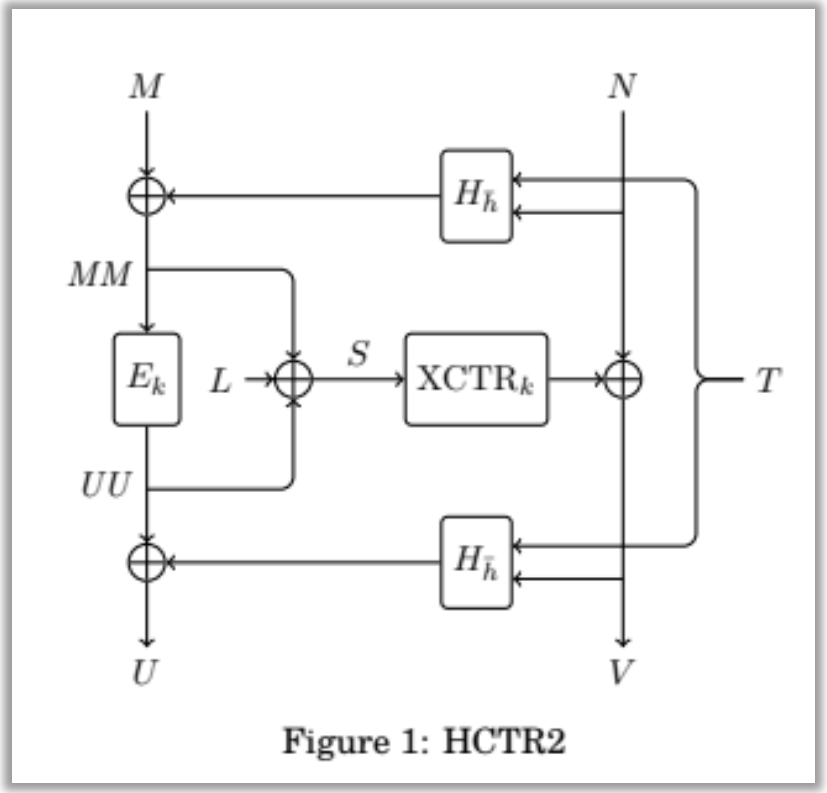


Figure 1: HCTR2

```python
def encrypt(self, pt, key, tweak):
    blocksize = self.lengths()['block']
    assert len(key) == self.lengths()['key']
    assert len(pt) >= blocksize
    hash_key = self._schedule(key, 0)
    l = self._schedule(key, 1)
    m = pt[0:blocksize]
    n = pt[blocksize:]
    mm = strxor(m, self._hash(hash_key, n, tweak))
    uu = self._block.encrypt(mm, key=key)
    s = strxor(strxor(mm, uu), l)
    v = self._xctr.encrypt(n, nonce=s, key=key)
    u = strxor(uu, self._hash(hash_key, v, tweak))
    return u + v
```

E_k와 XCTR_k의 Key를 다르게 사용하기 위해서

```python
def gen(self, l, nonce, key):
    assert len(key) == self.lengths()['key']
    assert len(nonce) == self.lengths()['nonce']
    res = b''
    count = 0
    while len(res) < l:
        count += 1
        countblock = count.to_bytes(len(nonce), byteorder='little')
        res += self._block.encrypt(strxor(nonce, countblock), key)
    return res[:l]

# encrypt and decrypt are the same
def encrypt(self, plaintext, nonce, key):
    return strxor(plaintext, self.gen(len(plaintext), nonce, key))
```

1. HCTR2

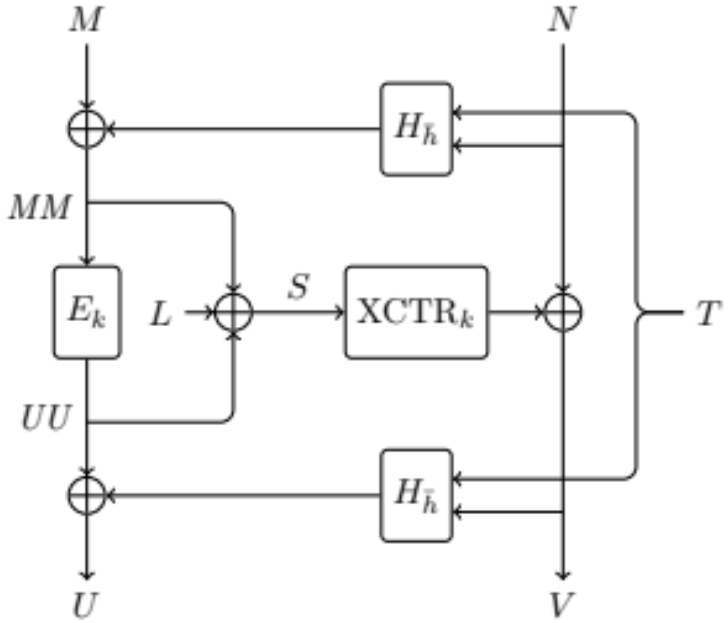


Figure 1: HCTR2

```python
def hash(self, key, message):
    blocksize = self.lengths()['key']
    assert len(message) % blocksize == 0
    hgen = self.gf.from_bytes(key, byteorder="little")
    hpoly = hgen * self.polyval_const
    hash_result = self.gf.from_int(0)
    for i in range(0, len(message), blocksize):
        hash_result += self.gf.from_bytes(
            message[i:i + blocksize], byteorder='little')
        hash_result *= hpoly
    return hash_result.to_bytes(byteorder='little')
```

병렬화 가능

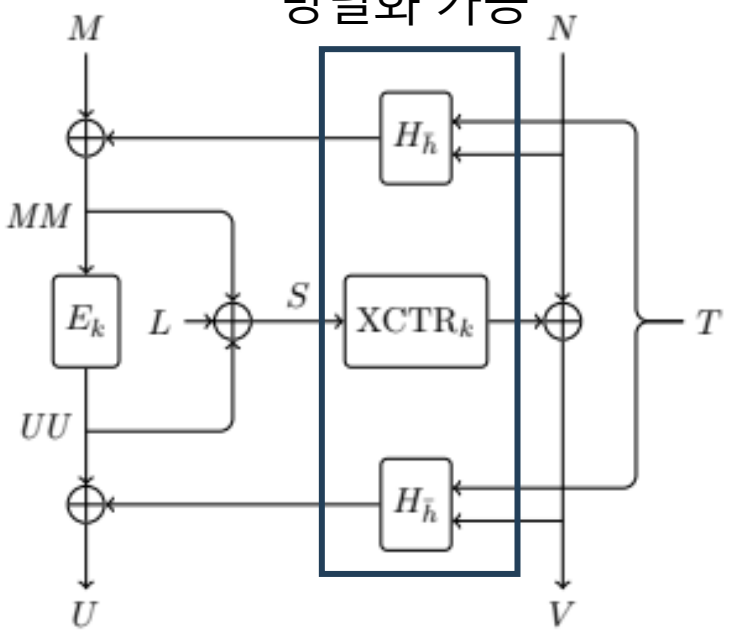


Figure 1: HCTR2

$$\mathrm{XCTR}_k(S) = E_k(S \oplus \mathrm{bin}(1)) \| E_k(S \oplus \mathrm{bin}(2)) \| E_k(S \oplus \mathrm{bin}(3)) \| \cdots$$

```
AES-128-HCTR2 encryption (generic)          97.522 cpb (29736 KB/s)
AES-128-HCTR2 decryption (generic)          97.673 cpb (29690 KB/s)
AES-128-HCTR2 encryption (simd)              1.028 cpb (2822118 KB/s)
AES-128-HCTR2 decryption (simd)              1.026 cpb (2826418 KB/s)

AES-192-HCTR2 encryption (generic)         111.191 cpb (26081 KB/s)
AES-192-HCTR2 decryption (generic)         111.183 cpb (26083 KB/s)
AES-192-HCTR2 encryption (simd)              1.078 cpb (2689587 KB/s)
AES-192-HCTR2 decryption (simd)              1.080 cpb (2684687 KB/s)

AES-256-HCTR2 encryption (generic)         117.611 cpb (24657 KB/s)
AES-256-HCTR2 decryption (generic)         117.655 cpb (24648 KB/s)
AES-256-HCTR2 encryption (simd)              1.128 cpb (2570715 KB/s)
AES-256-HCTR2 decryption (simd)              1.127 cpb (2572699 KB/s)
```

```
AES-128-XCTR encryption (generic)           15.757 cpb (184045 KB/s)
AES-128-XCTR decryption (generic)           15.761 cpb (183999 KB/s)
AES-128-XCTR encryption (simd)               0.286 cpb (10142302 KB/s)
AES-128-XCTR decryption (simd)               0.831 cpb (3488271 KB/s)

AES-192-XCTR encryption (generic)           18.923 cpb (153252 KB/s)
AES-192-XCTR decryption (generic)           18.974 cpb (152838 KB/s)
AES-192-XCTR encryption (simd)               0.333 cpb (8705519 KB/s)
AES-192-XCTR decryption (simd)               0.329 cpb (8820525 KB/s)

AES-256-XCTR encryption (generic)           22.079 cpb (131348 KB/s)
AES-256-XCTR decryption (generic)           22.139 cpb (130988 KB/s)
AES-256-XCTR encryption (simd)               0.375 cpb (7724673 KB/s)
AES-256-XCTR decryption (simd)               0.380 cpb (7629300 KB/s)

AES-128-XTS encryption (simd)                0.409 cpb (7084354 KB/s)
AES-128-XTS decryption (simd)                0.413 cpb (7030158 KB/s)

AES-192-XTS encryption (simd)                0.493 cpb (5880987 KB/s)
AES-192-XTS decryption (simd)                0.493 cpb (5888303 KB/s)

AES-256-XTS encryption (simd)                0.580 cpb (5001520 KB/s)
AES-256-XTS decryption (simd)                0.581 cpb (4992289 KB/s)
```

```
AIRA-128-HCTR2 encryption (generic)         111.395 cpb (26033 KB/s)

AIRA-192-HCTR2 encryption (generic)         120.053 cpb (24156 KB/s)

AIRA-256-HCTR2 encryption (generic)         116.842 cpb (24819 KB/s)

Polyval hashing (generic)                    45.066 cpb (64349 KB/s)
Polyval hashing (clmul)                       0.363 cpb (7996685 KB/s)

AIRA-128-XCTR encryption (generic)          25.363 cpb (114341 KB/s)

AIRA-192-XCTR encryption (generic)          29.686 cpb (97689 KB/s)

AIRA-256-XCTR encryption (generic)          33.973 cpb (85361 KB/s)
```

감 사 합 니 다