ChatGPT

암호학 기본 개념 정리 (IoT/엣지 초급용) + 경량암호 & PRESENT 한눈에 보기

목차

1. 개요

암호화는 **평문(원본 데이터)**을 **암호문(읽을 수 없는 형태)**으로 바꾸어 **기밀성**을 보장하는 기술이다. 하지만 실제 시스템(특히 IoT/엣지)에서는 기밀성 하나만으로 끝나지 않는다. 데이터가 **변조되지 않았는지(무결성)**, 상대가 **진짜인지(인증)**, 그리고 나중에 "난 안 했다"라고 **발뺌하지 못하게(부인방지)** 하는 것까지 함께 설계해야 한다.
현대 보안 설계는 보통 다음 방식으로 **여러 기술을 조합**한다.

- **대칭키 암호**(빠르고 저전력)로 본문 데이터 암호화
- **운용 모드**(CTR/GCM 등)로 스트림/파일 전체 처리
- **AEAD**(예: AES-GCM, ASCON-AEAD)로 **기밀성+무결성** 동시 확보
- **공개키 암호**(RSA/ECC)로 세션키 교환, 전자서명(인증·부인방지)
- **해시/HMAC**으로 무결성 확인, **KDF**(Argon2 등)로 비밀번호 안전 저장

엣지/IoT 관점: 연산·메모리·전력 제약이 크므로 **경량암호**(PRESENT, ASCON, SIMON/SIMECK 등)와 **키·IV 관리**를 특히 신경 써야 한다.

---

2. 보안의 4대 속성 (CIAA)

2.1 기밀성 (Confidentiality)

정의: 인가되지 않은 자가 데이터를 **볼 수 없게** 하는 속성.
왜 중요? 이미지·개인정보·기계 로그 등이 유출되면 직접 피해는 물론 신뢰도도 하락한다.
어떻게? 신뢰할 수 있는 **대칭키 암호**(AES, PRESENT 등)로 암호화하고, 키는 **eFUSE/OTP(일회성 프로그래밍 메모리)**나 **보안칩(SE/TPM)**, **PUF** 등 안전한 저장소에 보관한다.
예시: 암호화하지 않고 SD카드에 이미지를 저장하면, 카드를 빼내어 내용을 바로 볼 수 있다 → 기밀성 실패.
체크리스트 - [ ] 강력한 알고리즘/충분한 키 길이 선택 (임시 PoC라도 약식 알고리즘은 사용 금지)
- [ ] **키 분리 보관**(코드/펌웨어와 분리) 및 **키 교체 절차** 문서화
- [ ] **IV/Nonce 재사용 금지** (CTR/GCM, ASCON 등 스트림 모드·AEAD에서 특히 중요)

2.2 무결성 (Integrity)

정의: 데이터가 전송/저장 중 **변조되지 않았음**을 보장하는 속성.
왜 중요? 암호화만 하고 무결성 검증이 없으면, 공격자가 암호문 일부를 바꿔도 **의미 있는 평문 변조**를 일으킬 수 있다 (운용 모드에 따라 다르지만 충분히 위험).
어떻게? AEAD(AES-GCM, ASCON-AEAD 등)를 사용해 암호화와 동시에 태그 검증을 하거나, 최소한 **HMAC**을 추가하여 변조 여부를 확인한다.
예시: SD카드의 암호문 일부가 손상되었는데 태그 검증이 없다면, 복호화 결과가 달라져도 눈치채지 못한다.
체크리스트 - [ ] 태그 검증 실패 시 **데이터 폐기** (경고만 띄우고 사용하면 안 됨)
- [ ] 저장 매체/전송 경로의 **CRC**는 무결성 보조일 뿐, **암호학적 무결성**이 아님
- [ ] 태그·메타데이터(AAD, 버전, nonce 등)를 **함께 저장**하고 재현 가능하게 관리

2.3 인증 (Authentication)

정의: 상대방 또는 데이터의 **출처를 확인**하는 속성.
왜 중요? 가짜 기기나 공격자가 시스템에 접속해 데이터를 주입할 수 있다.
어떻게? 전자서명(ECDSA/EdDSA)이나 **챌린지-응답**(미리 공유한 대칭키 사용)으로 장치·서버 상호 인증을 수행한다. 큰 시스템에서는 X.509 인증서를 써서 규모가 커져도 신뢰 체인을 유지한다.
예시: 공격자가 가짜 카메라 모듈을 연결해 조작된 영상을 전송 → 인증 기능이 없으면 시스템이 그 영상을 진짜로 받아들인다.
체크리스트 - [ ] 최초 등록(프로비저닝) 시 **장치별 키/인증서** 발급
- [ ] 인증 실패, 인증서 **만료/폐지**(OCSP/CRL) 시 동작 정의
- [ ] **인증(Authentication)**과 **권한(Authorization)**은 구분할 것 — 인증은 신원 확인, 권한은 역할/정책에 따른 접근 제어

2.4 부인방지 (Non-repudiation)

정의: 행위자가 나중에 “내가 안 했다”고 **발뺌하지 못하게** 하는 속성.
왜 중요? 데이터 생성·전송에 대한 **책임 추적성**과 법적 증거 확보와 직결된다.
어떻게? 개인키로 **서명**하고 공개키로 **검증**한다 (서명을 “복호화”한다고 표현하지는 않음). 또한 로그에 **서명/태그/타임스탬프**를 함께 보관해 추후 위변조 여부를 확인 가능하게 한다.
체크리스트 - [ ] 서명에 사용되는 **개인키 보호** (암호화하여 저장하거나 HSM/TPM 등 하드웨어에 저장)
- [ ] **시간 동기화**를 통해 신뢰할 수 있는 타임스탬프 확보, 로그 위변조 방지

---

3. 암호 프리미티브

3.1 대칭키 암호

개념: **하나의 동일한 키**로 암호화와 복호화를 수행한다. 내부 구조는 SPN(AES/PRESENT)이나 Feistel(SIMON/SIMECK) 등 다양하지만 공통 목표는 **혼돈(Confusion)**과 **확산(Diffusion)**을 충분히 주는 것이다.
장점: 빠르고 저전력이며 구현이 단순하다. **대용량 데이터** 보호에 적합하다.
단점: **키 배포** 문제가 있다 (상대방에게 키를 안전하게 전달하는 것이 숙제). 그리고 무결성은 별도로 처리해야 한다 (MAC 또는 AEAD 추가).
운용 모드: 블록암호는 ECB/CBC/CTR/GCM 등의 **모드**로 긴 데이터(파일/스트림)를 처리한다. 실제로는 **CTR/GCM**이 주력(패딩 이슈가 적고 병렬화에 유리하기 때문).
IoT/엣지 팁: - **PRESENT**: 64비트 블록, 80/128비트 키, 31라운드. 4비트 S-box × 16개 + 비트 전치로 구성. **소면적/저전력**에 유리하며, RTL 설계 시 모듈 쪼개기가 쉽다.
- **AES**: 128비트 블록, 128/192/256비트 키. HW 가속(ASIC 명령어 등)이 있다면 **AES-GCM**이 표준 중 가장 많이 사용된다.
- **LEA/CHAM/HIGHT**: 임베디드 친화적인 국내외 경량 블록암호들. 구현 자료가 풍부하다.

실수 방지 - [ ] **약한 알고리즘**(DES 등)이나 **짧은 키** 금지 (테스트 용도라도 습관 들이지 말 것)
- [ ] CTR/GCM 등 스트림성 모드에서 **IV/Nonce 재사용 절대 금지** — 가장 흔하면서 치명적인 실수
- [ ] 키·IV 생성은 반드시 **안전한 RNG**(TRNG/DRBG) 사용

3.2 비대칭키(공개키) 암호

개념: **공개키/개인키** 한 쌍을 사용한다. 누구나 공개키로 암호화/서명 검증할 수 있지만, 오직 개인키 소유자만 복호화/서명 생성이 가능하다.
주요 역할:
- **키 교환**: ECDH(또는 RSA로 암호화한 키 전송) 등을 통해 세션키 합의
- **전자서명**: ECDSA/EdDSA 또는 RSA-PSS로 서명 생성 → 공개키로 검증 (**부인방지**)
장점: 키 배포가 상대적으로 쉽다 (공개키를 마음껏 배포 가능), 인증·서명 기능 제공
단점: 대칭키 대비 **속도가 느리고 연산이 무겁다** → 본문 데이터 암호화에는 부적합
현실적 사용: TLS처럼 **초기 핸드셰이크(세션 수립)**에만 비대칭키를 쓰고, 이후 **세션키**로 **대칭키 암호(AEAD)**를 사용한다.

실수 방지 - [ ] 개인키는 **절대 평문으로 저장 금지** (암호화하여 보관, 가능하면 HSM/SE/TPM 등 사용)
- [ ] RSA는 **적절한 패딩** 사용 (예: PSS, OAEP), ECC는 권고 **곡선과 키 길이** 준수
- [ ] 인증서(X.509) **유효기간 관리**와 **폐기(CRL/OCSP)**, 체인 검증 구현을 놓치지 말 것

3.3 해시 함수

개념: **임의 길이 입력 → 고정 길이 출력**으로 매핑하는 **단방향 함수**이다. 해시는 한 방향으로만 계산 가능하고, **복호화라는 개념은 없다**. 주로 **무결성 검증**이나 **데이터 지문 생성**에 사용된다.
핵심 성질: - **역상 저항성 (Preimage Resistance)**: 해시값만 보고 원본 입력을 찾기 어려워야 한다.
- **제2역상 저항성 (Second-preimage)**: 어떤 입력에 대한 해시값과 똑같은 해시값을 내는 **다른 입력**을 찾기 어려워야 한다.
- **충돌 저항성 (Collision Resistance)**: 해시값이 같은 **서로 다른 두 입력**을 찾기 매우 어려워야 한다.
주의 (오해 정정): 해시는 “암호화”가 아니므로 **복호화**가 불가능하다. 즉, 한 번 해시된 데이터는 수학적 성질과 출력 길이에 의존해 보호되며, 원본을 복원할 수 없다.

공격 모델과 대응 - **레인보우 테이블/사전 공격**: 미리 계산된 해시 테이블로 원본을 추정하는 공격.
→ 입력에 **솔트(Salt)**를 추가하여 테이블을 무력화한다.
- **충돌 공격**: SHA-1/MD5 등은 서로 다른 입력인데 해시가 같게 나오는 충돌을 인위적으로 만들 수 있다.
→ **SHA-256/512** 이상의 알고리즘을 사용하고, MD5/SHA-1은 지양한다.
- **길이 확장 공격**: 해시 알고리즘 자체로는 메시지에 일부 데이터를 이어붙였을 때 취약할 수 있다.
→ **HMAC**을 사용하면 이러한 공격을 방어할 수 있다.

패스워드 보관 (별도 권고)
- 단순 해시+솔트보다 **전용 KDF**(**Argon2**, scrypt, bcrypt, PBKDF2 등)을 사용할 것.
- KDF의 **워크 팩터**(메모리 사용량, 반복 횟수)를 높여 공격 비용을 늘릴 것.

3.4 메시지 인증 (HMAC)

개념: **공유된 비밀 키**와 해시 함수를 결합하여 만든 **메시지 인증 코드**. 무결성과 송신자 인증(누가 보냈는지)을 동시에 검증할 수 있다.
장점: 구현이 간단하고 빠르며, HMAC 구조상 **길이 확장 공격에 안전**하다.
제한: 송신자와 수신자가 **같은 키**를 사용하므로 **부인방지 기능은 없다** (제3자가 "누가 이걸 만들었는지" 증명하기는 어려움).
사용 예: HMAC-SHA256(가장 흔함), IPsec 인증, 일부 TLS 모드 등.
실수 방지 - [ ] 키 길이는 충분히 길게 (128비트 이상 권장)
- [ ] 메시지에 **메타데이터(AAD)**까지 포함하여 해당 **문맥까지 인증** (예: 프로토콜 버전, 메시지 종류 등)
- [ ] 수신 측은 태그 검증 시 **상수 시간 비교**를 할 것 (타이밍 차이를 통한 키 유출 위험 방지)

3.5 디지털 서명 & PKI (X.509)

디지털 서명: 송신자가 **개인키로 메시지 해시값에 서명**하고, 누구나 **공개키로 검증**할 수 있는 기술. 이를 통해 **부인방지**와 **출처 인증**을 제공한다.
대표 알고리즘: ECDSA, EdDSA, RSA-PSS 등
PKI (공개키 기반구조): 공개키를 **신뢰성 있게 배포**하기 위한 인프라. **루트 CA → 중간 CA → 서버/클라이언트 인증서** 형태의 신뢰 체인을 만들고, 인증서 **만료/폐지**(OCSP/CRL)를 관리한다.

실무 예시 (예: IoT 장치 등록): 1. 장치가 자체적으로 키쌍 생성 (가능하면 안전한 HW 내부에서)
2. CSR(인증서 서명 요청) 제출 → CA가 **장치 인증서** 발급
3. 서비스 접속 시 **서버-장치 상호 인증** 수행 (서버는 장치 인증서 검증, 장치는 서버 인증서 검증)
4. 주기적인 **인증서 갱신/폐지**로 신뢰 관계 유지

실수 방지 - [ ] 개인키는 **외부로 유출하지 않기** (가능하면 Secure Element/TPM/TEE 내부에서 키 생성 및 서명)
- [ ] 인증서 **체인 검증**(중간 CA, 루트 CA)과 신뢰 저장소 관리, 만료·폐지 처리 구현
- [ ] "서명을 복호화한다"와 같은 표현을 사용하지 말고, **서명은 검증**으로 설명할 것

---

4. 블록암호 운용 모드 (ECB / CBC / CTR / GCM)

전제: 블록암호(Block Cipher)는 고정 크기(예: 64비트 또는 128비트) 블록을 입력받아 같은 크기의 블록으로 변환하는 암호 알고리즘이다.
파일이나 스트림같이 **긴 데이터를 암호화**하려면 **운용 모드(Mode of Operation)**가 필요하다.

용어 짚고 넘어가기: - **IV (Initialization Vector)**: 첫 블록을 랜덤하게 섞기 위한 초기값 (비밀값은 아니지만 예측 어려워야 함).
- **Nonce (Number used once)**: “한 번만 쓰는 숫자”. IV와 개념은 비슷하지만 **절대 재사용하지 않는 유일값**이라는 점을 더 강조한 용어.
- **패딩 (Padding)**: 블록 크기의 배수가 되도록 마지막 부분을 채우는 작업.
- **태그 (Tag)**: 무결성 검증을 위한 짧은 인증 값 (예: 16바이트).

4.1 ECB (Electronic Codebook)

개념: 각 블록을 **독립적으로** 암/복호화한다. (암호문 = E(K, 평문 블록))
장점: 구현이 매우 단순하며 패딩만 처리하면 된다.
치명적 단점: **같은 평문 블록은 같은 암호문 블록**이 나오므로, 데이터 패턴이 그대로 노출된다.
결론: 실제 사용 **금지 수준** (학습이나 디버깅 용도 외에는 쓰지 말 것).

4.2 CBC (Cipher Block Chaining)

개념: `C_i = E(K, P_i ⊕ C_{i-1})` 형태로 이전 암호문과 **연쇄(Chain)** 한다. 첫 블록은 `C_0 = IV`로 처리.
장점: ECB의 패턴 노출 문제를 해결한다. 구현도 비교적 간단하다.
단점/주의:
- **직렬 처리**만 가능 (이전 결과가 다음 입력에 필요) → 하드웨어에서 병렬 처리 어려움.
- **패딩** 필요.
- **무결성 미제공** (암호화만 함). 별도 **HMAC** 또는 **AEAD** 추가 필요.
사용처: 과거 시스템과의 호환이 필요한 경우 등에 제한적으로 사용.

4.3 CTR (Counter)

개념: 블록암호를 **키스트림 생성기**로 활용한다. 평문과 XOR할 키스트림을 `KS_i = E(K, Nonce || counter_i)`로 생성하고, 암호문 `C_i = P_i ⊕ KS_i` 로 만든다.
장점:
- **병렬화** 용이 (각 블록별 카운터만 다르게 넣고 암호화하면 되므로).
- **패딩 불필요** (평문 길이대로 처리).
단점/주의:
- **Malleable(조작 가능)**: 무결성을 제공하지 않아 암호문 일부를 변경하면 평문 일부도 임의로 변경 가능 → **HMAC 또는 AEAD**로 보완해야 한다.
- **(동일한 키, 동일한 nonce) 재사용 금지**: 한 번이라도 동일한 키와 nonce로 CTR을 시작하면 키스트림이 반복되어 두 암호문을 XOR하면 **평문 XOR**가 노출된다.

HW 팁: IoT/FPGA에서 **구현이 간단하고 빠르다**. PRESENT 같은 경량 블록암호에도 잘 맞는다.

카운터 증가 규칙 (중요): - 카운터는 일반적으로 **우측(LSB)부터 +1** (빅엔디언 카운터) 증가시킨다.
- Nonce와 Counter를 합쳐 **고정 길이**(예: 96비트 Nonce + 32비트 counter = 128비트 입력)를 만들고, **오버플로** 발생 시 **세션 종료** 또는 **상위 필드 증가** 등으로 중복을 피한다.
- 문서에 **엔디언 방식, 증가 규칙, 오버플로 처리 방법**을 명확히 적어둬야 동일한 구현을 재현할 수 있다.

4.4 GCM (Galois/Counter Mode) — AEAD

개념: CTR 모드로 암호화 + **GHASH**(갈루아 필드 기반 다항식 해시)로 태그 생성 → **기밀성 + 무결성** 동시 제공.
장점: CTR 장점을 이어받아 병렬화가 우수하며, 표준으로 널리 쓰여 라이브러리/레퍼런스가 풍부하다.
주의사항(매우 중요):
- **96비트 Nonce**를 권장 (GCM 표준에서는 이때 추가 해시 연산 생략 가능 등 최적화).
- **(동일한 키, 동일한 Nonce) 재사용 금지** — 한 번 재사용하면 **기밀성과 무결성 모두 붕괴**된다.
- 태그 검증 실패 시 **데이터 폐기** (경고만 하고 계속 처리하면 안 됨).
- **태그 길이**: 128비트(16바이트)가 기본 권장. 구현 단순화나 패킷 크기 절감을 위해 96비트, 112비트 등으로 줄일 수 있지만, 보안 여유도 줄어드므로 제품에서는 가급적 128비트를 유지하는 것이 좋다.

4.5 실수 방지 체크리스트 (모드 공통)

- [ ] ECB **사용 금지**
- [ ] CTR/GCM 사용 시 **Nonce/IV 유일성** 보장 (예: 부팅 횟수 + 카운터 조합, 임의 난수 추가 등 명확한 정책)
- [ ] 태그가 있는 모드(AEAD)는 **반드시 태그 검증** → 실패 시 데이터 폐기
- [ ] 암호문 저장 시 **메타데이터**(AAD/버전/Nonce/길이 등)를 함께 기록하여, 복호 시 동일한 조건으로 처리

---

5. 스트림 암호 vs 블록 암호

> **스트림 암호 (Stream Cipher)**: 한 비트 또는 한 바이트씩 흐르는 데이터에 **키스트림**을 XOR하여 암/복호화하는 방식.
> **블록 암호 (Block Cipher)**: 고정 길이 블록 단위로 변환하는 암호. (다만 CTR 등 모드를 쓰면 스트림처럼 동작시킬 수 있다.)

용어 짚고 넘어가기: - **키스트림 (Keystream)**: 난수처럼 보이는 비트열로, 평문과 XOR하여 암호화를 수행하는 데 사용.
- **생일한계 (Birthday Bound)**: 블록 길이가 n비트일 때, 약 **2^(n/2)** 블록 수준에서 **충돌 가능성**이 통계적으로 눈에 띄게 커지는 경계.

5.1 스트림 암호

장점:
- 암복호 대기 지연이 낮고, 패딩이 필요 없다.
- 비트/바이트 단위로 순차 처리하기 좋아 실시간 스트리밍에 적합.
주의:
- **같은 (키, nonce) 재사용 시 치명적**: 두 암호문의 XOR를 취하면 평문 XOR가 드러나서 원문 정보가 노출된다.
- 무결성을 제공하지 않는다 → **HMAC 또는 AEAD**로 보완 필요.
예시: **ChaCha20** (널리 쓰이는 스트림 암호), 그리고 사실상 **CTR 모드의 블록암호**도 스트림 암호처럼 동작한다.

5.2 블록 암호

장점:
- 수십 년간의 분석과 표준화로 신뢰성이 검증된 알고리즘이 많다.
- 하드웨어 구현이 비교적 쉽고(S-box 및 배선 최적화), **파이프라인** 설계로 고속화가 가능하다.

주의:
- **블록 크기의 생일한계**: 64비트 블록(PRESENT 등)을 매우 대용량으로 사용하면 2^(64/2) 수준에서 블록 충돌 위험이 생긴다 → **Nonce/카운터 관리**를 더욱 엄격히 해야 한다.
- 운용 모드를 잘못 선택하거나 (예: ECB) IV/Nonce를 잘못 취급하면 보안이 쉽게 붕괴한다.

생일한계 예시: 64비트 블록 암호는 약 **2^(64/2) = 2^32 ≈ 43억 개 블록**(약 34GB, 블록당 8바이트 가정) 처리 규모에서 충돌 확률이 무시 못 할 수준으로 올라간다. 그래서 장시간 스트리밍하거나 대용량 저장을 할 때는 **Nonce/카운터의 유일성**을 더욱 철저히 관리하거나, 가능하면 128비트 블록 암호(AES 등)나 AEAD 모드를 고려한다.

5.3 선택 가이드 (엣지/FPGA)

- **기능 검증(암복호만 우선)**: **CTR 모드** 추천 (구현 간단·패딩 필요 없음·병렬화 쉬움)
- **제품 적용(무결성까지)**: **AEAD 계열** (예: ASCON-AEAD, AES-GCM 등) 추천
- **대용량/장시간 스트림**: 64비트 블록 암호(PRESENT 등) 사용 시 **Nonce/카운터 충돌 방지**에 최우선 주의. 가능하면 128비트 블록 암호(AES 등)나 AEAD 모드 고려.

---

6. 구조: Feistel vs SPN (Substitution–Permutation Network)

공통 목표:
- **혼돈(Confusion)**: 키와 출력의 관계를 **복잡하게 꼬이도록** 만들어 추측이 어렵게 함 (S-box 같은 비선형 연산 활용)
- **확산(Diffusion)**: 입력의 한 비트 변화가 출력 전체에 영향을 주도록 **퍼뜨림** (비트 전치, 행렬 곱 등의 선형 변환 활용)

용어 짚고 넘어가기: - **S-box (Substitution box)**: 작은 룩업 테이블 형태의 **비선형 치환**. (예: 4비트 입력 → 4비트 출력)
- **Permutation (전치/퍼뮤테이션)**: 비트나 바이트 위치를 **고정된 규칙으로 섞는** 연산(선형 확산층 역할).

6.1 Feistel 구조

형태: 입력을 좌/우 (L/R) 두 부분으로 나눈 뒤, **한쪽만** 함수 F로 처리하고 XOR하여 교환하는 방식을 반복.
수식:

```
L_{i+1} = R_i
R_{i+1} = L_i ⊕ F(R_i, K_i)
```

복호화: 같은 F 함수를 쓰면서 **키 순서만 반대로** 적용하면 원복된다. → 별도의 복호화 알고리즘이 필요 없다.
대표 예: **DES**, **SIMON**, **SIMECK** 등 (SIMON/SIMECK은 ARX: 덧셈/회전/XOR으로 구성).
HW 관점: F 함수가 얼마나 단순하고 빠른지가 관건. **복호화가 용이**한 점이 구현상 장점이다.

6.2 SPN 구조

형태: **S-box (비선형 치환)** 단계와 **P-레이어 (확산 전치)** 단계를 여러 라운드 반복한다.
복호화 시에는 **역 S-box**와 **역 P-레이어**를 적용해야 한다 (Feistel과 달리 복호를 따로 설계해야 함).

대표 예: **AES**, **PRESENT**, **PIPO** 등.
HW 관점:
- 여러 S-box를 **병렬로 배치**하여 **처리량을 높이기** 쉽다.
- 반대로 S-box를 하나만 두고 **직렬로 재사용**하여 **초소형 구현**도 가능하다.
- P-레이어는 고정 배선 연결이므로 FPGA에서 배선 최적화로 **전력↓ 지연↓** 효과를 기대할 수 있다.

6.3 비교 요약

항목	Feistel	SPN
복호화 구현	**쉬움** (키 역순 적용)	**별도 설계 필요** (역 S-box/역 확산)
HW 병렬화	F 함수 설계에 좌우됨	**S-box 병렬화 용이**
알고리즘 예시	SIMON, SIMECK, DES	AES, PRESENT, PIPO
설계 포인트	F 함수의 품질	S-box 비선형성 + 확산 강도

---

7. AEAD (Authenticated Encryption with Associated Data)

> **AEAD**는 **기밀성(암호화)**과 **무결성/인증(태그)**을 **동시에** 제공하는 암호 운용 방식이다.

용어 짚고 넘어가기: - **AAD (Associated Data)**: 암호화하지는 않지만 **태그 계산에 포함**하는 메타데이터. 예를 들어 프로토콜 헤더, 버전, 장치 ID, 파일명 등이 될 수 있다. (암호문에는 나오지 않지만 수신측이 동일하게 알고 있어야 태그 검증에 포함 가능)
- **태그 (Tag)**: 수신자가 데이터의 무결성을 검증하기 위해 함께 전달되는 **인증 값**. (태그 검증 실패 시 해당 데이터는 폐기해야 함)

7.1 왜 AEAD인가?

- 암호화만 적용하면 **조작(변조)**을 막지 못한다. 공격자가 암호문을 바꿔치기 하거나 일부를 변경해도 알아채지 못하면 큰 문제가 된다.
- 별도로 HMAC 등을 붙여서 무결성을 제공할 수도 있지만, AEAD는 암호화와 인증이 일체화되어 **설계 및 구현 실수 가능성을 줄인다**.

7.2 대표 AEAD 알고리즘

- **AES-GCM**: CTR 모드 + 갈루아 필드 해시(GHASH) 조합으로 태그를 생성. **HW 가속**이 있는 환경에서는 매우 빠르고 이미 표준으로 자리잡았다.
- **ASCON-AEAD**: NIST 지정 경량 암호 표준. **순열 기반 스펀지 구조**를 사용한 경량 AEAD로, 코드/메모리 footprint가 작아 **저사양 장치**에 유리하다.

7.3 사용 규칙 (중요!)

- **(동일한 키, 동일한 Nonce) 재사용 금지**: 같은 키로 동일 Nonce를 두 번 사용하면 **기밀성과 무결성 모두 붕괴**된다. 절대 한 번 쓴 Nonce를 같은 키로 다시 쓰지 말 것.
- **Nonce 길이/형식**: AES-GCM은 **96비트 Nonce**를 권장 (안전하고 최적화에 유리). Nonce는 보통 **카운터 + 디바이스ID + 랜덤값** 등을 조합해 전세계에서 유일하게 구성한다.
- **태그 검증 실패 처리**: **즉시 데이터 폐기**. 부분적으로라도 사용하면 안 된다 (위변조된 데이터 사용 방지).
- **AAD 기록**: 버전/헤더/메타데이터 등을 AAD로 포함했다면, 해당 값을 **별도로 보관**하거나 프로토콜상 명시해서 수신 측이 동일하게 검증할 수 있게 해야 한다.

7.4 저장 포맷 예시 (이미지 암호화 → SD 카드)

```
[Header/AAD]
  version:    u16    (프로토콜 버전)
  algo:       enum   (예: PRESENT-CTR, AES-GCM, ASCON-128a 등)
  nonce:      12 or 16 bytes (Nonce 길이 정책에 따름)
  frame_no:   u32    (프레임 번호 또는 타임스탬프)
  length:     u32    (ciphertext 길이)
[Ciphertext ...]
[Tag]         16 bytes (AES-GCM이나 ASCON-AEAD 기준 태그)
```

- **복호 과정**: 저장된 키/Nonce/AAD 정보를 사용해 태그를 먼저 검증 → 통과하면 암호문을 복호화.
- **이점**: 파일이 손상되거나 위변조되었으면 **복호화 전에** 감지하여 오류를 처리할 수 있다.

7.5 프로젝트 적용 가이드

- **기능 검증 단계**: 우선 구현이 쉬운 **CTR 모드**로 시작 (패딩 필요 없고, 암복호 경로 동일). 이때 **Nonce 재사용 금지** 규칙을 문서화.
- **제품화 단계**: **AEAD 모드**로 전환 (예: **ASCON-AEAD** 또는 **AES-GCM**). 동시에,
- **Nonce 생성기** 설계 (장치 고유값 + 카운터 + 난수 조합 등),
- **태그 실패 처리 경로** 마련 (오류 코드, 예외 로그 등),
- **메타데이터(AAD) 포맷**을 확정하여 문서화.
- **HW 최적화 고려**: 파이프라인 vs 직렬화 구조 선택, RNG(TRNG/DRBG) 품질 확보, **부채널 공격 완화**(연산 시간/메모리 접근 패턴 통일 등 1차 대비).

7.6 빠른 체크리스트

- [ ] **AEAD**(권장) 또는 **CTR+HMAC** 중 하나를 택해 **무결성** 보장
- [ ] **Nonce 정책**: 재부팅 시 카운터 초기화 문제, 장치별 고유값, 난수 사용 등 **유일성** 확립 및 로그 기록
- [ ] **태그 실패 시 폐기**: 에러 코드 및 로그 처리 포함
- [ ] **AAD 설계**: 버전, 알고리즘, Nonce, 데이터 길이, 파일명 등 필요한 정보를 **고정 형식**으로 포함
- [ ] 테스트: **공개 테스트 벡터**로 알고리즘 검증 + **파일 왕복 (encrypt→decrypt)** + **에러 주입 테스트**(태그 변조, 잘못된 Nonce 등) 실시

---

8. 경량암호 (Lightweight Crypto) 개요

> **경량암호**는 **자원 제약(면적/전력/메모리/지연)**이 큰 IoT/엣지/임베디드 기기를 위해 설계된 암호 알고리즘을 말한다.
> 목표는 "**충분한 안전성**을 유지하면서도 **작게, 적은 전력으로, 빠르게** 동작"하는 것이다.

용어 짚고 넘어가기: - **면적(Area)**: ASIC 게이트 등가 수 또는 FPGA의 LUT/FF 수 등 **칩 자원 규모**. (칩 크기, 단가와 직결)
- **전력(Power)**: 동작 시 소비 전력 (배터리 수명, 발열과 직결).
- **지연/처리량 (Latency/Throughput)**: 한 블록을 처리하는 데 걸리는 사이클 수 / 초당 처리 비트 수.
- **직렬화 (Serialization)**: 하드웨어 연산 유닛을 작게 만들고 **여러 사이클 돌려 쓰는** 방식 (면적↓, 지연↑).
- **파이프라인 (Pipeline)**: 연산 단계를 쪼개어 여러 블록을 **동시에 겹쳐 처리**하는 방식 (처리량↑, 면적↑).

8.1 설계 트레이드오프

- **작은 S-box** + **단순 확산층(전치나 회전)** → 하드웨어 면적과 전력 절감
- **직렬화**로 극소 면적 구현 가능 (예: 4비트 S-box 1개만 만들어 16번 써먹기) ↔ **병렬화**로 고처리량 구현 (4비트 S-box 16개 병렬 사용)
- **64비트 블록** 채택 (블록이 작으면 회로 소형화에 유리) ↔ **생일한계** 도달이 빠름 (대용량 처리 시 제약 증가)

8.2 운용상 원칙 (엣지/FPGA 관점)

- 기능 검증(암복호만 우선) 단계: **CTR 모드** (단순, 패딩 불필요, 병렬화 유리)
- 제품 적용(무결성 요구) 단계: **AEAD** 모드 (예: **ASCON-AEAD**, **AES-GCM**)
- **키·IV/Nonce 관리**를 처음부터 설계 (유일성, 생성 규칙, 저장 위치, 수명주기 명시)
- **부채널(전력/EM/타이밍/글리치 등) 대비 최소 조치**: 상수 시간 연산, 메모리 접근 패턴 고정, (제품 단계라면) 1차 마스킹 등

---

9. 알고리즘 카탈로그 (ASCON / SIMON / SIMECK / PRESENT / 국산)

> 각 알고리즘의 **핵심 개념, 특징, 사용처**를 간략히 정리.

공통 용어 정리: - **SPN (Substitution–Permutation Network)**: S-box (비선형 치환) + P-레이어(선형 확산) 반복 구조. - **Feistel 구조**: 입력을 L/R 반으로 나눠 한쪽을 F 함수 처리 후 XOR 교환하는 구조 (복호화가 쉬움). - **ARX (Add–Rotate–XOR)**: 덧셈, 회전, XOR만으로 구성된 경량 연산 방식 (하드웨어에 유리).

9.1 ASCON (경량 AEAD/해시)

- **유형/역할**: 블록암호가 아닌 **순열 기반 스펀지 구조**의 **AEAD/해시 알고리즘**.
  (스펀지: 내부 상태에 데이터를 흡수(absorb)하고 짜내기(squeeze) 하는 구조)
- **장점**: 코드와 메모리 footprint가 작고, AEAD 방식이라 **무결성 태그**까지 한 번에 제공. **저사양 장치**에 최적화됨.
- **주의**: Nonce 재사용 금지 (앞서 강조한 대로), 공개된 테스트 벡터와 교차 검증 필수.
- **추천 사용**: **AEAD가 필요한** 센서 데이터 전송, 엣지 로그 저장, 마이크로컨트롤러 간 통신 등.

9.2 SIMON / SIMECK (경량 블록암호, Feistel/ARX)

- **구조**: Feistel 구조에 **회전, 비트 AND, XOR** 등의 ARX 연산 활용. (룩업테이블 거의 없이 논리 게이트로만 구현 가능)
- **장점**: **하드웨어 구현 매우 용이**하고, 면적/전력 효율이 뛰어나다. 다양한 블록/키 크기 조합으로 유연성도 있다.
- **주의**: 변종이 많으므로 **정확한 파라미터 세트**(예: SIMON-128/128 등)를 명시하고, 공식 논문/자료의 테스트 벡터로 검증해야 한다.
- **추천 사용**: 극소형 하드웨어 (저가 소형 FPGA/ASIC), 직렬화 극대화 설계 시.

9.3 PRESENT (경량 블록암호, SPN 구조)

- **구조**: 4비트 S-box × 16개 병렬 + 비트 단위 전치(P-레이어), 총 **31라운드**.
- **장점**: **매우 소형, 저전력** 구현에 적합하고, 구조가 단순해 RTL 설계시 모듈 분리가 용이하다 (예: S-box 하나로 순차처리 또는 16개 병렬 등 융통성).
- **주의**: **64비트 블록**이라 아주 장시간/대용량 운용 시 **생일한계**에 빨리 도달할 수 있다. 따라서 **Nonce/카운터 관리**를 철저히 하거나, 경우에 따라 128비트 블록 암호를 선택할지 고려한다.
- **추천 사용**: 면적이 극도로 제한된 FPGA/ASIC 구현, 교육용/연구용, 제품 PoC 등.

9.4 국산 알고리즘 (LEA / CHAM / HIGHT / PIPO 등)

- **LEA/CHAM**: 128비트 블록 ARX 계열 (LEA는 ARX, CHAM은 ARX+비틀기), 소프트웨어/하드웨어에 모두 친화적이고 국내 구현 사례가 많다.
- **HIGHT**: 64비트 블록 경량 블록암호. IoT 기기에서 소형 구현 사례가 다수 있다 (과거 표준).
- **PIPO**: 64비트 블록 경량 SPN 구조 (학계 제안, 교육용 오픈소스 구현 등 존재).
- **추천**: 국내 자료 접근성이 좋아서 레퍼런스나 실습 자료 활용이 용이하다.

9.5 빠른 선택 가이드

- **무결성 필요 (AEAD)**: **ASCON-AEAD** (저사양) 또는 **AES-GCM** (HW 가속 가능 시 최우선)
- **암복호만 (기능 검증/초소형)**: **PRESENT-CTR** 또는 **SIMON-CTR / SIMECK-CTR**
- **대용량/장시간**: 128비트 블록 암호 (AES, LEA 등) 또는 처음부터 **AEAD 모드** 사용 권장

---

10. PRESENT 한 장 요약 (구조·이점·주의)

용어 다시 정리: - **S-box (4비트)**: 0~15 (4비트) 입력 → 0~15 출력으로 매핑하는 **비선형 치환 표**.
- **P-레이어 (Permutation layer)**: 64비트 상태의 **비트 위치를 고정 규칙으로 재배열**하여 확산을 주는 층.
- **화이트닝 (AddRoundKey)**: 매 라운드 (및 최종 출력 단계)에 **라운드키와 상태를 XOR**하여 키의 영향을 주입.

10.1 파라미터 & 라운드 구조

- **블록/키 크기**: 64비트 블록, **80비트 또는 128비트 키**, **31라운드** 암호
- **암호화 라운드 구성**:
- **AddRoundKey**: 상태(64비트)와 라운드키(64비트)를 XOR
- **S-box 층**: 4비트 S-box를 **16개 병렬** 적용 (64비트 → 64비트)

- **P-레이어**: 64비트 비트를 **고정된 규칙으로 자리 이동** (확산)
  (마지막 라운드 이후에는 AddRoundKey 한 번 더 실행해서 종료)
- **복호화**: 역 S-box, 역 P-레이어를 사용하고 **라운드키는 역순**으로 적용.

구현 팁: S-box를 **1개만 두고** 16번 순차 적용하면 **초소형 설계**가 가능하고,
S-box를 **16개 모두 배치**하면 한 사이클에 한 라운드 처리가 가능해 **고속화**가 가능하다.
P-레이어는 단순 비트 셔플이므로 **조합 논리**로 구현해도 타이밍에만 맞추면 된다.

10.2 키 스케줄 (요지)

- **PRESENT-80**: 80비트 키 레지스터를 매 라운드 **왼쪽으로 61비트 로테이트** → 상위 4비트에 S-box 1개 적용 → 특정 비트에 라운드 카운터 XOR → 차례로 라운드키 추출
- **PRESENT-128**: 128비트 키 레지스터를 사용. 상위 바이트들에 S-box 적용 + 라운드 카운터 XOR 등의 과정 (세부는 표준 문서 참고).
- **검증 포인트**: **라운드 카운터**가 정확히 XOR되었는지, 로테이트 비트수/방향 착오 없는지, S-box 입력 비트 정렬이 맞는지 등.

10.3 하드웨어 아키텍처 선택 (면적 ↔ 처리량)

아키텍처 유형	설명	장점	단점	활용처
완전 직렬 (초소형)	S-box 1개로 16번 연산 = 1 라운드	면적 최소, 전력↓	지연↑, 처리량 낮음	초소형 ASIC/FPGA
라운드 단위 (표준형)	S-box 16개, 1라운드씩 처리	균형 잡힌 설계	면적/처리량 중간	일반적 PoC/제품
풀 파이프라인 (고속)	31단 파이프라인 (라운드별 분할)	처리량 최고	면적↑↑, 초기 지연↑	고속 처리 필요 경로

처리량 대략 계산:
`Throughput ≈ (블록 크기 / 블록당 사이클 수) × f_clk`
예) 블록 64비트, 블록당 31사이클, f_clk = 100MHz → 약 206 Mbit/s 처리량.

10.4 모드 적용 및 시스템 통합

- **CTR 래핑**(권장): `키스트림 = E_K(Nonce || Ctr)` 생성 → `암호문 = 평문 XOR 키스트림`.
- 이때 **Nonce/카운터 관리**를 문서화 (부팅 카운터 + 프레임번호 + 랜덤 등 조합).
- CTR은 복호화도 동일 경로로 처리 (암호화와 똑같이 XOR).
- **AEAD 전환**: 무결성까지 필요하다면 상위 프로토콜에서 **ASCON-AEAD**나 **AES-GCM**으로 감싸는 식으로 보완.
- **메타데이터 (AAD)**: 버전, 알고리즘, Nonce, 데이터 길이, 파일명 등을 함께 저장하여 이후 복원 시 사용.

10.5 부채널 및 신뢰성 포인트

- **상수 시간(시간 불변) 동작**: 조건분기 제거, S-box 접근/연산 시간 일정
- **메모리 패턴 고정**: S-box를 LUT 대신 논리로 구현하거나, 테이블 참조 시 접근 패턴이 키와 무관하게
- **(제품 단계) 1차 마스킹**: S-box 입력과 출력에 난수 마스크 XOR 처리 (전력/EM 측채 대비)

- **Fault 주입 대비**: 중요한 연산은 이중 계산 또는 체크값 삽입 등 결함 공격 감지 장치
- **테스트**: 공개된 **테스트 벡터**로 블록 단위 검증 → CTR 모드 연속 처리 검증 → **파일 전체 암복호 왕복** + **에러 주입(Nonce/태그 변경)** 테스트까지 수행

10.6 구현 체크리스트 (요약)

- [ ] S-box 매핑 테이블 값/비트 순서/엔디언 일관성 확인
- [ ] 키 스케줄: **rol 61**(PRESENT-80) 동작, S-box 적용, 라운드 상수 XOR 정확성
- [ ] 라운드 수 **31**회 누락 없이 수행, **최종 AddRoundKey** 적용
- [ ] CTR 모드: **(키, Nonce) 유일성** 보장, 카운터 오버플로 처리 방안
- [ ] 타이밍/합성: P-레이어 배선 타이밍 체크, 필요한 경우 파이프라인 단계 삽입
- [ ] 부채널 대비: 상수 시간 수행, 메모리 접근 패턴 고정 → (제품) 마스킹 추가 및 TVLA 검증
- [ ] 검증: 테스트 벡터, 스트림 연속 암복호, 실제 파일 입출력 비교, 오류 상황 주입 테스트 (Nonce 중복, 태그 불일치 등)

---

11. HW 구현: 이득과 함정 (SoC/FPGA)

> **왜 하드웨어로 구현하나?** — **고처리량, 저지연, 저전력** 구현이 가능하고, 펌웨어처럼 쉽게 해킹으로 코드를 들여다볼 수 없다.
> **어려운 점은?** — **부채널 공격**(전력/전자파/타이밍/글리치 등), **키 안전 저장**, **업데이트/수명주기 관리** 이슈를 함께 고려해야 한다.

용어 짚고 넘어가기: - **부채널 (Side Channel)**: 수학적 취약점이 아닌 **전력 소비, 연산 시간, EM 방출** 등 물리적 정보를 통해 암호 키 등이 새는 현상.
- **TVLA (Test Vector Leakage Assessment)**: 전력/EM 등의 누설을 **통계적 검정**하여 정보 누출 유무를 판단하는 방법 (일종의 t-테스트 적용).
- **TRNG/DRBG**: **True Random Number Generator** (하드웨어 난수) / **Deterministic Random Bit Generator** (의사난수). IV/Nonce 생성, 마스킹, 챌린지값 등에 필요.
- **eFUSE/OTP/SE/TPM/PUF**: 키 저장/보호 방식들. eFUSE/OTP=칩 내 일회성 프로그래밍 메모리, SE/TPM=별도 보안 칩/모듈, PUF=칩 고유의 물리 특성으로 키 생성.

11.1 이득 (Pros)

- **처리량/지연 최적화**: 파이프라인, 병렬 연산 등으로 **Gbps급 처리량**도 가능.
- **전력 효율**: 동일 성능일 때 SW 대비 에너지 소비가 적다 (특정 작업에 특화되었으므로).
- **공격 표면 축소**: 소프트웨어는 메모리 덤프 등이 쉽지만, 하드웨어 회로는 리버스 엔지니어링이 상대적으로 어렵다. (물론 **물리적 측정 공격**은 별도 이슈)

11.2 함정 (Cons)

- **부채널 취약**: (예) S-box 테이블 접근 패턴이나 연산 시점 차이로 전력/EM 패턴이 드러나 키와 상관관계를 가질 수 있다.
  → **상수 시간 동작**, **접근 패턴 통일**, (제품 단계에서는) **마스킹, 노이즈 주입, 차폐** 등 대책 필요.
- **키 수명주기 관리**: 공장에서 **키 프로비저닝** → 현장 배포 → **교체/폐기**까지 절차를 정의해야 한다.
- **업데이트 비용**: FPGA인 경우 bitstream 배포 시에도 **서명 검증**(무결성/신뢰성)과 **롤백 방지** 등 고려. ASIC은 펌웨어 업데이트로 보완해야 하는데 유연성 제한.
- **타이밍/합성 복병**: P-레이어처럼 배선만 긴 경로의 타이밍 이슈, S-box 논리 최적화, 클럭 도메인 교차(CDC) 처리 등 디지털 설계상의 난제가 있다.

11.3 아키텍처 선택 (요약 비교)

유형	설명	장점	단점	추천 상황
직렬형	S-box 1개 등 최소 유닛 재사용	면적 최소, 전력 ↓	속도 느림, 지연↑	초소형 FPGA/ASIC
라운드 반복형	라운드당 필요한 유닛 배치	균형 잡힘	자원 소모 보통	일반적 제품 구현
풀 파이프라인형	라운드별 파이프라인 처리	처리량 최고	면적↑↑, 설계 복잡	고속/대역폭 요구 시

11.4 보안 부트 (Secure/Measured Boot) 간단 메모

- **Secure Boot**: 부팅 시 **부트로더/비트스트림에 전자서명 검증**을 통과해야만 실행하도록 하여, 인가되지 않은 코드가 실행되지 않게 하는 것.
- **Measured Boot**: 부팅 이미지를 해시하여 **추적용 로그**(PCR 등)에 저장해 두는 것. 부팅 과정에 **무결성 검증 기록**을 남겨 사후 감사 가능.
- FPGA/PSoC 보드에서는 **부트 체인**(ROM → FSBL → 비트스트림 → OS)의 각 단계에 이런 서명/해시 검증을 적용하여, **부팅부터 신뢰**를 쌓는 게 중요하다.

---

12. 구현 체크리스트 (End-to-End)

> 단계별로 **실패 포인트**를 줄이기 위한 체크리스트. (기능 검증 단계: CTR 모드 → 제품 단계: AEAD 적용 기준)

12.1 기획/설계 단계 (Threat Model + 요구 사항)

- [ ] **자산 정의**: 보호 대상 (이미지, 키, 로그 등)과 가정하는 공격자 모델 (물리 접근? 원격 공격?) 명확히
- [ ] **목표 결정**: 1차는 기능 검증(암복호만)인지, 최종은 제품 보안수준(무결성 포함)인지
- [ ] **알고리즘/모드 선택**: PRESENT-CTR (기능 시험용) → **ASCON-AEAD** 또는 **AES-GCM** (제품용)
- [ ] **키 길이/블록 크기 검토**: 64비트 블록의 생일한계 고려 (필요시 128비트 블록 암호로 변경 검토) 및 IV 정책 문서화
- [ ] **성능 지표**: 필요한 처리량, 허용 지연, 사용 가능한 LUT/FF/메모리, 전력 예산 등 명문화

12.2 RTL 설계

- [ ] **모듈 분할**: `cipher_core` (라운드 함수) / `key_schedule` / `ctr_process` (CTR 카운터 처리) / `top` 등으로 분리
- [ ] **S-box 구현**: LUT나 케이스문으로 **상수 시간 동작** 보장 (키에 따라 조건분기 없는 형태)
- [ ] **P-레이어**: 고정 배선으로 연결, 합성 시 타이밍 검토
- [ ] **키 스케줄**: PRESENT-80의 경우 **좌회전 61비트** + S-box + 라운드 상수 XOR 정확히 구현
- [ ] **상수 시간화**: 암호 알고리즘 경로에서 입력(키/데이터)에 따라 걸리는 시간이 변하지 않게
- [ ] **RNG 인터페이스**: TRNG/DRBG로부터 **Nonce/카운터 초기값**을 안전하게 받는 경로 마련

12.3 시뮬레이션/검증

- [ ] **공개 테스트 벡터**로 블록 암복호 정확성 확인 (라운드별 내부 상태도 가능하면 체크)
- [ ] **CTR 연속 암호화 테스트**: 고정 키/Nonce로 다수 블록 암호화 결과를 소프트웨어 구현과 비교
- [ ] **파일 왕복 (Round-trip)**: 임의의 파일/버퍼를 암호화 후 복호화하여 원문과 동일한지 확인

- [ ] **에러 주입**: 복호화 시 일부러 Nonce 1비트 변경 / 카운터 순서 어긋남 / (AEAD일 경우) 태그 1비트 변경 등을 해보고 올바르게 오류 처리(폐기) 되는지 확인
- [ ] **테스트 커버리지**: 키 스케줄 경계값 (라운드 상수), 카운터 오버플로, 리셋 처리, 예외 상황 등을 모두 시뮬레이션

12.4 합성/FPGA 구현

- [ ] **타이밍 여유**: 목표 주파수에서 Fmax 만족 여부, 멀티사이클 경로 설정 (필요시), 클럭 도메인 교차 (CDC) 안전 처리
- [ ] **리소스 사용량**: LUT/FF/BRAM이 예산 내인지 확인 (초과 시 아키텍처 조정)
- [ ] **전력**: 시뮬레이션 기반 스위칭 활발도 추정, 불필요한 토글 줄이기 (클럭 게이팅 등)
- [ ] **I/O 연동**: 카메라/SD카드/AXI-Stream 등 인터페이스 신호 타이밍 맞춤 및 제약 (XDC) 작성

12.5 펌웨어/드라이버 연동

- [ ] **레지스터 맵 정의**: `KEY[], NONCE[], CTRL, STATUS, ERR, DATA_IN, DATA_OUT` 등 레지스터 구성과 동작 명세
- [ ] **Nonce 정책 구현**: 부팅 카운터 + 난수를 합성해 Nonce 생성, **사용한 Nonce는 영구 기록** (중복 검출을 위해)
- [ ] **에러 처리 경로**: 태그 불일치, Nonce 재사용 감지 시 **즉시 데이터 폐기 및 이벤트 로그**
- [ ] **메타데이터 (AAD) 포맷**: 버전, 알고리즘, Nonce, 길이, 파일 ID 등을 포함한 **고정된 헤더**를 정의하고 문서화

12.6 제품화 보안 (고도화)

- [ ] **키 주입**: 공장 프로비저닝 절차 수립 (시리얼 넘버당 키 생성, 기록 보관 등)
- [ ] **키 저장**: eFUSE/OTP 또는 보안 모듈(SE/TPM/PUF) 활용 여부 결정 (키 교체/폐기 시나리오 포함)
- [ ] **Secure Boot 구현**: 부트로더/FPGA 비트스트림 전자서명 및 검증 적용, 롤백 방지
- [ ] **부채널 평가**: 최소한의 TVLA 수행 (전력/EM), 가능하면 1차 마스킹 적용
- [ ] **취약점 관리**: 알고리즘/구현 최신 취약점 트래킹, 라이브러리 버전 업데이트 및 롤백 플랜

---

13. 동형암호 (HE) — 범위 밖 짧은 메모

> **개념**: 서버가 **데이터를 직접 보지 않고도** (기밀성 유지) 그 위에서 연산/통계를 수행할 수 있게 하는 암호 기술.

용어 짚고 넘어가기: - **PHE (Partial HE)**: 한 가지 연산만 보존하는 동형암호 (예: **RSA**는 곱셈 동형, **Paillier**는 덧셈 동형).
- **FHE (Fully HE)**: 이론적으로 **임의의 연산**(회로)을 암호문 상태에서 수행 가능한 동형암호.
- **노이즈 / 부트스트랩**: 동형 연산을 할수록 암호문에 **노이즈**가 쌓여서 복호화 불가능해지는데, 이를 **다시 줄여주는 과정**(부트스트래핑)이 필요하다. (아주 비싼 연산)
- **CKKS**: 수치 연산(실수 근사 계산)을 위한 FHE 스킴 중 하나 (소수점 계산을 근사 동형연산으로 가능케 함).

13.1 장단점

- **장점**: 민감 데이터를 암호화한 상태로 클라우드에 보내 **프라이버시 보호**를 극대화하면서 데이터 분석/머신러닝 예측이 가능. (데이터 유출 위험 감소)
- **단점**: **매우 느리고 메모리 사용량 큼**. 암호문 크기도 평문의 10~100배 이상 커지는 경우가 많다. 현 시점 IoT/엣지 환경에서 **실시간 처리는 사실상 불가능**.

13.2 언제 쓰나?

- 의료/금융 데이터같이, 클라우드에 분석 맡기자니 유출 위험이 너무 큰 경우 (성능 희생해서라도 기밀 유지가 최우선일 때).
- **본 프로젝트 (카메라→FPGA→SD 암호화)** 같은 케이스에는 **지나친 기술**이다. 여기는 AEAD 또는 CTR+HMAC 조합이면 충분하고, 동형암호는 굳이 쓰지 않는다.

---

14. 참고/추가읽기 (한글 위주 자료 + 표준 문서)

개념 이해를 높이고 실제 구현에 도움을 주는 자료들을 추렸다.

- **KISA 암호 키 관리 안내서** – 키 생성/보관/배포/폐기까지 수명주기 가이드
- **KISA 부채널 공격 취약성 평가** – 전력/EM/타이밍/글리치 공격과 TVLA 방법에 대한 개요 및 평가 절차
- **PRESENT 알고리즘 상세** – 구조/키 스케줄/테스트 벡터 등이 정리된 한글 논문, 특허 자료 (라운드별 처리와 rol 61 등의 구현 팁)
- **SIMON/SIMECK 개요** – Feistel/ARX 경량암호 소개와 구현 참고 (다양한 파라미터 세트 주의)
- **NIST Lightweight Crypto 표준 (ASCON)** – 경량 AEAD/해시 표준 문서와 권장 파라미터, 테스트 벡터
- **AES-GCM 권고 사항** – Nonce 96비트 권장 이유, 태그 검증 실패 시 대응 등 모범 사례
- **패스워드 KDF 가이드** – Argon2/scrypt/bcrypt 등의 권장 파라미터 및 적용 사례
- **Secure/Measured Boot 개념** – 부트 체인 단계별 서명/해시 적용과 롤백 방지에 대한 요약 정리

디버깅 팁: 구현 중 막히면,
① **공개 테스트 벡터**로 블록/스트림 결과부터 맞춰보고 →
② **파일 단위 암복호 왕복** 테스트 →
③ **에러 주입**(Nonce 중복, 태그 변조 등) 테스트 →
④ (HW일 경우) **타이밍/전력 분석** 순으로 원인을 좁혀가면, 문제 해결이 훨씬 수월하다.
