BAB II
KAJIAN LITERATUR

2.1 Tinjauan Pustaka

Pada bagian ini berisikan landasan teori dan akan dijelaskan tinjauan pustaka yang berkaitan dengan penelitian yang dilakukan.

2.1.1 Kriptografi Ringan (Lightweight Cryptography/LWC)

Kriptografi ringan ditujukan untuk perangkat berdaya, memori, dan komputasi terbatas (sensor *IoT*, perangkat medis portabel, modul *embedded*). Program standardisasi NIST LWC memformalkan kebutuhan keamanan minimum serta mendorong rancangan yang efisien untuk skenario pesan pendek dan footprint kecil, dengan laporan status finalis/pemenang disajikan dalam NIST IR 8454. Laporan tersebut merangkum alasan pemilihan kandidat, profil ancaman yang dipertimbangkan, dan arah interoperabilitas untuk adopsi industri, menjadi rujukan primer mengapa evaluasi keluaran (termasuk sifat acak *ciphertext*) relevan dalam konteks implementasi nyata pada perangkat terbatas [2].

2.1.2 ASCON sebagai standar LWC

Ascon-AEAD128 merupakan salah satu skema authenticated encryption with associated data (AEAD) berbasis keluarga Ascon yang distandarkan oleh NIST untuk perangkat dengan sumber daya terbatas. Skema ini dirancang untuk menyediakan kerahasiaan (*confidentiality*) dan integritas autentik (*authenticity*) secara simultan terhadap pesan yang dienkripsi, sekaligus menjamin keutuhan associated data yang tidak dienkripsi, namun harus dilindungi dari pemalsuan. Spesifikasi resmi Ascon-AEAD128 dideskripsikan dalam NIST SP 800-232 [2].

Ascon-AEAD128 menggunakan beberapa parameter utama [2]:

1) Kunci rahasia (K)

   K adalah kunci simetris berukuran 128 bit yang dibagikan antara pengirim dan penerima yang berwenang. Keamanan skema sepenuhnya bergantung pada kerahasiaan kunci ini.

2) Nonce (N)

Nonce adalah nilai 128bit yang harus unik untuk setiap proses enkripsi dengan kunci yang sama. Nonce tidak perlu dirahasiakan, tetapi pengulangan nonce untuk pasangan kunci yang sama dapat menurunkan keamanan dan membuka peluang serangan.

3) Associated Data (A)

Associated Data adalah data berdimensi variabel yang tidak dienkripsi, namun dilindungi integritas dan keasliannya melalui tag autentikasi. Contohnya meliputi header protokol, identitas perangkat, atau metadata lain yang penting diverifikasi namun tetap dapat dibaca.

4) Plaintext (P)

Plaintext adalah pesan yang akan diberikan perlindungan kerahasiaan dan integritas. Panjangnya bersifat variabel.

5) Ciphertext (C)

Ciphertext merupakan hasil enkripsi plaintext dengan panjang yang sama dengan P.

6) Tag autentikasi (T)

Tag adalah nilai autentikasi sepanjang 128 bit yang dihasilkan pada akhir proses enkripsi. Tag ini mengikat (K, N, A, C) sehingga setiap modifikasi pada salah satu elemen tersebut akan menyebabkan verifikasi tag gagal saat dekripsi.

7) State internal (S)

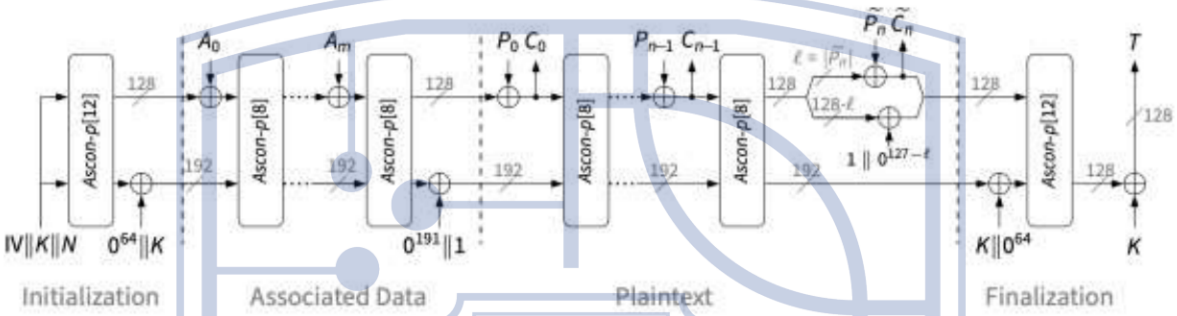
Ascon menggunakan state internal berukuran 320 bit yang direpresentasikan sebagai lima buah kata 64 bit. Seluruh operasi Ascon-AEAD128 dibangun di atas transformasi permutasi Ascon-p[r], yang bekerja pada state ini.

Dalam deskripsi algoritma digunakan notasi berikut [2]:

- $X||Y$: konkatenasi bitstring X dan Y.
- X⊕Y : operasi XOR bit per bit.
- |X| : panjang bit dari X.
- S[i:j] : subset bit dari state S, dari indeks i hingga j.
- parse(M, b) : fungsi yang membagi pesan M menjadi blok-blok berukuran b bit, dengan satu blok terakhir yang mungkin tidak penuh.
- pad(X, b) : fungsi yang menambahkan bit 1 diikuti bit 0 hingga panjang total mencapai b bit, digunakan untuk blok terakhir yang tidak penuh.
- *Ascon-p[12]* dan *Ascon-p[8]*: permutasi internal dengan 12 dan 8 ronde yang menyediakan confusion dan diffusion pada state. NIST Publications

Ascon-AEAD128 menggunakan konfigurasi sponge-like dengan rate sebesar 128 bit dan capacity sebesar 192 bit. Rate menentukan bagian state yang berinteraksi langsung dengan data (A dan P), sedangkan capacity berperan sebagai bagian "internal" yang memberikan margin keamanan terhadap serangan kriptanalitik [2].

Ascon-AEAD128 bekerja melalui empat fase utama yang saling berurutan, yaitu inisialisasi, pemrosesan *associated data*, pemrosesan plaintext, dan finalisasi yang diilustrasikan sebagai berikut.



Gambar 2.1 Ilustrasi algoritma ASCON-AEAD128 [2]

Keempat fase ini membentuk satu alur terpadu yang memastikan bahwa kunci rahasia, nonce, data tambahan, dan plaintext seluruhnya berkontribusi terhadap pembentukan ciphertext dan tag autentikasi secara terikat dan aman.

Pada fase inisialisasi, algoritma terlebih dahulu membangun *internal state* berukuran 320 bit sebagai fondasi seluruh operasi. State ini dibentuk melalui konkatenasi nilai inisialisasi tetap (IV) 64 bit, kunci rahasia K sepanjang 128 bit, dan nonce N sepanjang 128 bit, sehingga diperoleh susunan awal $S = IV||K||N$. State tersebut kemudian diproses menggunakan permutasi *Ascon-p[12]*, yaitu permutasi non-linear bersusun ronde yang bekerja di atas kelima kata 64 bit pembentuk state. Setelah proses permutasi, kunci rahasia K kembali diinjeksikan ke dalam state dengan cara di-XOR-kan pada 128 bit terakhir, sedangkan 192 bit pertama dibiarkan tidak berubah. Tahapan ini dirancang untuk mengikat kunci secara kuat ke dalam kondisi internal algoritma, sehingga pihak yang tidak mengetahui kunci tidak dapat merekayasa state awal maupun melacak kembali hubungan langsung antara masukan eksternal dan state internal [2].

Fase berikutnya adalah pemrosesan *associated data* (A), yang berfungsi untuk memasukkan data tambahan yang tidak dienkripsi namun harus dilindungi integritasnya. Jika A tidak kosong, data ini terlebih dahulu diparsing menjadi blok-blok berukuran 128 bit; blok terakhir yang tidak penuh dipenuhi menggunakan skema *padding* dengan penambahan

bit ‘1’ diikuti deretan bit ‘0’ hingga mencapai panjang 128 bit. Setiap blok *associated data* kemudian diabsorpsi ke dalam state dengan operasi XOR pada 128 bit pertama (bagian *rate*), dan setelah setiap blok diproses, permutasi *Ascon-p[8]* diterapkan untuk mendistribusikan pengaruh blok tersebut ke seluruh state. Dengan demikian, setiap variasi pada A akan mengubah evolusi state internal. Setelah seluruh *associated data* selesai diproses, algoritma menambahkan satu bit domain separation dengan melakukan XOR terhadap konstanta yang hanya mengubah bit terakhir state. Pemberian bit pemisah ini secara eksplisit menandai berakhirnya fase *associated data* dan mencegah terjadinya ambiguitas struktural antara jalur pemrosesan A dan plaintext dalam analisis keamanan. Jika A kosong, algoritma langsung menerapkan langkah domain separation ini tanpa proses absorbsi blok [2].

Setelah domain separation, algoritma memasuki fase pemrosesan plaintext yang sekaligus menghasilkan ciphertext. Plaintext P diparsing menjadi blok-blok 128 bit penuh dan satu blok terakhir yang mungkin berukuran kurang dari 128 bit. Untuk setiap blok penuh, nilai blok tersebut di-XOR-kan ke 128 bit pertama state; hasil dari 128 bit pertama setelah operasi XOR ini langsung diekstraksi sebagai blok ciphertext yang bersesuaian. Selanjutnya, permutasi *Ascon-p[8]* diterapkan untuk memperbaharui state sebelum memproses blok berikutnya. Dengan konstruksi ini, ciphertext bukan sekadar hasil XOR antara plaintext dengan keystream klasik, melainkan keluaran dari state internal yang terus berkembang dan yang sejak awal telah terkait dengan kunci, nonce, dan *associated data*. Untuk blok terakhir yang tidak penuh, plaintext parsial terlebih dahulu dipenuhi dengan skema *padding* hingga 128 bit, kemudian di-XOR-kan ke bagian *rate* state. Ciphertext bagian akhir diperoleh hanya dari sejumlah bit pertama yang sesuai dengan panjang plaintext aktual, sementara sisa bit hasil padding tidak disertakan dalam keluaran. Seluruh blok ciphertext yang diperoleh kemudian digabungkan membentuk ciphertext akhir C, yang panjangnya identik dengan plaintext semula [2].

Fase terakhir adalah finalisasi dan pembentukan tag autentikasi. Pada tahap ini, kunci rahasia K kembali dimasukkan ke dalam state sebagai langkah penguatan keterikatan antara state akhir dengan kunci. Integrasi ini dilakukan dengan meng-XOR-kan K pada bagian tengah state (setelah 128 bit pertama), sementara 64 bit terakhir diisi dengan nol sesuai definisi algoritma. State yang telah dimodifikasi tersebut kemudian diproses dengan permutasi *Ascon-p[12]*, sehingga seluruh informasi mengenai kunci, nonce, *associated data*, dan ciphertext yang telah diabsorbsi sebelumnya tercampur secara non-linear di dalam state. Tag autentikasi T kemudian dibentuk dengan mengekstraksi 128 bit terakhir state dan melakukan operasi XOR dengan kunci K. Nilai T inilah yang dikirimkan bersama ciphertext

dan digunakan pada sisi penerima untuk memverifikasi keaslian serta keutuhan keseluruhan transkrip (N, A, C) di bawah kunci yang sama. Apabila pada proses dekripsi nilai tag yang dihitung ulang tidak identik dengan T yang diterima, maka algoritma menyatakan kegagalan dan plaintext tidak dikeluarkan, sehingga mencegah pemalsuan pesan maupun manipulasi data tanpa terdeteksi [2].

Dari sudut pandang desain, Ascon-AEAD128 memadukan prinsip *sponge/duplex* dengan permutasi ringan Ascon-p untuk mencapai efisiensi tinggi sekaligus keamanan yang terukur. Setiap perubahan pada nonce, associated data, atau ciphertext akan mempengaruhi state internal sehingga menghasilkan tag yang berbeda. Jika, pada proses dekripsi, tag yang dihitung ulang tidak sesuai dengan T yang diterima, algoritma mengembalikan kegagalan (fail) dan plaintext tidak dikeluarkan, sehingga mencegah serangan pemalsuan pesan [2].

2.1.3 TinyJAMBU

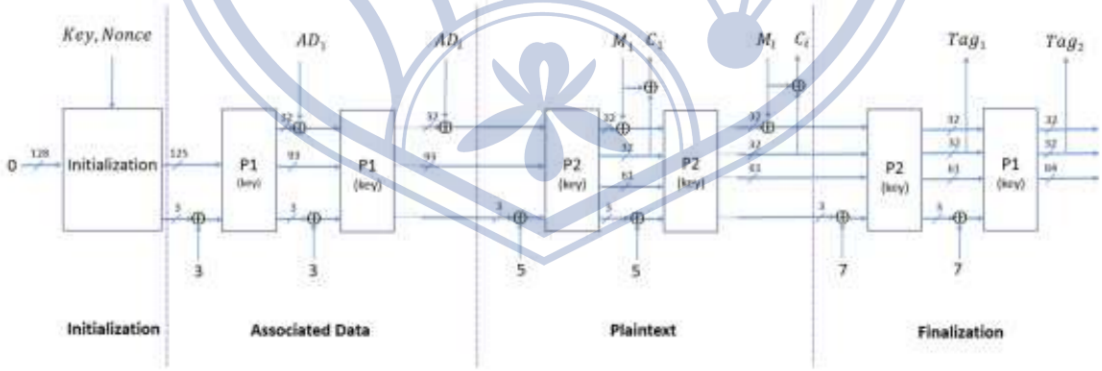
TinyJAMBU adalah keluarga algoritma *authenticated encryption with associated data* (AEAD) yang dirancang untuk perangkat berdaya dan ber-memori sangat terbatas. Algoritma ini mengadopsi mode enkripsi terotentikasi berbasis suatu *keyed permutation* 128-bit yang dibangun di atas *nonlinear feedback shift register* (NLFSR), sehingga mampu memberikan tingkat keamanan setara 128-bit dengan jejak perangkat keras yang sangat kecil dan struktur implementasi yang sederhana. Dalam rancangan TinyJAMBU, keamanan, efisiensi, dan kesederhanaan diharmonisasikan melalui pemrosesan terintegrasi antara kunci rahasia, *nonce*, *associated data*, plaintext, ciphertext, dan tag autentikasi [5].

Secara umum, TinyJAMBU mendukung tiga ukuran kunci, yaitu 128, 192, dan 256 bit, dengan *nonce* sepanjang 96 bit dan tag autentikasi 64 bit, sementara ukuran state internal tetap 128 bit. Kunci rahasia dilambangkan dengan K, dengan panjang K_{len} yang sesuai varian (128/192/256 bit). *Nonce* dilambangkan sebagai nilai 96 bit yang wajib unik untuk setiap pasangan kunci–pesan, dan berfungsi mencegah pengulangan keystream maupun struktur internal yang dapat dieksploitasi. Pesan asli yang akan dienkripsi dinyatakan sebagai plaintext M dengan panjang M_{len} bit, sedangkan data yang hanya diautentikasi tetapi tidak dienkripsi, seperti header protokol atau metadata, dinyatakan sebagai *associated data* AD dengan panjang AD_{len} bit. Proses enkripsi menghasilkan ciphertext C dan sebuah tag autentikasi 64 bit, T, yang bersama-sama menjamin kerahasiaan serta keutuhan dan keaslian seluruh *transcript* $(K, NONCE, AD, C)$. Di dalam algoritma, digunakan state internal 128 bit

yang dilambangkan dengan $S = (s_0, s_1, \ldots, s_{127})$, yang terus diperbarui sepanjang seluruh fase algoritma [5].

Inti operasional TinyJAMBU adalah permutasi berparameter P_n, yaitu *keyed permutation* 128-bit yang terdiri atas n putaran fungsi pembaruan state. Pada setiap putaran, *nonlinear feedback shift register* digunakan untuk menghasilkan sebuah bit umpan balik (*feedback bit*) berdasarkan kombinasi beberapa bit state terpilih, operasi logika non-linear, dan satu bit kunci yang dipetakan secara siklik terhadap panjang kunci. Bit umpan balik ini kemudian dimasukkan ke posisi bit paling signifikan setelah seluruh bit state digeser, sehingga setiap putaran menyusun transformasi yang bergantung pada kunci dan sangat sensitif terhadap perubahan state. Bentuk umum fungsi pembaruan ini memanfaatkan operasi XOR ($\oplus$), AND (&), NOT, serta pemilihan beberapa posisi bit (tap) untuk menghasilkan struktur non-linear yang kuat, tanpa memerlukan *key schedule* terpisah. Notasi P_{640}, P_{1024}, dan seterusnya menyatakan banyaknya putaran fungsi pembaruan yang diaplikasikan secara berurutan pada state [5].

Proses kerja TinyJAMBU dapat dipandang sebagai rangkaian empat fase utama yang semuanya menggunakan state yang sama dengan pemisahan domain yang tegas melalui *frame bits*. Untuk membedakan konteks data yang sedang diproses, TinyJAMBU memperkenalkan tiga bit penanda domain (*FrameBits*) yang disisipkan ke dalam state sebelum permutasi dijalankan. Frame bits ini memiliki nilai berbeda untuk setiap fase, antara lain penanda untuk pemrosesan nonce, untuk *associated data*, untuk plaintext/ciphertext, serta untuk fase finalisasi/tag, yang dapat diilustrasikan sebagai berikut.



Gambar 2.2 Ilustrasi algoritma TinyJAMBU [5]

Penyisipan *frame bits* memastikan bahwa kontribusi nonce, AD, pesan, dan langkah finalisasi tidak dapat dipertukarkan atau dikaburkan, sekaligus mencegah serangan yang memanfaatkan kesamaan struktur antara fase yang berbeda [5].

Pada fase inisialisasi, kunci rahasia pertama-tama digunakan untuk merandomisasi state melalui penerapan permutasi berkali-kali (misalnya P_{1024} state awal bernilai nol. Tahap ini menghasilkan state awal yang telah sepenuhnya bergantung pada kunci. Selanjutnya, nonce 96 bit dimasukkan secara bertahap dalam beberapa blok, di mana setiap langkah diawali dengan penyisipan *frame bits* khusus nonce, diikuti penerapan permutasi lagi, lalu penggabungan blok nonce ke bagian tertentu dari state. Kombinasi penyisipan domain dan permutasi berulang ini memastikan bahwa setiap pasangan $(K, NONCE)$ memunculkan state internal unik yang menjadi basis seluruh proses berikutnya, sehingga pengulangan nonce atau kunci dapat terdeteksi atau dibatasi sesuai model keamanan yang dianalisis dalam spesifikasi resmi [5].

Fase berikutnya adalah pemrosesan *associated data* yang bertujuan memasukkan pengaruh AD ke dalam state sehingga ikut terlindungi oleh tag autentikasi, meskipun tidak dienkripsi. Setiap blok AD diproses dengan cara: penanda domain AD disisipkan ke state, permutasi dijalankan sejumlah putaran yang ditentukan, lalu blok AD di-XOR-kan ke bagian state. Jika blok terakhir tidak penuh, bagian yang ada disisipkan, dan panjang blok parsial dikodekan ke state. Dengan cara tersebut, perbedaan urutan, panjang, ataupun isi AD akan menghasilkan perubahan state yang signifikan, sehingga setiap modifikasi terhadap AD akan tercermin pada nilai tag dan dapat terdeteksi penerima [5].

Setelah AD diproses, algoritma memasuki fase enkripsi plaintext. Di sini, kombinasi antara pembangkitan keystream dan penyerapannya kembali dilakukan secara serempak. Untuk setiap blok plaintext, penanda domain untuk data terenkripsi disisipkan, permutasi dijalankan, kemudian bagian tertentu dari state digunakan sebagai keystream yang di-XOR dengan plaintext untuk menghasilkan ciphertext, sementara plaintext (atau ciphertext, sesuai definisi mode) juga diserap ke dalam state melalui operasi XOR pada segmen bit lain. Jika blok terakhir parsial, perlakuan serupa dilakukan untuk sejumlah bit yang tersedia, dan panjang blok parsial kembali dikodekan ke dalam state. Dengan mekanisme ini, setiap bit ciphertext menjadi fungsi non-linear dari kunci, nonce, AD, serta semua bit plaintext sebelumnya, dan secara bersamaan memperkaya state yang kelak menentukan nilai tag autentikasi [5].

Pada fase finalisasi, setelah seluruh ciphertext dihasilkan, TinyJAMBU menggunakan penanda domain final untuk menandai bahwa tidak ada data baru yang akan dimasukkan. State kemudian dipermutasi kembali dalam dua tahap. Dari state hasil permutasi tersebut diambil dua bagian 32 bit yang kemudian dikonkatenasi menjadi tag autentikasi 64 bit. Tag ini merupakan ringkasan kriptografis dari seluruh interaksi data dalam

sesi tersebut. Di sisi dekripsi, penerima mengulangi seluruh proses dengan kunci, nonce, dan AD yang sama, mengekstrak plaintext dari ciphertext menggunakan keystream yang identik, lalu menjalankan kembali fase finalisasi untuk memperoleh tag lokal. Hanya jika tag lokal sama persis dengan tag yang diterima, ciphertext dan AD dinyatakan valid; jika tidak, seluruh pesan ditolak. Prosedur ini memastikan bahwa TinyJAMBU tidak hanya menjamin kerahasiaan data, tetapi juga integritas dan keaslian, dengan efisiensi tinggi yang selaras dengan kebutuhan kriptografi ringan [5].

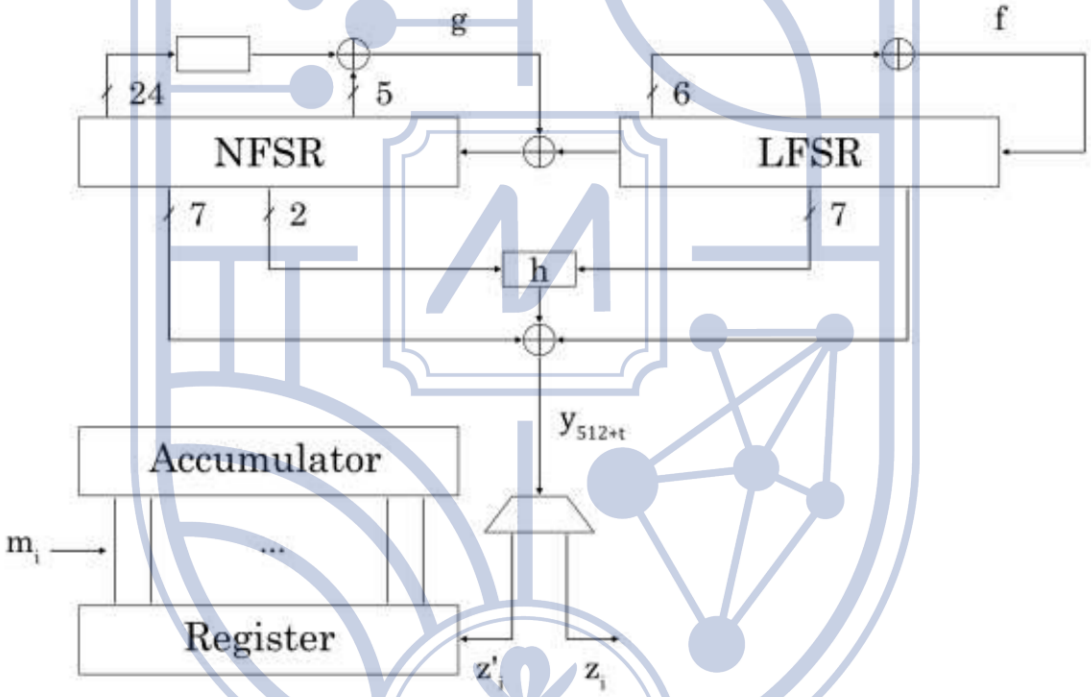
2.1.4 Grain-128AEADv2

Grain-128AEADv2 merupakan salah satu algoritma *authenticated encryption with associated data* (AEAD) yang dikembangkan dari keluarga *stream cipher* Grain dengan tingkat keamanan 128-bit dan dirancang khusus untuk lingkungan komputasi terbatas seperti perangkat IoT dan sistem tertanam. Algoritma ini mengintegrasikan dua tujuan utama, yakni kerahasiaan (*confidentiality*) dan keutuhan serta keaslian data (*integrity* dan *authenticity*), melalui mekanisme enkripsi berbasis *keystream* dan pembangkitan *authentication tag* dalam satu rancangan yang efisien. Dalam spesifikasinya, Grain-128AEADv2 menggunakan kunci rahasia berdimensi 128 bit dan *nonce* berdimensi 96 bit, di mana kombinasi keduanya harus unik untuk setiap proses enkripsi agar keamanan terjaga [9].

Secara internal, Grain-128AEADv2 dibangun di atas suatu *pre-output generator* yang berperan sebagai inti pembangkit deret bit pseudo-acak. Deret ini kemudian dimanfaatkan dalam dua fungsi yang saling berkaitan: pertama, sebagai *keystream* yang digunakan untuk mengenkripsi *plaintext* menjadi *ciphertext*; kedua, sebagai sumber bit autentikasi yang diproses lebih lanjut dalam suatu mekanisme *message authentication code* (MAC) berbasis *shift register* dan *accumulator*. Struktur internal utama *pre-output generator* terdiri atas dua buah *shift register*, yaitu *Linear Feedback Shift Register* (LFSR) sepanjang 128 bit dan *Non-linear Feedback Shift Register* (NFSR) sepanjang 128 bit. LFSR menggunakan fungsi umpan balik linear, sedangkan NFSR menggunakan fungsi umpan balik non-linear. Interaksi antara keduanya melalui fungsi-fungsi boolean yang terdefinisi dengan cermat menghasilkan evolusi *state* yang kompleks dan sulit diprediksi, sehingga memperkuat keamanan algoritma [9].

Parameter dan notasi yang digunakan dalam Grain-128AEADv2 didefinisikan secara sistematis. Kunci rahasia dilambangkan dengan k, terdiri atas 128 bit $(k_0, k_1, \ldots k_{127})$. *Nonce* atau *initialization vector* dilambangkan dengan IV, terdiri atas 96 bit

$(IV_0, IV_1, \ldots, IV_{95})$ dan harus berbeda untuk setiap proses enkripsi di bawah kunci yang sama. LFSR pada langkah waktu ke-t dinyatakan sebagai vektor $S_t = [S_{t,0}, S_{t,1}, \ldots, S_{t,127}]$, sedangkan NFSR sebagai $B_t = [b_{t,0}, b_{t,1}, \ldots, b_{t,127}]$ Kedua register ini diperbarui dengan fungsi umpan balik tertentu: LFSR menggunakan kombinasi linear bit-bit sebelumnya, sedangkan NFSR menggunakan kombinasi linear dan non-linear. Fungsi keluaran non-linear yang memanfaatkan beberapa bit terpilih dari LFSR dan NFSR menghasilkan sebuah bit keluaran antara yang disebut *pre-output* dan dinotasikan sebagai y_t. Dari deret y_t inilah kemudian diturunkan bit-bit keystream (z_i) dan bit autentikasi $(z_i(0))$ yang menjadi dasar enkripsi dan pembentukan tag [9]. Untuk prosedur kerja dari algoritma ini dapat di ilustrasikan sebagai berikut.



Gambar 2.3 Gambaran umum blok penyusun dalam Grain-128AEADv2 [9]

Prosedur kerja Grain-128AEADv2 diawali dengan fase inisialisasi. Pada fase ini, seluruh bit NFSR diisi dengan bit-bit kunci, sedangkan LFSR diisi dengan *nonce* pada 96 bit awal dan dilengkapi dengan pola tetap pada sisa bit untuk menjamin sifat diferensiasi awal *state*. Setelah pengisian awal, algoritma menjalankan sejumlah langkah *clocking* tanpa menghasilkan output eksternal. Pada setiap langkah, fungsi keluaran y_t dihitung menggunakan kombinasi bit-bit dari LFSR dan NFSR melalui fungsi boolean yang telah ditetapkan. Nilai y_t ini kemudian diumpankan kembali ke dalam fungsi umpan balik LFSR

dan NFSR sehingga menghasilkan difusi yang kuat antara kunci, *nonce*, dan *state* internal. Selain itu, pada bagian akhir fase inisialisasi, bit-bit kunci juga disisipkan kembali (*key re-injection*) ke dalam proses pembaruan register. Rangkaian operasi ini memastikan bahwa keadaan internal akhir setelah inisialisasi merupakan fungsi non-linear yang kompleks dari kunci dan *nonce*, sehingga tidak mudah direkonstruksi meskipun penyerang mengamati keluaran algoritma [9].

Setelah fase inisialisasi selesai, algoritma memasuki fase operasi yang digunakan untuk memproses pesan. Dari *state* akhir hasil inisialisasi, *pre-output generator* menghasilkan deret y_t yang kemudian dipetakan secara sistematis: sebagian bit digunakan sebagai keystream z_i untuk enkripsi, dan sebagian lagi digunakan sebagai bit autentikasi $z_i(0)$. Pesan asli (*plaintext*) direpresentasikan sebagai deret bit m_i , sementara hasil enkripsi (*ciphertext*) dinyatakan sebagai c_i . Proses enkripsi setiap bit dilakukan dengan operasi XOR sederhana, yaitu $c_i = m_i \oplus z_i$, sebagaimana karakteristik utama *stream cipher*. Dengan demikian, tingkat kerahasiaan sangat bergantung pada ketidak-terdugaan deret keystream yang diturunkan dari *state* internal Grain-128AEADv2 [9].

Secara paralel dengan enkripsi, Grain-128AEADv2 melaksanakan mekanisme autentikasi untuk menjamin integritas dan keaslian data. Untuk keperluan ini, digunakan dua komponen utama, yaitu *authentication shift register* dan *accumulator*, masing-masing berdimensi 64 bit. Bit autentikasi $z_i(0)$ yang diambil dari deret *pre-output* dimasukkan secara berurutan ke dalam *shift register*, sehingga register ini pada setiap langkah menyimpan kombinasi dinamis dari bit-bit autentikasi terakhir. Selanjutnya, accumulator diperbarui berdasarkan nilai bit pesan: bila suatu bit pesan (atau bit yang dimaknai dalam kerangka AEAD) bernilai 1, maka isi *shift register* pada saat itu dioperasikan XOR ke accumulator; bila bernilai 0, tidak ada perubahan. Dengan cara ini, nilai akhir accumulator menjadi fungsi dari seluruh deret bit yang diproses, yang telah terikat pada deret autentikasi yang berasal dari kunci dan *nonce*. Pada akhir proses, isi accumulator inilah yang digunakan sebagai *authentication tag* yang menyertai ciphertext. Penerima yang memiliki kunci dan *nonce* yang sama akan melakukan proses yang sama; kecocokan tag menunjukkan bahwa ciphertext dan data terkait tidak mengalami perubahan [9].

Sebagai algoritma AEAD, Grain-128AEADv2 tidak hanya mengenkripsi pesan, tetapi juga mendukung *associated data* yang tidak dienkripsi namun tetap diautentikasi. Data terkait ini, yang dapat berupa header protokol atau metadata lain, diikutsertakan dalam proses pembaruan mekanisme autentikasi melalui pengaturan pola masukan tanpa dikenai operasi XOR dengan keystream. Dengan demikian, data tersebut tetap dapat diverifikasi

keasliannya melalui tag, sekalipun disampaikan dalam bentuk terbuka. Pemisahan peran antara bit yang dienkripsi dan bit yang hanya diautentikasi diatur melalui definisi posisi dan pemetaan bit dalam spesifikasi, namun seluruhnya tetap terikat pada struktur internal yang sama sehingga keamanan tetap konsisten dalam satu kerangka matematis [9].

Secara keseluruhan, Grain-128AEADv2 memadukan desain *stream cipher* yang ringan dan efisien dengan mekanisme autentikasi terintegrasi berbasis *shift register* dan accumulator. Kombinasi LFSR dan NFSR dengan fungsi non-linear yang terpilih, fase inisialisasi yang memperkuat keterikatan antara kunci dan *nonce*, serta pemisahan terstruktur antara bit keystream dan bit autentikasi menghasilkan sebuah algoritma AEAD yang sesuai untuk lingkungan sumber daya terbatas, namun tetap memenuhi kebutuhan keamanan modern dengan tingkat kekuatan 128-bit. Narasi ini dapat digunakan sebagai dasar penjelasan formal pada bab tinjauan pustaka atau landasan teori dalam dokumen akademik yang membahas Grain-128AEADv2 [9].

2.1.5 AES-128

Sebagai Baseline (Tolok Ukur) Komparasi, Pemilihan AES (*Advanced Encryption Standard*) sebagai *baseline* dalam penelitian ini didasarkan pada posisinya sebagai standar enkripsi simetris global yang paling banyak dianalisis, diimplementasikan, dan teruji ketahanannya terhadap berbagai serangan kriptanalisis [25, 26]. Berbagai literatur, termasuk survei komprehensif mengenai arsitektur AES [26] dan analisis perluasan keamanannya [25], menegaskan bahwa jaringan *Substitution-Permutation Network* (SPN) pada AES menghasilkan profil kerandoman (efek *confusion* dan *diffusion*) yang menjadi *gold standard* atau tolok ukur absolut bagi algoritma kriptografi modern.

Untuk menjawab kebutuhan metodologis akan perbandingan yang adil dan setara (*apple-to-apple*) dengan kandidat algoritma Kriptografi Ringan (yang mayoritas beroperasi sebagai algoritma *Authenticated Encryption with Associated Data* / AEAD), AES dalam penelitian ini diimplementasikan secara spesifik menggunakan mode operasi **Galois/Counter Mode (GCM)**.

Parameter enkripsi AES-128-GCM diselaraskan sepenuhnya dengan algoritma uji (ASCON, TinyJAMBU, dan Grain) melalui konfigurasi berikut:

1. **Ukuran Kunci (Key Size):** Ditetapkan sebesar 128-bit, sejalan dengan standar keamanan minimum yang direkomendasikan saat ini dan sesuai dengan kapasitas algoritma LWC yang diuji.
2. **Ukuran Nonce/IV:** Ditetapkan sebesar 96-bit. Ini merupakan panjang operasional yang paling optimal dan direkomendasikan untuk mode GCM guna meminimalkan probabilitas tabrakan *counter* (collision) yang dapat merusak kerandoman *keystream*.
3. **Ekstraksi Output (Isolasi Ciphertext):** Pada penelitian ini, sistem diprogram untuk mengekstrak hanya *ciphertext* murni. *Authentication Tag* sepanjang 128-bit (yang dihasilkan dari proses GHASH pada mode GCM) dibuang secara eksplisit. Hal ini dilakukan agar pengukuran statistik benar-benar terfokus pada kualitas transformasi kerahasiaan enkripsi, setara dengan perlakuan isolasi *ciphertext* pada objek LWC.

2.1.6 Diferensiasi Konseptual: Entropi, Kerandoman Statistik, dan Keamanan Kriptografi

Dalam literatur keamanan informasi, istilah entropi, kerandoman, dan keamanan kriptografi kerap digunakan pada konteks yang berdekatan. Namun, guna menghindari ambiguitas operasional dalam penelitian ini, ketiga konsep tersebut dikunci dan dibedakan definisinya secara tegas sebagai berikut:

1. Entropi (Entropy)

Secara konseptual, entropi adalah ukuran teoretis dari ketidakpastian atau "kandungan informasi" murni di dalam sebuah sistem (berakar pada Teori Informasi Shannon). Dalam kriptografi, entropi merepresentasikan sifat intrinsik dari *input* atau sumber nilai acak dasar (seperti *True Random Number Generator* atau kualitas *Key*). Nilai entropi yang tinggi berarti sistem memiliki ruang kemungkinan yang sangat luas, sehingga mustahil bagi penyerang untuk menebak *state* awalnya. Entropi adalah "bahan bakar" bagi sistem keamanan [7, 24].

2. Kerandoman Statistik (Statistical Randomness)

Kerandoman statistik adalah **sifat empiris yang dapat diobservasi** pada deret keluaran (*output/ciphertext*). Sebuah algoritma enkripsi bersifat deterministik (jika input sama, output pasti sama), sehingga secara teoretis entropi pada *ciphertext* tidak akan pernah melebihi entropi dari *kunci*-nya. Namun, algoritma yang baik harus mampu menyebarkan

nilai tersebut sedemikian rupa sehingga deret *ciphertext* yang dihasilkan **tampak** acak sempurna (tidak memiliki pola bit, osilasi runtun seragam, dll). Kerandoman statistik inilah yang secara spesifik diukur menggunakan instrumen empiris seperti NIST STS [21].

3. Keamanan Kriptografi (Cryptographic Security)

Keamanan kriptografi adalah konsep payung (*umbrella concept*) yang paling luas, merujuk pada ketahanan sistem secara keseluruhan terhadap berbagai jenis serangan (*cryptanalysis*). Sebuah sistem disebut aman secara kriptografis jika ia mampu menahan serangan matematis (seperti *differential/linear cryptanalysis*), serangan aljabar, hingga serangan fisik (*side-channel attack*) [10, 17].

Hubungan Ketiganya dalam Konteks Penelitian

Ketiga konsep ini membentuk hierarki persyaratan, namun tidak bersinonim. Hubungannya dikunci dalam postulat berikut: *Sumber input harus memiliki **Entropi** yang tinggi untuk diproses oleh algoritma. Algoritma kemudian harus menghasilkan keluaran yang memiliki **Kerandoman Statistik** yang lulus uji (seperti NIST STS). Namun, memiliki kerandoman statistik yang baik hanyalah syarat awal (necessary condition); ia tidak serta-merta menjamin **Keamanan Kriptografi** yang absolut dari serangan peretasan tingkat tinggi.* Fokus utama dari tesis ini dibatasi pada evaluasi **Kerandoman Statistik**, sebagai *sanity check* fundamental sebelum algoritma LWC dinyatakan layak untuk dianalisis lebih lanjut pada tingkat keamanan kriptografinya [12, 25].

2.1.7 Pengujian statistik (NIST STS)

NIST SP 800-22 adalah standar *de facto* untuk menguji generator bilangan acak. Perangkat lunak NIST Statistical Test Suite (STS) didistribusikan secara resmi dalam format kode sumber bahasa C. Penggunaan compiler seperti GCC diperlukan untuk mengkompilasi program ini yang dibangun mengacu pada standar pengujian kerandoman NIST SP 800-22, instrumen standar yang dievaluasi secara luas untuk menguji independensi statistik luaran algoritma kriptografi [21]. Dalam konteks evaluasi algoritma enkripsi, lulus uji NIST STS merupakan indikator kuat bahwa algoritma tersebut memenuhi dua prinsip dasar Shannon, yaitu:

1. **Confusion (Konfusi):** Hubungan antara kunci dan *ciphertext* harus sangat kompleks. Uji seperti *Frequency Test* dan *Entropy* memvalidasi hal ini dengan memastikan tidak ada bias bit yang dapat dikaitkan dengan kunci.
2. **Diffusion (Difusi):** Perubahan satu bit pada *plaintext* harus menyebar ke banyak bit pada *ciphertext*. Uji seperti *Block Frequency* dan *Serial Test* memvalidasi properti ini dengan mendeteksi pola lokal yang mungkin muncul akibat difusi yang buruk.

Alat ini terdiri dari 15 uji statistik yang dirancang untuk mendeteksi berbagai jenis pola non-acak sebagai berikut :

1. **Frequency (Monobit) Test**: Memeriksa proporsi bit 0 dan 1 (harus mendekati 50%).
2. **Frequency Test within a Block**: Memeriksa proporsi *bit* dalam blok-blok kecil.
3. **Runs Test**: Menguji kerandoman transisi antara 0 dan 1.
4. **Longest Run of Ones**: Memeriksa panjang *run*-1 terpanjang dalam blok.
5. **Binary Matrix Rank Test**: Menguji ketergantungan *linear* antar sub-blok.
6. **DFT (Spectral) Test**: Mendeteksi pola periodik dalam *sequence*.
7. **Overlapping Template Matching Test**: Mendeteksi kemunculan pola bit tertentu yang terlalu sering.
8. **Non-overlapping Template Matching**: Mirip dengan Template Matching, tapi tanpa tumpang tindih.
9. **Maurer's Universal Statistical Test**: Mengukur kompresibilitas *sequence* (terkait entropi).
10. **Linear Complexity Test:** Menguji kompleksitas *linear sequence* (penting untuk *stream cipher*).
11. **Serial Test**: Memeriksa frekuensi seluruh pola m-bit yang mungkin.
12. **Approximate Entropy Test**: Membandingkan frekuensi pola blok yang tumpang tindih.
13. **Cumulative Sums (Cusum):** Menguji *random walk* dari *sequence*.
14. **Random Excursions**: Menguji jumlah kunjungan ke *state* tertentu dalam *random walk*.
15. **Random Excursions Variant**: varian dari kunjungan ke *state* tertentu dalam *random walk*.

Prinsip dasar pengujian ini adalah mengukur $p - value$ untuk setiap uji. Sebuah $p - value$ diinterpretasi sebagai probabilitas bahwa generator acak ideal akan menghasilkan

deret yang, secara statistik, "kurang acak" dibandingkan deret yang sedang diuji [7]. Jika $p - value$ sangat kecil (misalnya, di bawah ambang signifikansi $\alpha = 0{,}01$), maka hipotesis nol (bahwa deret tersebut acak) ditolak.

Untuk menguji sekumpulan sequence (misalnya m deret), STS tidak hanya melihat kelulusan individu $p \geq \alpha$), tetapi juga memeriksa apakah distribusi $p - value$ itu sendiri seragam. Jika $p - value$ dari banyak pengujian cenderung menumpuk di nilai rendah (meski masih di atas α), ini bisa menjadi indikasi bias. Keseragaman ini diuji menggunakan statistik *goodness-of-fit* (X^2)[21] dengan persamaan (2.1):

$$x^2 = \sum_{i=1}^{k} \frac{\left(F_i - \frac{m}{k}\right)^2}{\frac{m}{k}},\ P_T = 1 - F_{x^2_{k-1}}(X^2), \quad \text{.....(2.1)}$$

di mana m adalah jumlah sequence, $k = 10$ adalah jumlah bin, F_i adalah jumlah $p - value$ di bin ke-i, dan P_T adalah *p-value of p-values.* Selain itu, proporsi kelulusan ($\hat{p}$) dibandingkan dengan ekspektasi teoritis $p_\alpha = 1 - \alpha$ menggunakan ambang konservatif [21] seperti persamaan (2.2).

$$P_{min} = P_\alpha - 3\sqrt{\frac{p_\alpha(1-p_\alpha)}{m_{valid}}} \quad \text{.....(2.2)}$$

Suatu algoritma dianggap "sehat" secara statistik jika $\hat{p} \geq p_{min}$ dan $P_T \geq 10^{-4}$. Namun, terdapat pengecualian spesifik pada mekanisme penentuan ambang batas ini. Berdasarkan analisis statistik NIST STS [21], uji Random Excursions dan Random Excursions Variant memiliki karakteristik unik di mana ukuran sampel efektif (m') dapat berubah-ubah. Kedua uji ini mempersyaratkan sequence memiliki jumlah siklus (J) melebihi 500. Sequence yang tidak memenuhi syarat ini akan dikeluarkan dari perhitungan (tidak dihitung gagal, tetapi dianggap tidak valid untuk uji tersebut). Akibatnya, ambang batas kelulusan (P_{min}) untuk kedua uji ini tidak statis (seperti 96% pada uji lain), melainkan dihitung secara dinamis menyesuaikan jumlah sampel valid (m') yang tersisa menggunakan pendekatan statistik berikut.

$$\boldsymbol{P_{min} = m'(1 - \alpha) - 3\sqrt{m'\alpha(1-\alpha)}} \quad \text{.....(2.3)}$$

Sebagai implikasinya, jika jumlah sequence yang valid (m') menurun drastis, maka nilai batas lulus absolutnya pun akan menyesuaikan, yang sering kali terlihat lebih rendah secara angka namun tetap valid secara statistik [21].

Ambang Batas Dinamis (Teori Luengo et al.)

Pengecualian teoretis berlaku pada uji *Random Excursions* dan *Random Excursions Variant*. Kedua uji ini memiliki prasyarat di mana evaluasi hanya dapat dilakukan pada *sequence* yang lintasan acaknya (*random walk*) melintasi titik referensi nol sekurang-kurangnya 500 kali ($J > 500$). *Sequence* yang tidak memenuhi syarat ini akan diabaikan.

Berdasarkan studi analisis independensi NIST oleh Luengo et al. [21], kondisi ini menyebabkan jumlah sampel efektif (m') menjadi kurang dari ukuran sampel awal (m). Oleh karena itu, ambang batas kelulusannya dihitung secara dinamis:

$$P_{min\_dinamis} = m'(1-\alpha) - 3\sqrt{m'\alpha(1-\alpha)} \quad \text{.....(2.4)}$$

Keterangan Variabel:

- m' = Jumlah sampel *sequence* aktual yang divalidasi memenuhi prasyarat siklus $J > 500$.

Formula Keseragaman Distribusi P-Value (P_T)

Lulus secara proporsi tidaklah cukup; P-value yang dihasilkan oleh sekumpulan sampel harus terdistribusi secara seragam di sepanjang interval 0.0 hingga 1.0. NIST STS membagi interval ini ke dalam 10 sub-interval (*bin*). Keseragaman distribusi ini dievaluasi secara statistik menggunakan uji *Goodness-of-Fit Chi-Square* (χ^2) dengan rumus:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m/10)^2}{\frac{m}{10}} \quad \text{.....(2.5)}$$

Keterangan Variabel:

- χ^2 = Nilai ukur statistik Chi-Square.

- F_i = Frekuensi observasi (jumlah *sequence* yang nilai *p-value*-nya jatuh pada interval ke-i, di mana $i = 1, 2, \dots, 10$).
- m = Jumlah total *sequence* yang diuji.
- $m/10$ = Frekuensi harapan (*expected frequency*) yang ideal untuk setiap interval.

Nilai χ^2 tersebut kemudian ditransformasikan menjadi nilai metrik $P - value\ of\ P - values\ (P_T)$ menggunakan fungsi *Incomplete Gamma*. Algoritma dikategorikan menghasilkan distribusi yang seragam jika nilai $P_T \geq 0.0001$. Jika $P_T < 0.0001$, berarti *p-value* mengelompok pada titik tertentu, yang merupakan indikasi kuat adanya cacat struktural pada algoritma [21].

2.1.8 Gangguan/*fault* dan pola pada keluaran

Meskipun NIST STS menguji keluaran dalam kondisi operasi normal, penelitian tentang fault analysis menunjukkan relevansi pengujian statistik sebagai alat deteksi anomali. Fault analysis adalah cabang kriptanalisis yang meneliti bagaimana perilaku algoritma berubah ketika terjadi gangguan fisik (misalnya, lonjakan tegangan atau injeksi laser) pada perangkat keras.

Penelitian terdahulu telah menunjukkan bahwa gangguan semacam itu dapat menimbulkan pola statistik yang terukur pada keluaran ciphertext atau tag. Sebagai contoh:

1. Pada ASCON, studi oleh Das & Mazumdar [10] menunjukkan bahwa model persistent faults (di mana fault terjadi secara permanen, misal pada S-box) dapat menimbulkan bias statistik pada keluaran yang berpotensi membocorkan informasi kunci.
2. Pada TinyJAMBU, Salam et al. [4] mendemonstrasikan statistical fault analysis di mana injeksi kesalahan pada state internal dapat menghasilkan struktur yang dapat dideteksi pada ciphertext atau tag.
3. Pada Grain-128AEAD, Fang et al. [17] juga meneliti improved differential fault analysis yang menunjukkan sensitivitas stream cipher terhadap gangguan pada state internalnya.

Kajian-kajian ini memperkuat argumen bahwa keluaran kriptografi yang ideal haruslah acak secara statistik. Jika fault yang tidak disengaja atau disengaja dapat menciptakan pola, maka pengujian statistik seperti NIST STS berfungsi sebagai sanity check yang penting. Uji STS dapat bertindak sebagai indikator dini terhadap anomali implementasi, konfigurasi yang

salah, atau bahkan kondisi operasi di luar batas aman spesifikasi, yang melengkapi analisis keamanan struktural [12].

2.2 Penelitian Terdahulu

Sub-bab ini meninjau penelitian terdahulu yang beririsan langsung dengan empat objek studi utama (ASCON-128, TinyJAMBU-128, Grain-128AEAD, dan AES-128) guna mendudukkan posisi serta kontribusi penelitian ini secara presisi. Secara garis besar, penelitian terdahulu yang mendefinisikan objek-objek tersebut berfokus pada desain, spesifikasi, dan analisis keamanan secara teoretis. Turan et al. [2] memformalkan ASCON-128 sebagai standar Kriptografi Ringan (LWC) dalam dokumen NIST SP 800-232, yang menyediakan parameter operasional dan vektor uji resmi. Sementara itu, Wu et al. [5] dan Hell et al. [9] menyajikan dokumen spesifikasi finalis untuk TinyJAMBU-128 dan Grain-128AEAD, yang merinci arsitektur internal serta batasan operasional masing-masing algoritma.

Kelompok studi lain berfokus pada evaluasi komparatif dari aspek non-statistik. Sebagai contoh, Kaiser & Hoiness [19] membandingkan tingkat throughput ASCON dengan algoritma IoT populer lainnya. Sejalan dengan itu, Rana et al. [15] dan Aziz et al. [16] menganalisis efisiensi dan konsumsi daya cipher ringan, yang menjadi metrik krusial bagi perangkat terbatas. Selanjutnya, kelompok penelitian ketiga secara spesifik menyoroti kriptanalisis fisik, terutama fault analysis. Seperti yang telah dibahas pada sub-bab sebelumnya, studi oleh Das & Mazumdar [10] terhadap ASCON, Salam et al. [4] terhadap TinyJAMBU, dan Fang et al. [17] terhadap Grain-128AEAD mendemonstrasikan bahwa gangguan fisik (physical injection) dapat merusak state algoritma dan menghasilkan anomali statistik.

Untuk memetakan arah riset terkini (2021–2025) secara komprehensif, Tabel 2.1 menyajikan matriks *State of the Art* (SoTA) yang menjadi landasan pijak sekaligus menyoroti celah ilmiah yang akan diisi oleh penelitian ini.

Tabel 2.1 State of The Art (SoTA) dan posisi penelitian

Peneliti (Tahun) & Referensi	**Fokus Utama Penelitian**	**Algoritma yang Dikaji**	**Metode / Instrumen Utama**	**Temuan Utama dan Keterbatasan (*Gap*)**

Turan et al., (2025) [2]; Wu et al., (2021) [5]; Hell et al., (2021) [9]	Spesifikasi Resmi, Desain Arsitektur, & Standar LWC NIST	ASCON, TinyJAMBU, Grain-128AEAD	Analisis Teoretis, Pemaparan *Test Vector*, Batasan Parameter	Mendefinisikan arsitektur internal dan parameter standar algoritma secara matematis. **Gap:** Merupakan dokumen landasan teoretis, tidak menyediakan validasi empiris independen terhadap luaran statistik (*ciphertext*).
Kaiser & Hoiness (2023) [19]; Rana et al., (2024) [15]; Aziz et al., (2024) [16]	Performa Kinerja (*Performance*) & Efisiensi Hardware untuk IoT	ASCON, AES, dan beberapa algoritma *cipher* ringan lainnya	*Benchmarking Throughput*, Pengukuran Konsumsi Daya & RAM	Membuktikan algoritma LWC jauh lebih efisien dari AES pada mikrokontroler (IoT). **Gap:** Fokus murni pada metrik perangkat keras (kecepatan/daya), sama sekali mengabaikan analisis kualitas keamanan statistiknya.
Das & Mazumdar (2024) [10]; Salam et al., (2024) [4]; Fang et al., (2024) [17]	Kriptanalisis Fisik (Daya/Sirkuit) dan *Fault Analysis*	ASCON, TinyJAMBU, Grain-128AEAD	*Statistical Fault Injection*, *Differential Fault Analysis*	Menunjukkan bahwa injeksi fisik (*glitch*) merusak *state* dan memunculkan anomali statistik yang membocorkan kunci.

				Gap: Evaluasi dilakukan pada kondisi algoritma dirusak (anomali fisik), bukan menguji *randomness* pada kondisi operasional normal.
Gebeyehu (2022) [25]; JCSRA Survey AES (2024) [26]	Standar Keamanan Enkripsi Simetris dan Pengembangannya	AES-128 (sebagai standar industri)	Analisis Struktural SPN dan Tinjauan Pustaka Sistematis	Mengukuhkan posisi AES sebagai *Gold Standard* enkripsi global yang sangat *robust*. **Gap:** Tidak mengkomparasikan kualitas entropi AES secara langsung (*head-to-head*) dengan arsitektur LWC era baru.
Penelitian Ini	**Evaluasi Kualitas Kerandoman Statistik Ciphertext (Kondisi Normal)**	**ASCON-128, TinyJAMBU-128, Grain-128AEAD vs AES-128-GCM (*Baseline*)**	**NIST Statistical Test Suite (STS) SP 800-22 Rev 1a**	**Menutup celah dengan menyediakan bukti empiris *apple-to-apple* terkait kemurnian kerandoman *ciphertext* algoritma LWC yang dipertandingkan secara langsung dengan AES-128.**

Celah Penelitian (Research Gap) dan Kontribusi: Sintesis dari tinjauan di atas menunjukkan sebuah celah yang jelas. Penelitian terdahulu terbagi dalam tiga fokus utama:

1. **Desain dan Spesifikasi** [2, 5, 9]: Berfokus pada arsitektur internal, vektor uji, dan definisi standar.
2. **Metrik Performa dan Efisiensi** [15, 16, 19]: Berfokus pada kecepatan (mis. *throughput*), konsumsi daya, dan jejak memori, yang penting untuk perangkat IoT.
3. **Analisis Keamanan di Bawah Gangguan** [4, 10, 17]: Berfokus pada skenario anomali seperti *fault analysis* untuk menguji ketahanan algoritma.

Namun, *validasi statistik keluaran (ciphertext) dalam kondisi operasi normal* sebagai fokus utama evaluasi komparatif masih kurang. Studi performa [19] tidak menjamin kerandoman statistik, dan studi *fault* [4, 10, 17] menganalisis skenario anomali, bukan operasi standar.

Penelitian ini dirancang secara spesifik untuk *melengkapi* (complement) celah tersebut. *Perbaikan* atau *kontribusi* yang dilakukan dalam penelitian ini adalah menyediakan *baseline* empiris yang seragam:

- Penelitian ini tidak mengukur *throughput* seperti Kaiser & Hoiness [19] atau menginjeksi *fault* seperti Salam et al. [4] dan Das & Mazumdar [10].
- Sebaliknya, penelitian ini memvalidasi apakah *ciphertext* yang dihasilkan dalam operasi normal oleh desain-desain LWC terpilih ini [2, 5, 9] menunjukkan kualitas kerandoman statistik yang setara dengan *baseline* mapan (AES-128), menggunakan metodologi standar NIST STS [21].

Kontribusi ini penting sebagai *sanity check* fundamental yang melengkapi analisis performa dan analisis keamanan yang sudah ada. Ini menjawab pertanyaan: "Apakah algoritma yang efisien dan aman secara desain ini juga menghasilkan keluaran yang tidak dapat dibedakan dari deret acak murni dalam kondisi penggunaan normal?"

Untuk perbandingan yang adil, kriteria evaluasi STS yang digunakan konsisten, seperti dibahas di 2.1.7, menggunakan taraf signifikansi $\alpha = 0{,}01$ per uji, pengujian keseragaman P_T (Rumus 2.1), dan ambang proporsi lulus konservatif P_{min} (2.2).

2.3 Kerangka Konseptual

Kerangka ini secara eksplisit dirancang untuk *melengkapi* dan *memperbaiki* pemahaman yang ditinggalkan oleh penelitian sebelumnya. Tinjauan di 2.2 mengidentifikasi bahwa penelitian terdahulu telah berfokus pada:

1. **Aspek Performa (Efisiensi):** Seperti *throughput* [19] serta konsumsi daya dan memori [15, 16].
2. **Aspek Keamanan (Kondisi Anomali):** Seperti ketahanan terhadap *fault analysis* [4, 10, 17].

Kerangka konseptual ini *menjembatani* kedua fokus tersebut dengan menambahkan satu dimensi evaluasi fundamental yang hilang: **Validasi Statistik pada Kondisi Normal**.

Perbaikan atau kelengkapan yang ditawarkan oleh kerangka kerja ini adalah:

1. Penelitian ini *melengkapi* studi performa (seperti Kaiser & Hoiness [19]) dengan menjawab pertanyaan: "Apakah algoritma yang cepat (efisien) itu juga menghasilkan keluaran yang acak secara statistik?" Efisiensi [15, 16] tidak ada artinya jika keluarannya memiliki pola statistik yang dapat dieksploitasi.
2. Penelitian ini *melengkapi* studi *fault analysis* [4, 10, 17] dengan menyediakan *baseline* kondisi normal. Ini krusial untuk membedakan apakah anomali statistik disebabkan oleh *fault* (seperti temuan Salam et al. [4]) atau memang sudah ada sebagai bias inheren dalam algoritma (yang akan dideteksi oleh penelitian ini).

Dengan demikian, kerangka ini menggambarkan kontribusi penelitian: menyediakan evaluasi komparatif yang seragam atas kerandoman *ciphertext* menggunakan NIST STS [21]. Ini adalah sebuah *sanity check* fundamental yang melengkapi analisis desain [2, 12], analisis performa [19], dan analisis *fault* [4, 10, 17] yang sudah ada.

Untuk mengisi celah ini secara operasional, *jenis algoritma* (ASCON-128, TinyJAMBU-128, Grain-128AEAD, AES-128) ditetapkan sebagai variabel pembeda. *Kualitas kerandoman statistik ciphertext*, yang diukur melalui metrik NIST STS (Pass% dan P_T), menjadi variabel terikat.

Hubungan konseptual yang diuji adalah bahwa *perbedaan rancangan* (permutation-based [2, 5] vs. stream-cipher AEAD [9], ukuran state, dsb.) *dapat memengaruhi* pola statistik pada *ciphertext* ketika diuji dengan prosedur yang setara.

Secara operasional, variabel kontrol (input terkontrol, kunci/nonce acak non-reuse, panjang deret bit yang memadai) [lihat Bab 3] digunakan untuk memastikan kesetaraan uji. Artefak yang diuji dibatasi pada *ciphertext murni* untuk merepresentasikan keluaran enkripsi secara adil. Unit observasi adalah *sequence ciphertext* ($\sim 10^6$ bit), dan unit analisis adalah *agregasi per algoritma-per uji* (Pass% dan P_T).

Suatu uji dinilai "sehat" bila Pass% $\geq P_{min}$ (sesuai Rumus 2.2) dan $P_T \geq 10^{-4}$ [21]. Alur konseptualnya adalah: (1) Generasi korpus *ciphertext* terkontrol per algoritma; (2) Pengujian STS; (3) Agregasi hasil (Pass%, P_T); (4) Keputusan "sehat/tidak" per uji; (5) Komparasi lintas algoritma terhadap *baseline* AES-128.

Dengan kerangka ini, *hipotesis kerja* diformulasikan sebagai:

- H_0 : Tidak ada penyimpangan statistik signifikan dari acak (semua algoritma "sehat" menurut kriteria STS) pada kondisi uji yang terkontrol.
- H_1 : Setidaknya ada satu kombinasi algoritma-uji yang menyimpang secara statistik (Pass% $\geq P_{min}$ atau $P_T \geq 10^{-4}$ relatif terhadap *baseline* AES-128 atau sesama LWC.

Interpretasi hasil akhirnya tetap *konservatif*: keluaran "sehat" pada STS tidak membuktikan keamanan menyeluruh, sedangkan keluaran "tidak sehat" menjadi sinyal untuk audit parameter, implementasi, atau potensi anomali yang layak ditindaklanjuti.