NoTIP

NO TESTS IN PRODUCTION

Specifica tecnica - Crypto *SDK*$_G$

Versione	1.0.0
Data Modifica	2026-04-13
Utilizzo	Esterno

Abstract dei contenuti

Specifica tecnica della libreria Crypto-*SDK*$_G$. Architettura logica, design di dettaglio e testing.

Changelog$_G$

Versione	Data	Autori	*Verificatore*$_G$	Descrizione
1.0.0	2026-04-13	Leonardo Preo (responsabile)		Approvazione per ingresso in *baseline*$_G$ PB
0.0.1	2026-04-07	Matteo Mantoan	Francesco Marcon	Creazione del documento

Indice

1. Introduzione 4
2. Configurazione 4
3. Architettura logica 5
   3.1. Pattern architetturale: architettura a strati con Orchestrator 5
   3.2. Struttura dei moduli 5
   3.3. Componenti logiche 6
4. Design di dettaglio 7
   4.1. Gerarchia degli errori (`errors.ts`) 7
   4.2. `CryptoSdk`: Orchestrator (`crypto-sdk`$_G$`.ts`) 7
      4.2.1. Interfacce 7
      4.2.2. Campi 7
      4.2.3. Metodi pubblici 8
      4.2.4. Metodo privato: `decryptEnvelope` 8
   4.3. `DataApiService`: Adapter (`data-api.service.ts`) 8
   4.4. `CryptoEngine` (`crypto-engine.ts`) 8
      4.4.1. Metodo `decrypt` 8
   4.5. `KeyManager` (`key-manager.ts`) 9
      4.5.1. Pattern: cache-aside 9
   4.6. `DataApiRestClient` (`data-api-rest`$_G$`.client.ts`) 9
   4.7. `DataApiSseClient` (`data-api-sse`$_G$`.client.ts`) 9
      4.7.1. Pattern: channel ($producer_G$-$consumer_G$) 9
   4.8. `ManagementApiClient` (`management-api.client.ts`) 10
   4.9. `ManagementApiService` (`management-api.service.ts`) 10
   4.10. `authorizedFetch` (`http.ts`) 10
   4.11. Modelli di dominio (`models.ts`) 10
      4.11.1. Modelli di input (query) 10
      4.11.2. Modelli di output 11
      4.11.3. DTO_G (da schemi OpenAPI generati, definiti in `dto`$_G$`.ts`) 11
   4.12. Decisioni implementative 12
5. Metodologie di testing 14
   5.1. Test di unità 14
   5.2. Test di integrazione 16

1. Introduzione

Questo documento descrive l'architettura interna e le scelte implementative della libreria `@notip/crypto-sdk`$_G$. Sviluppata in *TypeScript*$_G$, questa libreria client-side ha lo scopo di nascondere la complessità dell'Opaque *Pipeline*$_G$ agli utilizzatori delle API NoTIP: gestisce il recupero delle chiavi crittografiche, la decrittazione AES-*GCM*$_G$ delle envelope e la validazione dei *payload*$_G$, esponendo un'interfaccia ad alto livello che restituisce direttamente misure in chiaro.

La libreria è distribuita come pacchetto npm (ESM + CJS) e utilizzata sia dal *frontend*$_G$ interno sia dai consumatori delle API pubbliche. Dipende esclusivamente da API web standard (Fetch API, *Web Crypto API*$_G$, `TextDecoder`, `AbortController`), garantendo la portabilità su tutti i runtime *JavaScript*$_G$ moderni: browser, *Node.js*$_G$ (>= 18), Deno e Bun.

Per l'esatta struttura dei *payload*$_G$ e i contratti delle interfacce, il codice sorgente costituisce la Single Source of Truth.

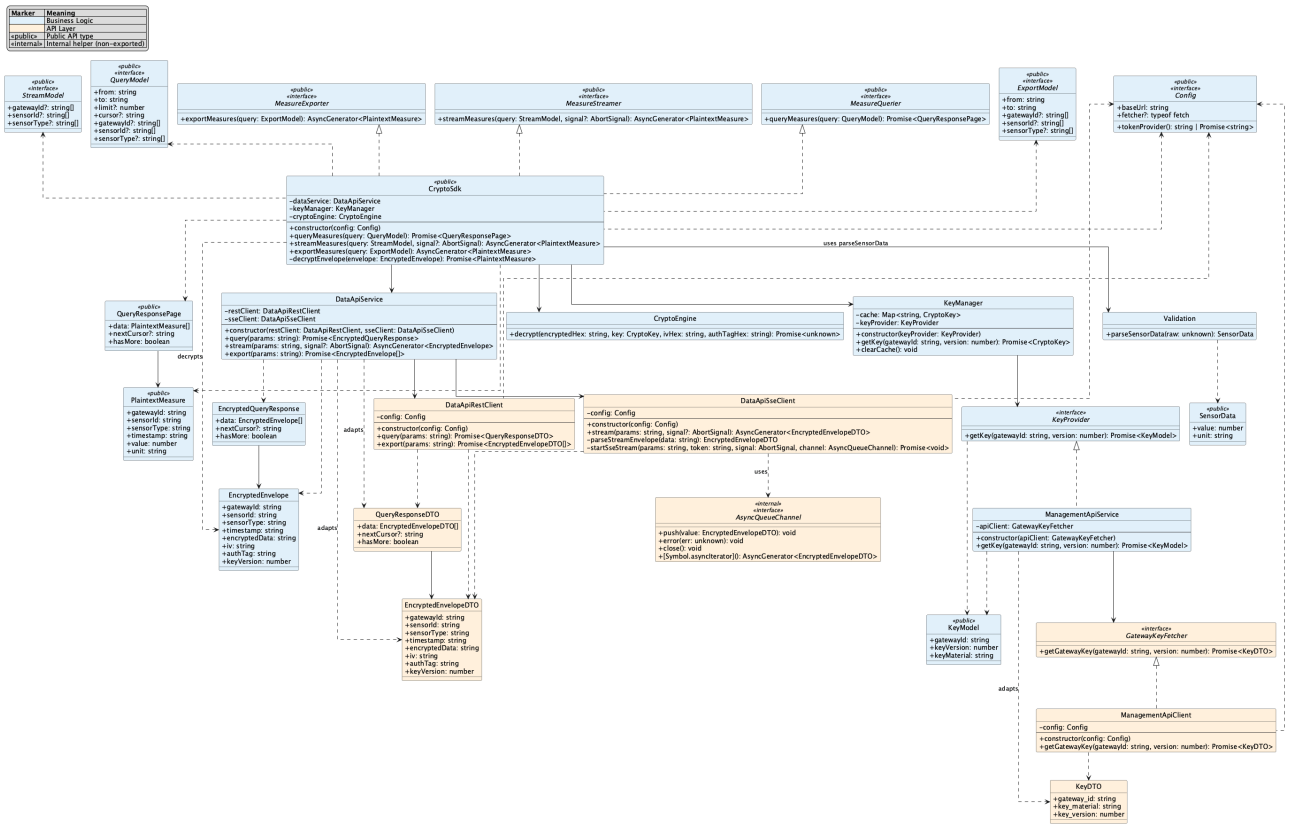
2. Configurazione

L'*SDK*$_G$ è configurato tramite un oggetto `Config` passato al costruttore di `CryptoSdk`.

Campo	Tipo	Default	Obbligatorio
`baseUrl`	`string`		Sì
`tokenProvider`	`() => string | Promise<string>`		Sì
`fetcher`	`typeof fetch`	`globalThis.fetch`	No

- `baseUrl`: URL base dell'istanza NoTIP (es. `https`$_G$`://api.notip.example.com`);
- `tokenProvider`: Funzione che restituisce (sincrona o asincrona) il *Bearer token*$_G$ per l'autenticazione. Permette l'integrazione con qualsiasi meccanismo di autenticazione (*Keycloak*$_G$, *OAuth2*$_G$, token statico);
- `fetcher`: Implementazione di `fetch`. Utile per ambienti non-browser (*Node.js*$_G$ < 18) o per iniettare mock nei test.

3. Architettura logica



3.1. Pattern architetturale: architettura a strati con Orchestrator

La libreria adotta un'architettura a strati, divisa in Business Logic e Api Clients, con il pattern **Orchestrator**. `CryptoSdk` coordina un workflow multi-step (conversione parametri → fetch dati cifrati → risoluzione chiavi → decrittazione → validazione → mapping a modelli di dominio).

I *consumer*$_G$ dipendono da tre interfacce ristrette (`MeasureQuerier`, `MeasureStreamer`, `MeasureExporter`) in applicazione del principio di Interface Segregation, e non dalla classe concreta `CryptoSdk`.

3.2. Struttura dei moduli

```
src/
├── index.ts                  API pubblica (re-export selettivo)
├── config.ts                 Interfaccia Config
├── crypto-sdk_G.ts           Orchestrator: wiring, conversione parametri, pipeline_G di
decrittazione
├── errors.ts                 Gerarchia errori (SdkError, ApiError, ValidationError,
DecryptionError)
├── http.ts                   authorizedFetch, HTTP helper con autenticazione
├── models.ts                 Modelli di dominio (plain TypeScript_G interfaces, no Zod)
├── dto_G.ts                  DTO_G type alias inferiti dagli schemi Zod auto-generati
├── validation.ts             Funzioni di validazione Zod con signature clean (es.
parseSensorData)
├── key-provider.ts           Interfaccia interna KeyProvider
├── crypto-engine.ts          Motore decrittazione AES-GCM_G
├── key-manager.ts            Caching delle CryptoKey (dipende da KeyProvider)
├── data-api-rest_G.client.ts  Client REST_G (query, export)
├── data-api-sse_G.client.ts   Client SSE_G (stream real-time)
```

```
├── data-api.service.ts       Confine DTO_G → modello di dominio; interfaccia unificata su
REST_G + SSE_G client
├── management-api.client.ts  Client Management API (chiavi), implementa GatewayKeyFetcher
├── management-api.service.ts Implementa KeyProvider, mapping DTO_G → KeyModel
└── generated/                Schemi Zod auto-generati da OpenAPI
    ├── notip-data-api-openapi.ts
    └── notip-management-api-openapi.ts
```

3.3. Componenti logiche

Strato	Moduli	Contenuto
API Pubblica	`index.ts`	Re-export selettivo: `CryptoSdk`, le tre interfacce ISP (`MeasureQuerier`, `MeasureStreamer`, `MeasureExporter`), `Config`, i modelli di dominio e le classi di errore. `KeyProvider` è esclusa dall'esportazione pubblica.
Orchestrator	`crypto-sdk`$_G$`.ts`	Costruisce il grafo di dipendenze da `Config`. Converte i modelli di query in parametri, orchestra la *pipeline*$_G$ fetch → decrypt → validate → map.
Servizi (Adapter)	`data-api.service.ts` `management-api.service.ts`	`DataApiService`: confine *DTO*$_G$ → modello di dominio per i dati cifrati; restituisce `EncryptedQueryResponse` e `EncryptedEnvelope`. `ManagementApiService`: implementa `KeyProvider`, mappa `KeyDTO` → `KeyModel`.

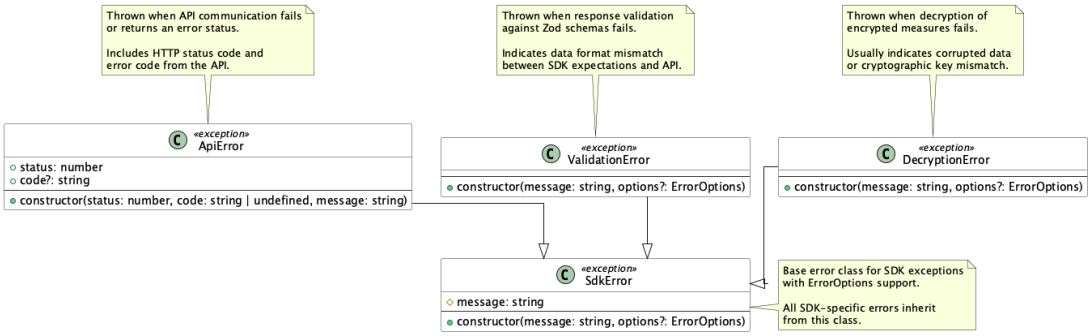
Client API	`data-api-rest`$_G$`.client.ts`
	`data-api-sse`$_G$`.client.ts`
	`management-api.client.ts`
		Comunicazione HTTP verso le API NoTIP. Validazione delle risposte con schemi Zod generati. Restituiscono *DTO*$_G$ al layer Servizio.

Infrastruttura	`http.ts`
	`crypto-engine.ts`
	`key-manager.ts`
		Helper HTTP con autenticazione, decrittazione AES-*GCM*$_G$ via *Web Crypto API*$_G$, caching delle chiavi.

Modelli	`models.ts`
	`dto`$_G$`.ts`
	`validation.ts`
	`key-provider.ts`
	`config.ts`
	`errors.ts`
	`generated/*`
		`models.ts`: plain *TypeScript*$_G$ interfaces per i modelli di dominio (no Zod). `dto`$_G$`.ts`: *DTO*$_G$ type alias, importati solo da Client e Service. `validation.ts`: funzioni di validazione Zod con signature clean. `key-provider.ts`: interfaccia interna `KeyProvider`. Nessuna logica di business.

4. Design di dettaglio

4.1. Gerarchia degli errori (`errors.ts`)



Tutti gli errori dell'SDK_G estendono `SdkError`, che a sua volta estende `Error` nativo. Supportano `ErrorOptions` (proprietà `cause`) per il chaining.

Classe	Semantica
`SdkError`	Errore base. Utilizzato per errori generici (es. chiave non trovata, errore canale SSE_G).
`ApiError`	Errore HTTP dalle API NoTIP. Espone `status: number` e `code: string | undefined` estratti dal body della risposta.
`ValidationError`	Validazione Zod fallita. Indica un mismatch tra lo schema atteso e la risposta ricevuta dall'API.
`DecryptionError`	Decrittazione AES-GCM_G fallita. Tipicamente indica dati corrotti o mismatch di chiave.

4.2. `CryptoSdk`: Orchestrator (`crypto-sdk_G.ts`)

Punto di ingresso dell'SDK_G. Il costruttore riceve un `Config` e costruisce l'intero grafo di dipendenze: `ManagementApiClient` ⟶ `ManagementApiService` ⟶ `KeyManager`, `DataApiRestClient` + `DataApiSseClient` ⟶ `DataApiService`, `CryptoEngine`.

Implementa le tre interfacce ISP: `MeasureQuerier`, `MeasureStreamer`, `MeasureExporter`.

4.2.1. Interfacce

Interfaccia	Metodo
`MeasureQuerier`	`queryMeasures(query: QueryModel): Promise<QueryResponsePage>`
`MeasureStreamer`	`streamMeasures(query: StreamModel, signal?: AbortSignal): AsyncGenerator<PlaintextMeasure>`
`MeasureExporter`	`exportMeasures(query: ExportModel): AsyncGenerator<PlaintextMeasure>`

4.2.2. Campi

Campo	Tipo	Note
`dataService`	`DataApiService`	Adapter unificato su $REST_G$ e SSE_G
`keyManager`	`KeyManager`	Caching e import delle chiavi crittografiche

`cryptoEngine`	`CryptoEngine`	Decrittazione AES-*GCM*$_G$

4.2.3. Metodi pubblici

Metodo	Responsabilità
`queryMeasures(query)`	Converte `QueryModel` in search params, delega a `DataApiService.query`, decifra ogni envelope e restituisce una pagina di misure in chiaro con supporto a cursore.
`streamMeasures(query, signal?)`	Converte `StreamModel` in search params, delega a `DataApiService.stream`, produce le misure decifrate una alla volta. La connessione *SSE*$_G$ è mantenuta aperta per tutta la durata del generatore. Per rilasciarla: uscire dal loop `for await...of` o invocare `abort()` sull'`AbortSignal`.
`exportMeasures(query)`	Converte `ExportModel` in search params, delega a `DataApiService.export`, produce le misure decifrate.

4.2.4. Metodo privato: `decryptEnvelope`

Pipeline$_G$ di decrittazione di una singola `EncryptedEnvelope`:

1. Recupera la `CryptoKey` dal `KeyManager` (cache hit o fetch tramite `KeyProvider` + import);
2. Invoca `CryptoEngine.decrypt` con ciphertext, *IV*$_G$ e auth tag (tutti in formato hex);
3. Valida il *payload*$_G$ decifrato tramite `parseSensorData()` da `validation.ts` (validazione di dominio, non di *DTO*$_G$);
4. Compone e restituisce un `PlaintextMeasure` combinando i *metadati*$_G$ dell'envelope con i dati decifrati.

4.3. `DataApiService`: Adapter (`data-api.service.ts`)

Costituisce il confine tra il layer Client (che restituisce *DTO*$_G$) e il layer Business Logic (che opera su modelli di dominio). Riceve `DataApiRestClient` e `DataApiSseClient` via constructor injection ed espone le envelope cifrate come modelli di dominio. Non decifra, non converte modelli di query in parametri.

Metodo	Responsabilità
`query(params: string): Promise<EncryptedQueryResponse>`	Delega a `DataApiRestClient.query`.
`stream(params, signal?): AsyncGenerator<EncryptedEnvelope>`	Delega a `DataApiSseClient.stream`.
`export(params: string): Promise<EncryptedEnvelope[]>`	Delega a `DataApiRestClient.export`.

4.4. `CryptoEngine` (`crypto-engine.ts`)

Classe stateless per la decrittazione AES-*GCM*$_G$ tramite *Web Crypto API*$_G$.

4.4.1. Metodo `decrypt`

```
async decrypt(
  encryptedHex: string, key: CryptoKey,
```

```
  ivHex: string, authTagHex: string
): Promise<unknown>
```

1. Converte ciphertext, *IV*$_G$ e auth tag da hex a `Uint8Array`;
2. Concatena ciphertext e auth tag (`ciphertext || authTag`) come richiesto da Web Crypto per AES-*GCM*$_G$;
3. Invoca `crypto.subtle.decrypt` con algoritmo `AES-GCM`$_G$;
4. Decodifica il *buffer*$_G$ risultante come UTF-8 e interpreta il *JSON*$_G$;
5. Qualsiasi errore è incapsulato in `DecryptionError`.

La funzione helper `hexToBytes` è privata al modulo.

4.5. `KeyManager` (`key-manager.ts`)

Gestisce il fetching e il caching delle chiavi crittografiche.

4.5.1. Pattern: cache-aside

Le `CryptoKey` sono memorizzate in una `Map<string, CryptoKey>` con chiave composita `"{gatewayId}-{version}"`. Al primo accesso per una coppia `(gatewayId, version)`, il `KeyModel` viene recuperato tramite l'interfaccia `KeyProvider` (iniettata nel costruttore), importato come `CryptoKey` AES-*GCM*$_G$ (decrypt-only) tramite `crypto.subtle.importKey`, e inserito in cache. Le chiamate successive restituiscono la chiave dalla cache senza ulteriori round-trip di rete. `KeyManager` dipende solo dal modello di dominio (`KeyModel`), non dai *DTO*$_G$ del transport layer.

Metodo	Responsabilità
`getKey(gatewayId, version): Promise<CryptoKey>`	Restituisce la `CryptoKey` dalla cache o la recupera, importa e memorizza.
`clearCache(): void`	Invalida l'intera cache delle chiavi.

La funzione helper `base64ToBytes` è privata al modulo e converte il materiale chiave da base64 a `Uint8Array`.

4.6. `DataApiRestClient` (`data-api-rest`$_G$`.client.ts`)

Client HTTP per gli *endpoint*$_G$ *REST*$_G$ della Data API.

Metodo	Responsabilità
`query(params: string): Promise<QueryResponseDTO>`	GET su `/data/measures/query?{params}`. Valida la risposta con `zQueryResponseDto`.
`export(params: string): Promise<EncryptedEnvelopeDTO[]>`	GET su `/data/measures/export?{params}`. Valida la risposta con `zMeasureControllerExportResponse`.

Entrambi i metodi delegano la chiamata HTTP ad `authorizedFetch` e lanciano `ValidationError` se la risposta non supera la validazione Zod.

4.7. `DataApiSseClient` (`data-api-sse`$_G$`.client.ts`)

Client *SSE*$_G$ per lo streaming real-time delle misure cifrate.

4.7.1. Pattern: channel (*producer*$_G$*-consumer*$_G$)

L'API `fetchEventSource` è callback-based. Per esporla come `AsyncGenerator`, il client usa un channel interno con coda: la callback `onmessage` inserisce le envelope validate nella coda

(*producer*$_G$), il generatore le consuma (*consumer*$_G$). La coda gestisce anche gli errori (`onerror` → `channel.error`) e la chiusura (`onclose` → `channel.close`).

```
async *stream(params: string, signal?: AbortSignal):
  AsyncGenerator<EncryptedEnvelopeDTO>
```

- Crea un `AbortController` interno collegato all'eventuale `signal` esterno;
- Apre la connessione *SSE*$_G$ con `fetchEventSource`;
- Ogni evento *SSE*$_G$ ricevuto è validato con `zEncryptedEnvelopeDto` prima di essere inserito nella coda;
- Il blocco `finally` del generatore invoca `abort()` e attende la terminazione del fetch, garantendo il cleanup della connessione.

4.8. `ManagementApiClient` (`management-api.client.ts`)

Client HTTP per l'*endpoint*$_G$ chiavi del Management API. Implementa l'interfaccia `GatewayKeyFetcher` per consentire la dependency injection nel servizio che lo consuma:

Interfaccia	Metodo
`GatewayKeyFetcher`	`getGatewayKey(gatewayId: string, version: number): Promise<KeyDTO>`

Metodo	Responsabilità
`getGatewayKey(gatewayId, version)`	GET su `/mgmt/keys?id={gatewayId}`. Filtra per versione. Lancia `SdkError` se la versione non è trovata.

4.9. `ManagementApiService` (`management-api.service.ts`)

Implementa l'interfaccia `KeyProvider` (definita in `key-provider.ts`). Riceve `GatewayKeyFetcher` via constructor injection e mappa i *DTO*$_G$ (`KeyDTO`, formato snake_case) in modelli di dominio (`KeyModel`, formato camelCase). La responsabilità del mapping *DTO*$_G$ → modello è concentrata in questo servizio.

Metodo	Responsabilità
`getKey(gatewayId, version): Promise<KeyModel>`	Recupera una chiave specifica tramite `GatewayKeyFetcher` e la mappa in `KeyModel`.

4.10. `authorizedFetch` (`http.ts`)

Funzione helper che decora `fetch` aggiungendo l'header `Authorization: Bearer {token}`. In caso di risposta non-ok, tenta di leggere il body *JSON*$_G$ per estrarre `code` e `message`, e lancia un `ApiError`.

4.11. Modelli di dominio (`models.ts`)

4.11.1. Modelli di input (query)

Tipo	Campi
`QueryModel`	`from: string`, `to: string`, `limit?: number`, `cursor?: string`, `gatewayId?: string[]`, `sensorId?: string[]`, `sensorType?: string[]`

`StreamModel`	`gatewayId?: string[]`, `sensorId?: string[]`, `sensorType?: string[]`
`ExportModel`	`from: string`, `to: string`, `gatewayId?: string[]`, `sensorId?: string[]`, `sensorType?: string[]`

4.11.2. Modelli di output

Tipo	Descrizione e campi
`EncryptedEnvelope`	Envelope cifrata restituita dal layer Servizio. Campi: `gatewayId`, `sensorId`, `sensorType`, `timestamp`, `encryptedData`, iv_G, `authTag` (tutti `string`), `keyVersion: number`.
`EncryptedQueryResponse`	Risposta paginata di envelope cifrate. Campi: `data: EncryptedEnvelope[]`, `nextCursor?: string`, `hasMore: boolean`.
`PlaintextMeasure`	Misura decifrata. Campi: `gatewayId`, `sensorId`, `sensorType`, `timestamp` (tutti `string`), `value: number`, `unit: string`.
`QueryResponsePage`	Pagina di risultati paginata. Campi: `data: PlaintextMeasure[]`, `nextCursor?: string`, `hasMore: boolean`.
`SensorData`	$Payload_G$ decifrato grezzo. Campi: `value: number`, `unit: string`. Validato tramite `parseSensorData()` da `validation.ts`.
`KeyModel`	Chiave crittografica. Campi: `gatewayId: string`, `keyVersion: number`, `keyMaterial: string`.

4.11.3. DTO_G (da schemi OpenAPI generati, definiti in `dto_G.ts`)

I DTO_G sono type alias inferiti dagli schemi Zod auto-generati da `@hey-api/openapi-ts`. Sono definiti in `dto_G.ts` e importati esclusivamente dai Client e dai Service. Il layer Business Logic non li vede:

Tipo	Schema sorgente
`EncryptedEnvelopeDTO`	`zEncryptedEnvelopeDto`. Campi: `gatewayId`, `sensorId`, `sensorType`, `timestamp`, `encryptedData`, iv_G, `authTag` (tutti `string`), `keyVersion: number`.
`QueryResponseDTO`	`zQueryResponseDto`. Campi: `data: EncryptedEnvelopeDTO[]`, `nextCursor?: string`, `hasMore: boolean`.
`KeyDTO`	`zKeysResponseDto`. Campi: `gateway_id`, `key_material` (entrambi `string`), `key_version: number`.

4.12. Decisioni implementative

▸ **Validazione Zod a ogni boundary**

Tutte le risposte HTTP e tutti i *payload*$_G$ *SSE*$_G$ sono validati con schemi Zod prima dell'uso. Anche il *payload*$_G$ decifrato (`SensorData`) è validato dopo la decrittazione. Questo garantisce fail-fast in caso di incompatibilità con le API, con errori tipizzati (`ValidationError`) che indicano l'esatta posizione del mismatch.

▸ **Schemi OpenAPI generati**

I file in `generated/` sono prodotti da `@hey-api/openapi-ts` a partire dalle specifiche OpenAPI dei servizi Data API e Management API. Lo script `fetch-dtos` rigenera i file allineandoli all'ultima versione delle API. Questo elimina il drift manuale tra *SDK*$_G$ e servizi.

▸ **Token provider come funzione**

`Config.tokenProvider` è una funzione `() => string | Promise<string>` anziché un token statico. Consente al *consumer*$_G$ (*frontend*$_G$ o servizio esterno) di gestire autonomamente il refresh dei token *Keycloak*$_G$/*OAuth2*$_G$ senza che l'*SDK*$_G$ debba conoscere il meccanismo di autenticazione.

▸ **Bring Your Own Fetch (BYOF)**

`Config.fetcher` consente di sostituire `globalThis.fetch` con un'implementazione custom, applicando il principio di inversione delle dipendenze alla primitiva HTTP. Questo mantiene la compatibilità con framework che forniscono una propria implementazione di `fetch` (es. Next.js, Nuxt) e permette l'iniezione di mock nei test per ottenere esecuzioni deterministiche senza server HTTP reale.

▸ **AsyncGenerator per stream e export**

`streamMeasures` e `exportMeasures` restituiscono `AsyncGenerator<PlaintextMeasure>` anziché array. Per lo stream *SSE*$_G$, questo è l'unico approccio possibile (flusso potenzialmente infinito). Per l'export, il generatore consente l'elaborazione incrementale senza caricare tutte le misure in memoria simultaneamente.

▸ **Channel con coda per bridge callback → AsyncGenerator**

`DataApiSseClient` usa un channel interno (coda + Promise) per convertire le callback di `fetchEventSource` (`onmessage`, `onclose`, `onerror`) in un `AsyncGenerator`. Il pattern evita busy-waiting: il *consumer*$_G$ si sospende su una Promise quando la coda è vuota, e viene risvegliato dalla callback del *producer*$_G$.

▸ **Interface Segregation sulla superficie pubblica**

I *consumer*$_G$ che utilizzano solo la query non devono dipendere dall'infrastruttura di streaming e viceversa. Tre interfacce ristrette (`MeasureQuerier`, `MeasureStreamer`, `MeasureExporter`)

consentono a ogni *consumer*$_G$ di dipendere esclusivamente dalla capability che utilizza. `CryptoSdk` implementa tutte e tre.

- **Service come confine *DTO*$_G$/modello**

I services come `DataApiService` costituiscono il confine formale tra il layer API Client (*DTO*$_G$) e il layer Business Logic (modelli di dominio). Riceve `DataApiRestClient` e `DataApiSseClient` via constructor injection e dichiara i tipi di ritorno come modelli di dominio (`EncryptedQueryResponse`, `EncryptedEnvelope`), sfruttando la compatibilità strutturale tra *DTO*$_G$ e modello di dominio (stessi campi, tipi diversi nel sistema di tipi).

- **Cache-aside per le CryptoKey**

`KeyManager` mantiene una `Map<string, CryptoKey>` che evita di ripetere sia la chiamata al `KeyProvider` sia l'operazione `crypto.subtle.importKey` (costosa) per le chiavi già utilizzate. La chiave di cache è la stringa composita `"{gatewayId}-{version}"`. La cache non ha *TTL*$_G$: le chiavi crittografiche sono immutabili per versione.

- ***Web Crypto API*$_G$ per la decrittazione**

La decrittazione usa esclusivamente `crypto.subtle` (*Web Crypto API*$_G$), disponibile nativamente in tutti i browser moderni e in *Node.js*$_G$ $\geq$ 15. Questo elimina la dipendenza da librerie crittografiche di terze parti e garantisce che le operazioni crittografiche siano eseguite in modo sicuro dal runtime.

5. Metodologie di testing

I test usano Vitest con coverage V8. Le dipendenze esterne (HTTP, *SSE*$_G$) sono sostituite tramite il campo `Config.fetcher` (mock) o `vi.mock` (per `fetchEventSource`). Le operazioni crittografiche usano la *Web Crypto API*$_G$ reale per garantire la correttezza end-to-end della *pipeline*$_G$ di cifratura/decifratura.

5.1. Test di unità

`CryptoEngine`

Caso di test	Postcondizione verificata
Decrittazione AES-*GCM*$_G$ di *JSON*$_G$ cifrato	Il *payload*$_G$ decifrato corrisponde all'originale
Chiave errata	Lancio di `DecryptionError`
Auth tag manomesso	Lancio di `DecryptionError`
Oggetto *JSON*$_G$ vuoto	Decrittazione corretta; risultato `{}`

`KeyManager`

Caso di test	Postcondizione verificata
Cache miss, prima richiesta per `(gatewayId, version)`	`KeyProvider.getKey` invocato; `CryptoKey` restituita con algoritmo AES-*GCM*$_G$ e usage `decrypt`
Cache hit, seconda richiesta stessa coppia	Stessa istanza `CryptoKey` restituita; `KeyProvider.getKey` non richiamato
Versioni diverse per stesso *gateway*$_G$	Due `CryptoKey` distinte; `KeyProvider.getKey` invocato due volte
`clearCache`: invalidazione cache	Dopo `clearCache`, la richiesta successiva causa un nuovo fetch

`DataApiRestClient`

Caso di test	Postcondizione verificata
`query`: risposta valida	URL e header `Authorization` corretti; risposta validata
`query`: risposta HTTP non-ok	Lancio di `ApiError`
`query`: *DTO*$_G$ invalido	Lancio di `ValidationError`
`export`: risposta valida	Array di `EncryptedEnvelopeDTO` restituito; URL corretto
`export`: *DTO*$_G$ invalido	Lancio di `ValidationError`

`DataApiSseClient`

Caso di test	Postcondizione verificata
Stream con due envelope valide	Generatore produce entrambe le envelope nell'ordine corretto

URL e header corretti	`fetchEventSource` invocato con URL completo, header `Authorization` e fetcher custom
Errore *SSE*$_G$	Lancio di `SdkError` con messaggio «*SSE*$_G$ stream error»
Data line vuota	Linea ignorata; solo le envelope valide sono prodotte
DTO$_G$ envelope invalido	Lancio di `ValidationError`
Abort signal esterno	Signal propagato all'`AbortController` interno; connessione terminata

`ManagementApiClient`

Caso di test	Postcondizione verificata
`getGatewayKey`: versione trovata	`KeyDTO` corretto restituito; URL con query param `id`
`getGatewayKey`: versione non trovata	Lancio di `SdkError`
`getGatewayKey`: token provider asincrono	Token asincrono risolto e inserito nell'header

`ManagementApiService`

Caso di test	Postcondizione verificata
`getKey`: fetch e mapping singola chiave	`GatewayKeyFetcher.getGatewayKey` invocato con parametri corretti; `KeyModel` restituito con campi camelCase
`getKey`: propagazione errori dal client	Errore `SdkError` propagato al chiamante

`DataApiService`

Le dipendenze (`DataApiRestClient`, `DataApiSseClient`) sono sostituite da mock iniettati nel costruttore. I test verificano la pura delega senza trasformazione.

Caso di test	Postcondizione verificata
`query`: delega al *rest*$_G$ client	Risposta restituita invariata; parametri propagati
`stream`: delega al *SSE*$_G$ client	Envelope prodotte dal generatore; signal propagato
`export`: delega al *rest*$_G$ client	Array di envelope restituito invariato

`CryptoSdk` (test di unità con *pipeline*$_G$ completa)

Caso di test	Postcondizione verificata

`queryMeasures`: decrittazione end-to-end	Misure in chiaro corrispondono al $payload_G$ originale cifrato; paginazione (`hasMore`, `nextCursor`) esposta
`queryMeasures`: risposta query invalida	Lancio di `ValidationError`
`queryMeasures`: $payload_G$ decifrato invalido	Lancio di `ValidationError` (post-decrittazione)
`queryMeasures`: paginazione con `nextCursor` e `hasMore`	Cursore e flag propagati nella risposta
`exportMeasures`: decrittazione end-to-end	Generatore produce misure decifrate; valori corrispondono all'originale
`streamMeasures`: decrittazione end-to-end via SSE_G	Generatore produce misure decifrate dallo stream SSE_G simulato
`streamMeasures`: propagazione `AbortSignal`	`AbortSignal` passato al client SSE_G sottostante

5.2. Test di integrazione

I test di integrazione (`tests/integration.spec.ts`) esercitano `CryptoSdk` come black-box con un fetcher mock basato su routing URL. Le operazioni crittografiche sono reali (*Web Crypto API*$_G$): ogni test cifra un $payload_G$ con AES-GCM_G e verifica che l'SDK_G lo decifri correttamente.

Caso di test	**Postcondizione verificata**
Key caching: stessa coppia `(gatewayId, version)`	Chiave recuperata una sola volta per tre envelope con stessa versione
Key caching: versioni diverse dello stesso $gateway_G$	Fetch separati per ogni versione; entrambe le misure decifrate correttamente
Key caching: $gateway_G$ indipendenti	Cache separata per $gateway_G$; un fetch per ciascuno
Token provider asincrono	Header `Authorization` contiene il token risolto dalla Promise
`queryMeasures`: pagina singola	Dati decifrati; `hasMore` false; `nextCursor` assente
`queryMeasures`: paginazione	`hasMore` true; `nextCursor` propagato
`exportMeasures`: ordine preservato	Le misure sono prodotte nell'ordine originale
`streamMeasures`: decrittazione e caching	Misure decifrate; chiave recuperata una sola volta
Errore: versione chiave inesistente	Lancio di `SdkError`
Errore: chiave errata per envelope	Lancio di `DecryptionError`