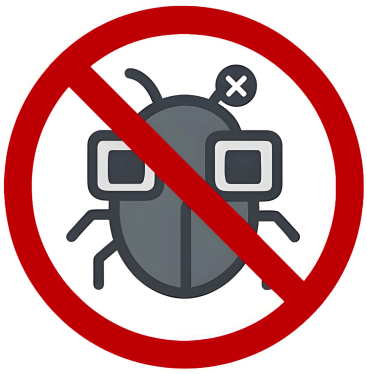

NoTIP

NO TESTS IN PRODUCTION

Manuale client API

Versione	1.0.0
Data Modifica	2026-04-13
Utilizzo	Esterno

Abstract dei contenuti

Manuale tecnico degli endpoint API esposti dalla piattaforma NoTIP, destinato a sviluppatori e integratori esterni.

Changelog

Versione	Data	Autori	Verificatore	Descrizione
1.0.0	2026-04-13	Leonardo Preo (responsabile)		Approvazione per ingresso in baseline PB
0.3.0	2026-04-13	Alessandro Mazzariol	Francesco Marcon	Aggiornamento di endpoint errati e aggiunta conclusione del documento
0.2.0	2026-04-12	Alessandro Mazzariol	Francesco Marcon	Aggiunta della sezione dettagliata sull'integrazione con la CryptoSdk, esempi di codice TypeScript e gestione degli errori
0.1.1	2026-04-11	Alessandro Mazzariol	Alessandro Contarini	Documentazione degli endpoint Management API (Gateways, Users, Alerts, Thresholds, Commands, Costs e Audit)
0.1.0	2026-04-10	Alessandro Mazzariol	Valerio Solito	Prima stesura delle sezioni Prerequisiti (autenticazione JWT) e Data API (Measures e Sensors)
0.0.1	2026-04-09	Alessandro Mazzariol	Valerio Solito	Creazione del documento e impostazione del template

Indice

Indice delle tabelle

1. Introduzione

Il presente documento costituisce il manuale tecnico della piattaforma NoTIP dedicato alle funzionalità esposte tramite le interfacce API. L'obiettivo è descrivere in modo chiaro e puntuale le modalità di utilizzo e integrazione degli *endpoint*$_G$ API messi a disposizione dalla piattaforma. I destinatari del presente manuale sono principalmente sviluppatori e amministratori di infrastruttura che intendono integrare i servizi offerti da NoTIP nelle proprie applicazioni o gestire l'infrastruttura del *Tenant*$_G$.

Il documento è organizzato come segue: la sezione **Prerequisiti** illustra i passaggi necessari per ottenere l'accesso autenticato alle API; la sezione ***Endpoint*$_G$ API** documenta nel dettaglio ogni *endpoint*$_G$ disponibile, organizzato per area funzionale; la sezione **Integrazione con la CryptoSdk** descrive l'*SDK*$_G$ ufficiale fornito da NoTIP per l'accesso semplificato alle misure cifrate.

1.1. Glossario

I termini tecnici rilevanti per la comprensione del manuale sono definiti nella sezione **Glossario** a fondo documento; nel testo tali termini sono contrassegnati con pedice *G*. Per il glossario completo del progetto, si rimanda al Glossario v3.0.0.

2. Prerequisiti

- Ottenere le credenziali di accesso: per poter utilizzare le API è necessario richiedere le credenziali di accesso a un amministratore del *Tenant*$_G$, il quale deve aver previamente registrato l'applicazione Client nel *Tenant*$_G$ di riferimento. Il Client può essere registrato in un unico *Tenant*$_G$ e i dati accessibili sono circoscritti all'ambiente di quel *Tenant*$_G$.
- Ottenere un token *JWT*$_G$ di accesso: una volta ottenute le credenziali di accesso, è necessario effettuare una richiesta HTTP POST al server di autenticazione (*Keycloak*$_G$) per ottenere un token *JWT*$_G$ valido.

Operazione	Risposta

`POST http://localhost/auth/realms/notip/protocol/openid-connect/token`
`Content-Type: application/x-www-form-urlencoded`
`grant_type=client_credentials &`
`client_id={client_id} &`
`client_secret={client_secret}`
	`{`
	`"access_token":`
	`"eyJhbGciOiJSUzI1NiIsInR5cCI...",`
	`"expires_in": 900,`
	`"refresh_expires_in": 0,`
	`"token_type": "Bearer",`
	`"not-before-policy": 0,`
	`"scope": "notip-claims notip-roles"`
	`}`

Tabella 1: Operazione POST /token

dove `{client_id}` e `{client_secret}` sono rispettivamente l'ID e il segreto del Client registrato nel *Tenant*$_G$. Il token ha una durata di 900 secondi (15 minuti), come indicato dal campo `expires_in` nella risposta. Alla scadenza è necessario ripetere la richiesta di autenticazione per ottenere un nuovo token: le richieste effettuate con un token scaduto vengono rifiutate con codice HTTP `401 Unauthorized`.

3. *Endpoint*$_G$ API

Una volta ottenuto un token *JWT*$_G$ valido, esso deve essere incluso nell'header `Authorization` di ogni richiesta API nel seguente formato: `Authorization: Bearer {access_token}`

Gli *endpoint*$_G$ sono suddivisi in due famiglie logiche, identificabili dal prefisso del percorso:
- **Data API** (prefisso `api/data/`): espone gli *endpoint*$_G$ per accedere alle misure rilevate dai sensori e ai metadati dei sensori registrati nel *Tenant*$_G$.
- **Management API** (prefisso `api/mgmt/`): espone gli *endpoint*$_G$ per la gestione dell'infrastruttura del *Tenant*$_G$, inclusi *gateway*$_G$, utenti, client API, alert, soglie di allarme, comandi remoti, costi e log di audit.

I campi opzionali nei parametri e nelle risposte sono indicati con il suffisso `?` nel nome del campo (es. `"field?": "type"`).

3.0.1. Enums

I tipi enumerati riportati di seguito definiscono i valori ammessi per specifici campi dei parametri e delle risposte negli *endpoint*$_G$ del Management API. Ogni volta che la documentazione di un *endpoint*$_G$ indica un tipo enumerato come valore atteso (es. `"status": "GatewayStatus"`), i valori accettabili sono esclusivamente quelli elencati nella tabella seguente.

Enum	Valori
UsersRole	**`system_admin`**, **`tenant_admin`**, **`tenant_user`**
GatewayStatus	**`gateway_online`**, **`gateway_offline`**, **`gateway_suspended`**
AlertType	**`gateway_offline`**
CommandType	**`config`**, **`firmware`**, **`suspend`**
CommandStatus	**`queued`**, **`ack`**, **`nack`**, **`expired`**, **`timeout`**

Tabella 2: Enumerazioni principali del microservizio `notip-management-api`

3.0.2. Measures

Gli *endpoint*$_G$ del gruppo Measures consentono di accedere alle misure rilevate dai sensori del *Tenant*$_G$. Le misure vengono restituite in forma cifrata (*AES-256*$_G$-GCM): per accedere ai valori in chiaro è necessario decifrare ogni envelope utilizzando la chiave AES corrispondente alla coppia `(gatewayId, keyVersion)`. L'*SDK*$_G$ `@notip/crypto-sdk`$_G$ automatizza questo processo; in alternativa, la decrittografia può essere implementata manualmente dal chiamante.

3.0.2.1. `GET api/data/measures/query`

Campo	Valore
Descrizione	Restituisce una query paginata delle misure cifrate filtrabili per intervallo temporale, *gateway*$_G$, sensore e tipo di sensore con finestra temporale massima di 24h.
Query Parameters	`{ "from": "string", "to": "string", "limit?": "number", "gatewayId?": "string", "sensorId?": "string", "sensorType?": "string", "cursor?": "string" }`
Body Request	-
Response	`{ "data": [{ "gatewayId": "string", "sensorId": "string",`

	"sensorType": "string", "timestamp": "string", "encryptedData": "string", "iv": "string", "authTag": "string", "keyVersion": "number", }], "nextCursor?": "string", "hasMore": "boolean" }

3.0.2.2. `GET api/data/measures/export`

Campo	Valore
Descrizione	Restituisce l'export completo in formato CSV delle misure cifrate in un intervallo temporale, senza paginazione, con finestra temporale massima di 24h.

Query Parameters	{
	"from": "string",
	"to": "string",
	"gatewayId?": "string",
	"sensorId?": "string",
	"sensorType?": "string"
	}

Body Request	-

Response	{"data":
	[{
	"gatewayId": "string",
	"sensorId": "string",
	"sensorType": "string",
	"timestamp": "string",
	"encryptedData": "string",
	"iv": "string",
	"authTag": "string",
	"keyVersion": "number"
	}]
	}

3.0.2.3. `GET api/data/measures/stream`

Campo	Valore
Descrizione	Espone uno stream Server-Sent Events di misure cifrate filtrabile per *gateway*$_G$, sensore e tipo di sensore.

Query Parameters	{
	"gatewayId?": "string",
	"sensorId?": "string",
	"sensorType?": "string"
	}

Body Request	-

Response	`text/event-stream`:
	{
	"gatewayId": "string",
	"sensorId": "string",
	"sensorType": "string",
	"timestamp": "string",
	"encryptedData": "string",
	"iv": "string",
	"authTag": "string",
	"keyVersion": "number"
	}

3.0.3. Sensors

L'*endpoint*$_G$ del gruppo Sensors consente di consultare i sensori registrati nel *Tenant*$_G$. Ogni sensore è identificato univocamente da un `sensor_id`, è associato a un *gateway*$_G$ tramite `gateway_id` e classificato per tipologia tramite `sensor_type`. Queste informazioni sono utili per filtrare le misure negli *endpoint*$_G$ del gruppo Measures, dove `sensorId` e `sensorType` possono essere impiegati come parametri di query.

3.0.3.1. `GET api/data/sensors`

Campo	Valore
Descrizione	Restituisce i sensori associati al *Tenant*$_G$ di appartenenza.
Query Parameters	-
Body Request	-
Response	`[{ "sensorId": "string", "sensorType": "string", "gatewayId": "string", "lastSeen": "string" }]`

3.0.4. Gateways

Gli *endpoint*$_G$ del gruppo Gateways consentono di visualizzare e gestire i *gateway*$_G$ fisici registrati nel *Tenant*$_G$. Un *gateway*$_G$ può trovarsi in uno dei tre stati definiti dall'enum `GatewayStatus` (`gateway_online`, `gateway_offline`, `gateway_suspended`) e può ricevere comandi remoti tramite gli *endpoint*$_G$ del gruppo Commands. Il flag `provisioned` indica se il *gateway*$_G$ ha completato la procedura di *provisioning*$_G$ iniziale.

3.0.4.1. `GET api/mgmt/gateways`

Campo	Valore
Descrizione	Restituisce i *Gateway*$_G$ del *Tenant*$_G$ di appartenenza.
Query Parameters	-
Body Request	-
Response	`[{ "id": "string", "name": "string", "status": "GatewayStatus", "last_seen_at": "string", "provisioned": "boolean", "firmware_version": "string", "send_frequency_ms": "number" }]`

3.0.4.2. `GET api/mgmt/gateways/`

Campo	Valore
Descrizione	Restituisce il dettaglio di un *Gateway*$_G$ specifico.
Query Parameters	`{ "id": "string" }`
Body Request	-
Response	`{ "id": "string", "name": "string",`

	<pre>{
	  "status": "GatewayStatus",
	  "last_seen_at": "string",
	  "provisioned": "boolean",
	  "firmware_version": "string",
	  "send_frequency_ms": "number"
	}</pre>

3.0.4.3. `PATCH api/mgmt/gateways/`

Campo	Valore
Descrizione	Aggiorna il nome di un *Gateway*$_G$ specifico.
Query Parameters	<pre>{ "id": "string" }</pre>
Body Request	<pre>{ "name": "string" }</pre>

Response	<pre>{
	  "id": "string",
	  "name": "string",
	  "status": "GatewayStatus",
	  "updated_at": "string"
	}</pre>

3.0.4.4. `DELETE api/mgmt/gateways/`

Campo	Valore
Descrizione	Elimina un *Gateway*$_G$ specifico.
Query Parameters	<pre>{ "id": "string" }</pre>
Body Request	-
Response	<pre>{ "status": 200 }</pre>

3.0.5. Users

Gli *endpoint*$_G$ del gruppo Users consentono di gestire gli utenti del *Tenant*$_G$. Ogni utente è associato a un ruolo (`UsersRole`) che determina i propri permessi di accesso alla piattaforma: `tenant_admin` dispone di privilegi completi sulla gestione del *Tenant*$_G$, mentre `tenant_user` ha accesso in sola lettura alle risorse. La creazione di un utente avviene contestualmente all'assegnazione del ruolo e delle credenziali di accesso.

3.0.5.1. `GET api/mgmt/users`

Campo	Valore
Descrizione	Restituisce gli utenti del *Tenant*$_G$ di appartenenza.
Query Parameters	-
Body Request	-

Response	<pre>[{
	  "id": "string",
	  "name": "string",
	  "email": "string",
	  "role": "UsersRole",
	  "last_access": "string"
	}]</pre>

3.0.5.2. `GET api/mgmt/users/`

Campo	Valore
Descrizione	Restituisce il dettaglio di un utente specifico.

Query Parameters	`{ "id": "string" }`
Body Request	-
Response	`{ "id": "string", "name": "string", "email": "string", "role": "UsersRole", "last_access": "string" }`

3.0.5.3. `POST api/mgmt/users`

Campo	Valore
Descrizione	Crea un utente nel *Tenant*$_G$.
Query Parameters	-
Body Request	`{ "name": "string", "email": "string", "role": "UsersRole", "password": "string" }`
Response	`{ "id": "string", "name": "string", "email": "string", "role": "UsersRole", "created_at": "string" }`

3.0.5.4. `PATCH api/mgmt/users/`

Campo	Valore
Descrizione	Aggiorna un utente del *Tenant*$_G$ specifico.
Query Parameters	`{ "id": "string" }`
Body Request	`{ "name": "string", "email": "string", "role": "UsersRole", "permissions": ["string"] }`
Response	`{ "id": "string", "name": "string", "email": "string", "role": "UsersRole", "updated_at": "string" }`

3.0.6. API Clients

Gli *endpoint*$_G$ del gruppo API Clients consentono di gestire i client applicativi registrati nel *Tenant*$_G$. Un client API è identificato da un `client_id` e da un `client_secret`, impiegati per ottenere token di accesso tramite il flusso `client_credentials` descritto nella sezione Prerequisiti. Si noti che il `client_secret` viene restituito esclusivamente al momento della creazione del client e non è recuperabile successivamente: in caso di smarrimento è necessario eliminare il client e ricrearne uno nuovo.

3.0.6.1. `GET api/mgmt/api-clients`

Campo	Valore
Descrizione	Restituisce i client API del *Tenant*$_G$.
Query Parameters	-
Body Request	-
Response	`[{ "id": "string", "name": "string", "client_id": "string", "created_at": "string" }]`

3.0.6.2. `POST api/mgmt/api-clients`

Campo	Valore
Descrizione	Crea un nuovo client API.
Query Parameters	-
Body Request	`{ "name": "string" }`
Response	`{ "id": "string", "name": "string", "client_id": "string", "client_secret": "string", "created_at": "string" }`

3.0.6.3. `DELETE api/mgmt/api-clients/`

Campo	Valore
Descrizione	Elimina un client API.
Query Parameters	`{ "id": "string" }`
Body Request	-
Response	`{ "status": 200 }`

3.0.7. Alerts

Gli *endpoint*$_G$ del gruppo Alerts consentono di consultare gli alert generati dal sistema e di configurare i timeout di irraggiungibilità dei *gateway*$_G$. Quando un *gateway*$_G$ non invia dati entro il timeout configurato, il sistema genera un alert di tipo `gateway_offline`. La configurazione può essere definita a livello di *Tenant*$_G$ come valore di default e sovrascritta per singolo *gateway*$_G$ tramite override specifici; in assenza di un override, si applica il timeout di default del *Tenant*$_G$.

3.0.7.1. `GET api/mgmt/alerts/config`

Campo	Valore
Descrizione	Restituisce la configurazione alert di default.
Query Parameters	-
Body Request	-
Response	`{ "default_timeout_ms": "integer", "gateway_overrides": [{`

	<pre>{
	  "gateway_id": "string",
	  "timeout_ms": "integer"
	}]
	}</pre>

3.0.7.2. `GET api/mgmt/alerts`

Campo	Valore
Descrizione	Restituisce gli alert del *Tenant*$_G$ nel range richiesto.

Query Parameters	<pre>{
	  "from": "string",
	  "to": "string",
	  "gateway_id?": "string"
	}</pre>

Body Request	-

Response	<pre>[{
	  "id": "string",
	  "gateway_id": "string",
	  "type": "AlertType",
	  "details": {
	  "last_seen": "string",
	  "timeout_configured": "number"
	  },
	  "created_at": "string"
	}]</pre>

3.0.7.3. `PUT api/mgmt/alerts/config/default`

Campo	Valore
Descrizione	Imposta la configurazione alert di default.
Query Parameters	-
Body Request	<pre>{ "tenant_unreachable_timeout_ms": "number" }</pre>

Response	<pre>{
	  "tenant_id": "string",
	  "timeout_ms": "number",
	  "updated_at": "string"
	}</pre>

3.0.7.4. `PUT api/mgmt/alerts/config/gateway`$_G$`/`

Campo	Valore
Descrizione	Imposta la configurazione alert specifica per un *Gateway*$_G$.
Query Parameters	<pre>{ "gatewayId": "string" }</pre>
Body Request	<pre>{ "gateway_unreachable_timeout_ms": "number" }</pre>

Response	<pre>{
	  "gateway_id": "string",
	  "timeout_ms": "number",
	  "updated_at": "string"
	}</pre>

3.0.7.5. `DELETE api/mgmt/alerts/config/gateway`$_G$`/`

Campo	Valore
Descrizione	Elimina la configurazione alert specifica per un *Gateway*$_G$, tornando a utilizzare la configurazione di default.
Query Parameters	<pre>{ "gatewayId": "string" }</pre>

Body Request	-

Response	{
	"status": 200
	}

3.0.8. Thresholds

Gli *endpoint*$_G$ del gruppo Thresholds consentono di definire i range di valori accettabili per le misure dei sensori. È possibile impostare una soglia di default per ciascuna tipologia di sensore (`sensor_type`) e sovrascriverla per un sensore specifico tramite il relativo `sensor_id`. Quando una misura supera i valori `min_value` o `max_value` configurati, il sistema la considera anomala. In assenza di un override per sensore, si applica la soglia di default della tipologia corrispondente.

3.0.8.1. `GET api/mgmt/thresholds`

Campo	Valore
Descrizione	Restituisce le soglie del *Tenant*$_G$.
Query Parameters	-
Body Request	-
Response	[{ "sensor_type?": "string", "sensor_id?": "string", "min_value": "number", "max_value": "number", "updated_at": "string" }]

3.0.8.2. `PUT api/mgmt/thresholds/default`

Campo	Valore
Descrizione	Imposta la soglia di default per tipologia.
Query Parameters	-

Body Request	{
	"sensor_type": "string",
	"min_value": "number",
	"max_value": "number"
	}

Response	{
	"sensor_type": "string",
	"min_value": "number",
	"max_value": "number",
	"updated_at": "string"
	}

3.0.8.3. `PUT api/mgmt/thresholds/sensor/`

Campo	Valore
Descrizione	Imposta o aggiorna la soglia per uno specifico sensore (override della soglia di default).

Query Parameters	{
	"sensorId": "string"
	}

Body Request	{ "min_value": "number", "max_value": "number",

	<pre>{
	  "sensor_type": "string"
	}</pre>

Response	<pre>{
	  "sensor_id": "string",
	  "min_value": "number",
	  "max_value": "number",
	  "updated_at": "string"
	}</pre>

3.0.8.4. `DELETE api/mgmt/thresholds/sensor/`

Campo	Valore
Descrizione	Elimina la soglia specifica di un sensore, ripristinando quella di default per tipologia.

Query Parameters	<pre>{
	  "sensorId": "string"
	}</pre>

Body Request	-
Response	<pre>{ "status": 200 }</pre>

3.0.8.5. `DELETE api/mgmt/thresholds/type/`

Campo	Valore
Descrizione	Elimina la soglia di default per una intera tipologia di sensori.

Query Parameters	<pre>{
	  "sensorType": "string"
	}</pre>

Body Request	-
Response	<pre>{ "status": 200 }</pre>

3.0.9. Commands

Gli *endpoint*$_G$ del gruppo Commands consentono di inviare comandi remoti ai *gateway*$_G$. I comandi vengono accodati lato server e consegnati al *gateway*$_G$ alla prima opportunità di connessione disponibile. Lo stato di avanzamento di ogni comando è rappresentato dall'enum `CommandStatus` (`queued`, `ack`, `nack`, `expired`, `timeout`) e può essere interrogato tramite l'apposito *endpoint*$_G$ di stato, fornendo il `command_id` restituito al momento dell'invio.

3.0.9.1. `POST api/mgmt/cmd/:gatewayId/config`

Campo	Valore
Descrizione	Invia una configurazione a un *Gateway*$_G$.

Query Parameters	<pre>{
	  "gatewayId": "string"
	}</pre>

Body Request	<pre>{
	  "send_frequency_ms?": "number",
	  "status?": "string"
	}</pre>

Response	<pre>{
	  "command_id": "string",
	  "status": "queued",
	  "issued_at": "string"
	}</pre>

3.0.9.2. `POST api/mgmt/cmd/:gatewayId/firmware`

Campo	Valore
Descrizione	Invia un comando di aggiornamento firmware a uno specifico *Gateway*$_G$.
Query Parameters	`{ "gatewayId": "string" }`
Body Request	`{ "firmware_version": "string", "download_url": "string" }`
Response	`{ "command_id": "string", "status": "queued", "issued_at": "string" }`

3.0.9.3. `GET api/mgmt/cmd/:gatewayId/status/:commandId`

Campo	Valore
Descrizione	Restituisce lo stato corrente di esecuzione di un comando specifico.
Query Parameters	`{ "gatewayId": "string", "commandId": "string" }`
Body Request	-
Response	`{ "command_id": "string", "status": "CommandStatus", "timestamp": "string" }`

3.0.10. Costs

L'*endpoint*$_G$ del gruppo Costs espone un riepilogo dell'utilizzo corrente delle risorse del *Tenant*$_G$, in termini di spazio di archiviazione occupato (`storage_gb`) e banda trasmessa (`bandwidth_gb`). Questi valori possono essere utilizzati per monitorare i consumi e pianificare la capacità infrastrutturale del *Tenant*$_G$.

3.0.10.1. `GET api/mgmt/costs`

Campo	Valore
Descrizione	Restituisce i costi correnti del *Tenant*$_G$.
Query Parameters	-
Body Request	-
Response	`{ "storage_gb": "number", "bandwidth_gb": "number" }`

3.0.11. Audit

L'*endpoint*$_G$ del gruppo Audit restituisce il registro cronologico delle operazioni effettuate dagli utenti del *Tenant*$_G$. Ogni voce riporta l'utente che ha eseguito l'azione, la risorsa interessata, i dettagli dell'operazione e il relativo timestamp. Il range temporale è obbligatorio:

i parametri `from` e `to` delimitano la finestra di interrogazione, mentre `userId` e `action` permettono di restringere ulteriormente i risultati a specifici utenti o tipologie di operazione.

3.0.11.1. `GET api/mgmt/audit`

Campo	Valore
Descrizione	Restituisce i log di audit del *tenant*$_G$ nel range richiesto. I parametri `from` e `to` sono obbligatori, `userId` e `action` opzionali.
Query Parameters	`{ "from": "string", "to": "string", "userId?": "string", "action?": "string" }`
Body Request	-
Response	`[{ "id": "string", "user_id": "string", "action": "string", "resource": "string", "details": {}, "timestamp": "string" }]`

4. Integrazione con la CryptoSdk

NoTIP fornisce un *SDK*$_G$ dedicato, `@notip/crypto-sdk`$_G$, che automatizza l'intero flusso di recupero, decrittografia e restituzione delle misure cifrate. L'*SDK*$_G$ incapsula le chiamate ai due *endpoint*$_G$ API (Data API e Management API) e gestisce internamente la decrittografia *AES-256*$_G$-GCM tramite le Web Crypto API, esponendo allo sviluppatore esclusivamente le misure in chiaro.

L'utilizzo dell'*SDK*$_G$ è consigliato in tutti gli scenari in cui l'obiettivo principale è consumare le misure dei sensori, poiché elimina la necessità di gestire manualmente la crittografia, il ciclo di vita delle chiavi di decrittografia e la validazione del payload. Per le operazioni di gestione dell'infrastruttura — *gateway*$_G$, utenti, alert, soglie, comandi — rimane necessario effettuare chiamate dirette agli *endpoint*$_G$ del Management API, in quanto tali funzionalità non sono esposte dall'*SDK*$_G$.

4.0.1. Installazione

L'*SDK*$_G$ è pubblicato su npm e supporta sia il formato ES Module (ESM) che CommonJS (CJS), rendendolo compatibile con i principali ambienti JavaScript moderni. Per installarlo è sufficiente eseguire:

```
npm install @notip/crypto-sdk_G
```

Per il corretto funzionamento sono richieste le seguenti dipendenze runtime:

- `zod` (v4.x) — utilizzata per la validazione degli schemi dei payload decifrati
- `@microsoft/fetch-event-source` — utilizzata per la gestione delle connessioni Server-Sent Events
- Un runtime che esponga `globalThis.crypto.subtle` (browser moderni, Node.js 16+, Deno, Cloudflare Workers)

4.0.2. Configurazione

La costruzione di un'istanza `CryptoSdk` richiede il passaggio di un oggetto di configurazione di tipo `Config`. I tre campi disponibili sono descritti nella tabella seguente. In particolare, `tokenProvider` viene invocata automaticamente prima di ogni chiamata API: è responsabilità del chiamante garantire che la funzione restituisca sempre un token valido, gestendo internamente il rinnovo alla scadenza del token corrente.

Campo	Descrizione	Obbligatorio
`baseUrl`	URL base dell'istanza NoTIP (es. `"http://localhost"`)	Sì
`tokenProvider`	Funzione (sync o async) che restituisce un *JWT*$_G$ valido. Viene invocata automaticamente ad ogni chiamata API.	Sì
`fetcher`	Implementazione custom di `fetch`. Se omesso, viene usato `globalThis.fetch`.	No

```ts
import { CryptoSdk } from "@notip/crypto-sdk_G";

const tokenProvider = async (): Promise<string> => {
  const resp = await fetch(
    "http://localhost/auth/realms/notip/protocol/openid-connect/token",
    {
      method: "POST",
      headers: { "Content-Type": "application/x-www-form-urlencoded" },
      body: new URLSearchParams({
        grant_type: "client_credentials",
        client_id: "my-client-id",
        client_secret: "my-client-secret",
      }),
    }
  );
  const json = await resp.json();
  return json.access_token;
};

const sdk_G = new CryptoSdk({
  baseUrl: "http://localhost",
  tokenProvider,
});
```

Codice 1: Esempio di configurazione base

4.0.3. Query: misure storiche paginate

Il metodo `queryMeasures` esegue una chiamata `GET /data/measures/query`, decifra ogni envelope ricevuto e restituisce una pagina di misure in chiaro con supporto alla paginazione tramite cursore. L'intervallo temporale definito dai parametri `from` e `to` non può superare 24 ore; per recuperare dati su finestre temporali più ampie è necessario effettuare chiamate successive con intervalli adiacenti. La paginazione basata su cursore consente di gestire volumi elevati di misure in modo efficiente: quando il campo `hasMore` della risposta è `true`, il valore di `nextCursor` deve essere passato alla chiamata successiva per ottenere la pagina seguente.

```
const page = await sdkG.queryMeasures({
  from: "2026-04-01T00:00:00.000Z",
  to: "2026-04-01T23:59:59.999Z",
  limit: 100,
  gatewayId: ["gw-001"],
  sensorType: ["temperature"],
});

// page: { data: PlaintextMeasure[], nextCursor?: string, hasMore: boolean }
for (const m of page.data) {
  console.log(m.timestamp, m.value, m.unit);
}

// Paginazione: se page.hasMore === true, usare page.nextCursor
const nextPage = await sdkG.queryMeasures({
  from: "2026-04-01T00:00:00.000Z",
  to: "2026-04-01T23:59:59.999Z",
  cursor: page.nextCursor,
});
```

Codice 2: Esempio di query

4.0.4. Stream: misure real-time via *SSE*$_G$

Il metodo `streamMeasures` apre una connessione Server-Sent Events (*SSE*$_G$) persistente verso `GET /data/measures/stream` e restituisce un `AsyncGenerator<PlaintextMeasure>`. Ogni misura ricevuta dallo stream viene decifrata al volo prima di essere ceduta al chiamante. Questo metodo è indicato per scenari in cui è necessario ricevere le misure in tempo reale nel momento in cui vengono prodotte dai sensori, senza ricorrere a polling periodico sull'*endpoint*$_G$ di query. La connessione rimane aperta fino a quando non viene invocato il metodo `abort()` sull'`AbortController` fornito come secondo argomento.

```
const controller = new AbortController();

for await (const measure of sdkG.streamMeasures(
  { gatewayId: ["gw-001"] },
  controller.signal
)) {
  console.log(measure.gatewayId, measure.sensorId, measure.value);
}

// Per chiudere lo stream:
controller.abort();
```

Codice 3: Esempio di streaming

4.0.5. Export: dump completo senza paginazione

Il metodo `exportMeasures` esegue una chiamata `GET /data/measures/export` e restituisce un `AsyncGenerator` che produce la totalità delle misure decifrate nell'intervallo temporale richiesto, senza paginazione. A differenza di `queryMeasures`, questo metodo è progettato per ottenere un dump completo dei dati in un unico flusso continuo, adatto a operazioni di analisi batch o di esportazione verso sistemi esterni. Anche per questo metodo si applica il vincolo della finestra temporale massima di 24 ore.

```
for await (const measure of sdkG.exportMeasures({
  from: "2026-04-01T00:00:00.000Z",
  to: "2026-04-02T00:00:00.000Z",
})) {
  console.log(measure.timestamp, measure.value);
}
```

Codice 4: Esempio di export

4.0.6. Modello dati: PlaintextMeasure

Tutti e tre i metodi (`queryMeasures`, `streamMeasures`, `exportMeasures`) restituiscono oggetti di tipo `PlaintextMeasure`, ovvero misure già decifrate e validate dallo schema atteso. I campi `value` e `unit` provengono dal payload JSON contenuto all'interno di ogni envelope cifrato, mentre i restanti campi sono metadati trasmessi dall'API senza cifratura.

Campo	Tipo	Descrizione
`gatewayId`	`string`	ID del *gateway*$_G$ che ha inviato la misura
`sensorId`	`string`	ID del sensore origine
`sensorType`	`string`	Tipologia del sensore (es. «temperature», «humidity»)
`timestamp`	`string`	Timestamp ISO 8601 della misurazione
`value`	`number`	Valore numerico decifrato
`unit`	`string`	Unità di misura (es. «°C», «%»)

4.0.7. Gestione delle chiavi di decrittografia

Le misure sono cifrate con l'algoritmo *AES-256*$_G$-GCM e ogni *gateway*$_G$ può disporre di più versioni di chiave nel tempo, identificate dal campo `keyVersion` presente in ogni envelope. La `CryptoSdk` gestisce automaticamente il ciclo di vita di queste chiavi senza richiedere alcun intervento da parte del chiamante. Il comportamento adottato è il seguente:

- **Recupero on-demand**: al primo incontro di un envelope con una coppia `(gatewayId, keyVersion)` sconosciuta, l'*SDK*$_G$ chiama internamente `GET /mgmt/keys?id={gatewayId}` per ottenere la chiave corrispondente.
- **Cache in-memory**: le chiavi importate come `CryptoKey` vengono memorizzate in una `Map` interna, identificate da `"gatewayId-keyVersion"`. Le chiamate successive per lo stesso *gateway*$_G$/versione non generano traffico di rete.
- **Sicurezza**: le chiavi vengono importate tramite `crypto.subtle.importKey("raw", ...)` come AES-GCM, `extractable: false`, con uso consentito a solo `["decrypt"]`. I byte grezzi della chiave vengono scartati dopo l'import — rimangono solo i gestori `CryptoKey` non esportabili.
- **Nessuna persistenza su disco**: le chiavi esistono solo nella memoria dell'istanza `CryptoSdk`. Alla distruzione dell'istanza, la cache viene persa.

Non è richiesto alcun intervento manuale per la gestione delle chiavi: l'*SDK*$_G$ le recupera, importa e memorizza in totale autonomia, garantendo che la cache venga popolata esclusivamente con chiavi non esportabili e utilizzabili soltanto per operazioni di decrittografia.

4.0.8. Gestione degli errori

L'*SDK*$_G$ espone una gerarchia di errori derivati dalla classe base `SdkError`, che consente una gestione granulare dei fallimenti nelle diverse fasi del flusso: chiamata HTTP, decrittografia e validazione del payload. Si raccomanda di intercettare le classi specifiche anziché la sola

classe base, poiché ciascuna fornisce informazioni diagnostiche distinte utili per il debug e per determinare la strategia di recupero più appropriata. Le classi disponibili sono le seguenti:

Classe	Quando viene lanciata
`ApiError`	Errore HTTP: risposta non 2xx da Data API o Management API. Contiene `status` (codice HTTP) e `code`.
`ValidationError`	Il payload decifrato non rispetta lo schema atteso (`{ value: number, unit: string }`).
`DecryptionError`	Fallimento della decrittografia AES-GCM: chiave errata, IV corrotto o auth tag non valido.
`SdkError`	Errore generico della *SDK*$_G$, base della gerarchia.

```
import {
  CryptoSdk,
  ApiError,
  ValidationError,
  DecryptionError,
} from "@notip/crypto-sdkG";

try {
  const page = await sdkG.queryMeasures({
    from: "2026-04-01T00:00:00.000Z",
    to: "2026-04-01T23:59:59.999Z",
  });
} catch (err) {
  if (err instanceof ApiError) {
    // Es: token scaduto (401), gatewayG non trovato (404)
    console.error(`HTTP ${err.status}: ${err.message}`);
  } else if (err instanceof ValidationError) {
    // Il dato decifrato non è un JSON valido o non rispetta lo schema
    console.error("Dati sensore non validi:", err.message);
  } else if (err instanceof DecryptionError) {
    // Problema crittografico: chiave sbagliata o manomissione
    console.error("Decrittografia fallita:", err.message);
  } else {
    console.error("Errore sconosciuto:", err);
  }
}
```

Codice 5: Esempio di gestione errori

4.0.9. Architettura interna della CryptoSdk

La `CryptoSdk` è composta da sei componenti interni che collaborano per gestire in modo trasparente il recupero, la decrittografia e la validazione delle misure. La tabella seguente descrive il ruolo di ciascun componente; il diagramma successivo illustra il flusso dettagliato di una chiamata `queryMeasures`, comprensivo della risoluzione delle chiavi e della validazione del payload decifrato.

Componente	Ruolo
`CryptoSdk`	Orchestratore principale: espone i tre metodi pubblici e coordina i componenti interni
`DataApiService`	Facade che delega a `DataApiRestClient` (query/export HTTP) e `DataApiSseClient` (streaming SSE_G)
`ManagementApiClient`	Client HTTP per `GET /mgmt/keys?id={gatewayId}` — recupera le chiavi AES del $gateway_G$
`ManagementApiService`	Adatta il DTO restituito da `ManagementApiClient` al modello `KeyModel` richiesto da `KeyManager`
`KeyManager`	Gestisce cache in-memory + import delle chiavi come `CryptoKey` non estraibili
`CryptoEngine`	Esegue la decrittografia $AES\text{-}256_G$-GCM via `crypto.subtle.decrypt`

Tabella 3: Flusso completo di una chiamata `queryMeasures`

Il flusso completo di una chiamata `queryMeasures` è il seguente:

```
queryMeasures({ from, to, gatewayId, ... })
    |
    ▼
DataApiService.query(params)
    |
    └─► GET /data/measures/query?...  (Bearer token_G da tokenProvider)
        |
        └─► Response: { data: EncryptedEnvelopeDTO[], ... }
            |
            └─► Per ogni envelope: decryptEnvelope(envelope)
                |
                ├─► KeyManager.getKey(gatewayId, keyVersion)
                |       |
                |       ├── cache hit → CryptoKey cached
                |       └── cache miss → GET /mgmt/keys?id=...
                |                           |
                |                           └─► importKey("raw", ...) → AES-GCM
                |
                ├─► CryptoEngine.decrypt(encryptedData, key, iv, authTag)
                |       |
                |       └─► ciphertext || authTag → crypto.subtle.decrypt
                |               |
                |               └─► JSON.parse → { value, unit }
                |
                └─► zSensorData.safeParse(result)
                        |
                        ├─► VALID → PlaintextMeasure
                        └─► INVALID → ValidationError
```

4.0.10. Quando usare la CryptoSdk rispetto alle chiamate API dirette

La scelta tra l'utilizzo della `CryptoSdk` e le chiamate dirette agli $endpoint_G$ dipende dallo scenario applicativo. La tabella seguente riassume le differenze principali per le funzionalità più rilevanti.

Scenario	Usare CryptoSdk	Chiamate API dirette

Decrittografia automatica	Sì — gestita internamente con cache chiavi	No — va implementata manualmente *AES-256*$_G$-GCM + gestione chiavi
Validazione dati	Sì — tramite Zod schema	No — a carico del chiamante
Streaming *SSE*$_G$	Sì — AsyncGenerator con AbortSignal	Sì — ma va gestito il parsing *SSE*$_G$ manualmente
Flessibilità *endpoint*$_G$	Limitata ai 3 metodi esposti (query, stream, export)	Totale — accesso a tutti gli *endpoint*$_G$ Management API
Gestione infrastrutture	No — la *SDK*$_G$ si focalizza solo sulle misure	Sì — gateways, users, alerts, thresholds, commands, costs, audit

In sintesi: se l'obiettivo è **leggere e decifrare misure** dai *gateway*$_G$, la `CryptoSdk` è la scelta consigliata, in quanto elimina la necessità di gestire manualmente crittografia, chiavi e validazione. Se invece il requisito riguarda la **gestione dell'infrastruttura** del *Tenant*$_G$ (*gateway*$_G$, utenti, alert, comandi), è necessario effettuare chiamate dirette agli *endpoint*$_G$ del Management API descritti nelle sezioni precedenti, poiché tali funzionalità non sono esposte dall'*SDK*$_G$.

5. Considerazioni finali

Il presente manuale costituisce una guida di riferimento agli *endpoint*$_G$ che NoTIP espone ai client esterni per accedere ai dati di un *Tenant*$_G$ e gestire l'infrastruttura associata. Si precisa che il presente documento non ha l'obiettivo di illustrare le modalità di integrazione tecnica degli *endpoint*$_G$ in sistemi terzi, bensì di descrivere con precisione il contratto e il comportamento atteso di ciascun *endpoint*$_G$ esposto.

Per ulteriori informazioni sulla piattaforma, sul modello di sicurezza adottato o sulle procedure di registrazione di un Client, si raccomanda di consultare la documentazione di progetto disponibile sul sito ufficiale del gruppo o di contattare l'amministratore del proprio *Tenant*$_G$ di riferimento.

6. Glossario

AES-256$_G$ Algoritmo di cifratura simmetrica a 256 bit utilizzato per proteggere i payload telemetrici con decifratura esclusivamente client-side.

Bearer Token$_G$ Token di autorizzazione trasmesso nelle richieste HTTP, di solito nell'header Authorization, per dimostrare che il client ha il diritto di accedere a una risorsa.

Endpoint$_G$ URL o percorso specifico di un'API che rappresenta una risorsa o un'azione disponibile per i client, identificato univocamente nel sistema.

Gateway$_G$ Dispositivo fisico o software che funge da punto di accesso e intermediario per la comunicazione tra reti, sensori o sistemi diversi.

JWT$_G$ Acronimo di JSON Web Token, standard aperto per la creazione di token di accesso che consentono l'autenticazione e lo scambio sicuro di informazioni tra parti.

KeyCloak$_G$ Piattaforma Open Source che permette di centralizzare l'autenticazione e l'autorizzazione per applicazioni e servizi moderni.

OAuth2$_G$ Standard di autorizzazione che consente l'accesso delegato ai servizi, permettendo l'autenticazione tramite provider terzi senza esporre le credenziali.

Provisioning$_G$ Processo di registrazione e configurazione iniziale di un *gateway*$_G$ nella piattaforma.

REST$_G$ Acronimo di Representational State Transfer, stile architetturale per la progettazione di API basato su HTTP e rappresentazioni di risorse.

SDK$_G$ Acronimo di Software Development Kit, insieme di librerie, strumenti e documentazione che aiuta a integrare o sviluppare un software in modo più rapido e coerente.

SSE$_G$ Protocollo HTTP unidirezionale (Server-Sent Events) che permette al server di inviare aggiornamenti in tempo reale al client attraverso una connessione persistente.

Telemetria$_G$ Raccolta e trasmissione automatica di dati di misurazione e diagnostica da dispositivi remoti a un sistema centrale per monitoraggio e analisi.

Tenant$_G$ Entità cliente in un'architettura multi-tenancy che condivide l'infrastruttura ma con segregazione completa dei dati e delle risorse.