Verschlüsseltes Modbus RTU

Diese Seite beschreibt, wie ein NILAB-Integriertlinearmotor (NLi / GDi-Familie) in eine SPS-basierte Maschinensteuerung mit dem NILAB-verschlüsselten Modbus-RTU-Protokoll integriert wird.

Die Verschlüsselungsschicht läuft transparent über Standard-Modbus-RTU über RS-485. Es sind keine Änderungen an der physischen Verkabelung oder an der Standard-Modbus-RTU-Registerkarte erforderlich. Antriebe mit deaktivierter Verschlüsselung kommunizieren als Standard-Modbus-RTU-Slaves und sind vollständig abwärtskompatibel mit jedem generischen Modbus-Master.

Die kryptografische Implementierung wird als **geschlossene, vor-kompilierte Bibliothek** für jede unterstützte Entwicklungsumgebung (CODESYS, C/C++, .NET) bereitgestellt. Der interne Algorithmus ist Eigentum von NILAB GmbH. Die Bibliothek ist für Integratoren im Rahmen einer NILAB-Technologiepartnerschaft verfügbar — kontaktieren Sie NILAB für den Zugang.

Wenn die Verschlüsselung auf dem Antrieb aktiviert ist, reagiert der Antrieb **nur** auf verschlüsselte Frames. Jede Klartext-Modbus-Anfrage (FC3, FC6, FC16) wird stillschweigend verworfen. Sie müssen die NILAB-Verschlüsselungsbibliothek verwenden, um mit einem verschlüsselungsaktivierten Antrieb zu kommunizieren.

1. Konzept- und Protokollübersicht

1.1 Was die Verschlüsselungsschicht bietet

Die NILAB-verschlüsselte Modbus-RTU-Schicht fügt drei Sicherheitseigenschaften zum Standard-Modbus-RTU-Protokoll hinzu:

Element
Authentizität — jeder Frame trägt ein kryptografisches Tag. Der Antrieb verifiziert dieses Tag, bevor er eine Anfrage verarbeitet. Gefälschte, beschädigte oder wiedergespielte Frames werden stillschweigend ohne Antwort verworfen.
Vertraulichkeit — die Modbus-PDU (Funktionscode und Registerdaten) ist verschlüsselt. Ein Beobachter auf dem RS-485-Bus kann keine Registerwerte oder Sollwerte lesen.
Anti-Replay — ein monotoner Frame-Zähler in jedem Frame verhindert, dass ein erfasster gültiger Frame zu einem späteren Zeitpunkt erneut injiziert wird.

Der Schutz ist bidirektional: sowohl die Master→Antrieb-Anfrage als auch die Antrieb→Master-Antwort sind verschlüsselt und authentifiziert.

1.2 Schlüssel

Jeder NILAB-Antrieb wird ab Werk mit einem **eindeutigen 128-Bit-AES-Schlüssel** ausgestattet, der im geschützten Flash-Speicher gespeichert ist. Der Schlüssel ist an die Seriennummer des Antriebs in der NILAB-Schlüsseldatenbank gebunden.

Um den Schlüssel für einen bestimmten Antrieb zu erhalten, kontaktieren Sie NILAB GmbH mit der Seriennummer des Antriebs. Der Schlüssel wird als 32-stelliger Hexadezimalstring geliefert, zum Beispiel:

```
1D23586E43A218620EC5C7486274CAD0
```

Dieser Schlüssel muss als Geheimnis behandelt werden. Er muss im geschützten Speicher auf der SPS-Seite gespeichert werden und darf niemals über den Bus übertragen oder in Systemprotokolle geschrieben werden.

1.3 Format des verschlüsselten Frames

Der verschlüsselte Frame ersetzt den Standard-Modbus-Funktionscode und die PDU durch einen Wrapper mit fester Struktur, gekennzeichnet durch Funktionscode **0x65**:

```
Standard-Modbus-RTU (Klartext):
[ Knoten-ID (1) ][ FC (1) ][ PDU (N) ][ CRC16 (2) ]

NILAB-verschlüsseltes Modbus-RTU:
[ Knoten-ID (1) ][ 0x65 (1) ][ Zähler (4) ][ Chiffretext (N) ][ Tag (8) ][
CRC16 (2) ]
```

Feld	Größe (Bytes)	Beschreibung
Knoten-ID	1	Modbus-Slave-Adresse, unverändert
0x65	1	NILAB-verschlüsselter PDU-Marker
Zähler	4	Frame-Zähler, Big-Endian, streng steigend
Chiffretext	N	Verschlüsselte Modbus-PDU (gleicher Inhalt wie die Klartext-PDU)
Tag	8	Kryptografisches Authentifizierungstag über Zähler + Chiffretext
CRC16	2	Standard-Modbus-CRC16 über alle vorangehenden Bytes

Der durch den Verschlüsselungswrapper hinzugefügte Overhead beträgt **12 Bytes** pro Frame (4 Zähler + 8 Tag) gegenüber dem äquivalenten Klartext-Frame.

Das **CRC16** verwendet das Standard-Modbus-Polynom und die Byte-Reihenfolge (niederwertiges Byte zuerst) und wird über den gesamten Frame berechnet, einschließlich Knoten-ID, Funktionscode 0x65, Zähler, Chiffretext und Tag — genau wie bei Standard-Modbus-RTU.

1.4 Frame-Zähler

Der Frame-Zähler ist ein 32-Bit-Ganzzahl ohne Vorzeichen, Big-Endian übertragen (höchstwertiges Byte zuerst). Er ist pro Richtung unabhängig:

Element
Der **Master (SPS)** erhöht seinen TX-Zähler mit jeder Anfrage, die er sendet.
Der **Antrieb** verfolgt den letzten akzeptierten Master-TX-Zähler und lehnt jeden Frame ab, dessen Zähler nicht streng größer als der letzte akzeptierte Wert ist.

Der Antrieb verwendet seinen eigenen TX-Zähler für verschlüsselte Antworten; der Master verfolgt diesen, um wiedergespielte Antworten zu erkennen.

Zähler beginnen bei 0 bei der ersten Inbetriebnahme. **Sie müssen im nichtflüchtigen Speicher gespeichert und beim Start wiederhergestellt werden** — siehe Abschnitt 5.

1.5 Zusammenfassung des Antriebsverhaltens

Zustand des Antriebs	**Klartext-FC3/FC6 vom Master**	**Verschlüsselte FC-0x65 vom Master (korrekter Schlüssel)**	**Broadcast-Sync (FC 0x80, Knoten 0x00)**
Verschlüsselung **deaktiviert** (Werksstandard)	Akzeptiert, normale Antwort	Nicht verstanden, Exception-Antwort	Immer akzeptiert
Verschlüsselung **aktiviert**	Stillschweigend verworfen	Akzeptiert, verschlüsselte Antwort	Immer akzeptiert

2. Erste Schritte

2.1 Voraussetzungen

Element
Ein NILAB-NLi- oder GDi-Integriertlinearmotor mit Verschlüsselungsfirmware (fragen Sie NILAB nach der Firmware-Version, die Verschlüsselung unterstützt).
Der 16-Byte-AES-Schlüssel für den spezifischen Antrieb, erhalten von NILAB (siehe Abschnitt 1.2).
Die NILAB-Krypto-Bibliothek für Ihre Entwicklungsumgebung (siehe Abschnitt 3).
RS-485-Verkabelung: Standard-Modbus-RTU-Bus, 2-Draht, mit korrekter Terminierung.

2.2 Kommunikationsparameter

Das verschlüsselte Protokoll verwendet dieselben seriellen Parameter wie das Standard-NILAB-Modbus-RTU:

Parameter	**Wert**
Baudrate	115200 bps (Standard)
Datenbits	8
Parity	Keine
Stoppbits	1
Protokoll	Modbus RTU
Knoten-ID	Am Antrieb konfiguriert (Standard: 1)

2.3 Framegrößen für Zeitberechnungen

Die gesamte Framegröße hängt von der inneren PDU-Größe ab. Für die häufigsten Operationen:

Operation	Innere PDU (Bytes)	Gesamter verschlüsselter Frame (Bytes)
FC6 schreiben, 1 Register	5	1+1+4+5+8+2 = **21**
FC3 lesen, 1 Register	5 (Anfrage)	1+1+4+5+8+2 = **21**
FC3 lesen, 1 Register	4 (Antwort)	1+1+4+4+8+2 = **20**
FC3 lesen, N Register	3+2N (Antwort)	1+1+4+(3+2N)+8+2 = **19+2N**

Verwenden Sie diese Werte, um angemessene serielle Empfangszeitouts auf der SPS-Seite einzustellen.

---

3. NILAB-Krypto-Bibliothek

NILAB stellt eine geschlossene, vor-kompilierte Bibliothek für jede unterstützte SPS-Entwicklungsumgebung bereit. Die Bibliothek exponiert eine minimale, klar definierte API (siehe Abschnitt 4) und behandelt alle kryptografischen Operationen intern. Für die Integration ist kein Wissen über den zugrunde liegenden Algorithmus erforderlich.

3.2 Erhalten der Bibliothek

Die Bibliothek wird im Rahmen einer NILAB-Technologiepartnerschaft verteilt. Um Zugang anzufordern:

Element
Kontaktieren Sie NILAB GmbH mit der Betreffzeile „Encrypted Modbus Library Request“
Geben Sie Ihre Entwicklungsumgebung an (siehe Tabelle oben)
Geben Sie die Seriennummer(n) der NILAB-Antriebe an, die Sie integrieren

NILAB wird das Bibliothekspaket zusammen mit den entsprechenden Antriebsschlüsseln liefern.

---

4. Bibliotheks-API

Alle Bibliotheksvarianten exponiert dieselbe logische API, angepasst an die Konventionen der Zielumgebung. Die folgende Beschreibung verwendet Pseudocode-Notation.

4.1 Initialisierung

```
\
NILAB_Init(ctx, key[16], tx_counter, rx_counter)\
```

Initialisiert einen Krypto-Kontext für eine Antriebsverbindung.

Parameter	Typ	Beschreibung
ctx	undurchsichtiges Handle	Kontextobjekt — einen pro Antrieb zuweisen
key	byte[16]	Der 16-Byte-AES-Schlüssel für diesen Antrieb
tx_counter	uint32	Anfänglicher TX-Zähler (0 für erste Inbetriebnahme, gespeicherter Wert beim Neustart)
rx_counter	uint32	Anfänglicher RX-Zähler (0 für erste Inbetriebnahme, gespeicherter Wert beim Neustart)

4.2 Schreiben eines Anfrage-Frames (FC6)

```
frame_len = NILAB_BuildWriteFrame(ctx, node_id, reg_address, value,
frame_buf)
```

Erstellt einen vollständigen verschlüsselten FC6-Schreib-Frame, bereit zum Senden über RS-485.

Parameter	Typ	Beschreibung
ctx	Handle	Initialisierter Kontext
node_id	uint8	Modbus-Slave-Adresse
reg_address	uint16	Zu schreibende Registeradresse
value	uint16	Zu schreibender Wert
frame_buf	byte[]	Ausgabepuffer (mindestens 23 Bytes)
Rückgabewert	int	Anzahl der zu sendenden Bytes

4.3 Erstellen eines Leseanfrage-Frames (FC3)

```
frame_len = NILAB_BuildReadFrame(ctx, node_id, reg_address, qty, frame_buf)
```

Erstellt einen vollständigen verschlüsselten FC3-Leseframe für `qty` aufeinanderfolgende Register.

4.4 Antwort-Frame parsen und entschlüsseln

```
status = NILAB_ParseResponse(ctx, raw_frame, frame_len, node_id,
reg_values_out)
```

Verifiziert das Authentifizierungstag, prüft den Zähler und entschlüsselt die Antriebsantwort.

Rückgabewert	Bedeutung
NILAB_OK (0)	Frame authentifiziert und erfolgreich entschlüsselt. reg_values_out gültig.
NILAB_ERR_AUTH (1)	Authentifizierungstag stimmt nicht überein — Frame gefälscht, beschädigt oder falscher Schlüssel
NILAB_ERR_REPLAY (2)	Frame-Zähler steigt nicht — möglicher Replay-Angriff
NILAB_ERR_SHORT (3)	Frame zu kurz, um eine gültige verschlüsselte Antwort zu sein
NILAB_ERR_TIMEOUT(4)	Keine Antwort innerhalb des Timeouts empfangen

4.5 Aktuelle Zählerwerte lesen (für NVM-Persistenz)

```
tx_counter = NILAB_GetTxCounter(ctx)
rx_counter = NILAB_GetRxCounter(ctx)
```

Gibt die aktuellen Zählerwerte zurück. Speichern Sie diese regelmäßig und beim Herunterfahren im nichtflüchtigen Speicher (siehe Abschnitt 5).

---

5. Zählerpersistenz

Der Anti-Replay-Mechanismus erfordert, dass der Master-TX-Zähler über Netzzyklen hinweg streng steigend ist. Sowohl `tx_counter` als auch `rx_counter` müssen im nichtflüchtigen Speicher gespeichert und beim Start wiederhergestellt werden.

Empfohlene Strategie: Bei jedem SPS-Start den Kontext mit dem letzten gespeicherten Zählerwert plus einer Sicherheitsmarge (z. B. +1000) initialisieren. Dies stellt sicher, dass selbst wenn der NVM-Schreibvorgang beim Herunterfahren verzögert wurde, der wiederhergestellte Zähler immer noch höher ist als jeder Zähler, den der Antrieb in der vorherigen Sitzung akzeptiert hat.

Edit

5.1 CODESYS (RETAIN-Variablen)

```
VAR \RETAIN
  nTxCounterNVM : UDINT := 0;
  nRxCounterNVM : UDINT := 0;
END_VAR

(* Beim Start, einmal: *)
NILAB_Init(ctx, key, nTxCounterNVM + 1000, nRxCounterNVM);

(* Nach jeder erfolgreichen Transaktion: *)
nTxCounterNVM := NILAB_GetTxCounter(ctx);
nRxCounterNVM := NILAB_GetRxCounter(ctx);
```

CODESYS-RETAIN-Variablen werden bei einem Netzzyklus automatisch auf Flash gespeichert — kein zusätzlicher NVM-Schreibvorgang erforderlich.

5.2 C / Bare-Metal

```
/* Beim Start: */
uint32_t tx_saved, rx_saved;
```

```
NVM_Read(&tx_saved, &rx_saved);   /* Ihre NVM-Lesefunktion */
NILAB_Init(&ctx, key, tx_saved + 1000, rx_saved);

/* Nach jeder erfolgreichen Transaktion: */
NVM_Write(NILAB_GetTxCounter(&ctx), NILAB_GetRxCounter(&ctx));
```

5.3 Erste Inbetriebnahme

Bei der ersten Inbetriebnahme (keine NVM-Daten verfügbar), übergeben Sie 0 für beide Zähler. Der Antrieb akzeptiert den ersten Frame ab Zähler 0 und setzt seinen internen Referenzwert entsprechend.

---

6. Integrationsbeispiele

Die folgenden Beispiele zeigen die typische Aufrufsequenz für die drei unterstützten Umgebungen. Sie gehen davon aus, dass die Bibliothek in das Projekt importiert und der Kontext im Startabschnitt initialisiert wurde.

6.1 CODESYS V3.5 — Strukturierter Text

Fügen Sie die NILAB-Bibliothek zu Ihrem CODESYS-Projekt hinzu über **Tools → Library Manager → Add Library → NILAB_ModbusCrypto**.

```
(*
 ---------------------------------------------------------------
 Start / Initialisierung (einmal von PLC_PRG aufrufen, erster Scan)
 --------------------------------------------------------------- *)
\VAR
  ctx   : NILAB_Ctx;          (* Krypto-Kontext -- einen pro Antrieb *)
  key          : ARRAY[0..15] OF BYTE := [
                    16#1D, 16#23, 16#58, 16#6E, 16#43, 16#A2, 16#18, 16#62,
                    16#0E, 16#C5, 16#C7, 16#48, 16#62, 16#74, 16#CA, 16#D0
];
  aFrame       : ARRAY[0..31] OF BYTE;
  aResponse    : ARRAY[0..31] OF BYTE;
  nFrameLen    : UDINT;
  nStatus      : INT;
  nRegValue    : WORD;
  bInitDone    : BOOL := FALSE;
END_VAR

IF NOT bInitDone \THEN
  NILAB_Init(ctx := ctx,
             key := key,
             tx_counter := nTxCounterNVM + 1000,
```

```
              rx_counter := nRxCounterNVM);
  bInitDone := TRUE;
END_IF

(*
 ---------------------------------------------------------------
 Register 0x0010 = 0x1234 auf Knoten \1 schreiben
 --------------------------------------------------------------- *)
nFrameLen := NILAB_BuildWriteFrame(
              ctx         := ctx,
              node_id     := 1,
              reg_address := 16#0010,
              value       := 16#1234,
              frame_buf   := aFrame);

(* aFrame (nFrameLen Bytes) über Ihren seriellen/RS-485-Block senden *)
ComSend(port := COM1, data := aFrame, length := nFrameLen);

(* Auf Antwort warten, dann: *)
ComReceive(port := COM1, data := aResponse, length => nRespLen);

nStatus := NILAB_ParseResponse(
             ctx       := ctx,
             raw_frame := aResponse,
             frame_len := nRespLen,
             node_id   := 1,
             reg_values_out := nRegValue);

IF nStatus = NILAB_OK \THEN
  (* Schreiben bestätigt, nRegValue enthält den zurückgespiegelten Wert *)
  nTxCounterNVM := NILAB_GetTxCounter(ctx);
  nRxCounterNVM := NILAB_GetRxCounter(ctx);
\ELSE
  (* Fehler behandeln: nStatus = NILAB_ERR_AUTH, NILAB_ERR_REPLAY usw. *)
END_IF

(*
 ---------------------------------------------------------------
 Register 0x0020 auf Knoten \1 lesen
 --------------------------------------------------------------- *)
nFrameLen := NILAB_BuildReadFrame(
              ctx         := ctx,
              node_id     := 1,
              reg_address := 16#0020,
              qty         := 1,
              frame_buf   := aFrame);

ComSend(port := COM1, data := aFrame, length := nFrameLen);
ComReceive(port := COM1, data := aResponse, length => nRespLen);
```

```
nStatus := NILAB_ParseResponse(
             ctx           := ctx,
             raw_frame     := aResponse,
             frame_len     := nRespLen,
             node_id       := 1,
             reg_values_out := nRegValue);

IF nStatus = NILAB_OK \THEN
  wMotorStatus := nRegValue;   (* den Registerwert verwenden *)
\END_IF
```

Ersetzen Sie ComSend / ComReceive durch die tatsächlichen seriellen Funktionsbausteine, die in Ihrer CODESYS-Laufzeit verfügbar sind (z. B. SL_SER_SND / SL_SER_RCV bei Wago, RS bei generischem IEC 61131-3, ModbusSerial-Masterblöcke, wenn Sie die Funktionscode-Ebene umgehen). Die NILAB-Bibliothek arbeitet auf Rohbyte-Ebene und ist unabhängig vom verwendeten seriellen Übertragungsblock.

6.2 C / C++ (Bare-Metal, RTOS, Linux)

Fügen Sie nilab_modbus_crypto. h zu Ihrem Include-Pfad hinzu und verlinken Sie gegen libnilab_modbus_crypto. a (ARM) oder nilab_modbus_crypto. lib (x86 Windows).

```
#include "nilab_modbus_crypto.h"

/*
 --- Initialisierung (einmal beim Start aufrufen) --- */
static NILAB_Ctx g_ctx;

static const uint8_t g_key[16] = {
  0x1D, 0x23, 0x58, 0x6E, 0x43, 0xA2, 0x18, 0x62,
  0x0E, 0xC5, 0xC7, 0x48, 0x62, 0x74, 0xCA, \0xD0
};

void motor_comm_init(void)
{
  uint32_t tx_saved, rx_saved;
  NVM_Read(&tx_saved, &rx_saved);   /* Ihr NVM-Lesevorgang */
  NILAB_Init(&g_ctx, g_key, tx_saved + 1000, rx_saved);
}

/*
 --- Register 0x0010 = 0x1234 auf Knoten 1 schreiben --- */
int motor_write_register(uint16_t reg_addr, uint16_t value)
{
  uint8_t frame[32];
  uint8_t response[32];

  int flen = NILAB_BuildWriteFrame(&g_ctx, 1, reg_addr, value, frame);
```

```
  rs485_send(frame, flen);   /* Ihr RS-485-Senden */
  int rlen = rs485_receive(response, sizeof(response), 200 /*ms*/);

  NILAB_Status st = NILAB_ParseResponse(&g_ctx, response, rlen, 1, NULL);

  if (st == NILAB_OK) {
      NVM_Write(NILAB_GetTxCounter(&g_ctx), NILAB_GetRxCounter(&g_ctx));
  }
  return (int)st;
}

/*
 --- Register 0x0020 auf Knoten 1 lesen --- */
int motor_read_register(uint16_t reg_addr, uint16_t *value_out)
{
  uint8_t frame[32];
  uint8_t response[32];

  int flen = NILAB_BuildReadFrame(&g_ctx, 1, reg_addr, 1, frame);

  rs485_send(frame, flen);
  int rlen = rs485_receive(response, sizeof(response), 200);

  NILAB_Status st = NILAB_ParseResponse(&g_ctx, response, rlen, 1,
value_out);
  return (int)st;
}
```

6.3 Ladder Logic (generisch)

Ladder Logic hat keine native Unterstützung für Byte-Level-kryptografische Operationen. Der empfohlene Ansatz für Ladder-basierte SPS ist:

```
Network 1: Enable-Block bei steigender Flanke von StartComm \Kontakt
--[P]--[StartComm]----[FB_NILAB.xEnable := TRUE]--

Network 2: Schreibmodus \wählen
--[WriteCmd]----------[FB_NILAB.xWrite := TRUE]---
--[/WriteCmd]---------[FB_NILAB.xWrite := FALSE]--

Network 3: Registeradresse und \Wert übergeben
--[MOVE]-- nTargetReg  --> FB_NILAB.\nRegAddress
--[MOVE]-- nSetpoint   --> FB_NILAB.nWriteValue

Network 4: Ergebnis bei \Done verarbeiten
--[FB_NILAB.xDone]---[ProcessResultSubroutine]----

Network 5: Fehlerbehandlung
```

```
--[FB_NILAB.xError]--[SET]--AlarmBit-------------
```

Für SPS, die **nicht** Funktionsbausteine oder Strukturtext-Unterroutinen unterstützen (reine Relais-Ladder-Systeme), ist direkte Verschlüsselung auf SPS-Ebene nicht machbar. In diesem Fall kontaktieren Sie NILAB für Informationen über den **NILAB Crypto Gateway** — ein kompaktes externes Modul, das als transparenter Verschlüsselungs-Bridge zwischen dem Standard-Modbus-RTU-Port der SPS und dem Antriebsbus fungiert.

---

7. Fehlerbehandlung

Alle API-Funktionen geben einen Statuscode zurück. Die folgende Tabelle beschreibt die korrekte Reaktion auf jede Fehlerbedingung in einer Maschinenanwendung:

Statuscode	Bedeutung	Empfohlene Aktion
NILAB_OK	Transaktion erfolgreich	Normaler Betrieb, NVM-Zähler aktualisieren
NILAB_ERR_AUTH	Frame-Authentifizierung fehlgeschlagen	Ereignis protokollieren, bis zu 3 Mal wiederholen; wenn anhaltend, Schlüssel prüfen
NILAB_ERR_REPLAY	Zähler stimmt nicht überein oder Replay erkannt	Ereignis protokollieren, Kontext mit NILAB_Init und gespeicherten Zählern zurücksetzen
NILAB_ERR_SHORT	Antwort-Frame zu kurz oder falsche Knoten-ID	RS-485-Verkabelung und Terminierung prüfen
NILAB_ERR_TIMEOUT	Keine Antwort innerhalb des Timeouts	Antriebsstromversorgung, Adresse und Baudrate prüfen

In einer sicherheitsrelevanten Achsanwendung sollte jeder anhaltende `NILAB_ERR_AUTH`- oder `NILAB_ERR_REPLAY`-Fehler einen sofortigen Achsstopp und einen Bedieneralarm auslösen. Diese Fehler sollten im normalen Betrieb nicht auftreten und können auf Manipulation oder einen Hardwarefehler hinweisen.

---

8. Fehlerbehebung

Symptom	Wahrscheinliche Ursache	Lösung
Antrieb reagiert auf keinen Frame	Verschlüsselung aktiviert, SPS sendet Klartext	NILAB-Bibliothek verwenden. Alle Klartext-Frames werden konstruktionsbedingt abgelehnt.
Antrieb reagiert nur auf den ersten Frame nach dem Einschalten	Zähler stimmen nach SPS-Neustart nicht überein	Gespeicherte Zähler beim Start wiederherstellen (Abschnitt 5)
Anhaltende NILAB_ERR_AUTH-Fehler	Falscher Schlüssel für diesen Antrieb	Schlüssel gegen NILAB-Schlüsseldatenbank mit Seriennummer prüfen

Symptom	Wahrscheinliche Ursache	Lösung
Gelegentliche NILAB_ERR_AUTH-Fehler	RS-485-Busrauschen beschädigt Frames	Kabel, Terminierungswiderstände (120 Ω an beiden Enden), Kabellänge prüfen
NILAB_ERR_REPLAY nach SPS-Neustart	Gespeicherter Zähler ist niedriger als der Antrieb erwartet	NVM-Sicherheitsmarge von +1000 auf +10000 erhöhen und neu inbetriebnehmen
Kommunikation funktioniert, aber Antrieb ignoriert Bewegungsbefehle	Antrieb im Fehler-/Schutzzustand	NILAB Starter verwenden, um das Fehlerregister des Antriebs vor SPS-Steuerung zu prüfen

9. Referenz

Kontakt und Support

Für Bibliothekszugang, Schlüsselbereitstellung oder technische Integrationsunterstützung:

Element
Web: www.nilab.at
Technisches Supportportal: www.ni-lab.online

Um eine Sicherheitslücke in NILAB-Produkten zu melden: Sicherheitslücke melden