Modbus RTU crittografato

Questa pagina descrive come integrare un motore lineare integrato NILAB (famiglia NLi / GDi) in un controller di macchina basato su PLC usando il protocollo Modbus RTU crittografato NILAB.

Il livello di crittografia opera in modo trasparente sopra il Modbus RTU standard su RS-485. Non sono richieste modifiche al cablaggio fisico o alla mappa dei registri Modbus RTU standard. I drive con crittografia disabilitata comunicano come slave Modbus RTU standard e sono completamente retrocompatibili con qualsiasi master Modbus generico.

L'implementazione crittografica è fornita come **libreria chiusa, pre-compilata** per ogni ambiente di sviluppo supportato (CODESYS, C/C++, .NET). L'algoritmo interno è proprietario di NILAB GmbH. La libreria è disponibile per gli integratori nell'ambito di un accordo di partner tecnologico NILAB — contatta NILAB per l'accesso.

Se la crittografia è abilitata sul drive, il drive risponderà **solo** a frame crittografati. Qualsiasi richiesta Modbus in chiaro (FC3, FC6, FC16) sarà scartata silenziosamente. Devi usare la libreria di crittografia NILAB per comunicare con un drive con crittografia abilitata.

1. Panoramica del concetto e del protocollo

1.1 Cosa fornisce il livello di crittografia

Il livello Modbus RTU crittografato NILAB aggiunge tre proprietà di sicurezza al protocollo Modbus RTU standard:

Voce
Autenticità — ogni frame porta un tag crittografico. Il drive verifica questo tag prima di elaborare qualsiasi richiesta. I frame falsificati, corrotti o ripetuti vengono scartati silenziosamente senza alcuna risposta.
Riservatezza — la PDU Modbus (codice funzione e dati dei registri) è crittografata. Un osservatore sul bus RS-485 non può leggere i valori dei registri o i setpoint.
Anti-replay — un contatore di frame monotono incorporato in ogni frame impedisce che un frame valido catturato venga reiniettato in un momento successivo.

La protezione è bidirezionale: sia la richiesta master→drive che la risposta drive→master sono crittografate e autenticate.

1.2 Chiave

Ogni drive NILAB è dotato in fabbrica di una **chiave AES 128-bit univoca**, memorizzata in memoria flash protetta. La chiave è legata al numero di serie del drive nel database chiavi NILAB.

Per ottenere la chiave per un drive specifico, contatta NILAB GmbH con il numero di serie del drive. La chiave è consegnata come stringa esadecimale di 32 caratteri, ad esempio:

```
1D23586E43A218620EC5C7486274CAD0
```

Questa chiave deve essere trattata come segreta. Deve essere memorizzata in memoria protetta sul lato PLC e non deve mai essere trasmessa sul bus o scritta nei log di sistema.

1.3 Formato del frame crittografato

Il frame crittografato sostituisce il codice funzione e la PDU Modbus standard con un wrapper a struttura fissa identificato dal codice funzione **0x65**:

```
Modbus RTU standard (in chiaro):
[ Node ID (1) ][ FC (1) ][ PDU (N) ][ CRC16 (2) ]

Modbus RTU crittografato NILAB:
[ Node ID (1) ][ 0x65 (1) ][ Counter (4) ][ Ciphertext (N) ][ Tag (8) ][
CRC16 (2) ]
```

Campo	Dimensione (byte)	Descrizione
Node ID	1	Indirizzo slave Modbus, invariato
0x65	1	Marcatore PDU crittografata NILAB
Counter	4	Contatore frame, big-endian, strettamente crescente
Ciphertext	N	PDU Modbus crittografata (stesso contenuto della PDU in chiaro)
Tag	8	Tag di autenticazione crittografico su Counter + Ciphertext
CRC16	2	CRC16 Modbus standard su tutti i byte precedenti

L'overhead aggiunto dal wrapper di crittografia è **12 byte** per frame (4 counter + 8 tag) rispetto al frame in chiaro equivalente.

Il **CRC16** usa il polinomio Modbus standard e l'ordine dei byte (byte basso per primo) ed è calcolato sull'intero frame includendo Node ID, codice funzione 0x65, counter, ciphertext e tag — esattamente come nel Modbus RTU standard.

1.4 Contatore frame

Il contatore frame è un intero senza segno a 32 bit, trasmesso big-endian (byte più significativo per primo). È indipendente per direzione:

Voce
Il **master (PLC)** incrementa il suo contatore TX con ogni richiesta che invia.
Il **drive** tiene traccia dell'ultimo contatore TX master accettato e rifiuta qualsiasi frame con un contatore che non sia strettamente maggiore dell'ultimo valore accettato.
Il drive usa il proprio contatore TX per le risposte crittografate; il master tiene traccia di questo per rilevare risposte replay.

I contatori iniziano a 0 alla prima messa in servizio. **Devono essere salvati in memoria non volatile** e ripristinati all'avvio — vedi Sezione 5.

1.5 Riepilogo del comportamento del drive

Stato del drive	FC3/FC6 in chiaro dal master	FC 0x65 crittografato dal master (chiave corretta)	Sync broadcast (FC 0x80, node 0x00)
Crittografia **disabilitata** (default di fabbrica)	Accettato, risposta normale	Non compreso, risposta eccezione	Sempre accettato
Crittografia **abilitata**	Scartato silenziosamente	Accettato, risposta crittografata	Sempre accettato

---

2. Come iniziare

2.1 Prerequisiti

Voce
Un motore lineare integrato NLi o GDi NILAB con firmware di crittografia (chiedi a NILAB la versione di firmware che supporta la crittografia).
La chiave AES di 16 byte per il drive specifico, ottenuta da NILAB (vedi Sezione 1.2).
La libreria crypto NILAB per il tuo ambiente di sviluppo (vedi Sezione 3).
Cablaggio RS-485: bus Modbus RTU standard, 2 fili, con corretta terminazione.

2.2 Parametri di comunicazione

Il protocollo crittografato usa gli stessi parametri seriali del Modbus RTU NILAB standard:

Parametro	Valore
Baud rate	115200 bps (default)
Data bits	8
Parity	Nessuna
Stop bits	1
Protocollo	Modbus RTU
Node ID	Configurato sul drive (default: 1)

2.3 Dimensioni dei frame per i calcoli di timing

La dimensione totale del frame dipende dalla dimensione della PDU interna. Per le operazioni più comuni:

Operazione	PDU interna (byte)	Frame crittografato totale (byte)
FC6 scrittura, 1 registro	5	1+1+4+5+8+2 = **21**
FC3 lettura, 1 registro	5 (richiesta)	1+1+4+5+8+2 = **21**
FC3 lettura, 1 registro	4 (risposta)	1+1+4+4+8+2 = **20**
FC3 lettura, N registri	3+2N (risposta)	1+1+4+(3+2N)+8+2 = **19+2N**

Usa questi valori per impostare timeout di ricezione seriale appropriati sul lato PLC.

3. Libreria crypto NILAB

NILAB fornisce una libreria chiusa, pre-compilata per ogni ambiente di sviluppo PLC supportato. La libreria espone un'API minimale e ben definita (vedi Sezione 4) e gestisce tutte le operazioni crittografiche internamente. Non è richiesta conoscenza dell'algoritmo sottostante per l'integrazione.

3.2 Ottenere la libreria

La libreria è distribuita nell'ambito di un accordo di partner tecnologico NILAB. Per richiedere l'accesso:

Voce
Contatta NILAB GmbH con oggetto “Encrypted Modbus Library Request”
Specifica il tuo ambiente di sviluppo (vedi tabella sopra)
Fornisci i numeri di serie dei drive NILAB che stai integrando

NILAB fornirà il pacchetto libreria insieme alle chiavi del drive corrispondenti.

4. API della libreria

Tutte le varianti della libreria espongono la stessa API logica, adattata alle convenzioni dell'ambiente di destinazione. La seguente descrizione usa notazione pseudocodice.

4.1 Inizializzazione

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

Inizializza un contesto crypto per una connessione drive.

Parametro	Tipo	Descrizione
ctx	handle opaco	Oggetto contesto — assegnane uno per drive
key	byte[16]	La chiave AES di 16 byte per questo drive
tx_counter	uint32	Contatore TX iniziale (0 per prima messa in servizio, valore salvato al riavvio)
rx_counter	uint32	Contatore RX iniziale (0 per prima messa in servizio, valore salvato al riavvio)

4.2 Costruire un frame di richiesta di scrittura (FC6)

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

Costruisce un frame di scrittura FC6 crittografato completo, pronto per la trasmissione su RS-485.

Parametro	Tipo	Descrizione
ctx	handle	Contesto inizializzato
node_id	uint8	Indirizzo slave Modbus
reg_address	uint16	Indirizzo registro da scrivere
value	uint16	Valore da scrivere
frame_buf	byte[]	Buffer di uscita (minimo 23 byte)
valore di ritorno	int	Numero di byte da trasmettere

4.3 Costruire un frame di richiesta di lettura (FC3)

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

Costruisce un frame di lettura FC3 crittografato completo per `qty` registri consecutivi.

4.4 Analizzare e decrittare un frame di risposta

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

Verifica il tag di autenticazione, controlla il contatore e decritta la risposta del drive.

Valore di ritorno	Significato
NILAB_OK (0)	Frame autenticato e decrittato con successo. reg_values_out valido.
NILAB_ERR_AUTH (1)	Tag di autenticazione non corrisponde — frame falsificato, corrotto o chiave errata
NILAB_ERR_REPLAY (2)	Contatore frame non crescente — possibile attacco replay
NILAB_ERR_SHORT (3)	Frame troppo corto per essere una risposta crittografata valida
NILAB_ERR_TIMEOUT(4)	Nessuna risposta ricevuta entro il timeout

4.5 Leggere i valori del contatore correnti (per la persistenza NVM)

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

Restituisce i valori del contatore correnti. Salva questi in memoria non volatile periodicamente e allo spegnimento (vedi Sezione 5).

5. Persistenza del contatore

Il meccanismo anti-replay richiede che il contatore TX master sia strettamente crescente attraverso i cicli di alimentazione. Sia `tx_counter` che `rx_counter` devono essere salvati in memoria non volatile e ripristinati all'avvio.

Strategia consigliata: ad ogni avvio PLC, inizializza il contesto con l'ultimo valore del contatore salvato più un margine di sicurezza (es. +1000). Questo garantisce che anche se la scrittura NVM è stata ritardata allo spegnimento, il contatore ripristinato è comunque più alto di qualsiasi contatore che il drive ha accettato nella sessione precedente.

Edit

5.1 CODESYS (variabili RETAIN)

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

(* All'avvio, una volta: *)
NILAB_Init(ctx, key, nTxCounterNVM + 1000, nRxCounterNVM);

(* Dopo ogni transazione riuscita: *)
nTxCounterNVM := NILAB_GetTxCounter(ctx);
nRxCounterNVM := NILAB_GetRxCounter(ctx);
```

Le variabili RETAIN CODESYS vengono salvate automaticamente su flash al ciclo di alimentazione — non è necessaria alcuna scrittura NVM aggiuntiva.

5.2 C / bare-metal

```
/* All'avvio: */
uint32_t tx_saved, rx_saved;
NVM_Read(&tx_saved, &rx_saved);   /* la tua funzione di lettura NVM */
NILAB_Init(&ctx, key, tx_saved + 1000, rx_saved);

/* Dopo ogni transazione riuscita: */
NVM_Write(NILAB_GetTxCounter(&ctx), NILAB_GetRxCounter(&ctx));
```

5.3 Prima messa in servizio

Alla prima messa in servizio (nessun dato NVM disponibile), passa 0 per entrambi i contatori. Il drive accetterà il primo frame dal contatore 0 e imposterà il suo riferimento interno di conseguenza.

---

6. Esempi di integrazione

Gli esempi seguenti mostrano la sequenza di chiamata tipica per i tre ambienti supportati. Essi presuppongono che la libreria sia stata importata nel progetto e che il contesto sia stato inizializzato nella sezione di avvio.

6.1 CODESYS V3.5 — Structured Text

Aggiungi la libreria NILAB al tuo progetto CODESYS tramite **Tools → Library Manager → Add Library → NILAB_ModbusCrypto**.

```
(*
 ---------------------------------------------------------------
 Avvio / inizializzazione (chiama una volta da PLC_PRG, primo scan)
 --------------------------------------------------------------- *)
\VAR
  ctx   : NILAB_Ctx;          (* Contesto crypto -- uno per drive *)
  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

(*
 ---------------------------------------------------------------
 Scrivere registro 0x0010 = 0x1234 sul node \1
 --------------------------------------------------------------- *)
nFrameLen := NILAB_BuildWriteFrame(
              ctx         := ctx,
```

```
              node_id     := 1,
              reg_address := 16#0010,
              value       := 16#1234,
              frame_buf   := aFrame);

(* Inviare aFrame (nFrameLen byte) tramite il tuo blocco seriale/RS-485 *)
ComSend(port := COM1, data := aFrame, length := nFrameLen);

(* Attendere la risposta, poi: *)
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
  (* scrittura confermata, nRegValue contiene il valore restituito *)
  nTxCounterNVM := NILAB_GetTxCounter(ctx);
  nRxCounterNVM := NILAB_GetRxCounter(ctx);
\ELSE
  (* gestire l'errore: nStatus = NILAB_ERR_AUTH, NILAB_ERR_REPLAY, ecc. *)
END_IF

(*
 ---------------------------------------------------------------
 Leggere registro 0x0020 sul node \1
 --------------------------------------------------------------- *)
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;   (* usare il valore del registro *)
```

```
\END_IF
```

Sostituisci ComSend / ComReceive con i blocchi di funzione seriale effettivi disponibili nel tuo runtime CODESYS (es. SL_SER_SND / SL_SER_RCV su Wago, RS su IEC 61131-3 generico, blocchi master ModbusSerial se bypassi il livello del codice funzione). La libreria NILAB opera a livello di byte grezzi ed è indipendente dal blocco di trasporto seriale utilizzato.

6.2 C / C++ (bare-metal, RTOS, Linux)

Aggiungi nilab_modbus_crypto. h al tuo percorso di include e collega contro libnilab_modbus_crypto. a (ARM) o nilab_modbus_crypto. lib (x86 Windows).

```
#include "nilab_modbus_crypto.h"

/*
 --- Inizializzazione (chiama una volta all'avvio) --- */
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);   /* la tua lettura NVM */
  NILAB_Init(&g_ctx, g_key, tx_saved + 1000, rx_saved);
}

/*
 --- Scrivere registro 0x0010 = 0x1234 sul node 1 --- */
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);   /* il tuo invio RS-485 */
  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;
}
```

```
/*
 --- Leggere registro 0x0020 sul node 1 --- */
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 (generico)

Ladder Logic non ha supporto nativo per operazioni crittografiche a livello di byte. L'approccio consigliato per PLC basati su Ladder è:

```
Network 1: Blocco abilitazione sul fronte di salita di StartComm \contatto
--[P]--[StartComm]----[FB_NILAB.xEnable := TRUE]--

Network 2: Seleziona modalità di \scrittura
--[WriteCmd]----------[FB_NILAB.xWrite := TRUE]---
--[/WriteCmd]---------[FB_NILAB.xWrite := FALSE]--

Network 3: Passa indirizzo registro e \valore
--[MOVE]-- nTargetReg  --> FB_NILAB.\nRegAddress
--[MOVE]-- nSetpoint   --> FB_NILAB.nWriteValue

Network 4: Elabora il risultato su \Done
--[FB_NILAB.xDone]---[ProcessResultSubroutine]----

Network 5: Gestione \errori
--[FB_NILAB.xError]--[SET]--AlarmBit-------------
```

Per PLC che **non** supportano blocchi di funzione o subroutine Structured Text (sistemi a relè puro), la crittografia diretta non è fattibile a livello PLC. In questo caso, contatta NILAB per informazioni sul **NILAB Crypto Gateway** — un modulo esterno compatto che funge da ponte di crittografia trasparente tra la porta Modbus RTU standard del PLC e il bus dei drive.

---

7. Gestione degli errori

Tutte le funzioni API restituiscono un codice di stato. La seguente tabella descrive la risposta corretta a ogni condizione di errore in un'applicazione di macchina:

Codice di stato	Significato	Azione consigliata
NILAB_OK	Transazione riuscita	Operazione normale, aggiorna i contatori NVM
NILAB_ERR_AUTH	Autenticazione frame fallita	Registra evento, riprova fino a 3 volte; se persistente, controlla la chiave
NILAB_ERR_REPLAY	Contatore non corrispondente o replay rilevato	Registra evento, reimposta il contesto con NILAB_Init e contatori salvati
NILAB_ERR_SHORT	Frame di risposta troppo corto o node ID errato	Controlla cablaggio e terminazione RS-485
NILAB_ERR_TIMEOUT	Nessuna risposta ricevuta entro il timeout	Controlla alimentazione drive, indirizzo e baud rate

In un'applicazione di asse rilevante per la sicurezza, qualsiasi errore persistente `NILAB_ERR_AUTH` o `NILAB_ERR_REPLAY` dovrebbe attivare un arresto immediato dell'asse e un allarme operatore. Questi errori non dovrebbero verificarsi in operazione normale e possono indicare manomissione o guasto hardware.

---

8. Risoluzione dei problemi

Sintomo	Causa probabile	Soluzione
Il drive non risponde a nessun frame	Crittografia abilitata, PLC invia in chiaro	Usa la libreria NILAB. Tutti i frame in chiaro sono rifiutati by design.
Il drive risponde solo al primo frame dopo l'accensione	Contatore non corrispondente dopo riavvio PLC	Ripristina i contatori salvati all'avvio (Sezione 5)
Errori NILAB_ERR_AUTH persistenti	Chiave errata per questo drive	Verifica la chiave contro il database chiavi NILAB usando il numero di serie del drive
Errori NILAB_ERR_AUTH intermittenti	Rumore bus RS-485 che corrompe i frame	Controlla cavo, resistori di terminazione (120 Ω a entrambe le estremità), lunghezza cavo
NILAB_ERR_REPLAY dopo riavvio PLC	Contatore salvato più basso di quanto il drive si aspetta	Aumenta il margine di sicurezza NVM da +1000 a +10000 e ricommissiona
La comunicazione funziona ma il drive ignora i comandi di movimento	Drive in stato di errore/protezione	Usa NILAB Starter per controllare il registro guasti del drive prima del controllo PLC

---

9. Riferimento

Contatto e supporto

Per accesso alla libreria, provisioning delle chiavi o supporto tecnico di integrazione:

Voce
Web: www.nilab.at
Portale di supporto tecnico: www.ni-lab.online

Per segnalare una vulnerabilità di sicurezza nei prodotti NILAB: Segnala una vulnerabilità

From:
https://www.nilab.at/dokuwiki/ - **NiLAB GmbH Knowledgebase**

Permanent link:
https://www.nilab.at/dokuwiki/doku.php?id=it:integrated_drive_motors:modbus_crypto

Last update: **2026/09/11 16:41**