Key Provisioning for Minimal Fieldbus Systems

Guideline on symmetric key provisioning for constraint embedded fieldbus devices

White Paper EmSA-WP-104

Version 1.12
1 June 2026

Jointly authored by

Embedded Systems Academy GmbH	**Embedded Systems Academy, Inc.**
Bahnhofstraße 17	**84 W. Santa Clara St., Suite 700**
30890 Barsinghausen	**San Jose, CA 95113**
Germany	**United States**

www.em-sa.com

Contents

1 Scope and Objective

This document presents a key provisioning model for minimal fieldbus systems where the devices are too constrained to use asymmetric cryptography. The model covers the full lifecycle from factory delivery to end of life. It defines a layered, prioritized set of pre-shared keys, the operations that install each layer and the destructive recovery paths that return a device to its uncommissioned state. CANopen is used as an example.

A node may also hold cryptographic material that is outside the scope of this document. Software-update keys used to authenticate firmware images, for example, are typically separated from the keys that protect operational traffic on the bus. This document focuses on the keys used directly for and during fieldbus communication; keys held for other purposes have their own lifecycle and are not discussed further.

Fieldbus security implementations using this key concept include SOFA (Secure Object Fieldbus Access, see our white paper EmSA-WP-105) and SPsec (Small-Packet Network Security Sublayer).

SOFA is aimed at securing single, selected accesses: service parameters, configuration objects and infrequently invoked functions such as firmware activation, configuration locking or mode change. It is not designed for protecting the high-rate cyclic process data (periodic measurements, drive-loop updates) that dominates a running fieldbus. For that traffic class a dedicated security sublayer sitting directly above the data link layer is the better fit. The SPsec specifications address it explicitly. The SPsec specifications are available at EmSA's technical library at: https://www.esacademy.com/en/library/spsec.html

1.1 System Overview

Our target system is a single machine in which a main control unit talks to several embedded devices over a fieldbus segment. The main control unit is the only node with an external interface to a higher control level, and that external interface enforces its own security controls and is out of scope here.

We assume the device firmware provides an authenticated encryption with associated data (AEAD) function, a key derivation function (KDF), secure non-volatile key storage and a true random number generator. Where the platform lacks any of these, the cryptographic measures described here are not feasible.

2 Root of Trust Options

The first design decision is the root of trust.

2.1 Devices With Asymmetric Capability

If every device on the bus can store and protect a manufacturer-issued private key and verify a certificate chain at production rates, the certificate is the natural root of trust. Provisioning then follows the certificate-based onboarding patterns established by IEC 62443-2-4, and key agreement uses a standard authenticated handshake. We recommend this path whenever the hardware allows it. The remainder of this document addresses the cases in which the hardware does not.

2.2 Symmetric-Only Constrained Devices

Many fieldbus nodes are too constrained for asymmetric primitives at the rates and frame sizes that fieldbuses impose. For these devices the realistic protection on the bus is AEAD using a symmetric cipher. We recommend AES-128-GCM where the platform supports it. Symmetric AEAD requires pre-shared keys, and the rest of this document defines how those keys are installed, rotated and destroyed.

3 Provisioning Lifecycle

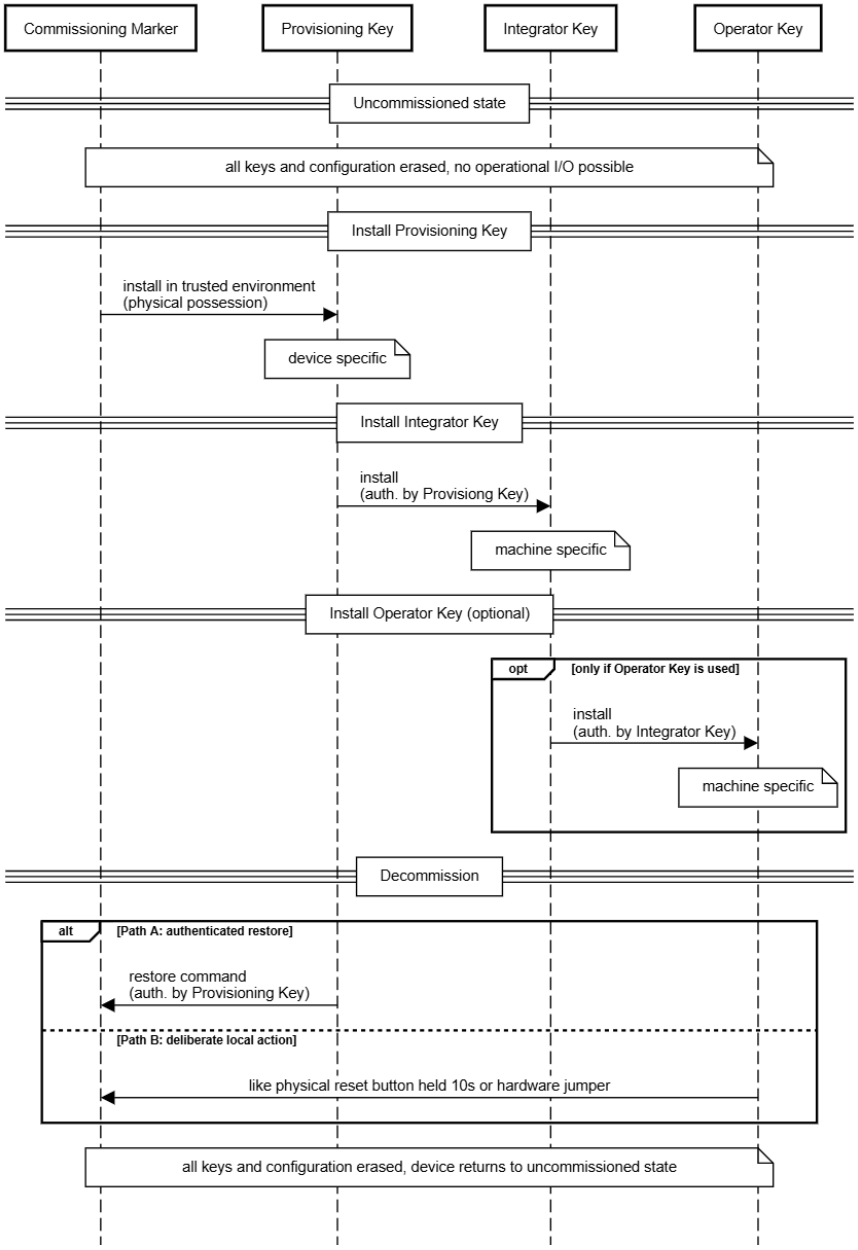

FIGURE 1: KEY LIFE CYCLE OVERVIEW

The model uses two mandatory and one optional base-key layers above an explicit uncommissioned state.

- Provisioning Key (mandatory)
- Integrator Key (mandatory)
- Operator Key (optional)

Each base key is installed under protection of one of the previous ones (Operator Key can be installed by protection of Provisioning or Integrator Key), has its own lifecycle and can be destroyed independently. How those base keys are then turned into a per-session key for actual secure access is described in the next chapter.

Figure 1 summarizes the full lifecycle as a sequence of key installations. The horizontal participants are the key roles a device knows: the factory commissioning marker, the Provisioning Key, the Integrator Key and the optional Operator Key. Each arrow shows which key authorizes the installation or replacement of the next. Every authenticated install arrow further implies a salt write and a session-key derivation on the bus, both omitted from Figure 1 for clarity; the salt mechanism is described in the "Using the Keys" chapter and shown in Figure 3. The two arrows that return to the commissioning marker show the two decommissioning paths, both destructive.

3.1 Uncommissioned State and Factory Commissioning Marker

A device leaves the factory in an uncommissioned state. In this state it does not perform any operational input or output function. It refuses all bus commands except the single command that installs the Provisioning Key. The state of the device, not a credential, controls what it accepts.

We name the known value that the device recognizes in this state the factory commissioning marker (technically may be an unprotected write with a zero key). The factory commissioning marker is not an authentication secret, not a password, and not a cryptographic key. It is a non-secret factory default value used only while the device is in the uncommissioned state. It does not enable operational use of the device and it becomes invalid once device-specific provisioning has been completed.

Naming this value explicitly matters for audit and conformity assessment. A device described as shipping with a "zero key" reads as a default password violation. A device described as shipping in an uncommissioned state that accepts a single provisioning command from a documented commissioning environment reads as a state machine, and the relevant standards recognize that pattern.

> **⚠ SECURITY**
> The single command an uncommissioned device accepts, the Provisioning Key install, must be issued only in an isolated commissioning environment with no untrusted node on the bus. This isolation, not the non-secret commissioning marker, is the control on which the security of the uncommissioned state depends: without it, an attacker on the bus could install their own Provisioning Key and take ownership of the device.

3.2 Device-Unique Provisioning Key

The first cryptographic credential installed on the device is the Provisioning Key. The Provisioning Key shall be unique per device. It has three purposes:

- authorize installation and replacement of the Integrator Key,
- authorize destructive factory restore by remote command,
- bind any device-specific reset or re-keying operation to a credential held by the integrator.

The Provisioning Key may be installed by the manufacturer during production, in which case it shall be delivered out of band to the integrator. A label inside the enclosure is the established industrial practice and is consistent with the expectations of both IEC 62443-2-4 and CRA Annex II, provided the device is shipped with intact tamper evidence.

The Provisioning Key may also be installed by the integrator in a trusted commissioning environment, using a commissioning tool that generates the key, installs it and records it in the integrator's key management system. Both delivery models are acceptable, and the device cannot tell them apart; the assurance therefore rests entirely on the out-of-band delivery and on the isolated commissioning environment. Where the optional Key ID is used, the same tool generates and installs a Key ID alongside the key, so the key management system can later identify the installed key by reading its Key ID.

Installation of the Provisioning Key on an uncommissioned device must occur in an isolated commissioning environment. Acceptable environments include a dedicated commissioning bench with no other bus participants or a new-machine installation in which no other devices are yet powered on the bus. The device shall enforce the key length required by the Cryptographic Primitives chapter and shall refuse trivially weak Provisioning Keys, such as all-zero or known-default values, so that a tool attempting to install a low-entropy value is rejected by the device itself.

> **⚠ SECURITY**
> The Provisioning Key must be unique per device. Install it only on an uncommissioned device in an isolated commissioning environment, and have the device enforce the key

length and refuse trivially weak keys. A shared or weak Provisioning Key undermines every credential installed under it.

3.3 Integrator Key

The Integrator Key is installed under protection of the Provisioning Key. By primary intention it is the shared secret of one machine, and all nodes in the same machine hold the same Integrator Key. Where system design requires different handling, it may instead be a device-unique key, with the corresponding change in trust-domain scope. It has two purposes:

- it defines the machine's trust domain,
- it is one of the input bases from which session keys are derived.

We treat the Integrator Key as a group secret at machine scope and document its consequences explicitly. If one node is physically extracted and the Integrator Key is recovered, the entire machine trust domain is exposed. Replacement of a suspect node should therefore trigger replacement of the Integrator Key on every node in the machine.

⚠ SECURITY
By primary intention the Integrator Key is a machine-wide group secret: every node in the machine holds the same key, defining the machine trust domain. Where system design requires different handling, it may instead be device-unique, and the Operator Key may be handled the same way. In the machine-wide case, recovering one node's Integrator Key exposes the whole machine trust domain, so replacing a suspect node must trigger replacement of the Integrator Key on every node in that machine.

Where supported, the Integrator Key (or a Session Key derived from it) shall be used to protect default configurations. Configuration protection can be implemented by authenticating configuration blobs sent to devices or by locking / unlocking write access to configurable parameters.

3.4 Optional Operator Key

The Operator Key is an optional third base key, installed under protection of the Provisioning or Integrator Key. Where the Integrator Key authorizes site deployment, the Operator Key authorizes day-to-day runtime use and configuration. Separating the two keys lets an integrator hand a machine over to an operator without exposing the Integrator Key and lets the operator rotate the runtime credential without involving the integrator. Deployments that do not need this separation may skip the Operator Key and run all secure access under sessions derived from the Integrator Key.

The Operator Key is bound to the same trust domain as the Integrator Key, and installation, replacement and destruction follow the same rules. Like the Integrator Key, it is machine-wide by primary intention and may be made device-unique where the system design requires it.

3.5 Optional Key ID

The base keys are write-only and can never be read back, so a tool has no way to ask a device which key it currently holds. An optional Key ID closes that gap. A Key ID is a readable, non-secret identifier associated one-to-one with a stored base key: each Provisioning, Integrator or Operator Key may carry its own Key ID. The Key ID identifies the key but is not the key, and it reveals nothing about the key material.

Keys cannot be read out, but Key IDs can, even by a session that holds no valid key. This lets the integrator or operator keep a system-level key database that maps each Key ID to its key material, extending the key management system introduced under Device-Unique Provisioning Key. A tool reads the Key ID from a device, looks up the matching key in that database and proceeds, without the key ever crossing the bus.

The Key ID is optional in this model. It does not feed the key derivation function, it does not affect the derived session key and it is not transmitted with the protected traffic.

A device holds the Key ID in a readable parameter rather than carrying it with traffic. Reserved values indicate that no valid key is present. The exact width, reserved encodings and storage layout are left to the fieldbus profile; the CANopen example below gives a concrete instance.

3.6 Factory Restore

Recovery from a lost Provisioning Key or a suspected compromise uses two symmetric paths, both shown in the lower half of Figure 1. The first is an authenticated restore command sent over the bus, authorized by the current Provisioning Key. The second is a deliberate local action, for example a recessed reset button held for 10 seconds, or a hardware jumper that requires opening the enclosure. The local path requires physical possession of the device and provides recovery when the Provisioning Key has been lost.

Factory restore is destructive. It erases the Provisioning Key, the Integrator Key, any operational session key, all stored configuration and any cryptographic material derived from these. Where Key IDs are used, restore also resets every Key ID to the reserved no-key value, so a restored device reports no valid key in any slot. After restore the device returns to the same uncommissioned state in which it left the factory. Operational input

and output remain disabled until a new Provisioning Key has been installed. The device may retain a small, non-erasable reset event counter or timestamp for incident response, provided this record carries no sensitive material.

> **⚠ SECURITY**
> Factory restore is destructive and irreversible. It erases the Provisioning Key, the Integrator Key, any session key, all stored configuration and all derived material, and returns the device to the uncommissioned state with operational input and output disabled until a new Provisioning Key is installed.

Restore should be visible to the operator, for example through a defined LED pattern or a diagnostic bit, so that an unauthorized reset can be detected after the fact. We do not recommend any third recovery path. Every additional path is one more thing an auditor must evaluate and one more potential bypass.

4 Using the Keys

The base keys defined in the previous chapter never appear on the wire as the payload-protecting key. Every secure access between a client and a server uses a session key derived from the chosen base key (Provisioning, Integrator or Operator) and a salt established on the server at session start. The salt is a non-secret derivation input. The base key never leaves the device, the derived session key is never transmitted, and the downstream secure protocol provides replay protection, freshness and nonce uniqueness above this layer. Figure 2 shows the inputs to this derivation and the session key it produces.

> ⚠ **SECURITY**
> Base keys never appear on the wire and never leave the device, and the derived session key is never transmitted. Both peers derive the session key independently from the base key and the salt. Any implementation that sends a base key or a session key over the bus, even once and even encrypted, breaks the model.

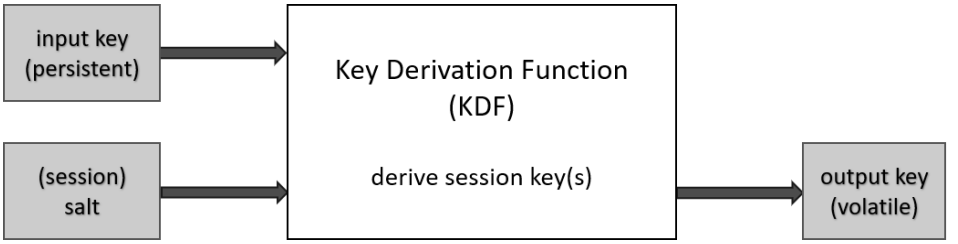

FIGURE 2: KEY DERIVATION FUNCTION

4.1 Salt Lifecycle on the Server

At every power-up and at every reset the server initializes its salt slot to zero. While the slot reads zero, no session key has been derived and no secure access is possible. The server accepts exactly one plain, unauthenticated write to the salt slot; that write sets the salt and locks the slot. Any further salt write is refused until the next power cycle or reset. The salt remains active for the lifetime of the power cycle. Its length is fixed by the applicable fieldbus profile.

> ⚠ **SECURITY**
> The salt slot accepts exactly one plain, unauthenticated write per power cycle; that write sets the salt and locks the slot until the next reset or power cycle. The server enforces this, not the client. There is no in-band salt rotation; a fresh salt requires a reset.

This rule is enforced by the server, not by the client. A client that needs a fresh salt on a running device must trigger a reset of that device by a cold restart or a documented service action. There is no in-band salt rotation.

> **⚠ SECURITY**
> The salt is non-secret and the salt slot may be read back for diagnostic purposes. Knowing the salt does not weaken the session key, which depends on the base key the attacker does not have.

4.2 Session Key Derivation

Once the base key and the salt are both present on the server, both peers compute the same session key in parallel by applying the key derivation function specified in the Cryptographic Primitives chapter to the base key and the stored salt. The KDF also accepts an optional context label, needed only where the same base key serves more than one purpose or peer that must derive distinct session keys; the per-server separation described in this section uses it.

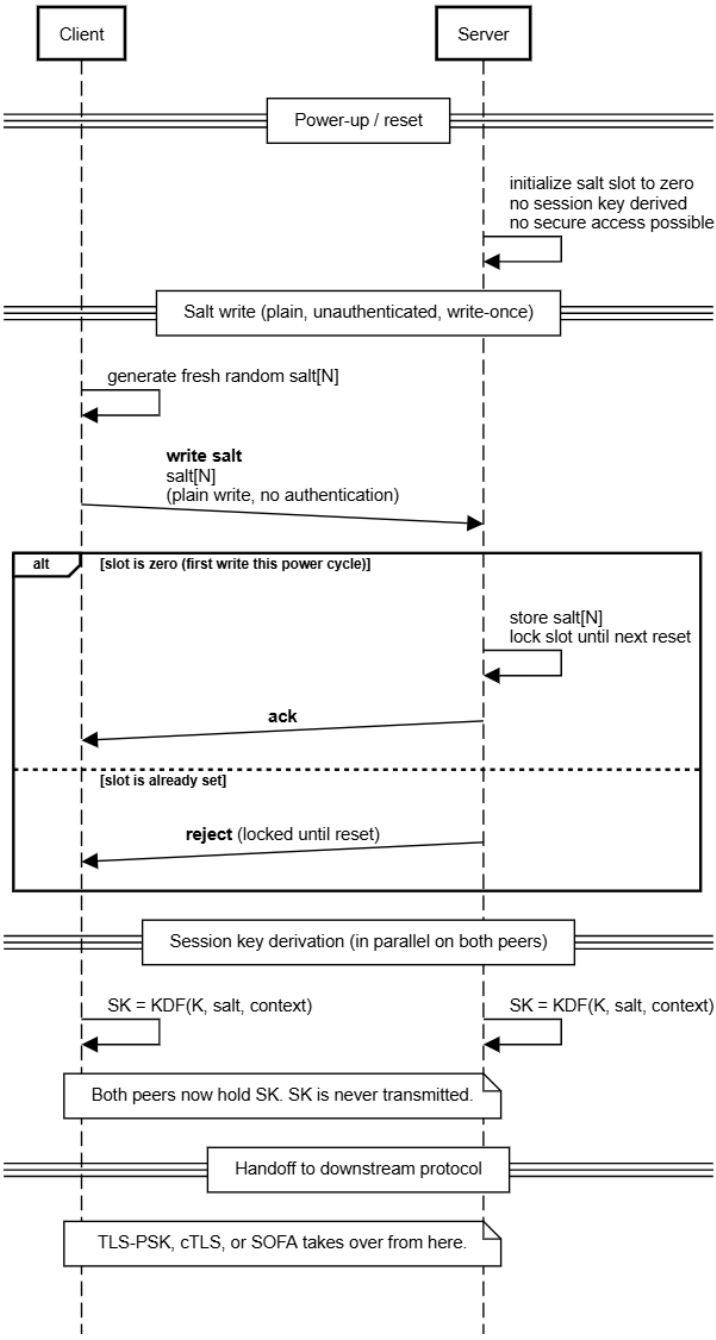

FIGURE 3: KEY DERIVATION WITH SHARED SALT

The derivation is deterministic; both peers reach the same session key without further exchange. The derived key is the key that the downstream protocol uses to protect data on the wire. The derived key is never transmitted.

The derivation deliberately does not bind any peer identifier. The same base key and the same salt always yield the same session key, regardless of which client or server runs the KDF. A client with constrained resources can therefore write the same salt to several servers in the same machine and reuse one derived session key across all of them: one base key, one salt, one derived key, one set of session buffers on the client.

The trade-off is the loss of per-server independence; a compromise of one server exposes the session key that protects every other server the client talks to under the same salt. Because the base keys are machine-wide and no peer identity is bound, two servers given the same salt derive the identical session key even when sharing was not intended, so if compromised, one can impersonate the other to the client for that power cycle. Deployments that need per-server independence must either give the client the budget for a per-server security context or bind a server-specific value, such as the node identifier, into the KDF context label so that each pairing yields a distinct session key; that value need not be secret, but the client must obtain it over a trustworthy channel because the fieldbus does not authenticate it.

Figure 3 shows the session-start sequence between one client and one server. The server boots with the salt slot at zero. The client writes a fresh salt to that slot in a plain, unauthenticated write. From that point on, both peers hold the same derived session key and the downstream protocol takes over. Because the salt write and its acknowledgement are unauthenticated, the client gains no assurance at this layer that the node acknowledging the write is the intended server rather than another holder of the same base key; that assurance comes only from the downstream protocol once protected exchanges begin.

4.3 Binding to a Downstream Protocol

The session key produced in this chapter is the pre-shared key for a downstream secure protocol. Three established options apply directly to constrained fieldbus systems:

- TLS-PSK (RFC 4279), when the platform supports the full TLS record layer and the frame budget allows the TLS handshake and record overhead.
- cTLS, the compact TLS profile, when frame budgets are tighter and a reduced handshake is acceptable.
- SOFA, the Secure Object Fieldbus Access profile published by EmSA, when the target is small-packet fieldbus traffic (CAN, CAN FD, CANopen, CANopen FD) and the

per-frame overhead must stay inside the bus frame. SOFA is specified in EmSA-WP-105.

Each of these protocols provides its own per-frame nonce construction, replay protection and counter handling. SOFA, for example, mixes a client random and a server random into every exchange and binds an explicit counter into the AEAD associated data. The white paper at hand does not restate those rules; it specifies only the key material and the salt that the downstream protocol consumes.

4.4 Threats Against the Salt Write

The salt write is intentionally unauthenticated. An attacker on the bus can write anything to the salt slot, but only once per power cycle and only before the legitimate client has written. The resulting threats are bounded:

- Pre-positioned salt. An attacker that writes the salt before the legitimate client comes online has not learned anything useful. The derived session key is still secret because it depends on the base key the attacker does not hold. The attacker cannot forge frames, cannot decrypt traffic and cannot recover the base key. The session is unusable to the legitimate client, which is the denial-of-service case.
- Salt replay forcing nonce reuse. Within a power cycle this is prevented by construction: the slot accepts exactly one write, so an attacker cannot replay an old salt to force the server into a previously used session-key regime during that cycle. Across power cycles the salt may recur, because it is client-generated and the server applies no uniqueness check, so the salt is not required to be unique across cycles. Nonce uniqueness for the protected traffic is guaranteed by the downstream protocol's per-frame nonce construction, not by the salt mechanism.
- Denial of service. An attacker that wins the race to write the salt locks the slot for the rest of the power cycle, so the legitimate client cannot communicate until the server is power-cycled. This deserves only a side note: any attacker with wire access to a fieldbus already has near-unlimited denial-of-service options, from frame flooding and bit-level corruption to physical disruption. The salt-write race is one more of these; it adds no new class of risk and, like the rest, is contained by the physical and operational trust boundary rather than by the protocol.

The salt write does not give the attacker any path to forgery, decryption or base-key recovery. An attacker who does not hold a machine base key gains only a single denial-of-service primitive per power cycle; an attacker who holds the relevant base key can additionally impersonate the intended server to the client, as noted under Session Key Derivation.

5 Cryptographic Primitives

Our recommended primitives for the symmetric-only profile are:

- KDF: HKDF-SHA-256, as specified in RFC 5869, with an optional context label to separate purposes or peers that share the same base key.
- AEAD: AES-128-GCM is the default, all other AEAD capable algorithms are a potential usable alternative. Authentication tags of 32 to 64 bits are acceptable on both classical CAN and CAN FD; longer (full 16-byte) tags are an option for deployments that need a wider per-key usage budget. A truncated tag of n bits gives a forgery probability of about 2^-n per verification attempt, so a deployment that truncates must bound the number of accepted verification failures and rekey within a per-key budget defined by the downstream protocol specification.
- Random: a true random number generator, or a cryptographically secure deterministic RNG seeded from one, with NIST SP 800-90B health tests, used for nonces and for Provisioning Key generation in the integrator's tool.
- Key sizes: 128 bits minimum for AEAD keys, with key storage provisioned for up to 256 bits where the platform allows, to accommodate larger keys or future algorithms.

This selection is consistent with BSI TR-02102 for constrained embedded systems.

6 Application Example: CANopen

Classical CANopen SDO communication carries at most 7 bytes per frame. A full 16-byte authentication tag significantly increases message size and the realistic security overhead must be smaller. We segment configuration traffic, provisioning traffic and session establishment over multiple frames, where the full AEAD overhead is acceptable. Process data uses a truncated tag, typically 32 or 64 bits, consistent with BSI TR-02102 guidance on tag truncation for time-critical short messages.

CAN FD carries up to 64 bytes per frame. The full AEAD overhead fits inside one frame for the majority of application messages, simplifying the design considerably. Where bandwidth allows, we recommend CAN FD for new designs that require authenticated encryption end to end. Tags of 32 to 64 bits remain a valid choice on CAN FD as well, for deployments that prefer the smaller per-frame overhead.

The Provisioning, Integrator and Operator Keys are installed and rotated using authenticated sessions on the bus. The factory commissioning marker is the well-known value that bootstraps the very first such session and is used only while the device is in the uncommissioned state.

For key installation and salt distribution Object Dictionary entries of type `DOMAIN` or `OCTET_STRING` and access type write-only can be used. Key installation is accepted only if all required conditions are met.

- The Provisioning Key can be installed only if the device is in the uncommissioned state.
- The Integrator Key can be installed (or overwritten) only if that write access is authenticated by the Provisioning Key.
- The Operator Key, if used, can be installed (or overwritten) only if that write access is authenticated by the Provisioning Key or the Integrator Key.
- Session keys are derived on both peers from the chosen base key (Integrator or Operator) and the server-side salt; they are never installed directly.
- The salt is written by the client to an Object Dictionary entry of type `OCTET_STRING` with write-once-after-reset access. The entry is zero after power-up; the first write sets it and locks it until the next reset.

A device that implements the optional Key ID exposes one readable Object Dictionary entry of type `UNSIGNED32` per Key ID, alongside each write-only key entry. The entry can be read even by an unauthenticated session, and the reserved values 0 and `0xFFFFFFFF` indicate that no valid key is installed in the corresponding slot.

7 Alignment With Standards and Regulations

7.1 IEC 62443-4-2

The three-layer hierarchy and its lifecycle map onto component requirements CR 1.5 (authenticator management), CR 3.1 (communication integrity), CR 4.1 (information confidentiality) and CR 4.3 (use of cryptography). The uncommissioned state supports a claim against CR 1.5: the device ships with no usable authenticator, and the first installed credential is unique per device. The Integrator Key delimits a machine-scope trust domain that maps onto a zone in the zone-and-conduit sense, consistent with IEC 62443-3-3.

7.2 EU Cyber Resilience Act

Annex I requires a secure default configuration and the absence of known exploitable vulnerabilities. A device that ships in an uncommissioned state, refuses to operate without a Provisioning Key and shares no cryptographic secret with any other device of the same model supports a claim against both requirements, provided the single command it accepts while uncommissioned, the Provisioning Key install, is reachable only in the isolated commissioning environment required above; that isolation, not the marker being non-secret, is what prevents an attacker from installing their own Provisioning Key. The factory commissioning marker is not a default password, because the device cannot perform any protected function before commissioning. Annex II responsibilities are split: the manufacturer documents the commissioning environment requirement and the recovery procedure, and the integrator commissions the device in that environment.

7.3 NIS2

NIS2 applies to operators of essential and important entities. Article 21 requires risk-appropriate technical and organizational measures, including access control, asset management, cryptography where appropriate, supply-chain security, maintenance security and physical protection. The provisioning model in this document gives the operator the technical building blocks to satisfy those obligations: a device-unique credential for the maintenance path, a machine-scoped integrator credential for the operational path, and destructive recovery for the end-of-life path.

7.4 BSI TR-02102

AES-128-GCM and HKDF-SHA-256 are within the cryptographic recommendations of BSI TR-02102 for constrained embedded systems. Tag truncation to 32 to 64 bits is consistent with the guidance for short, time-critical messages and is bounded by the uniqueness value carried in the authenticated payload.

8 Summary and Conclusion

We recommend a layered model for minimal fieldbus systems that cannot use asymmetric cryptography at production rates. A device ships in an uncommissioned state recognized by a factory commissioning marker that is not a secret. The integrator installs a device-unique Provisioning Key, then a machine-specific Integrator Key, and optionally an Operator Key for day-to-day runtime use, each in a trusted commissioning environment. The integrator may also record an optional Key ID per key, so the installed keys can later be identified for maintenance, audit and rollover. On every power cycle the client writes a salt to the server in a write-once-after-reset slot; both peers then derive a session key from the chosen base key and the salt. The derived session key is consumed by a downstream secure protocol (TLS-PSK, cTLS or SOFA as specified in EmSA-WP-105) that handles replay protection, freshness and nonce uniqueness above this layer. Factory restore is destructive and uses either the Provisioning Key or a deliberate local action, and the device returns to the same uncommissioned state in which it left the factory.

Applied to CANopen, this model fits naturally into an authenticated sublayer above CANopen. Applied to other small-packet fieldbus systems, the same lifecycle, the same key hierarchy and the same primitives carry across without modification. The result is a single, defensible provisioning pattern an integrator can apply across a portfolio of constrained fieldbus devices.

9 Figures: Sequence Diagram Sources

The sequence diagrams in this document, Figures 1 and 3, are produced by https://sequencediagram.org/ from the listings below. Figure 2 (Key Derivation Function) is a block diagram, not a sequence diagram, and has no listing here.

Figure 1:

```
title: Key Life Cycle Overview
participant "Commissioning Marker" as Marker
participant "Provisioning Key" as Prov
participant "Integrator Key" as Integr
participant "Operator Key" as Oper

==Uncommissioned state==

note over Marker, Oper: all keys and configuration erased, no operational I/O possible

==Install Provisioning Key==

Marker->Prov: install in trusted environment\n(physical possession)


note over Prov: device specific
==Install Integrator Key==

Prov->Integr: install\n(auth. by Provisioning Key)

note over Integr: machine specific

==Install Operator Key (optional)==

opt only if Operator Key is used
Integr->Oper: install\n(auth. by Integrator Key)
note over Oper: machine specific
end

==Decommission==

alt Path A: authenticated restore
  Prov->Marker: restore command\n(auth. by Provisioning Key)
else Path B: deliberate local action
Marker->Marker: physical reset button held 10 s or hardware jumper, no key required
end

note over Marker, Oper: all keys and configuration erased, device returns to uncommis-
sioned state
```

Figure 3:

```
title: Key Derivation with Shared Salt
participant Client
participant Server

==Power-up / reset==

Server->Server: initialize salt slot to zero\nno session key derived\nno secure access
possible

==Salt write (plain, unauthenticated, write-once)==
```

```
Client->Client: generate fresh random salt[N]

Client->(1)Server: **write salt**\nsalt[N]\n(plain write, no authentication)

alt slot is zero (first write this power cycle)
  Server->Server: store salt[N]\nlock slot until next reset
  Server->(1)Client: **ack**
else slot is already set
  Server->(1)Client: **reject** (locked until reset)
end

==Session key derivation (in parallel on both peers)==

parallel
Client->Client: SK = KDF(K, salt, context)
Server->Server: SK = KDF(K, salt, context)
parallel off

note over Client, Server: Both peers now hold SK. SK is never transmitted.

==Handoff to downstream protocol==

note over Client, Server: TLS-PSK, cTLS, or SOFA takes over from here.
```