Key Provisioning for Minimal Fieldbus Systems

Guideline on symmetric key provisioning for constraint embedded fieldbus devices

White Paper EmSA-WP-104

Version 1.27
21 July 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

© Embedded Systems Academy, 2026. Permission is granted to copy, distribute, and use this material for non-commercial educational purposes with attribution. Commercial use requires explicit permission.

The authors are not liable for defects or indirect, incidental, special, or consequential damages, including loss of anticipated profits or benefits, arising from the use of this document or warranty breaches, even if advised of such possibilities.

The information presented in this document is believed to be accurate. Responsibility for errors, omission of information, or consequences resulting from the use of this information cannot be assumed by the authors.

Contents

1 Scope and Objective 4
1.1 System Overview 5
1.2 Terms and Definitions 5
2 Device Capability Levels 8
3 Root of Trust Options 9
3.1 Devices With Asymmetric Capability 9
3.2 Symmetric-Only Constrained Devices 9
3.3 Two Ways to Anchor the Trust 9
3.3.1 Integrator-Rooted 9
3.3.2 Manufacturer-Rooted 10
4 Provisioning Lifecycle 11
4.1 Uncommissioned State and Device Claim Token 12
4.2 Device-Unique Provisioning Key 13
4.3 Optional Factory Device Identity 14
4.4 Claiming Ownership at Handover 15
4.4.1 Issuing the Ownership Voucher 15
4.4.2 Claim by Manufacturer Voucher 17
4.4.3 Claim by Token Gate 20
4.5 Integrator Key 21
4.6 Optional Application-Specific Keys 21
4.7 Optional Key ID 22
4.8 Re-Provisioning a Commissioned Device 22
4.9 Factory Restore 23
5 Delegation of Authority 26
6 Using the Keys 27
6.1 Binding to a Downstream Protocol 27

7 Security Profiles 28
8 Declaration and Discovery 29
9 Cryptographic Primitives 30
10 Application Example: CANopen 31
11 Alignment With Standards and Regulations 33
11.1 IEC 62443-4-2 33
11.2 EU Cyber Resilience Act 33
11.3 NIS2 34
11.4 BSI TR-02102 34
12 Summary and Conclusion 35
13 Figures: Sequence Diagram Sources 36

1 Scope and Objective

This document presents a device provisioning, ownership and key lifecycle model for minimal fieldbus systems. The model is symmetric at its core and spans an ordered ladder of device capability, from devices that can run no asymmetric cryptography at all to devices that can verify or produce an occasional signature and agree a key. It covers the full lifecycle from factory delivery to end of life: the ordered set of pre-shared keys, the operations that install each layer, the ownership and delegation model that authorizes those operations, and the graded recovery paths that return a device to its uncommissioned state. The model is fieldbus-independent; CANopen appears only as an illustration and carries no normative weight.

The core of the model needs no asymmetric primitive and is conformant with this document on the most constrained devices. Where a device can afford a public-key operation, the same model uses that capability to raise the assurance of ownership, identity and handover, without changing the symmetric session-key derivation and without introducing a certificate chain. The Device Capability Levels chapter defines the ladder, and the Security Profiles chapter names the combinations most deployments will choose. Where a device can afford a public-key signature, an optional factory device identity lets the integrator verify the device at handover and anchors the provisioning chain to that proof, as described under Optional Factory Device Identity.

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).

One class of security layer above this key model targets single, selected accesses: service parameters, configuration objects and infrequently invoked functions such as firmware activation, configuration locking or mode switch. 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.

1.2 Terms and Definitions

This section fixes the vocabulary used throughout. Where a term belongs to the wider CAN security vocabulary, cansecurity.net carries the fuller treatment; where this document defines a term of its own, the chapter that introduces it gives the detail.

- **Device capability level.** The classification, C0 to C3, of what a device's cryptography can do: C0 symmetric only, C1 can also verify a signature, C2 can also produce one from an identity of its own, C3 can also agree a key. Each higher level is a superset of the one below.
- **Uncommissioned state**. The state a device ships in: it holds no credentials and refuses every bus command except the first Provisioning Key install.
- **Commissioning environment**. The secure, isolated bench, or new-machine bus with no other node powered, on which provisioning is performed.
- **Device Claim Token.** The value a device accepts while uncommissioned to authorize the first Provisioning Key install, and only then. It may be a well-known default, or a per-unit secret printed on the device or supplied with the delivery paperwork. It is not a base key, not a password and not an operational secret.
- **Token gate.** The claim method in which the device accepts the first install from an installer that presents the correct Device Claim Token. Whether the token is a well-known value or a per-unit secret is a provenance choice, not a different mechanism.
- **Ownership voucher**. The manufacturer-signed statement binding one device serial to one integrator public key.
- **Voucher gate.** The claim method in which the device accepts the first install only from the integrator named in a manufacturer-signed ownership voucher, verified against the manufacturer trust anchor. Requires at least capability level C1.
- **Owner epoch**. The monotonic counter carried in a voucher. A device refuses any voucher at or below the highest epoch it has already accepted.

- **Owner public key.** The integrator public key that a device records when it is claimed through the voucher gate, and against which it later verifies owner-authorized operations. See Ownership Transfer.
- **Ownership credential.** The credential a device retains that authorizes owner-level operations after commissioning, such as re-provisioning. It is the Provisioning Key at C0 and C1, and the owner public key at C2 and C3. It authorizes lifecycle operations and never protects operational traffic.
- **Delegation.** A fixed, authority-signed statement that grants a named delegate a scoped subset of the owner's authority, so that a tool need not hold the ownership credential itself. Defined in the Delegation of Authority chapter.
- **Integrator-rooted**. The anchoring model in which the device is born open and the integrator installs every root at first commissioning.
- **Manufacturer-rooted**. The anchoring model in which the device is born locked, carrying a factory device identity and the manufacturer's trust anchor.
- **Factory device identity**. The manufacturer-installed signature keypair. Permanent, and not erased by factory restore. IEEE 802.1AR calls it an IDevID.
- **Local device identity**. The integrator-installed signature keypair. Erased by factory restore. IEEE 802.1AR calls it an LDevID.
- **Trust anchor.** The public key a verifier holds in order to check a certificate or a signature.
- **Trust domain.** The set of nodes that share a common root of trust, typically one machine or installation, within which a credential is honored.
- **Base key**. The collective name for the Provisioning, Integrator and Operator Keys.
- **Provisioning Key.** The device-unique base key installed first, which authorizes installation of the Integrator Key and is the ownership credential at capability levels C0 and C1. It never protects operational traffic.
- **Integrator Key.** The base key installed under the Provisioning Key that defines a machine's trust domain and is a base for further key derivations. Whether it is shared across the machine, unique per device or ladder-derived is the integrator's choice.
- **Operator Key.** An optional application-specific base key installed under the Provisioning or Integrator Key.
- **Key ID**. The optional readable, non-secret identifier of an installed base key.
- **Re-provisioning.** The owner-authorized replacement of key material on a device that is already commissioned, by the party that already owns it. It crosses no ownership boundary and is distinct from commissioning.
- **Factory restore**. The destructive return to the uncommissioned state, which erases the base keys, the local device identity and all configuration.
- **Restore depth.** One of the four graded levels, D1 to D4, to which a factory restore erases device state, from clearing delegations only up to returning the device to unowned.

- **Security profile.** A named, recommended combination of capability level, claim gate and protection choices, so that a device and a tool can agree what to expect in a single declaration. Defined in the Security Profiles chapter.

2 Device Capability Levels

The provisioning model must serve devices of very different cryptographic means, from a constrained MCU with a software AES to a controller with a hardware signature engine. Rather than fork the model per device, we classify a device by an ordered capability level and let every later choice, the claim gate, the protection primitive and the recovery policy, refer to that level. The levels are ordered by hardware cost, so a device implements the highest level it can afford and inherits everything below it.

Level	Adds the ability to	Private key on device	What it enables
C0	symmetric AEAD and key derivation	no	the symmetric baseline
C1	verify a signature it is given	no	voucher-gated ownership, signature-authorized commands, delegation
C2	sign with an identity of its own	yes	device identity: attestation, anti-counterfeit, non-repudiation
C3	agree a key	yes	confidential key install

The most consequential boundary is between C1 and C2, because it is the boundary between authorization and attestation. A C1 device can enforce that only an authorized party acts on it, yet cannot prove its own identity to anyone. Many industrial deployments need the first property and not the second, and C1 lets them buy ownership control and delegated maintenance without the cost, and the key secrecy burden, of an on-device private key. A device that can verify but not sign is a deliberate and useful design point, not an incomplete one.

Capability is a property of the device and does not by itself decide who installs the roots of trust or how strictly ownership is controlled. Those are the separate choices of the following chapters. The Security Profiles chapter draws capability level, claim gate and protection together into the named combinations a deployment actually selects.

3 Root of Trust Options

The first design decision is the root of trust.

3.1 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.

Even a constrained device may still be able to compute an infrequent public-key signature where it cannot run a continuous asymmetric handshake. That capability is what the manufacturer-rooted model below rests on.

3.2 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.

3.3 Two Ways to Anchor the Trust

Device capability decides which primitives are available. It does not decide who installs the roots of trust, or when, and that is a separate choice with two ends. The two models below are the ends of a spectrum, not a menu: a device implements one of them.

3.3.1 Integrator-Rooted

The device is born open. It leaves the factory carrying no credential, the integrator installs every root at first commissioning, and the device locks itself afterwards. The manufacturer retains only firmware update authority. This is the simpler model for the manufacturer, and it is the one the rest of this document specifies by default. Its cost is a window of limited protection: from the moment the device leaves the production line until the integrator locks it, nothing but physical and supply-chain control stands between an attacker and ownership of the device. It fits a single integrator running a closed deployment. It needs no asymmetric primitive at all and is fully specified by the chapters that

follow, so a device that cannot afford a signature primitive has only this model available to it and remains conformant with this document.

3.3.2 Manufacturer-Rooted

The device is born locked. It leaves the production line already carrying a factory device identity and the manufacturer's trust anchor, so there is no window in which the device is open and unprovisioned. The integrator does not install trust into a blank device; it verifies an already-trustworthy one, claims it, and layers its own keys on top. This fits broad markets, several integrators drawing from one production run, and deployments where counterfeit devices are a real concern. It costs the manufacturer a signature on the production line and a trust anchor to maintain for the life of the product. IEEE 802.1AR is the established reference for the credential this model rests on, and names it an IDevID. The model is specified in the Optional Factory Device Identity and Ownership Transfer subsections below.

4 Provisioning Lifecycle

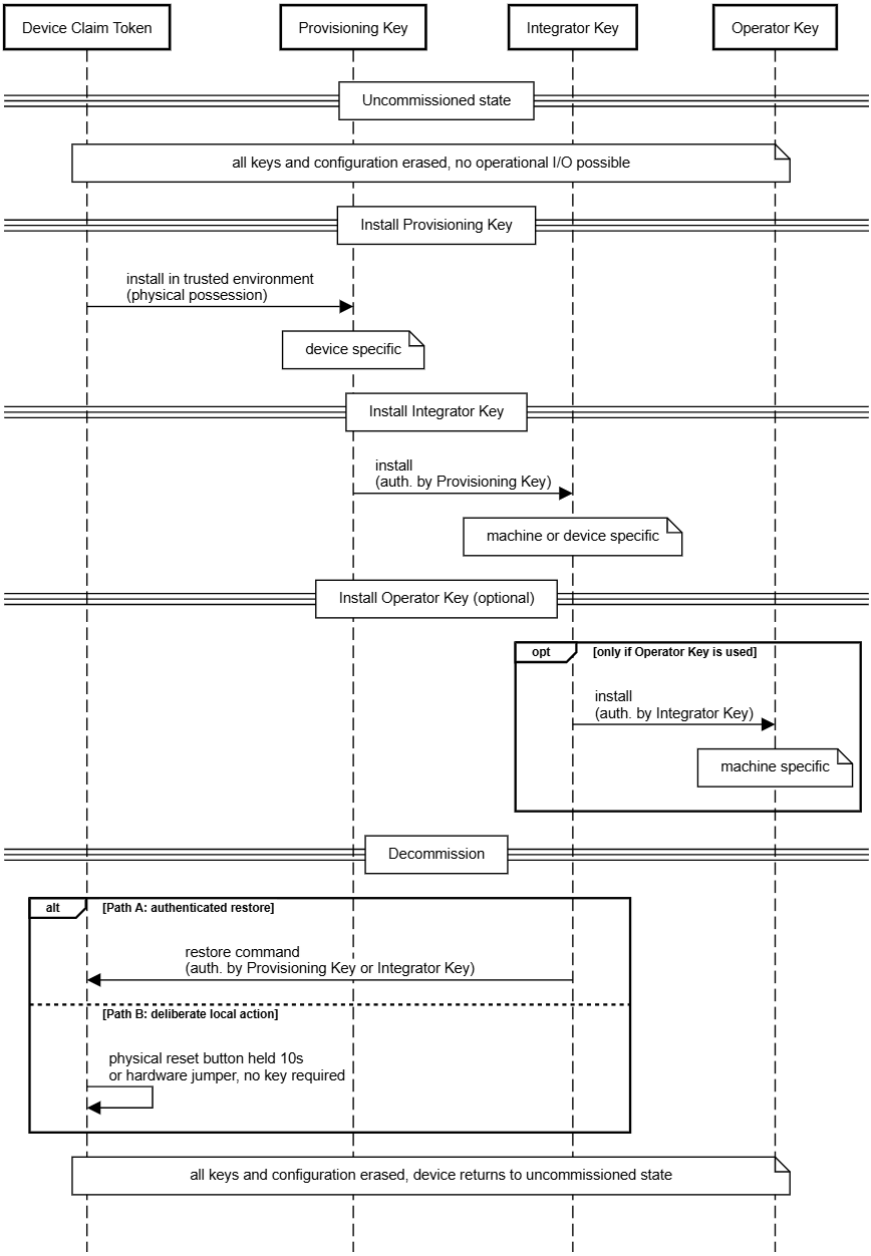


FIGURE 1: KEY LIFE CYCLE OVERVIEW

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 (the Operator Key may be installed under protection of the Provisioning or the 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 Using the Keys chapter.

Every authenticated install arrow further implies that the downstream protocol establishes its session or working key on the bus, omitted from Figure 1 for clarity.

4.1 Uncommissioned State and Device Claim Token

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 value the device recognizes in this state the Device Claim Token. It is not a base key, not a password and not an operational secret: it authorizes only the first Provisioning Key install while the device is uncommissioned, and becomes inert once provisioning has completed. Its provenance is a claim-gate choice described under Ownership Transfer: a well-known non-secret default, as simple as an unprotected write, or a per-unit token printed for one device.

Naming this value explicitly matters for audit and conformity assessment. 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 IEC 62443-4-2 and the CRA 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 Device Claim Token alone, 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.

4.2 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,
- serve as the ownership credential for owner-authorized operations the deployment defines, such as re-provisioning,
- bind any device-specific reset or re-keying operation to a credential held by the integrator.

At capability levels C0 and C1 the Provisioning Key is the device's ownership credential, retained rather than spent at commissioning. It authorizes lifecycle operations only and never protects operational traffic; session keys derive from the Integrator or Operator Key. This separation keeps ownership distinct from operation, so recovering a key that protects runtime traffic does not, by itself, let an attacker take ownership. At capability levels C2 and C3 the owner public key established during the claim takes over that role, and the Provisioning Key install is authorized by the voucher chain instead.

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 same label can instead carry a printed one-time token that gates the ownership claim without pre-installing the Provisioning Key; see Ownership Transfer.

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 (KMS). 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 must occur in an isolated commissioning environment. 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 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.

4.3 Optional Factory Device Identity

A device may carry an optional factory device identity that sits below the symmetric key hierarchy as its root anchor. This identity is a long-term signature keypair installed at manufacturing. Its private half never leaves the device, and its public half is not secret and is verifiable against the manufacturer's trust anchor. The identity is independent of the base keys and of the commissioning state: a device can prove its identity while uncommissioned, while commissioned or after a factory restore. Factory restore does not erase it.

The factory device identity does two things. First, it authenticates the identity read: a tool that reads the factory device identity receives a value signed by the device's private key, so the tool confirms it is talking to a genuine, specific device rather than an impostor on the bus. Second, it anchors the installation of the Provisioning Key. The device signs the install, so the commissioning tool binds the unique Provisioning Key to a verified device rather than trusting the bench alone. Because the Provisioning Key is unique per device, and every later base key installs under protection of the previous one, and every session key derives from a base key, the trust of every secure session traces back to this anchored install. A single verified handover therefore roots the whole symmetric chain.

The factory device identity authenticates provisioning and identity; it does not protect confidentiality or add key agreement. It does not encrypt the Provisioning Key value, which remains readable on the bus during the install and therefore still relies on the isolated commissioning environment for confidentiality. It does not add forward secrecy, and it does not by itself authenticate individual runtime sessions.

The factory device identity is optional. A device that does not carry one is conformant, and its commissioning relies on the isolated environment alone. Where the identity is used, the manufacturer generates the keypair during production and records or publishes the public key so that integrators can trust it, for example through a manufacturer trust anchor or an out-of-band identity record. Revocation is handled at that trust layer, not on the device. The signature primitive is given in the Cryptographic Primitives chapter, and the on-the-wire construction of the signed identity read and the anchored install

is defined by the downstream protocol; EmSA-WP-105 specifies it for CANopen and CANopen FD.

> ⚠ **SECURITY**
> The factory device identity private key must never be readable and must never leave the device. Only the public key is shared. The manufacturer generates the keypair during production, and factory restore does not erase it, so the device keeps a stable identity across its whole life. A device that exports or accepts a factory device identity private key breaks the root anchor the rest of the provisioning chain depends on.

4.4 ~~Claiming Ownership at Handover~~Ownership Transfer

Claiming ownership is the step that turns a factory-fresh unit into a device that belongs to one integrator and refuses every other. A device gates the claim by one of two methods, chosen at production, that trade friction against assurance. The token gate accepts the first Provisioning Key install from an installer that proves knowledge of the Device Claim Token, and its assurance depends on the token's provenance: a well-known token is trust on first use, where the isolated commissioning environment is the only control, while a per-unit printed token raises the bar to possession of a secret the manufacturer printed for that one device. The voucher gate accepts the install only from the integrator named in a manufacturer-signed ownership voucher, which the device verifies against a trust anchor and which requires capability level C1 or above. The voucher gate is described first, then the token gate; both end at the same point, an integrator-owned device holding its own Provisioning Key.

4.4.1 Issuing the Ownership Voucher

A device that ships born locked accepts an owner only on the strength of an ownership voucher: a short manufacturer-signed statement, naming one device serial, one integrator public key and an owner epoch, over which the manufacturer signs. It travels with the goods and is not secret; its value is that it cannot be forged, not that it is hidden. The construct is adapted from the voucher of RFC 8995 (BRSKI), reduced to what a device with no network stack and no clock can verify.

The voucher comes into existence off the bus, before commissioning, as Figure 2 shows. At manufacture the maker installs the device's factory identity and records its serial against that identity in an ownership database. When ownership transfers, the integrator's tool requests a voucher for the device, presenting the serial, its own integrator public key and evidence of ownership; the manufacturer verifies that evidence, allocates an owner epoch higher than any it has issued for that device before, builds the voucher and

signs it. The signed voucher is delivered to the integrator out of band, ready to be re-deemed at handover.

The owner epoch is what makes a change of hands safe. Because a local factory restore needs no key, a previous owner holding an old voucher could otherwise re-claim a resold device. The device stores the highest epoch it has ever accepted in a counter that factory restore does not erase, and refuses every voucher at or below it, so the moment a new owner's higher-epoch voucher is accepted the previous owner's voucher stops working.

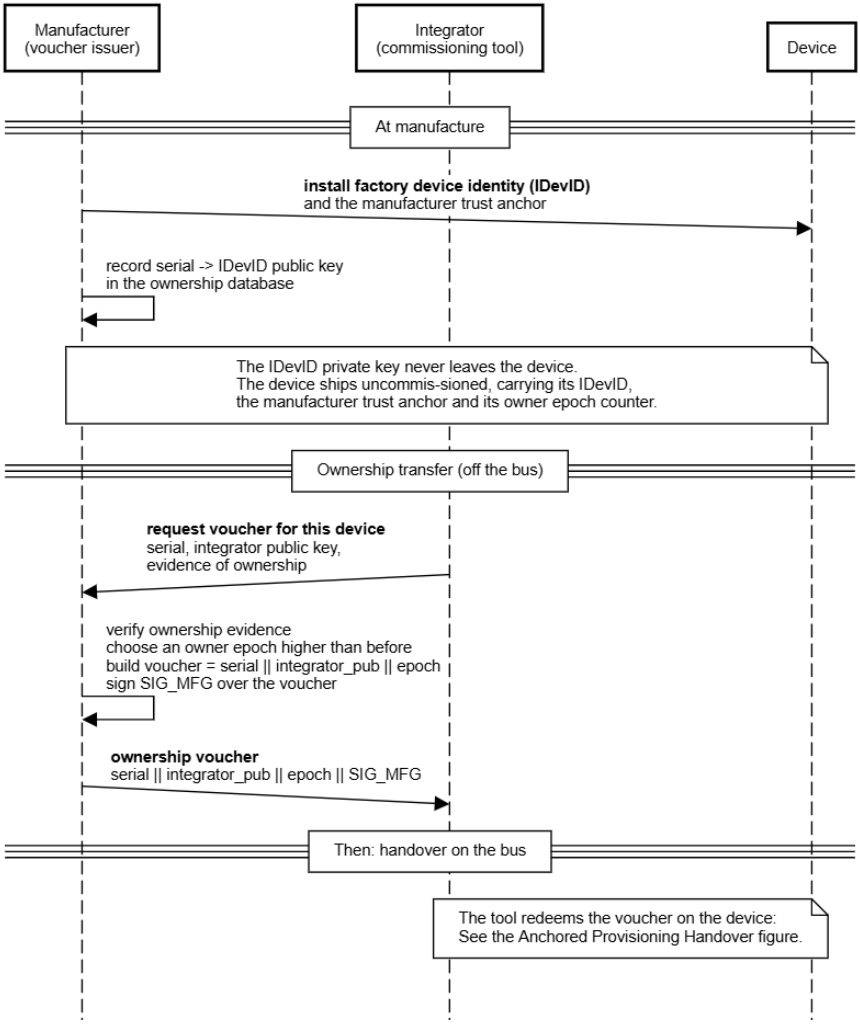


FIGURE 2: OWNERSHIP VOUCHER ISSUANCE

4.4.2 Claim by Manufacturer Voucher

With the voucher issued as above, the integrator redeems it on the bus. The factory device identity answers the genuineness question of any handover: is this a genuine device of the expected type and serial? The integrator's tool reads the signed identity and validates it against the manufacturer's trust anchor before it trusts the unit with anything. A device that carries no identity cannot answer that question cryptographically, and the integrator falls back on physical provenance and supply-chain control instead.

A fieldbus device has no reliable time source, so it cannot check a voucher's expiry, and a voucher is therefore a bearer statement that stays technically valid for the life of the device. This document does not answer that with a clock. It answers it with the state machine already defined for the uncommissioned state: a voucher is accepted only while the device is uncommissioned, and only once. Replaying a voucher to a commissioned device is refused. Replaying it to a restored device re-installs the same integrator the voucher already named, which gains an attacker nothing.

The handover assumes three things are already in place. The device holds its factory device identity keypair, the manufacturer trust anchor and its owner epoch counter, and it is uncommissioned and holds no base keys. The commissioning tool holds the same manufacturer trust anchor, obtained out of band, and an ownership voucher for this device. The trust anchor is a public key, so the tool verifies against it locally and never contacts the manufacturer during commissioning: the isolated environment required under Device-Unique Provisioning Key stays isolated. Revocation is handled at the trust layer, before the tool goes to the bench. The four steps then run as follows, and Figure 3 shows them in sequence. Every check in them is fail-closed: any verification that does not pass aborts the handover, and the device stays uncommissioned, unclaimed and without an installed key. Figure 3 does not draw those abort branches.

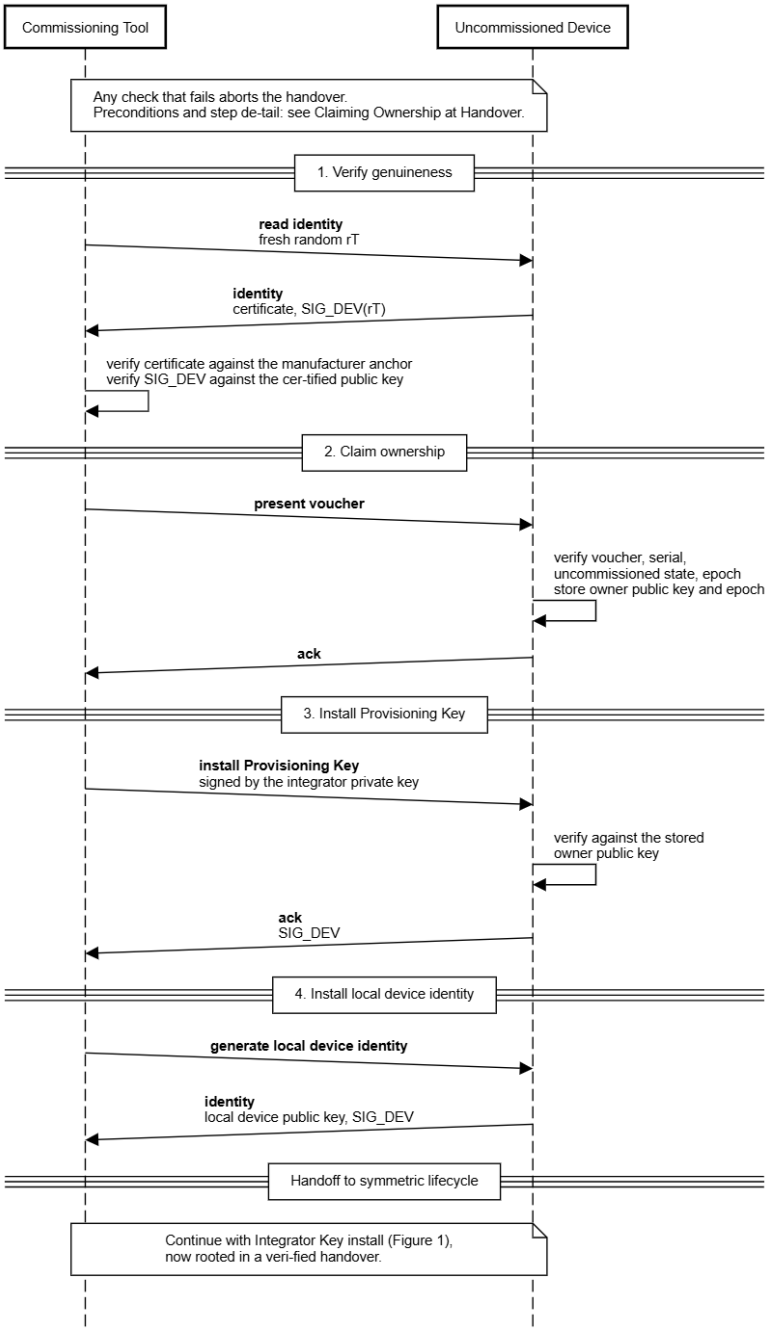


FIGURE 3: ANCHORED PROVISIONING HANDOVER

1. **Verify genuineness**. The tool reads the factory device identity, passing a fresh random number. The device returns its manufacturer-signed certificate and a signature over that random number. The tool verifies the certificate against the manufacturer trust anchor and the signature against the certified public key. The random number is what makes the answer fresh: without it, a response recorded from a genuine device could be replayed by an impostor.
2. **Claim ownership**. The tool presents the ownership voucher. The device verifies the manufacturer's signature on it against the same trust anchor, and checks that the serial names this device, that it is still uncommissioned, and that the owner epoch is above the highest epoch it has stored. On success it stores the integrator public key as its owner and the new epoch, and acknowledges.
3. **Install the Provisioning Key**. The tool installs the unique Provisioning Key, signed by the integrator private key. The device verifies that signature against the owner public key it stored in step 2, so the install is accepted only from the integrator the voucher named. The device acknowledges with a signature of its own, which lets the tool confirm the key reached the device it verified in step 1.
4. **Install the local device identity**. The device generates its own local device identity keypair and signs the new public key with its factory device identity before exporting it, so the integrator knows the key it records was generated on the device it has just verified. The integrator's key management system anchors it from there.

The order of the last two steps follows the rule this document applies to every other credential: each one installs under the protection of the previous one. The Provisioning Key is the first credential the device accepts from its new owner, and the local device identity then installs under it, exactly as the Integrator Key does. IEEE 802.1AR calls the two credentials an IDevID and an LDevID respectively. The factory device identity (IDevID) stays on the device and stays valid: it remains the proof of what the device is, and it survives factory restore so the device can be claimed again later. The local device identity (LDevID) is what the integrator's own tooling trusts from that point on, and factory restore erases it with the rest of the integrator's credentials. IEC 62443-2-4 places the onboarding responsibility this subsection describes on the integrator.

The claim authenticates possession; it does not make the handover confidential. The Provisioning Key value still crosses the bus in the clear during the install, and the isolated commissioning environment required under Device-Unique Provisioning Key remains the control that protects it. Encrypting the install would need a key agreement primitive in addition to the signature primitive which is out of scope for this document.

> **⚠ SECURITY**
> A voucher is unforgeable, not secret, and the device has no clock with which to expire it. Its freshness comes only from the rule that a voucher is accepted once, and only while the device is uncommissioned. A device that accepts a voucher in any other state, or accepts a second one, hands an attacker a path to take ownership of a device that is already in service.

4.4.3 Claim by Token Gate

The token gate closes the trust-on-first-use window with a symmetric secret and needs no asymmetric primitive, so it is available at capability level C0. In its well-known provenance the Device Claim Token is a published default and the gate reduces to trust on first use, where the isolated commissioning environment is the only control. In its per-unit provenance the manufacturer generates a distinct token for each device, prints it on the enclosure label described under Device-Unique Provisioning Key, and ships the device uncommissioned but holding that token. The device then accepts the first Provisioning Key install only from an installer that proves knowledge of the token, so an attacker who reaches the bus first but does not hold the token is refused. The integrator then installs its own Provisioning Key exactly as in the voucher path, and the per-unit token is spent: like a voucher, it is accepted once and only while the device is uncommissioned.

The printed one-time token and the voucher solve the same problem and differ in what each side must hold. The token needs no public-key operation on the device, which suits the most constrained parts, but the manufacturer must generate, print and retain a per-unit secret for every device, and the token sits on a label that physical access can read. The voucher needs an asymmetric verify on the device but the manufacturer holds only its signing key, and the voucher can be handed on in the open because it cannot be forged. A device that already carries a factory device identity has no reason to add a token; a device that cannot carry one gains most of the voucher's assurance from a token, at the cost of a per-unit secret database on the manufacturer's side.

The label delivery described under Device-Unique Provisioning Key is the degenerate form of this gate, in which the printed secret is the Provisioning Key itself and the device ships already provisioned rather than uncommissioned. That form is simpler, but the integrator then holds a key the manufacturer generated and could retain. Treating the printed secret as a printed one-time token instead, and letting the integrator install its own Provisioning Key after the claim, keeps the Provisioning Key the integrator's own secret, matching the ownership property of the voucher path. Both forms are acceptable, and the choice is the same out-of-band-delivery trade-off recorded under Device-Unique Provisioning Key.

> **⚠ SECURITY**
> A printed one-time token is a per-unit secret on a label, so it is only as strong as the physical control over the device before commissioning, exactly like the Provisioning Key label. Print it only where tamper evidence protects it, treat it as spent once the device is claimed, and never reuse one token across units. A token shared across a batch reduces the whole batch to trust on first use.

4.5 Integrator Key

The Integrator Key is installed under protection of the Provisioning Key. It may be shared by every node in one machine, unique per device, or derived through a ladder scheme. Which of these applies is a design decision for the integrator; this document neither assumes nor requires a particular choice, and neither does any implementation built on it. It has two purposes:

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

The choice trades key-management simplicity against exposure.

> ⚠ SECURITY
> Whether the Integrator Key is shared across a machine, unique per device or ladder-derived is the integrator's design decision, and no part of this model depends on a particular choice. Where it is shared, 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 it is device-unique, that blast radius does not apply.

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

4.6 Optional Application-Specific Keys

Beyond the two mandatory base keys, a system may install optional application-specific keys, and the Operator Key is the common example. It is installed under protection of the Provisioning or Integrator Key. Where the Integrator Key authorizes site deployment, an Operator Key authorizes day-to-day runtime use and configuration; separating the two lets an integrator hand a machine to an operator without exposing the Integrator Key, and lets the operator rotate the runtime credential without involving the integrator.

This is one option, not a requirement. A deployment may omit it and run all secure access under sessions derived from the Integrator Key, use a different application-specific key in its place, or define several such keys, each installed under or derived from the Integrator Key in the same way and bound to the same trust domain.

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, whether it is shared across the machine or device-unique is a design choice for the deployment.

4.7 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.

4.8 Re-Provisioning a Commissioned Device

The lifecycle so far installs each key once, at commissioning, across the ownership boundary the claim establishes. Deployments also replace key material on a device that is already owned: a component is swapped, a suspected key is rotated, a maintenance action re-keys a node. This operation crosses no ownership boundary and the same owner performs it repeatedly over the device's service life. It is therefore authorized differently from the first claim.

Re-provisioning is authorized by the ownership credential, the Provisioning Key at C0 and C1 or the owner public key at C2 and C3, and never by an operational key. It is permitted while the device is commissioned, it increments the owner epoch, and it is bound to the device identity and the new epoch so that a captured transfer cannot be replayed onto another device or rolled back to an earlier owner.

When a replaced component is removed, it leaves the machine still holding valid keys, and it may be resold, returned or examined by an attacker later. We therefore recommend that a component swap re-key every node in the trust domain and increment the domain's owner epoch, not only the new node. Bumping the epoch across the domain makes the departed node's keys stale by construction, which delivers revocation without a revocation list on any device.

> ⚠ SECURITY
> Re-provisioning must be authorized by the ownership credential, never by a key that protects operational traffic. If an operational key could authorize re-provisioning, a compromise of runtime traffic would become a change of ownership, and an attacker who recovered a session-protecting key could lock the legitimate owner out.

4.9 Factory Restore

Factory restore returns a device toward its uncommissioned state. It is graded by depth, so that routine maintenance need not erase ownership and an end-of-life release can. Its trigger is an application choice, because the same act that recovers a device an owner has locked itself out of is also the act a thief would use. The mechanics below define what each depth erases and what it must preserve; the depth and trigger policy a given product adopts are recommendations here, not requirements.

Device state falls into three storage classes, and restore is defined against them:

Class	Contents	Erased by restore
Immutable	serial, Device Claim Token, factory device identity, manufacturer trust anchor	never, at any depth
Monotonic	owner epoch	never cleared, only incremented
Erasable	base keys, ownership credential, owner public key, local device identity, delegations, field configuration	yes, in the graded amounts below

The four restore depths clear progressively more of the erasable class:

Depth	Additionally clears	Leaves	Typical trigger
D1	delegations	ownership and keys	staff turnover, lost tool
D2	operational key material	ownership	key rotation, component swap
D3	all field configuration	ownership credential (still owned)	return to unconfigured
D4	the ownership credential	immutable material and owner epoch (unowned)	resale, end of life

Two rules hold across every depth. A restore never erases the birth material, so a restored device returns to exactly the capability and identity it shipped with, never below it; an attacker cannot downgrade a device by resetting it. And the owner epoch increments on every restore regardless of depth, so that no previously issued authorization survives any restore. The difference between D3 and D4 is the anti-theft choice: D3 leaves the device bound to its integrator and useless to a thief, while D4 makes it genuinely off the shelf.

The restore trigger is application-specific, and we recommend matching it to the deployment rather than fixing it. Four settings cover the range: disabled, for permanently bound devices returned only through the manufacturer; presence only, for a cabinet where opening the enclosure is already meaningful; authorization only, for fleets and rented equipment where a physical trigger would be a liability; and both, for high-assurance field service. Only the last needs a mechanism beyond the others, because a command cannot prove that someone is standing at the device: presence is expressed as a physical action that opens a bounded window during which an authorized restore is accepted, not as the physical action performing the erase. We recommend that any restore require at least the authority that re-provisioning requires, the ownership credential, since restore is strictly more destructive, and that only D4 be reachable by physical presence alone, as the recovery path for an owner that has lost its ownership credential.

A physical restore is inherently safe at C2 and above and inherently exposed at C0 and C1. A restored C2 device still holds its manufacturer anchor, so a stolen unit is blanked but cannot be re-commissioned by the thief; a restored token-gated device needs only the Device Claim Token, which a thief may have taken with the hardware.

A full restore, depth D4, is destructive. It erases the Provisioning Key, the Integrator Key, the ownership credential and the owner public 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 a D4 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. The factory device identity is not erased by any restore: it is a permanent identity that survives into the uncommissioned state, so a restored device can still prove what it is and be re-commissioned through an anchored handover. The local device identity, by contrast, is erased with the rest of the integrator's credentials. The owner epoch is not erased either, so a restored device cannot be re-claimed with a voucher that an earlier owner still holds.

> **⚠ 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.

5 Delegation of Authority

Delegation lets the owner grant a tool a scoped subset of its authority, signed by the owner, so the tool can act without holding the ownership credential, which is what a maintenance workforce of many changing tools needs. It is one level deep, never a chain: the device checks the owner's signature on the delegation and the delegate's signature on the command, two checks against a fixed structure with no certificate path to build, so a verify-only device at capability level C1 can enforce it without a private key of its own.

The scope is a bitmask over the model's own privileged operations, claim, install key, re-provision, restore at each depth, and lock or unlock configuration, so a maintenance tool can be allowed to re-provision and perform a shallow restore yet forbidden to decommission or re-claim the device. Each delegation binds to a device serial or a group; an unrestricted delegation is not representable, since it would be indistinguishable from handing over the owner's key.

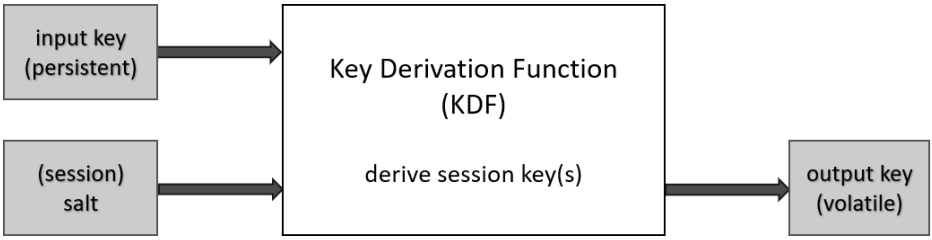
> ⚠ SECURITY
> A device with no reliable clock cannot enforce a delegation's expiry, so a stolen tool's delegation keeps working until the affected devices are re-keyed. Revocation is therefore the owner-epoch increment already used for re-keying, not a validity period; bound every delegation by scope and by serial or group.

6 Using the Keys

The base keys defined in the previous chapters are never used on the wire as the payload-protecting key. A base key stays in the device; the key that protects traffic is a working key that the downstream secure protocol derives from a base key. Both peers derive it independently, so nothing secret crosses the bus for a protected exchange to begin.

> ⚠ SECURITY
> Base keys are never used on the wire and never leave the device. Both peers derive it independently. A base key crosses the bus only during its own authenticated install, under protection of the previous base key. Any implementation that transmits the derived working key, or protects traffic with a base key instead of a session key, breaks the model.

How that working key is derived, and how freshness and replay protection are provided above it, is defined by the downstream secure protocol, not by this document, because the mechanism differs from one protocol to another. EmSA-WP-105 specifies one such mechanism, including its salt lifecycle, key-derivation formula, session-key reuse trade-offs and salt-write threat model; SPsec specifies another. This document fixes only the base keys that such a protocol consumes.



6.1 Binding to a Downstream Protocol

The working key the downstream protocol derives from a base key is the pre-shared key for that 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.
- The secure object-access profile of EmSA-WP-105, 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.

7 Security Profiles

Capability level, claim gate and the protection of operational access are independent choices allowing complex configuration combinations. To keep the common cases simple, we name six recommended combinations, called profiles, so that a device can declare one value and a tool can select against it. A device that matches no named profile reports the custom profile, and a tool then reads its individual capabilities. The six profiles, from least to most capable, are:

- Profile 0, Custom.
  Any combination not covered by a named profile. A tool must inspect the individual capabilities before it acts.
- Profile 1, Open.
  Capability C0, known, shared Device Claim Token. The controlled-environment baseline.
- Profile 2, Claimed.
  Capability C0, per-unit Device Claim Token. Ownership control with no asymmetric cryptography and no new silicon; the difference from Open is a manufacturing step. This is the recommended default for stationary industrial deployments.
- Profile 3, Authorized.
  Capability C1, token or voucher gate, signature-authorized commands and delegation, with no private key on the device. This is the level that adds cryptographic authorization and scoped field maintenance without an on-device secret.
- Profile 4, Identified.
  Capability C2, voucher gate, device identity. Adds attestation, anti-counterfeit and attribution.
- Profile 5, Confidential.
  Capability C3, voucher gate, confidential key install. Adds confidentiality of installed key material, for field re-keying or a bus segment on which a peer node may be hostile.

Each profile adds exactly one property to the one before it. The step that costs real hardware is from Profile 2 to Profile 3, where a device gains a signature-verification engine; the step from Profile 4 to Profile 5 is the one to weigh against need, since a plaintext install is acceptable wherever the commissioning environment is controlled.

8 Declaration and Discovery

Before any security exists on the wire, a tool must learn what a device can do. A device declares its profile and its capability level, and a tool reads that declaration first and, if it recognizes the profile, needs nothing further.

A tool must never let the device choose the security level. The declaration answers what the device can do, never what will therefore be used. A tool holds its own policy, a minimum profile it will accept, and refuses a device that reports less. Without this rule a forged declaration could talk a tool into weak commissioning, and the declaration is read before any security is in place.

At capability level C2 and above the declaration is covered by the device signature in the identity read, so a tool compares what it read in the clear against what came back signed and detects a forged declaration. At C0 and C1 the declaration cannot be authenticated, and a deployment must treat it as advisory and rely on the controlled environment accordingly.

The concrete object that carries the declaration, its address and its encoding, is fixed by the per-fieldbus mapping and is outside the scope of this document.

9 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.
- **Signature** (optional device identities): Ed25519, as specified in RFC 8032, offering about 128-bit security with 32-byte public keys, 64-byte signatures and an internal SHA-512. Ed25519 signing is deterministic and needs no per-signature randomness. Reserve storage for a 32-byte private key per identity the device carries; devices that cannot afford the primitive omit it.
    - Key agreement (optional, capability level C3): X25519, as specified in RFC 7748, used only for a confidential key install. It shares Curve25519 with the Ed25519 identity, so a device that already carries the signature primitive adds key agreement at little further cost, and a device that does not need a confidential install omits it.

A device in the manufacturer-rooted model both signs and verifies. Reserve storage for the 32-byte manufacturer trust anchor public key, for a second 32-byte private key where the integrator installs its own identity, and for the owner epoch counter. Verification costs roughly twice a signature on the same part, and both occur only at commissioning, never per session, so the rates a fieldbus imposes do not apply to them. NIST SP 800-57 covers the establishment, rotation and destruction lifecycle for the key material named here.

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

10 Application Example: CANopen

This chapter is a non-normative illustration. It shows how the fieldbus-independent model defined above lands on one concrete fieldbus. The object dictionary indices, data types and access rules that realize the model, and the object that carries the capability declaration, belong to the CANopen mapping and not to this document.

Classical CANopen SDO communication carries at most 7 bytes per frame. A full 16-byte authentication tag significantly increases message fragmentation, 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. In its well-known provenance the Device Claim Token is the value that bootstraps the very first such session, and is used only while the device is in the uncommissioned state.

For key installation, 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.

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.

A device that implements the optional factory device identity exposes its public key in a readable Object Dictionary entry of type DOMAIN or OCTET_STRING, so a peer or tool can retrieve it without authentication. The matching private key is generated at manufacturing and is never readable, and no Object Dictionary entry exposes it. The signed identity read and the anchored Provisioning Key install are segmented over several frames where they exceed one frame, consistent with the treatment of other provisioning traffic that exceeds a single frame. A device that implements the ownership voucher accepts it through a write-only Object Dictionary entry of type DOMAIN or OCTET_STRING that is writable only while the device is uncommissioned, and exposes the accepted owner epoch in a readable UNSIGNED32 entry alongside the Key IDs. The voucher exceeds one frame and is segmented in the same way.

11 Alignment With Standards and Regulations

11.1 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. Where the optional factory device identity is provisioned, it further supports CR 1.2 (identification and authentication): the device carries a manufacturer-anchored identity, and the Provisioning Key install is bound to it. The ownership voucher extends that claim to CR 1.5 (authenticator management), because the first authenticator a device accepts is then bound to a named owner rather than to whoever reaches the bus first, and it supports the onboarding responsibilities IEC 62443-2-4 places on the integrator. IEEE 802.1AR is the reference for the two-credential structure of a factory device identity and a local device identity.

The delegation and graded-restore mechanisms further support CR 2.1 (authorization enforcement) and the least-privilege principle: a maintenance credential is scoped to the operations it needs and cannot take ownership or decommission a device, and re-provisioning gives the owner a bounded, auditable path to rotate credentials in the field. The capability levels and named security profiles give an integrator a declared, verifiable assurance level per device, which supports the security-level reasoning of IEC 62443-3-3.

11.2 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 Device Claim Token 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.

11.3 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.

11.4 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.

12 Summary and Conclusion

On every power cycle the downstream secure protocol establishes a working key from the chosen base key by its own mechanism; by the mechanism that protocol specifies, described in EmSA-WP-105.

This model fits naturally into an authenticated sublayer above any higher-layer protocol. 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.

Across that range the model is one ladder: a capability level fixes what a device's cryptography can do, a claim gate fixes how ownership is taken, delegation lets an owner authorize field maintenance without surrendering its credential, graded restore fixes how a device is released, and a small set of named profiles fixes the combinations a deployment actually selects. A device implements the level it can afford and remains conformant at every rung.

13 Figures: Sequence Diagram Sources

The sequence diagrams in this document, Figures 1, 2 and 3, are produced by https://sequencediagram.org/ from the listings below.

Figure 1:

```
title: Key Life Cycle Overview
participant "Device Claim Token" 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 or device 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
  Integr->Marker: restore command\n(auth. by Provisioning Key or Integrator Key)
else Path B: deliberate local action
Marker->Marker: physical reset button held 10s \nor hardware jumper, no key required
end

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

Figure 2:

```
title: Ownership Voucher Issuance
participant "Manufacturer\n(voucher issuer)" as Mfg
participant "Integrator\n(commissioning tool)" as Tool
participant "Device" as Dev
==At manufacture==
Mfg->(1)Dev: **install factory device identity (IDevID)**\nand the manufacturer trust anchor
Mfg->Mfg: record serial -> IDevID public key\nin the ownership database
note over Mfg, Dev: The IDevID private key never leaves the device.\nThe device ships uncommissioned, carrying its IDevID,\nthe manufacturer trust anchor and its owner epoch counter.
==Ownership transfer (off the bus)==
```

```
Tool->(1)Mfg: **request voucher for this device**\nserial, integrator public key,\nev-
idence of ownership
Mfg->Mfg: verify ownership evidence\nchoose an owner epoch higher than before\nbuild
voucher = serial || integrator_pub || epoch\nsign SIG_MFG over the voucher
Mfg->(1)Tool: **ownership voucher**\nserial || integrator_pub || epoch || SIG_MFG
==Then: handover on the bus==
note over Tool, Dev: The tool redeems the voucher on the device:\nSee the Anchored
Provisioning Handover figure.
```

Figure 3:

```
title: Anchored Provisioning Handover
participant "Commissioning Tool" as Tool
participant "Uncommissioned Device" as Device

note over Tool, Device: Any check that fails aborts the handover.\nPreconditions and
step detail: see Ownership Transfer.

==1. Verify genuineness==

Tool->(1)Device: **read identity**\nfresh random rT
Device->(1)Tool: **identity**\ncertificate, SIG_DEV(rT)
Tool->Tool: verify certificate against the manufacturer anchor\nverify SIG_DEV against
the certified public key

==2. Claim ownership==

Tool->(1)Device: **present voucher**
Device->Device: verify voucher, serial,\nuncommissioned state, epoch\nstore owner pub-
lic key and epoch
Device->(1)Tool: **ack**

==3. Install Provisioning Key==

Tool->(1)Device: **install Provisioning Key**\nsigned by the integrator private key
Device->Device: verify against the stored\nowner public key
Device->(1)Tool: **ack**\nSIG_DEV

==4. Install local device identity==

Tool->(1)Device: **generate local device identity**
Device->(1)Tool: **identity**\nlocal device public key, SIG_DEV

==Handoff to symmetric lifecycle==

note over Tool, Device: Continue with Integrator Key install (Figure 1),\nnow rooted
in a verified handover.
```