NCC Group

Secure Device Provisioning Best Practices: Heavy Truck Edition

Written by Rob Wood

Sponsored by and drafted in collaboration with National Motor Freight Traffic Association Inc. and heavy vehicle industry partners.

Table of Contents

Executive Summary

The complexities of the heavy truck ecosystem pose challenges to the security of the ECU networks contained within the vehicles. This paper describes some of the major sources of complexity, and how each can be addressed to design and implement a secure robust ECU provisioning system. Such a system is required in order for the various ECUs to form and maintain trust relationships with each other and with external devices, for the entire lifespan of a truck. Many of these problems are well studied elsewhere by the security community, but here we apply these long-established security concepts to the heavy truck industry.

List of Best Practices

The following best practices are collected from the body of the document, and are all intended to protect the assets critical to a provisioning system: private key root of trust, ECU identities, certificates, and keys assigned to ECUs.

01. General Protection Goals
    a. Hold the root of trust of the provisioning system in an HSM (Hardware Security Module)
    b. Protect the integrity of ECU identities that are passed to the ECU during manufacturing
    c. Protect the integrity, confidentiality, and availability of certificates and keys assigned to ECUs
02. Design provisioning systems to function with intermittent connectivity
03. On-premises appliances need to be implemented as hardened devices
    a. Mitigate the risk of lost, stolen or physical compromise; do not store secrets persistently
    b. Designed and implemented to operate only on small batches of temporary information
04. Design provisioning systems as distributed systems
    a. Include delegation of provisioning authority in the design
    b. Limit the scope of authority of delegation by role, time, rate and quantity
05. Design for long lifespans
    a. Plan for regular deployment of security patches, signed by the authority (delegated or otherwise) of the provisioning system
    b. Include replacement of keys and certificates in the design
    c. Arrange disaster recovery and business continuity plans with suppliers and partners
    d. Include revocation and expiry of authenticators; revocation requires connectivity. Include cryptographic agility in the designs using a protocol version indicator, allowing algorithm co-existence but *not* protocol negotiation
06. Design for support of aftermarket authentication and provisioning of ECUs (e.g., repairs beyond the end-of-life date)

07. Design for re-provisioning in completely offline manner (e.g., repairs in remote locations)
    a. Include support for updates via long latency data paths such as USB
08. Avoid common mistakes
    a. Do not share secrets across ECUs
    b. Do not share provisioning secrets over weak channels, use out-of-band transfer
    c. Do not use Trust-On-First-Use unless in a secure environment
    d. Do not use plaintext key provisioning
    e. Do not use secret algorithms to protect provisioning or other sensitive assets; use public algorithms and secret keys
    f. Do not embed secrets in offline tools
    g. Do not allow unauthenticated privileged access (including JTAG/SWD)
    h. Do not use random number generator sources with insufficient entropy
    i. Do not use deprecated cryptography
    j. Do not reuse keys for multiple purposes
    k. Do not develop your own cryptographic implementations or design your own algorithms. Use a well established implementation library reviewed by the security community
09. Ensure in-vehicle systems have a minimum set of security requirements so that provisioning system security is not undermined
    a. Secure Boot
    b. Authenticated access
    c. Cryptographically appropriate entropy source
    d. Secure storage
    e. Secure firmware updates
    f. Implement security testing during the development lifecycle
    g. Implement privilege separation (hardware-enforced using the MMU/MPU where possible)
10. Design for communication security
    a. Implement end-to-end protection between the ECU and the back-end
    b. Replay attack resistance
    c. Brute force mitigations
    d. Certificate pinning
    e. Side channel attack resistance
    f. Mutual TLS
11. Design for storage security
    a. Partition mutable and static data
    b. Use authenticated encryption modes like AES-GCM
    c. Implement anti-rollback protections
    d. Use a hardware root of trust
    e. Implement access controls in the storage system
    f. Use a layered key storage system
12. Design for access controls

a. Avoid passwords; use key-based authentication with private keys stored in an HSM on the back-end
b. Do not store private keys or other secrets in the diagnostic or manufacturing tools
c. Isolate and separate roles where appropriate
d. Implement an audit trail with suitable logging; Do not log sensitive data

13. Design for key management
    a. Generate keys in a secure environment (in an HSM, or within the ECU)
    b. Select appropriate algorithms and key sizes for each protocol and application of cryptography
    c. Use hardware-backed key storage within the ECU
    d. Protect provisioned keys during transport using TLS or equivalent
    e. Use different signing keys (firmware and certificate signing) for each product or product family
    f. Include key revocation and replacement mechanisms
    g. Purge keys (from memory, storage, and backups) when no longer needed, at the earliest opportunity
    h. Implement secure scrap procedures, including blacklisting scrapped and stolen ECUs
    i. Use robust key ceremonies following the HSM vendor's guidelines during crucial activities

Introduction

Historically, threat models within the heavy truck industry have been well understood:

- Is someone going to steal a truck or its cargo?
- Is someone modifying log books (electronic or otherwise) to commit fraud?
- Are freight classifications being intentionally misapplied in order to unfairly compete or enable price collusion?

Under these threat models, the mitigations (such as regular audits), are well understood and have been applied for many decades. But like many other industries, the threat landscape is evolving to include many interconnected systems, which results in increased complexity.

Current vehicle architectures include complex segmented networks consisting of numerous Electronic Control Units (ECUs). These include:

- Powertrain Control Modules (PCMs) controlling the engine and related systems,
- Chassis or Body Control Modules (BCMs) controlling a variety of safety critical systems,
- Fleet management and telemetry systems used to monitor vehicle health and ensure cargo can be tracked and delivered on time,
- Telematics systems and event data recorders[1] serving insurance and compliance purposes,
- Internet-connected infotainment systems,
- Repair and diagnostic equipment that must safely and reliably interoperate

The development of newer technologies are laying a foundation to enable autonomous vehicles (AVs), which will need to safely adapt to the congested and varied transportation environment. Some of these technologies include:

- Vehicle-to-Infrastructure (V2I) technology that will allow vehicles to communicate with roadside infrastructure,
- Vehicle-to-Vehicle (V2V) that enables direct communication with other vehicles to share information about road maintenance, weather, and traffic conditions,
- Computer vision, Machine Learning (ML), expert systems, and other Artificial Intelligence (AI) technologies which are all making their way into connected vehicles at an exciting rate.

With all of this added complexity comes additional attack surfaces and threat models that need to be re-evaluated to compensate.

[1] https://en.wikipedia.org/wiki/Event_data_recorder

- The pervasive growth of systems attaching to vehicle networks results in an increased attack surface for safety critical ECUs. While CAN bus segmentation (using CAN bus gateways, etc.) are gaining traction to address this; moving towards AVs necessitates more interconnection of systems and more connectivity to the Internet, lessening the ability to appropriately segment the vehicle network. This increases the likelihood of security vulnerabilities turning into safety-critical issues.
- Remote diagnostics and over-the-air firmware update solutions (FOTA) provide additional remote connection points that may be vulnerable to attackers.
- If machine learning assets are not routinely monitored and adequate countermeasures are not integrated into the system, attackers can poison the data pool over time by providing adversarial samples as input to the system. This can lead to systems misclassifying objects or misinterpreting sensor input, which can have a real-world safety impact for road users.
- Connected systems such as telematics units may inadvertently leak location and identity data, which can enable surveillance and tracking of heavy truck fleets. This may allow targeted cargo theft or the collection of competitive and commercially sensitive vehicle performance data, which may in turn allow insider trading, unfair competition, and other business concerns.
- Ransomware and other denial-of-service (DoS) attacks can cripple a transport business[2] or even deprive entire cities of vital resources such as food, water, and fuel.
- Growth of software companies' involvement within the vehicle changes the business model for OEMs drastically. Expect vehicles to remain on the road for at least a decade and to have increasing dependence on third-party software companies, cloud solutions, etc. There has been a rise in mergers, acquisitions, and consolidations to support this growth industry. This thrash, while necessary, could result in growing concerns around supporting vehicles on the road in the long term (both from a functionality and a security perspective).

Ensuring the security of the system both now and for the defined lifetime is the overall goal. One of the foundations of a secure system is integrity:

- It must be possible to tell that a system has not been tampered with and that the system is fit for purpose.
- ECUs must protect the communication paths between each other.
- ECUs must form trust relationships with each other even when they are each manufactured by a different supplier.
- Interactions with external services like diagnostics and repair systems, V2V and V2I systems must be managed securely.

[2] https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world

This paper concerns itself primarily with the management of these keys, specifically how they are generated, safely stored, provisioned to devices within vehicles, and revoked in the event of a compromise. While the fundamental concepts are general and well-studied by the security community, their application to the complexities of the heavy truck ecosystem is not always immediately obvious. Consequently, many truck suppliers, OEMs, and operators struggle to implement robust modern solutions. When a single entity designs and implements the entire system, solving these issues would seem like straightforward engineering work. However, as businesses increase levels of outsourcing (e.g., sourcing of components and devices, inclusion of firmware libraries, ensuring the overall security, safety, and integrity of the vehicle and sub-systems becomes much more complicated.

Many security protections find their foundations in the mathematics of cryptography. This requires the use of secrets to protect data and to authenticate devices and firmware. We note that there have been some efforts to standardize some of these cryptographic concepts in the vehicle space in the form of ISO20828:2006[3] and others[4]. These have seen relatively poor adoption, likely owing to the lack of prescriptive and practical details and the lack of guidance to bootstrap the cryptographic protections. These gaps are what this document aims to cover. Failure to handle the secrets with care can result in compromise. When compromise occurs, systems with functionality that requires security guarantees become unreliable.

The heavy truck ecosystem will securely provision other data, in addition to keys, and in particular, an important security foundation is identity. Each ECU needs to be uniquely identifiable in a manner that avoids spoofing, counterfeiting, and credential theft. The serial numbers, MAC addresses, International Mobile Equipment Identity (IMEI), Vehicle Identification Numbers (VIN), etc. all need to be assigned and provisioned into the devices in a way that guarantees their integrity so that they can be trusted for device identification purposes throughout the lifetime of the vehicle. The same provisioning systems used to assign keys and other secrets to the systems can be used to manage these identities. Such identifiers are typically generated offline and programmed into the vehicle systems at the factory when suppliers first build devices. OEMs, in the case of the VIN, must also program this identifier at chassis build-time and during part replacements. In all cases, identifiers must be protected from being altered by unauthorized users. The authority to program some identifiers (like the VIN) must be transferable to various authorized parties throughout the vehicle lifetime and must certainly include repair facilities like those owned and operated by the motor freight carriers.

Finally, it is important to note that the word “security” is overloaded in many ways. Depending on the context, security encompasses many concepts, including confidentiality, integrity,

[3] https://www.sis.se/api/document/preview/907544/
[4] https://icmconference.org/wp-content/uploads/E11c-Whyte.pdf

authenticity[5], availability, privacy, consent[6], and safety. Throughout this paper, we try to avoid the ambiguous term in favor of the specific security traits that are relevant to the discussion.

Threat modeling

Threat modeling is a useful exercise that helps describe the overall *security objectives* of a system. Typically, these objectives are associated with classes of *threat actors* whose goal is to compromise the system's *critical assets* by violating specific security traits of the system. Finally, a threat model may also articulate *mitigations* that define how the system defends itself against certain classes of threats. The attack surface is an enumeration of external interfaces by which a threat actor can interact with the target system. The threat actor aims to discover vulnerabilities in how these interfaces consume inputs that pass across a trust boundary. Here we present a brief threat modeling summary of an example provisioning system (see Figure 1).

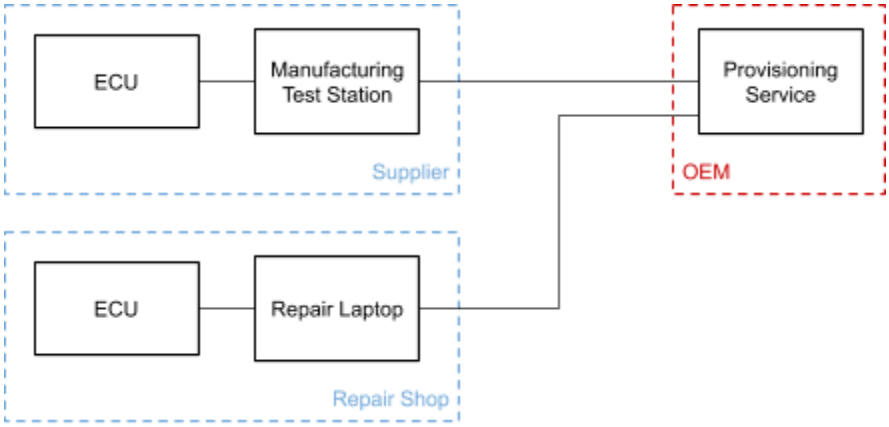


Figure 1: Minimal provisioning service

Assets

Any discussion of threat modeling needs to start with an enumeration of the assets to be protected. The list of assets the industry should consider in heavy trucks is long and varied; here we will focus our attention on the provisioning system itself and the assets the system provisions. The protection required for each asset may vary, and can include confidentiality, integrity, availability, etc. The following lists the major assets for our example provisioning system depicted above.

- The root of trust of a provisioning system will be a private key held within the back-end, preferably in a Hardware Security Module (HSM)[7]. Compromising the confidentiality of this secret would allow an attacker to set up their own system and provision their own devices.

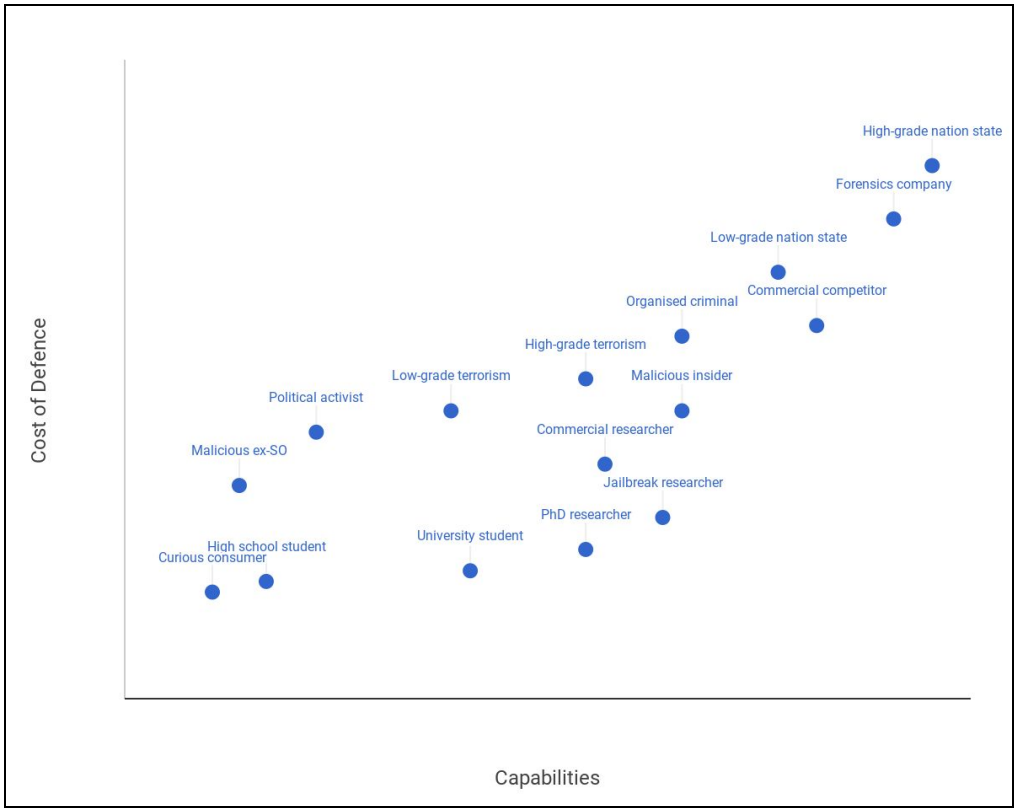
[5] https://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/
[6] https://fieldnotes.resistant.tech/optimal-privacy/
[7] https://en.wikipedia.org/wiki/Hardware_security_module

- The integrity of identities that the system manages and passes to ECU manufacturing lines for provisioning to ECUs are crucial to limiting counterfeiting operations and identifying legitimate devices.
- Integrity, confidentiality, and availability of certificates and keys assigned to individual ECUs are important for integrity and confidentiality protection of their data-at-rest and data-in-transit. The data itself may have additional privacy and consent requirements, especially within the European Union, where General Data Protection Regulation (GDPR) regulations are enforced.

Threat Actors

Like the assets, the threat actors of concern can vary widely, and should be considered before performing any risk analysis. While this list is ever-changing, it is important to acknowledge the nature of the likely threat actors. It may not be possible or practical to defend against all of these individuals or organizations, as doing so may require a significant amount of time, effort, and money. The chart below subjectively compares various commonly considered attackers in terms of attacker capability and cost to defend against each. The decision about where to set the risk threshold will depend on the individual business. We present a deeper analysis of each category of attacker and their capabilities in other NCC Group publications[8], which for brevity is not reproduced here.

8 https://www.nccgroup.trust/uk/our-research/security-of-things-an-implementers-guide-to-cyber-security-for-internet-of-things-devices-and-beyond/



Some example scenarios that may be relevant to the provisioning system threat model include:

- Counterfeiters, seeking to profit from inferior ECU devices, need to gain access to the provisioning system in order to have their devices recognized and interoperated within the vehicle.
- An aftermarket repair shop attempts to modify the identity of an ECU in order to load firmware from a different product SKU, thereby converting the low-cost SKU to a high-cost SKU, without regard for the safety impact of doing so.
- A contract manufacturing partner attempts to overproduce, selling the excess outside the normal sales channel.

Challenges

General Challenges

The primary challenges involved in a provisioning system revolve around access control and data protection of the sensitive assets (e.g., secret keys). These are actually well studied problems and have accepted solutions. When security solutions typically fail to meet the objectives, it is often a result of insufficient threat modeling. For example, failing to enumerate threat actors, or accepting the associated risks without understanding the real-world implications, can lead to a compromise of secrets or unauthorized access to provisioning systems.

Manufacturing Challenges

The manufacturing environment can be a challenge for product security, regardless of industry. It is uncommon in the trucking industry for factory employees to be made aware of the product security requirements, let alone be provided incentives to uphold them. In fact, they may have perverse incentives[9] that run counter to product security objectives. Employees of production facilities may bypass security measures in order to improve throughput and meet their own individual targets, which can cause production delays and a reduction in both product quality and security. This is difficult to detect if the production line does not include adequate end-of-line testing and quality assurance procedures. Suppliers may outsource to sister facilities or outside subcontractors to reduce labor costs; this outsourcing may be unknown to the original product designer and is often not checked unless the original production contract restricts such behavior, or if technical safeguards are in place. Corruption within the factory environment is commonplace, and a line worker can make many times their salary by leaking information or designs, or by providing network access to external counterfeiters, organized crime gangs, or competitors[10]. The provisioning system should help mitigate these challenges in the manufacturing environment.

A provisioning system typically requires exposing some provisioning infrastructure to the factory. This is true regardless of who owns the factory, but in the vehicle space, this is most often a Tier-1 supplier rather than the OEM. Some suppliers manufacture ECUs for trucks in geographies chosen primarily for the lower operating cost due to differences in labor markets. In such locations, reliable internet connectivity may not be a realistic expectation, and so any provisioning scheme should be designed to tolerate a certain amount of network downtime. A common resolution is to deploy a portion of the provisioning infrastructure directly within the factory. This can be extremely problematic if the factory is untrusted[11]. While the factory is frequently a third party (as depicted in Figure 2), no factory, even one that is OEM-owned, is immune to the aforementioned abuses.

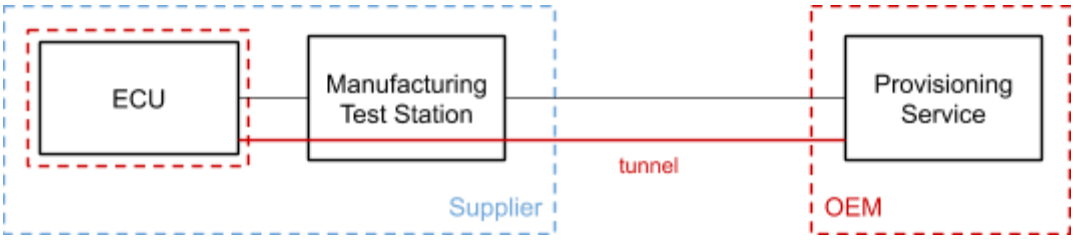


Figure 2: Provisioning system exposed to factory

[9] https://en.m.wikipedia.org/wiki/Perverse_incentive

[10] https://www.npr.org/2019/01/29/689663720/a-robot-named-tappy-huawei-conspired-to-steal-t-mobile-s-trade-secrets-says-doj?t=1563815834977

[11] https://www.nccgroup.trust/uk/our-research/secure-device-manufacturing-supply-chain-security-resilience/

To counter these concerns, the ECU should be designed to communicate securely with provisioning systems in such a way so that a Man-in-the-Middle (MitM) attack within the factory cannot compromise the data flows.

The data path may need a local on-premises appliance to account for network downtime between the factory and the provisioning service. Any portion of the provisioning system that is deployed on-premises needs to be implemented as a hardened appliance, with safeguards to mitigate risk in the event that it is lost, stolen, or physically compromised. This appliance should be implemented to operate only with temporary data and must connect back to the master system at the OEM at regular intervals for refreshes. A compromise of this appliance would then only grant the attacker a small duration of autonomy, thus limiting the potential damage.

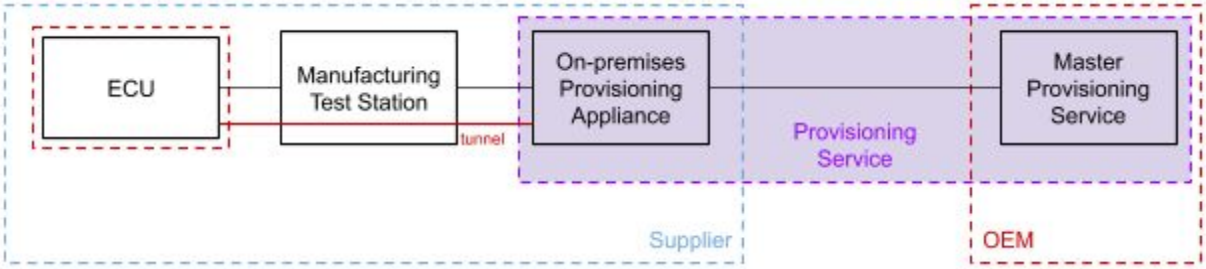


Figure 3: Application of on-premises appliance

Example implementations may consist of:

1. A remote provisioning system that generates device certificates. These certificates are delivered in batches at a regular cadence to an on-premises appliance within the factory, as defined by the network conditions and business and security risk analysis (practical frequencies are in the order of every 4 hours, once per day, etc.). The appliance provisions these certificates to the devices being manufactured as needed, but cannot generate new ones on its own. Enough volume should be delivered in each batch so that the expected amount of network downtime can be tolerated. This can be notably higher when the factory is in parts of the world with poor internet connectivity.
2. An on-premises appliance within the factory that contains an intermediate certificate authority, capable of signing device certificates as needed. The appliance should store the private key corresponding to the intermediate CA certificate only in volatile memory, and require an authenticated connection to the master provisioning system to securely acquire this cryptographic asset during the appliance startup sequence. The appliance must be configured to connect to the master provisioning system on a regular basis (say, every 1000 signing requests) to upload transaction logs for audit purposes, and to refresh the intermediate CA key as needed. This will prevent the appliance from signing additional device certificates in the event that it is lost or stolen.

Distributed Ecosystem

The industry is not vertically integrated, which results in a complex ecosystem of electronics within the vehicle. A modern truck contains ECUs from a number of different Tier-1 suppliers,

which may run firmware provided by other sub-suppliers, and run on processors with firmware provided by Tier-2 microprocessor and software suppliers.

The base vehicle platform can be further modified and enhanced by Body Builders, who may add various aftermarket features to the base design (e.g., Power Take Off (PTO), sleeper cabs, additional control panels for heavy lifting equipment). Here, Body Builders add additional ECUs to customize the truck for its application-specific requirements.

Finally, the operator may install their own preferred brand of Telematics Control Unit (TCU) to support fleet management, digital tachograph devices specific to their use-case, and other aftermarket ECUs to enable any number of additional feature-sets. Once in operation, the vehicle will interoperate with yet more third-party systems for repair and diagnostics, V2X, and other Intelligent Transport Systems (ITS). All of this complexity poses challenges when the ECUs need to form trust relationships with each other and external systems. Identifying the difference between legitimate devices and malicious or unauthorized devices on these networks is key to a secure system design. A provisioning system that is centrally managed is simpler to build and operate; however, this may not meet the needs of such a complex ecosystem of truck components.

Delegation of provisioning authority from the OEM to the Tier-1 suppliers may be required. This can be accomplished through the use of subordinate certificate authorities and certificate chains (described in more detail below). OEMs must delegate in a way that does not allow the supplier to assume responsibility for maintaining trust at a vehicle level. Rate limiting that effectively prevents more ECUs than ordered from being provisioned, implemented within hardened on-premises appliances, can be used to enforce such controls.

Vehicle Lifespan

There is a distinct longevity challenge for any security system in this space. Businesses expect heavy trucks to last many decades, and often heavy trucks have longevity in excess of one million miles[12], yet security attacks and defenses continue to evolve at an internet-age pace. This is in stark contrast to other markets: most consumer electronics are often designed to have a relatively short half-life (often as short as 6 months); consumer vehicles typically only last about 200,000 miles[13].

A crucial defense over such long timeframes is to provide regular software and firmware security-patch updates. As new issues and vulnerabilities are discovered, OEMs should provide access to fixes in aftersales. This cadence requires that active vulnerability management take place for the life of the vehicles, and that reliable and integrity-assured over-the-air firmware update capabilities be implemented. The heavy truck industry must evaluate the model for

[12] https://www.auto.edu/blog/13-solid-stats-about-driving-a-semi-truck-for-a-living/
[13] https://www.consumerreports.org/car-repair-maintenance/make-your-car-last-200-000-miles/

long-term support, as more functionality is dependent on software libraries developed by the tech industry, which has shorter product lifecycle requirements.

Likewise, authenticator management across multi-decade support lifetimes can be a daunting commitment. OEMs need a scalable solution to replace expired or revoked keys and certificates. Suppliers need to remanufacture replacement devices or refurbish ECUs long after the original production ends. The heavy truck industry needs to arrange disaster recovery and business continuity plans should any partner or supplier involved fail to uphold these lengthy commitments. The heavy truck industry can use software and key escrow to hedge against such risks.

Authenticators that outlive the protections placed on them may outlive their usefulness. For this reason, authenticators may be designed to either be revoked or to expire (or both). Revoking authenticators can be problematic in situations with limited connectivity; however, such connectivity, however infrequent, is required for revocation information to be communicated to the ECU endpoints. Expiry has a requirement for a reliable time source, which can come in various flavors (GPS time, count of the number of boot cycles, count of the number of back-end connections). Integrity protection of the timebase's storage is a must to prevent tampering such as rollback. If the source of the time measurement is external to the ECU (such as NTP or GPS), then communications with that source must be protected from tampering[14], and clock drift corrections must be applied in a smooth and monotonically increasing manner. The exact implementation will depend on the design constraints of the system, however, balancing this with the need for continuous vehicle operation is obviously important. The provisioning strategy needs to define both policy and system behavior upon expiry/revocation such that both needs can be met.

Finally, cryptographic algorithms, the core of many security protections, get weaker over time.

- The algorithms undergo constant cryptanalysis, where new mathematical attacks are discovered[15], and new implementation flaws are uncovered.
- Unforeseen system-level issues and novel attack techniques can be discovered, such as side-channel[16] and fault injection[17] attacks.
- Most algorithms rely on the computational infeasibility of brute forcing their solution, an assumption that degrades as new more powerful hardware is developed each year. Real world examples of long running attacks have set benchmarks for brute force capabilities [18]. It is vital to choose key sizes of sufficient strength to eliminate such possibilities.

[14] https://arxiv.org/pdf/1710.05798.pdf
[15] Xiaoyun Wang, et al: http://dx.doi.org/10.1007/11426639_2
[16] https://en.wikipedia.org/wiki/Side-channel_attack
[17] https://www.riscure.com/uploads/2018/06/Riscure_fault-injection-on-diagnosis-protocols-presentation.pdf
[18] https://en.wikipedia.org/wiki/RSA_Secret-Key_Challenge

- Quantum computing is an area of intense research and is expected to defeat most public key cryptosystems.[19] The implementation of quantum-safe algorithms is likely to be needed within the lifespan of current vehicles.

The lesson is this: what was considered secure yesterday, may not be considered secure tomorrow. Most pronounced is the problem concerning the security of any data-at-rest where the cryptographic protection is not ephemeral. To hedge against any rapid attack evolution (in particular unforeseen or rapid innovations), *cryptographic agility* is needed, whereby the protocols themselves can support upgrades to the fundamental algorithms that they use. The dynamic nature of cybersecurity risk is an outlier for the heavy truck industry. Existing risk analysis techniques, which characterize risk for mechanical parts with predictable failure modes, cannot be readily adapted to measure cybersecurity risk. As the heavy truck industry future-proofs new designs, the selection of protocols to support cryptographic agility is important to address the longevity requirements. Future-proofing a design for algorithm agility doesn't necessarily mean planning for significant resource overhead since, as discussed in later sections, newer algorithms are always developed to be performant and aren't necessarily slower than older ones.

It must be noted that designing such flexibility is often the source of its own problems if done incorrectly. Algorithm *agility* refers to the ability of a system to be easily upgraded to support new algorithms as old ones become obsolete. This is done properly with a protocol version indicator, allowing several versions to coexist on a given communication link or storage system. However, implementers aiming for algorithm agility often unintentionally implement algorithm *negotiation*, in which all individual algorithms are dynamically negotiated, leading to much implementation complexity, and unforeseen issues with "bad combinations," downgrade attacks, or worse[20]. There have been countless examples of vulnerabilities of this nature in TLS[21,22] that are instructive of what not to do.

Right to Repair

The *right to repair movement*[23] adds another layer of complexity. While heavy trucks are specifically excluded from Bill H-4362[24] in the US market, it is mentioned here as a risk, considering that tractors and smaller vehicles are already under scrutiny. Heavy trucks are not exempt from similar independent repair legislation in Europe[25] and sales of trucks for European export market must meet this requirement. A system built so secure that not even the owner can repair it may actually become illegal in some jurisdictions. This, in principle, is a good thing; the device may outlast the original manufacturer or one of its suppliers and still have a serviceable

[19] https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization
[20] https://en.wikipedia.org/wiki/Null_encryption
[21] https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/
[22] https://en.wikipedia.org/wiki/Cipher_suite#Vulnerabilities
[23] https://repair.org/automotive
[24] https://malegislature.gov/Bills/187/H4362
[25] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009R0595

life. The lack of vertical integration exacerbates this risk. Routinely, we see vehicles being sold in whole or in part to buyers in less developed countries, long past the end-of-life date planned by the OEM. There are vast numbers of repair shops beyond the OEM authorized dealers that keep vehicles on the road. Most larger operators have in-house repair shops as well. All of these mechanics need to at least be able to interface diagnostic tools with the electronic systems in the vehicle; in many cases, replacement of ECUs and sensors connected to these ECUs is necessary.

Supporting an ecosystem that supports the aftermarket authentication and provisioning of ECUs may be required. Code, keys, and systems may need to be placed in escrow to allow for circumstances such as a major supplier going out of business.

Offline Operation

Because of the complex ecosystem, design of provisioning services cannot assume connectivity is always available. Consider repair shops that must be able to access ECU systems for diagnostic purposes, but that may not have Internet connectivity for a variety of practical reasons (e.g., cost, remoteness). Because of this, certain systems may need to interact with the ECU for provisioning purposes in a completely offline manner. For such situations, often we find that the credentials or secrets necessary to access the ECUs privileged interfaces are erroneously embedded within the host-side diagnostic applications. This allows anyone with a copy of this software to extract it. With modern cellular connectivity, the need for truly offline operation may no longer be required in many cases. However there are still places, far from cities (like the northern tundra), where trucks regularly operate for much of their lifespan without hopes of even intermittent connectivity. While such isolation limits the opportunistic attacks that the vehicles may experience, targeted attacks are high on the list of threat models for such operators.

Solutions to support extended vehicle isolation may include:

- Direct vehicle-to-vehicle updates, allowing one vehicle to securely update from another that has more frequent connectivity.
- Support for updates via long latency data paths using physical links like USB and OBD2. In such an implementation, a normal cryptographic challenge/response protocol is used for authentication; however, the timing constraints are relaxed enough to permit many days or weeks between the challenge and the response, which allows for creative transport solutions such as using the postal service.
- Support for satellite and other remote connectivity in the diagnostic tools and systems.
- Lengthy deferral of updates to allow uninterrupted operation until such time as connectivity can be restored (i.e., allowing the operator/owner to accept the risk).

Common Mistakes

There are a number of ways that provisioning systems typically fail. Some of the most common design mistakes are discussed here as instructive examples of what not to do.

- **Shared provisioning secrets.** Provisioning each device with the same secret key is a common mistake. This leads to a situation where an attacker may compromise a single device, and then leverage the information gained to compromise all other devices with no extra effort, thus allowing attacks to scale in a very low-cost manner. This is the so-called, “break once, break everywhere” model, and it should be avoided.
- **Key exchange for symmetric keys.** Symmetric key cryptography is simpler and less resource-intensive than its asymmetric counterpart, however, it is often accompanied by a weak key exchange. Sharing of symmetric secrets requires an out-of-band channel to avoid interception by a passive adversary on the channel.
- **Trust-on-first-use.** A trust model where one party to the communication does not know about the other beforehand. Upon seeing the remote party for the first time, it records the details and trusts it from then on. This does work to form a strong authentication bond between the communicating parties, however, it is vulnerable to attack during the first connection when establishing the trust relationship. OEMs should only use this model if the first use or registration occurs in a trusted manufacturing facility. Truly trustworthy environments are a serious challenge of their own, even in OEM-controlled factory facilities. Care must also be taken to avoid mechanisms whereby the attacker can “reset” the state to make the provisioning system think it is seeing an ECU for the first time again.
- **Plaintext key provisioning.** This is common in systems where the secrets are generated offline and pushed down to the embedded system. Because the local physical connections to the devices (e.g., USB, serial, OCD) and the factory networking connections are rarely encrypted, adversaries with access to the factory systems are able to easily intercept these secrets. A safer solution is to generate keys within the ECU itself, and export necessary information to the back-end in an encrypted form.
- **Secret algorithms.** Secret algorithms used to generate passwords, keys, or other authentication tokens are still very common. For example, the *SecurityAccess* authentication mechanism in ISO 14229 Unified Diagnostic Services is ubiquitous throughout the automotive industry and relies on “secret” algorithms to derive the key from the seed. The danger here is that once the algorithm has been extracted and reverse engineered, the secret is completely compromised, creating situations which enable attackers to carry out their objectives.
- **Offline tools with embedded secrets.** Related to the previous mistake, offline diagnostic tools are a common place to find sensitive secrets. These are common in the automotive market for Unified Diagnostic Services, and the same has been found to hold true in the heavy truck space. Unique to the off-highway heavy vehicle market is the need to provide diagnostic support in remote unconnected parts of the world. This requires authentication schemes that are not reliant on Internet connectivity, and frequently this involves embedding a secret within the diagnostic software tools. If an attacker is able to reverse engineer a copy of the host-side diagnostic software, then they can extract the secret. This broken security model can be a challenge for OEMs

wanting to move to an always on-line solution where the secrets are stored remotely on back-end servers.

- **Unauthenticated privileged access:** Unauthenticated debug ports are still the most common means to compromise embedded devices when physical access is available, and heavy truck ECUs are no exception. The privileged interfaces exist for a variety of legitimate reasons, including manufacturing, Fault and Failure Analysis (FFA), development, and diagnostics. But leaving these interfaces unauthenticated, or weakly authenticated, allows attackers the same privileges. Classic examples of weak authentication include static passwords/keys (break once, break everywhere), or predictable keys derived from public information such as component serial numbers or VINs. A strong public key based challenge-response protocol is recommended to mitigate this risk.
- **Insufficient entropy.** Not enough randomness, or otherwise weak key generation, has led to predictable keys in many systems. Sometimes this stems from a failure to properly characterize the entropy source before use. More often this is either a) an integration issue, where one supplier leaves a software hook for another supplier to implement, and its implementation is forgotten or otherwise neglected, or b) a lack of entropy entirely, in which a constant or predictable value (like the time of day) is used as a seed to a predictable pseudo-random number generator function.
- **Deprecated Cryptography:** The use of outdated and obsolete cryptographic algorithms, modes, and key sizes is still a widespread problem across many industries, not just heavy trucking. Backward compatibility with legacy systems can explain some design patterns, but much of this is just momentum and lack of business drivers, such as customer input or regulations. A non-exhaustive list of algorithms that are commonly used, but that must be avoided include SHA1, MD5, DES, 3DES, and RC4.
- **Key Reuse.** Overloading the use and purpose of cryptographic keys is a bad practice, and can lead to several devastating cryptographic attacks. Some examples include:
    - Keys that are used for both signing and encryption can lead to a compromise of the keys themselves.
    - A single key used to sign files that are intended for different purposes (e.g., firmware, OEM configuration) may lead to image type confusion issues and allow an attacker to leverage the signer as an oracle to sign arbitrary files.
- **Other Improper Cryptography.** Cryptography is easily misused and error-prone. Even simple mistakes can lead to catastrophic results. Knowing the bounds of the problem and applying the correct cryptographic solutions requires expert advice and misuse-resistant libraries. The list of cryptographic mistakes is endless, but some of the common ones include padding oracle attacks, initialization vector and nonce reuse, length extension attacks, use of ECB mode, timing attacks, poor parameter validation, and small key size.

Solution Requirements

Building any secure embedded system and incorporating them into a complex vehicle network requires a large number of security measures and mitigations. We are limiting the focus of this document to only those security requirements directly associated with a robust provisioning system and the corresponding data; The security of the vehicle system as a whole is beyond our scope.

It is worth at least mentioning some of the other platform security measures on which such a vehicle system will depend to prevent its security from being undermined. Many of these are considered bare-minimum security measures for any modern product built this decade, yet many of these are still commonly missing or fatally flawed in embedded systems used in the trucking industry, even those still under development in 2019. We note that this list is by no means complete.

- **Secure Boot**, or an equivalent boot-time integrity validation system, is needed to ensure that any of the firmware responsible for implementing the ECU side of the provisioning system can itself be trusted to behave as designed. Such a boot scheme will involve each stage of the boot process cryptographically validating subsequent stages before it is executed. This will begin at the earliest stage in the hardware ROM so that there is no possibility of running firmware that is untrusted (assuming no implementation flaws). It must implement anti-rollback protection to prevent attackers from downgrading firmware to known-vulnerable versions.
- **Authenticated access** provides strong guarantees that only authorized agents are able to access privileged functionality of the device: functionality for developers, manufacturing, aftersale field diagnostics, and RMA refurbishment. Much of this privileged functionality can be used to undermine the security of the system if misused, so ensuring that it is only accessible to authorized agents is vital. Unauthenticated debug ports (commonly JTAG/SWD, UART, or USB) is a fatal security flaw for any device where a threat actor will have such physical access. Authentication needs to be implemented within all accessible code modules, including bootROMs, bootloaders, operating systems, and applications.
- An **entropy source** that is cryptographically random, (i.e., a hardware random number generator) is an important requirement for much of the cryptographic implementations that the system will rely on. Choosing a hardware component that supports this need is the simplest approach, but does require foresight. Common processors and System-on-Chips (SoCs) contain such functionality built directly into the silicon.
- **Secure storage** is important for anything that must be persisted and will provide integrity and confidentiality guarantees as needed, depending on the data. Only the OEM has a need to modify some of these data items, such as ECU-to-vehicle relationships, while others will need access for aftersales. These different scenarios will dictate the implementation of the security measures for the storage system. The integrity

guarantees will normally include anti-rollback protection preventing the entire filesystem (or subset thereof) from being reverted to an older, but still valid state, and preventing an attacker from selectively modifying or reverting individual data records.

- **Secure firmware update** functionality needs to be implemented with integrity assurances regardless of the delivery method (e.g., local USB, OBD2, CAN bus, over-the-air, removable flash memory cards). The ability to update devices for aftersales is especially important given the long lifespan of heavy trucks. As researchers constantly discover new vulnerabilities, providing a means to patch these in a timely manner is a primary defense to the vulnerabilities being exploited. It is simply not feasible to implement a system and expect it to remain secure without a means to fix security aftersale vulnerabilities. Example implementations are out of scope, but some are well documented elsewhere[26,27,28,29]. Common traits of such systems include transport security, pre-install image validation, and strong firmware manifest management.
- **Security testing** is different than regular functional and performance testing. While the latter aims to test whether the system does what it is supposed to, security testing explicitly tries to determine if the system does anything it is *not* supposed to. Such testing uses a variety of techniques including manual code review, automated static analysis, and dynamic analysis such as fuzzing[30]. If security testing is not incorporated into the development process, then security vulnerabilities will not be detected and fixed at the earliest and cheapest opportunity. Rather, you will need to rely on customer feedback to report discovered issues. External security reports can be unreliable: users may not have the expertise to effectively report security issues; customers may be more interested in suppressing the knowledge for fear that attackers may learn about the vulnerability; researchers may find it more lucrative to sell the information to bad actors. A secure software development process will incorporate robust security testing at multiple points. There are many public resources to help with this[31].

For the provisioning system itself, we divide the security requirements into a few broad categories: communication security, storage security, key management, and access control. We will discuss the high-level requirements only, as the actual implementation can vary widely and will depend on many implementation decisions.

Before discussing the security requirements in detail, some background on the cryptographic tools at our disposal is needed.

[26] https://uptane.github.io/
[27] https://github.com/Microsoft/CFU
[28] https://theupdateframework.github.io/
[29] https://arxiv.org/pdf/1807.05002.pdf
[30] https://www.f-secure.com/gb-en/consulting/our-thinking/15-minute-guide-to-fuzzing
[31] https://blogs.msdn.microsoft.com/microsoft_press/2016/04/19/free-ebook-the-security-development-lifecycle/

Cryptographic Concepts

This section contains some brief, non-mathematical descriptions of the cryptographic building blocks that are necessary for building a provisioning system. While it is useful to understand these concepts so that they can correctly be applied, the reader is warned that implementing and using cryptography is error-prone, and even small mistakes can be catastrophic. **Do not implement your own cryptographic implementations or design your own algorithms.** Instead, use a well established cryptographic implementation library reviewed by the security community, one that is specifically misuse-resistant. Appropriate choices include BoringSSL[32], BearSSL[33], NaCL[34], and mBedTLS[35].

Encryption is the mathematical encoding of data such that only an authorized agent with the correct key can decrypt it. This is a primary defense for confidentiality protection of data. Encryption is used both to protect data-in-transit and data-at-rest. Broadly, these algorithms can be grouped into two categories: symmetric and asymmetric algorithms.

AES is a common symmetric algorithm recommended for use in modern systems. It relies on a single shared key for both encryption and decryption. Consequently, both parties need to be equally trusted not to compromise the key. Symmetric block ciphers operate in different modes appropriate for the context. We always recommend using an **authenticated encryption**[36] scheme (such as AES-GCM, AES-EAX, or ChaCha20+Poly1305) to additionally provide integrity. Vulnerable modes like ECB, or obsolete algorithms like DES/3DES/RC4, are insufficient for new designs.

On the other hand, asymmetric algorithms use a pair of keys, a private and a public key. This allowed the security community to develop elegant protocols that support integrity protection, non-repudiation, identity validation, and other useful security properties. Here developers use algorithms such as RSA and ECC. One practical limitation is that asymmetric algorithms are generally much slower than their equivalent symmetric counterparts due to resource intensive computations (although for large equivalent key sizes -- see table 1 -- ECC is considerably faster than RSA)[37]. Another important limitation for RSA specifically, is that it supports only limited-size inputs, making encryption of larger messages or files difficult. Systems commonly overcome these limitations by using hybrid encryption schemes: symmetric algorithms with an ephemeral key for bulk cryptography and only using the asymmetric algorithms to protect the ephemeral key, or by using a Diffie-Hellman (DH) key exchange protocol to negotiate a shared symmetric secret for encryption.

---

[32] https://opensource.google.com/projects/boringssl
[33] https://bearssl.org/
[34] https://nacl.cr.yp.to/
[35] https://tls.mbed.org/
[36] https://en.wikipedia.org/wiki/Authenticated_encryption
[37] https://bearssl.org/speed.html

Signatures are a special use of asymmetric algorithms where a signature is computed over a message (e.g., file, packet) using a private key. The signature can then be validated easily using the public key. This provides strong integrity protection against forgery, as only those with the private key are able to sign messages that will validate with the public key.

Key exchange algorithms like DH and the elliptic curve equivalent (ECDH) are usually the most appropriate way to negotiate a *temporary* (ephemeral) shared symmetric secret. Here, only the public keys are shared between the communicating parties and used (along with the local private keys) to compute a shared symmetric secret. ECUs use key exchange algorithms for efficiently encrypting any data communication. One caveat is that DH/ECDH are themselves insufficient to protect against an active Man-in-the-Middle (MitM) who can provide malicious/replacement public keys to each party. To resolve this, ECUs must validate the public keys as authentic by the use of a trusted third party or out-of-band channel. This is most often accomplished by signing them with a trusted root Certificate Authority (CA) known to the device software or by pre-sharing public keys by, for example, embedding them in the firmware (itself validated by the Secure Boot process).

Cryptographic hash functions are mathematical methods by which data can be compressed into a "fingerprint" or digest. Product engineers can think of the hash digest as a checksum of sorts, but with additional cryptographic guarantees.

A good cryptographic hash function must:

- Not be reversible
- Not reveal information about the input message
- Be difficult to find collisions
- Meet practical constraints such as performance

Typical hash functions recommended for use are the SHA-2 family (SHA-256 and SHA-512) and the newer SHA-3. We do not recommend obsolete functions like MD5 and SHA-1[38], as they fail to meet all of the security objectives of a modern hash function. Hash functions are frequently used to provide integrity guarantees within protocols and during digital signature validations.

A reliable source of **entropy** is vital for generating random numbers, but computers are designed to be deterministic and repeatable. This is a problem for cryptographic operations that require randomness for their mathematical security. When attackers can influence the selection of "random" values, it reduces the complexity for breaking cryptographic protection mechanisms, which underpin a variety of security mechanisms within the ECU. To solve this, most embedded microcontrollers and SoCs include a hardware entropy source, typically based on some form of analog noise, like a noisy diode, linear feedback shift register (LFSR), or ring oscillators. For the

[38] https://shattered.io/

best results, pair the hardware random number generator (HWRNG) with a cryptographically secure pseudo-random number generator (CSPRNG) or a hash function to remove bias and improve performance. Detailed guidance for the creation of random number sources is available elsewhere[39].

Certificates can be a complex topic, but at their most basic they consist of a cryptographic public key and an identity that are together cryptographically signed by some other key in order that the certificate's integrity can be assured. Certificates frequently contain additional metadata, such as the issuer name, its intended purpose, issue and expiry dates, etc. The issuer creates a **certificate chain** by having one certificate signed by another (and so on) that ends at the **root certificate** or **certificate authority** (CA). Collectively, these concepts and the processes that surround them are sometimes referred to as a **Public Key Infrastructure** (PKI). This construct allows delegation of cryptographic authority to subordinate CAs, which may be a useful feature to satisfy some of the business constraints in the heavy truck supply chain. Here, an OEM (root CA) may issue subordinate CA keys to Tier-1 suppliers so that they may produce ECUs for use in the OEM vehicles. Scheduled refreshing of the subordinate CA and regular audits can limit abuse in the event of a supplier compromise. When attackers obtain access to private keys, they can assume the identity that the key represents to sign, encrypt, and decrypt data or software. Software supply chain attacks sometimes occur because there is insufficient access control, which may allow malware to be signed and distributed using the trusted certificates[40] .

Lightweight cryptography[41] is specifically intended to operate on resource-constrained embedded devices. This is not something generally intended for the automotive or IoT space, but for truly resource-constrained devices such as RFID tags. There is a contest to develop the standards, and at the time of this writing, no clear algorithm has been chosen as a winner. The contest itself does not define the term "resource constrained," and only gives high-level guidance as to what performance metrics will be measured. Any investigation into the applicability of lightweight cryptography needs to start with a careful study of the performance constraints of the ECU design in question. It bears noting that even the algorithms described in the previous sections (AES/ECC/SHA2) were specifically designed to be highly performant. As a general rule, a device with a processor running at 50MHz or higher with at least 256kB of RAM should be more than capable of running common algorithms without the need for new and exotic cryptography.

Threshold Cryptography is a construct that allows multiple parties to be involved in decryption of information. This can be used to mitigate a single point of failure. One common algorithm in

[39] https://csrc.nist.gov/projects/random-bit-generation

[40] https://csrc.nist.gov/CSRC/media/Projects/Supply-Chain-Risk-Management/documents/ssca/2017-winter/NCSC_Placemat.pdf

[41] https://csrc.nist.gov/projects/lightweight-cryptography

this category is *Shamir's Secret Sharing*[42] ("M of N"), which allows a private key to be split among N parties in such a way that at least M need to participate in order to decrypt data. Any group smaller than M does not reveal any secret information. Such mechanisms are strongly encouraged for critical key management functions, such as certificate authorities, where a disaster in one place (e.g., fire, flood, rogue employee) must not be allowed to prevent the ongoing system operation.

Communication Security

During provisioning, the communication channel between the provisioning systems and the ECU needs both confidentiality and integrity protection. This is commonly referred to as the security of the data-in-transit. The primary security properties of interest here are confidentiality and integrity (which includes replay-resistance). These can be satisfied by applying modern security protocols such as TLS 1.2 or later, which provides both.

When everything is working correctly and the communication pathways are end-to-end protected, then anything outside the trust boundary can be ignored from a security modeling perspective. This allows the model of our system to be drawn rather simply as follows.

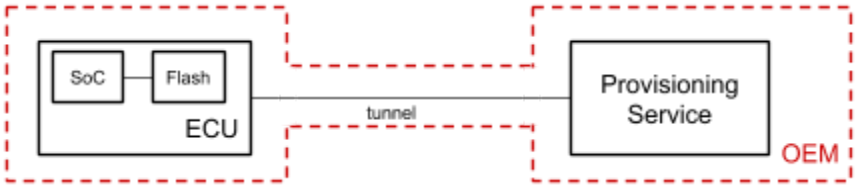


Figure 4: End-to-End secure communications

This will be true when all data passing through the intermediate systems is properly authenticated by the endpoints, thus eliminating the possibility of the data being compromised in transit. Note that the diagram assumes a single provisioning service entity, which may not be the case when delegating cryptographic authority to a subordinate provisioning service at the supplier. In this case, the view will be the same, but the provisioning service the ECU interacts with will be either a) the supplier's, not the OEM's, or b) an appliance installed on-premises by the OEM, which reports back to the master provisioning system at the OEM whenever network connectivity is available.

Additional security properties of interest in all communication protocols include:

- **Replay attack resistance**. Replay attacks involve re-issuing challenges, responses, or other packets. If the system cannot detect receiving duplicate messages, then an attacker need only intercept one valid message, and without bypassing confidentiality or integrity protections, can simply replay the message to cause the intended behavior.

[42] https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

Typically resisting such an attack involves including a monotonically increasing counter or a random nonce within the envelope of the protocol where there is assurance of this value's integrity.

- **Brute force mitigations**. Allowing an attacker unlimited attempts at guessing the correct credentials can allow them to guess passwords and keys. Typical brute-force mitigations provide a time penalty for each invalid attempt to limit the rate at which guesses can be made. To avoid penalizing legitimate activity (which can affect manufacturing yield), this penalty may only apply to invalid guesses after a certain threshold, such as 10 attempts.
- **Certificate pinning.** Validation of the public key certificates used to communicate with external systems, like the back-end provisioning service, is necessary to determine authenticity and trust. This avoids active MitM attackers who attempt to subvert the communication channel by abusing the handshake and key exchange process. Certificate pinning is a method by which the expected certificate (or its root) is made known to the endpoint, by including it (or a hash of it) in the firmware for comparison. The system can then reliably reject unexpected certificates. Caution should be taken to avoid pinning certificates that can expire before they can be replaced, as this has led to costly product recalls[43]. Revoking (prematurely expiring) pinned certificates can be similarly problematic, and caution and additional planning is warranted.
- **Side-channel attack resistance**. Side channels are ways in which a computer system can leak information about a secret without intending to do so. This can be through any number of means, including error messages, timing information, or commonly Radio Frequency, Electro-Magnetic or power related signals. A classic timing side channel is when a timing-dependent comparison method is used to compare a secret like a password. An attacker able to measure the timing of the comparison may learn how many bytes of the secret are correct, which allows much faster brute forcing. Such attacks can be devastating when applied to retrieve a shared secret: the "break once, break everywhere" model significantly reduces effort for conducting widespread attacks. Mitigating timing side-channels involves the use of timing-invariant comparisons and masked computations when dealing with secret information. To mitigate more complicated cryptographic side-channels, the selection of an accepted cryptographic library is vital. Any cryptographic library that is well maintained and undergoes regular security assessments or research[44] is likely to be a more reasonable choice than a library that has not seen security updates for a prolonged period. Many commercially available libraries unfortunately fit into this later group.
- **Mutual-TLS (mTLS)**. mTLS is an extension to the more common TLS protocol that allows both ends to validate the authenticity of the other. If your back-end provisioning system needs to reject requests from non-legitimate devices, such as counterfeits, then this may be a viable protocol strategy. While mTLS is a common solution providing such

[43] https://blog.logitech.com/2017/11/09/update-will-replace-logitech-harmony-links/

[44] https://www.nccgroup.trust/uk/our-research/technical-advisory-rohnp-key-extraction-side-channel-in-multiple-crypto-libraries/

functionality, there are other protocol design options that provide similar mutual authentication capabilities. Take care when implementing mTLS to avoid endpoint certificates that expire. Should they expire prematurely while the device is unconnected and sitting in a warehouse, the results could be costly[45].

Storage Security Requirements

After provisioning sensitive assets (e.g., secrets, keys, passwords, tokens, identities), they need to be protected (data-at-rest security). The main security properties of a robust storage system are integrity and confidentiality.

In this section, we specifically exclude the portions of the persistent memory that are immutable. While such data may also be stored in a "filesystem" structure, they do not generally change at runtime and should contain no sensitive secrets (beyond intellectual property). Thus, they can be better protected by other measures, such as write protection and the normal Secure Boot process (look for mechanisms such as dm-verity[46] on Linux/Android). This implies the requirement that the mutable and static portions of the filesystem are stored in separate partitions.

The integrity protection of a secure filesystem needs to include more than mere checksums. Checksums and CRCs can handily detect accidental data corruption, however, these are trivially forged by a malicious actor, so stronger methods are needed. Use a keyed hash-based message authentication code (HMAC)[47] using a hash from the SHA-2 family of functions in place of a checksum to provide this integrity protection. HMAC is specifically difficult to misuse, and so is highly recommended; however, this does not provide confidentiality.

An accepted solution to this problem is to use an authenticated encryption scheme, such as AES-GCM. This encrypts the data and also provides integrity protection such that modifications to the ciphertext can be accurately detected. The problem then shifts to managing the filesystem metadata and the filesystem encryption key.

Additional properties of a secure storage system include:

- **Anti-rollback protection**. As changes to the filesystem contents are made, the system evolves from one encrypted filesystem state to another. Both the old and new states are cryptographically valid, and so an attacker with direct access to the flash memory may simply revert the entire filesystem to the previous version without any need to bypass the encryption. This may allow the circumvention of higher level security features, such as data deletion, certificate expiry, or password attempt counters. To prevent this, a monotonically increasing counter needs to be implemented and stored outside the

[45] https://www.cepro.com/article/quirky_terribly_embarrassed_over_wink_home_automation_hub_recall
[46] https://www.kernel.org/doc/Documentation/device-mapper/verity.txt
[47] https://en.wikipedia.org/wiki/HMAC

filesystem. Compare the current value against the value stored in the encrypted filesystem to detect rollbacks. The Replay Protected Memory Block (RPMB) in eMMC/UFS flash devices is ideally suited to implementing monotonic counters.

- A **hardware root-of-trust** is important to prevent a number of attacks against the storage system, such as memory replacement and memory swap attacks. Most often this root of trust takes the form of a cryptographic key stored within the hardware itself. Common implementations include keys stored in fuses or other OTP memory, or other hardware specific implementations. Where such hardware support does not exist, keys may be derived during initial system boot from a SRAM-based Physically Unclonable Functions (PUF)[48].
- **Access control** methods are most important when there is a degree of privilege separation in the software. It is unfortunately common that security assessments of embedded systems reveal that every application runs with system-level privileges. Not all processes should be able to access all data in the persistent memory. Where possible a mandatory access control[49] system should be used (e.g., SELinux[50] on the Linux and Android operating systems). For capable microcontrollers and SoC-based devices, a Memory Management Unit (MMU) can be effectively used by the operating system to provide a great deal of hardware-enforced memory access protection[51]. For simpler ECUs, designed around less feature-rich microcontrollers and firmware that may have limited abilities to enforce privilege separation, some measure of access control may still be possible. The use of a memory protection unit (MPU) can restrict access to certain memory ranges on many simple microcontrollers. Careful use and segregation of data encryption keys between internal modules can limit the exposure of plaintext data between subsystems.

A complete design of a secure filesystem is beyond the scope of this paper, but generally a layered approach that is rooted in hardware is necessary:

1. A hardware key is stored in OTP/Fuses, or some other immutable location within the microcontroller. This is only accessible to the cryptographic module, and is inaccessible through the hardware debug interfaces (JTAG/SWD). Often the chip supplier programs this key even before it is shipped from the foundry.
2. The hardware key is then used to derive other keys as needed using appropriate key derivation algorithms (keys for provisioning and encrypting the RPMB block in the eMMC/UFS flash memory, for example).
3. The RPMB block is limited in size but used to store persistent keystore root keys, monotonic counters, and other useful things.

[48] https://trust-hub.org/publications/P9.pdf
[49] https://en.wikipedia.org/wiki/Mandatory_access_control
[50] https://opensource.com/business/13/11/selinux-policy-guide
[51] https://crysp.uwaterloo.ca/courses/cs458/W14-lectures/Module3-3up.pdf

4. These keystore root keys protect an extensible keystore where all other keys are stored. The encrypted keystore can safely be stored in the regular filesystem due to it being encrypted (with an authenticated encryption mode) and protected from rollback using the monotonic counters.
5. Individual files containing the provisioned data (e.g., keys, certificates, identities) are stored in the regular filesystem encrypted with the keys in the keystore.
6. The regular filesystem and keystore both handle access control.

Access Control Requirements

Design strong access controls, as you would expect, to ensure only authorized identities are able to access a specific resource. In the context of a provisioning system, we are talking primarily about gating who can access the back-end system to obtain valid provisioning data, and gating who can connect to the embedded devices to access provisioning functionality or data. For such systems, passwords, especially fixed or shared, are not advisable. We strongly recommend a public-key based challenge-response protocol[52]. In such a scheme, only the corresponding public key of the authentication service (required for response validation) is stored inside the firmware image, and no persistent secrets are required. The corresponding private key should be suitably protected (in an HSM) on a back-end system that is itself access-controlled. As noted in the common mistakes section, there are existing implementations that include the private key within the tools that access the embedded system, however, these tools are deployed in an uncontrolled offline manner. Tools that exhibit this behavior are not suitable to avoid compromise of the private key. Only after authentication with the back-end system will the system respond to challenges and generate the correct response.

Role Management

Authorized agents will use the provisioning system to provision devices. They must be restricted from provisioning devices that they are not authorized to provision. If factory A is building device X, and factory B is building device Y, then factory A should not be capable of provisioning device Y, and vice-versa. This is often accomplished by granting different levels of permissions and privileges and separate login credentials to each factory, production line, or provisioning agent.

While role isolation and credential management is a clear solution here, implementation can be fraught when considering that many of the factory test stations (and the provisioning system is no exception) operate in a headless and autonomous manner without operator involvement. It may be tempting to embed the credentials in the software running on the provisioning test station, however, this leads to stolen software leaking the credentials. Stronger protection is needed, and this can be better accomplished with whitelisting on the back-end where the credentials are verified. Any abuse of credentials should be detected and investigated, as discussed in the following section.

[52] https://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication

Logging and Audit

It is easy to overlook the importance of an audit trail when designing a system. While the use of logging systems for simple failure analysis and fault diagnosis is somewhat obvious and more commonplace, these systems are highly useful in the detection and prevention of unauthorized or malicious activities. A well-implemented logging system that captures who accessed which system, when, why, how, and where, can reveal vital information to investigators during an incident response scenario. Knowing when and where devices were manufactured or repaired can help identify subversive rogue actors in your manufacturing supply chains that might be seeking to produce counterfeits or lower quality (and less secure) devices for use in your heavy truck ecosystem. Finally, a relatively small amount of automation effort up-front can leverage a logging system into a proactive detection and early warning system for such abuse, which can be useful to limit the overall cost of attackers in the shadow supply chains[53].

What you decide to log from your provisioning system will depend on the implementation, but at a high level should include all of the following:

- User or agent identification
- User location, a subjective measure of where they are located; can be physical address or site, IP address, or any other suitable proxy for location
- Network addresses like TCP/IP and MAC addresses where available
- Timestamp, as determined by a trusted source
- Device identity (serial number, VIN, etc.) for any operations involving devices-under-test
- Operation performed, commands issues to the device, etc.
- Versions of software used on the device, test station, provisioning software, etc. as appropriate

For completeness, we must state that some things in provisioning systems must NOT be logged, including passwords, credentials, keys, and anything else that is subject to confidentiality requirements. Log files by design frequently have fewer protections than these sensitive data items require. Leaking them through logs or other telemetry is a common mistake.

For detection, analysis and automation should seek to identify baseline normal activity in the log, flagging anything out of the ordinary for investigation. The details of this analysis are beyond our discussion here. This functionality will depend highly on the provisioning environments of the specific devices in question, and the ability to correlate this data with other sources, such as purchasing and business-to-business systems. Many of the abusive patterns that have been seen historically and that can still be expected are summarized in detail in previous NCC Group publications[54]. Some of the more important ones for a provisioning system are:

[53] https://www.chipestimate.com/The-Shadow-Supply-Chain/blogs/484

[54] https://www.nccgroup.trust/uk/our-research/secure-device-manufacturing-supply-chain-security-resilience/

- Device activation data mismatched with manufacturing records
- Cloned device identifiers (such as devices appearing in multiple places simultaneously)
- Provisioning system credentials used from an incorrect site or station type
- Inexplicably high volume provisioning from a single provisioning station
- Unexpected activity during quiet times like weekends or holidays

Key Management

At its heart, a provisioning system like that discussed in this document is just a key management system. Strong key management involves the entire lifecycle of a key, token, or authenticator, from generation, storage, transport, revocation, and destruction.

Key Generation

Cryptographic key generation is an important first step in any crypto system. A guessable key provides very little security benefits. To this end, keys should be generated using only appropriate algorithms and leveraging only true random entropy sources. Key derivation functions (KDF) are used to convert the initial keying material into a strong key suitable for the cryptographic algorithm which will use the key. These can be very useful when you need to generate a key from a password, or otherwise generate a key in a repeatable way (for example, see the discussion regarding RPMB provisioning in the *Storage Security Requirements* section above).

Key sizes should be selected based on the performance required and the lifetime of protection desired. This can be tricky in heavy vehicles where legacy hardware dominates the market. For new systems (i.e., 2019 and beyond), choosing key sizes is practically quite simple: use the biggest that is supported by your platform for the chosen algorithm. This is because, with modern algorithms, any of the key sizes shown in the table below are practically fine for the life of a truck (barring major quantum breakthroughs). Regardless, choosing the correct algorithms is at least as important as choosing a key size[55]. NIST guidelines[56] for keys size and protection equivalents are as follows (note that smaller key sizes than those listed here are explicitly considered too weak to mention):

AES Key Size (bits)	RSA Key Size (bits)	ECC Key Size (bits)
128	3072	256
192	7680	384
256	15360	512

Table 1. Equivalent key sizes across different cryptographic algorithms

[55] https://paragonie.com/blog/2019/03/definitive-2019-guide-cryptographic-key-sizes-and-algorithm-recommendations
[56] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt1r4.pdf

Keys should be unique to each device, and devices should not share keys. As noted previously, there is a greater risk when attackers compromise shared keys.

Generate keys only within a trusted system where attackers cannot compromise keys before use. Generally, this means generating them within:

a) Embedded devices that will use them, and securely exporting the public key encrypted with the provisioning system's public key
b) A back-end system and securely transporting them to the embedded device

Because securing the communication path during provisioning can be a challenge, and has a prerequisite for key generation anyway, the first method is often preferred. Note that with RSA, key generation is particularly slow, which implies the need for the second solution, or preferably just avoiding RSA altogether[57].

Key Storage

There are at least two storage locations of interest: on the embedded device where individual device secrets are provisioned; and on the back-end, where master secrets need to be stored and used safely.

On the embedded device, there are several options for secure storage. Modern devices will typically store sensitive data in a secure filesystem implementation, which protects the data using a layer of cryptography providing confidentiality and integrity guarantees. It should also prevent replay/rollback attacks directly against the flash memory itself. The filesystem protections will be multi-layered and based on a hardware backed cryptographic key that is unique per device, and this key is typically stored in fuses on the main processor, ideally provisioned in the foundry. It should only be accessible to the cryptographic modules leveraging it. The ECU must protect cryptographic secrets from compromise by hardware debug interfaces like JTAG and SWD.

The SAE J3101[58] described Hardware Protected Security Environment (HPSE), and the Android Keystore[59] are both robust solutions to this problem.

Alternate implementations of hardware-backed key storage include TPM sealed storage[60], external EEPROMs, Secure Elements[61] and other external security coprocessors. Such implementations are most frequently prescribed by the suppliers of these solutions. These are difficult to integrate securely on their own without enabling low-cost circuit-level attacks[62], and so

[57] https://blog.trailofbits.com/2019/07/08/f/
[58] https://www.sae.org/standards/content/j3101/ (paywalled)
[59] https://source.android.com/security/keystore
[60] https://en.wikipedia.org/wiki/Sealed_storage
[61] https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Secure-Element-15May2018.pdf
[62] https://www.nccgroup.trust/us/our-research/tpm-genie-interposer-attacks-against-the-trusted-platform-module-serial-bus/

take additional care when physical attacks are within the ECU's threat model. Protecting the communication path between the coprocessor and the host microcontroller requires careful design and manufacturing considerations to bootstrap the trust relationship between those two components. Furthermore, many of these solutions suffer from the *confused deputy* problem, where the coprocessor cannot authenticate the service running on the host processor, and so a compromised host can simply ask the coprocessor to use its privileges on behalf of the attacker.

On the back-end, protection of master secrets is crucial. Here, businesses should implement a robust HSM-based process (discussed further in the *Key Ceremonies* section below).

Key Transport

During communication with the embedded device, it is important for the data path to be protected for both integrity and confidentiality.

In the factory, an attacker's malware that is running on a factory test station should not be able to compromise the provisioned data. Securing this data path against an active MitM is relatively straightforward using TLS and certificate-pinning. Additional protection for the last-mile to the device can be implemented if the SoC has a pre-provisioned unique secret from the chip supplier.

In the field, when re-provisioning keys or secrets, use the same general mechanisms. Ensure that these mechanisms support external users so that their right-to-repair is not adversely impacted.

Key Revocation

Key and certificate revocation is a hard problem when dealing with difficult-to-update embedded devices. If a master secret is compromised, then informing all deployed devices and systems to no longer use or trust that secret can be difficult, especially when not all devices are online all of the time. Furthermore, embedded devices frequently lack a reliable and secure time source, which is often the trigger for automatic expiry.

We recommend mitigating the challenges by segregating product families and rotating these master secrets when new products are introduced. Should an older, less security-robust product be compromised, then this segregation will prevent that from affecting newer devices.

ECUs should use a counter for key and certificate expiry, based on the number of times the ECU has connected to some external trusted system. Without an independent tamper-resistant time source (such as GPS), this will be a far more reliable metric.

Planned obsolescence can be a problem when talking about embedded devices installed in heavy trucks with multi-decade lifespans. For this reason, certificates that expire are not a recommended implementation unless the provisioning service can guarantee replacement.

Certificates that expire while devices sit in a warehouse or while they are unconnected in the field can lead to costly denial-of-service and product recalls.

A related but separate issue is when the algorithm itself becomes obsolete within the lifespan of the vehicle. This would typically be a result of a new cryptanalysis breakthrough. Replacement of the affected firmware is the ideal solution, however, this may not always be practical or cost effective at-scale. Depending on the severity of the weakness discovered, it may be possible to buy time by increasing the cadence with which the keys are revoked and replaced rather than the entire algorithm. Fortunately, the pace of attack evolution against modern cryptographic algorithms is relatively slow, and the weaknesses are revealed many years before practical attacks are available, which allows for advanced planning.

Key Destruction

When businesses no longer need secrets, take care to ensure that compromise does not occur. This is because they may still be useful to an attacker to abuse legacy systems. A secure destruction process must be implemented.

NIST[63] and other formal guidelines[64] for data destruction can be followed, but general rules can be used as well:

- When ephemeral secrets are not needed anymore, destroy them at the earliest opportunity. Overwrite them in RAM or erase from flash as the case dictates.
- For long-term secrets the same advice applies, but also consider unintended copies created by underlying storage systems, such as wear-leveling and garbage collection routines in the underlying software layers.
- Any stored copies in back-end system databases may also need to be purged. Do not forget about system backups.
- Secure scrap processes in manufacturing and refurbishment facilities are important to prevent credentials from scrapped devices from being reused on new or counterfeit devices. Device scrap processes should track devices using strong identities and not merely tracked by weight. Businesses should enhance their scrap processes to limit the number of devices/components/secrets/identities that funnel into grey market supply chains.
- Blacklisting of expired and scrapped credentials can prevent reuse. Businesses can also use these to flag stolen devices and is a valuable data set in the audit process.

Key Ceremonies

While much of the day-to-day operation of a provisioning system is automated with the appropriate software handling the business logic, the most fundamental aspects, such as bringing the system online and generating master secrets, will still involve human

[63] https://www.nist.gov/publications/nist-special-publication-800-88-revision-1-guidelines-media-sanitization
[64] https://www.ncsc.gov.uk/guidance/secure-sanitisation-storage-media

administrators. To keep these administrators honest and limit insider abuse, clearly defined processes, or ceremonies, must be followed to minimize risk and to create audit trails.

Insider abuse can occur when authorized individuals or employees act against the security of the system. When managing back-end HSM systems, businesses need strong guarantees because of the high importance of the master secrets. Multi-party authentication and strong audit processes should be a part of any key management plan. The number of administrators that must be involved in the ceremony will depend on the importance of the system. We strongly recommend at least two administrators for a heavy truck provisioning system.

Note that transparency is often a requirement of such ceremonies, and thus there are many publicly published examples[65]. Following the guidance of the HSM supplier is preferred when implementing these ceremonies. For a relatively closed ecosystem such as that implemented by the heavy truck industry, this level of transparency many not be required, however internal compliance audits will still be important.

Solution Practicalities

Tying all of the above requirements together, we present an example provisioning system as shown in the diagram below. Even in this simplified view, we can start to see some of the complexity involved:

- During manufacturing, ECUs with some appropriate firmware will connect to a host test station over a local physical interface (e.g., typically serial, USB, or CAN).
- The ECUs themselves, will run only cryptographically signed firmware to ensure the integrity of the process. The process of bootstrapping of this Secure Boot process (i.e. enabling it) must be carefully choreographed between the chip supplier and the OEM factory.
- The test station will run some software designed to orchestrate the manufacturing test and provisioning sequences. Through the physical connection, it will issue commands to the ECU.
- The test station will, as needed, download provisioning data from, and upload test logs and other data to services that are exposed on the factory network.
- One of these factory test services is the provisioning service. It is responsible for provisioning services: providing device identifiers, collecting or distributing cryptographic keys and certificates, and authenticating factory test stations and users. The provisioning service may be located on-premises at each factory, or may be centralized and remote at the OEM. When an ECU supplier sells to multiple OEMs, this may be further complicated by the interoperability requirements of each OEM's provisioning system.

[65] https://www.iana.org/dnssec/ceremonies/22

- When on-premises, the provisioning appliance is hardened against physical attacks and is designed to operate only with periodic contact with the master provisioning service in the back-end.
- Behind the master provisioning service, strong key management systems are in place, based around an HSM that protects against compromise or misuse of the sensitive key material.
- Master secrets, like firmware signing keys, and certificate signing keys should be rotated for each new product or product family to hedge against compromise and to limit cross compatibility issues.
- Robust analytics should be used to monitor the baseline activity of the provisioning system and detect anomalies that might indicate abuse of the system.

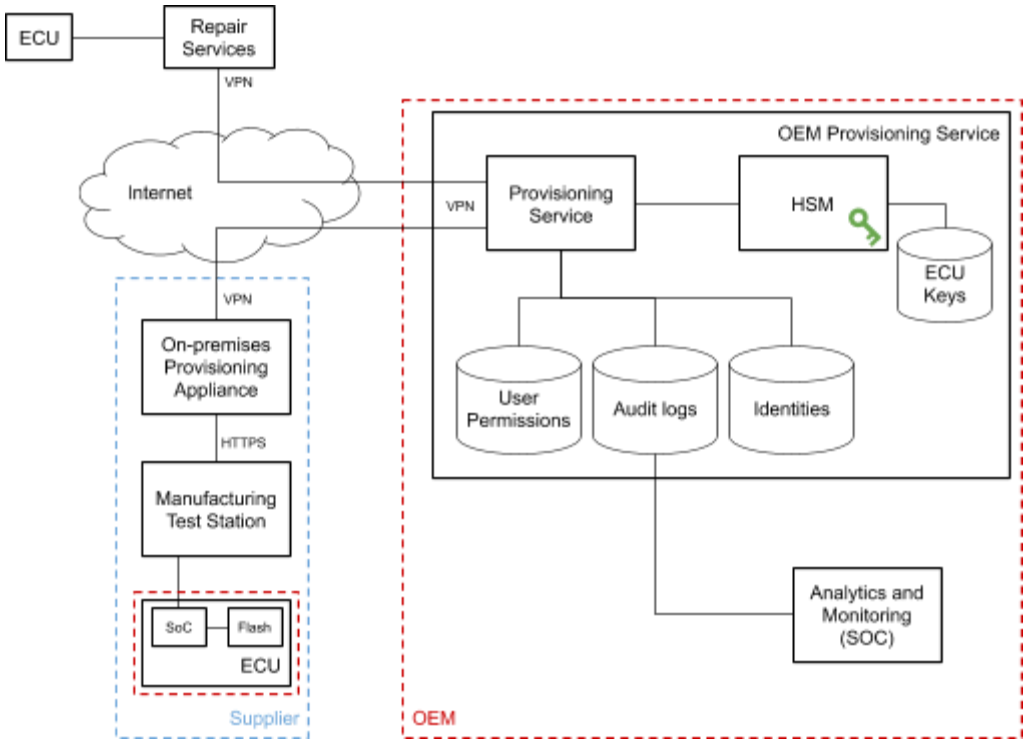


Figure 5: Example provisioning system architecture

Buy vs Build

References in the footnotes of this section are intended only to give a flavor of some of the commercial offerings that are available. Omissions or inclusions here are purely coincidental and do not constitute an endorsement by either the NMFTA or NCC Group.

Like any other part of the vehicle system, a buy-vs-build decision needs to be made by the business. NCC Group cannot make a full comparison of these options purely from the generic perspective taken in this paper. The weighing of options will be highly dependent on the design and implementation of existing ECUs and their manufacturing systems, and on the business

logic systems and processes that govern the vehicle ecosystem. All of this will differ from one OEM/supplier to another.

Building yourself might take slightly longer and involve more engineering effort, but it allows the solution to be highly tailored to the specific environment, vehicle ecosystem, and business processes as needed. When issues arise, or extensions and new features are desired, owning the system design, implementation, and deployment allows quick reactions to changing requirements and circumstances. This option depends on having the available resources to competently turn the requirements laid out here into a reality, and to operationalize such an infrastructure across the vehicle ecosystem.

Conversely, there are commercial entities offering provisioning solutions, or portions thereof, that can help satisfy the provisioning needs. Purchasing a system or service can get the ECU and vehicle to market much faster, and maybe even at a lower initial per-ECU price point. Most major cloud providers offer provisioning solutions to enroll and onboard devices into their IoT platforms[66,67,68]. If the vehicle systems are already using services from these cloud providers, then these might be a logical part of the solution. Integration into the manufacturing environment may vary. To aid with this portion of the puzzle, many contract manufacturing partners offer value-added services to support device provisioning[69], and may have existing options for integrating with both cloud services and devices. On the device-side, many chip suppliers offer solutions within their Board Support Packages (BSP) on which to build secure provisioning systems[70,71]. Furthermore, many chip resellers and suppliers have value-added services for pre-provisioning devices before they get to manufacturing[72]. Component pre-programming can be an important aspect to protect the “last mile” of the provisioning data path when the manufacturing environment itself is untrusted. Finally other suppliers exist that offer more or less complete solutions that are independent of manufacturing, component, and cloud providers[73,74,75,76,77,78,79].

Hybrid options of course exist, and are likely to be the most practical alternative. Most commercial solutions will fall short in some respects. Interoperability requirements with third party systems may dictate certain portions of the overall design. Solutions architects will need to

[66] Microsoft IoT Hub Device Provisioning Service: https://docs.microsoft.com/en-us/azure/iot-dps/concepts-security
[67] AWS IoT Device Provisioning: https://docs.aws.amazon.com/iot/latest/developerguide/iot-provision.html
[68] Zero touch device provisioning Google Cloud IoT Core: https://cloud.google.com/iot-core/
[69] https://www.arrow.com/en/research-and-events/articles/arrow-emea-nxp-secure-provisioning
[70] Infineon Secured Provisioning Services
[71] https://www.digikey.com/en/articles/techzone/2017/mar/efficient-and-secure-provisioning-for-the-iiot
[72] http://www.dataio.com/Technology/Sentrix
[73] https://www.deviceauthority.com/solutions/device-registration-onboarding-and-provisioning
[74] https://www.insidesecure.com/Products/Flexible-Provisioning-Solutions/Device-Provisioning
[75] https://www.certicom.com/content/certicom/en/products-and-services/asset-management-system.html
[76] https://www.securethingz.com/products/prog_services_product/
[77] https://www.nextplatform.com/micro-site-content/securely-provisioning-keys-cryptomanager-security-platform/
[78] https://teserakt.io
[79] https://www.arm.com/products/iot/pelion-iot-platform/device-management/provision

create or customize glue logic for integration between systems. The complexities of the vehicle ecosystem demand a complex provisioning system. Above all, if you are purchasing a system in whole or in part, then you need to ask the hard questions of the suppliers, and determine if they are correctly implementing all of the requirements.

Summary

A secure provisioning system is vital for securing ECUs in the face of modern and emerging threat models. The heavy trucking industry will achieve this by the application of strong cryptographic protections and key management processes. Applying these long-established security concepts to heavy trucking applications can be challenging due to the complex distributed ecosystem, manufacturing practicalities, long lifespan of the vehicles, and the remoteness in which they are sometimes required to operate. The security properties of any system should account for the protection of data-at-rest, data-in-transit, access controls, and more. By following the requirements laid out in this paper, the heavy trucking industry can build, integrate, or acquire a robust provisioning system that can support the desired security guarantees of the vehicle system, throughout the lifetime of the vehicle. A provisioning system is by no means sufficient to fully secure the ECUs and vehicle, however they provide a strong foundation on which a secure vehicle system can be built.

Glossary of Acronyms

AES: Advanced Encryption Standard.
AI: Artificial Intelligence.
AV: Automated Vehicles.
BCM: Body Control Module.
CA: Certificate Authority.
CAN Bus: Controller Area Network.
CSPRNG: Cryptographically Secure Random Number Generator.
DES/3DES: Data Encryption Standard.
DH: Diffie-Hellman (key exchange protocol).
DoS: Denial of Service.
ECB: Electronic Codebook (cipher mode).
ECC: Elliptic Curve Cryptography.
ECDH: Elliptic Curve DH.
ECU: Electronic Control Unit.
EEPROM: Electrically Erasable Programmable ROM.
eMMC: embedded Multimedia Card
FFA: Fault and Failure Analysis.
FOTA: Firmware Over-the-Air.
GCM: Galois Counter Mode (cipher mode).
GDPR: General Data Protection Regulation.
GPS: Global Positioning System.
HPSE: Hardware Protected Security Environment.
HSM: Hardware Security Module.
HWRNG: Hardware Random Number Generator.
IMEI: International Mobile Equipment Identity.
IoT: Internet of Things
ITS: Intelligent Transport System.
JTAG: Joint Test Action Group (hardware debug interface).
LFSR: Linear Feedback Shift Register.
MAC address: Media Access Control address
MD5: Message Digest Algorithm.
MitM: Man-in-the-Middle.
ML: Machine Learning.
MMU: Memory Management Unit.
MPU: Memory Protection Unit.
mTLS: Mutual TLS.
NIST: National Institute of Standards and Technology.
OBD2: On-Board diagnostics.
OCD: On-Chip Debugger.
OEM: Original Equipment Manufacturer.
OTP: One-time Programmable.
PCM: Powertrain Control Module.
PKI: Public Key Infrastructure.
PTO: Power Take Off.
PUF: Physically Unclonable Function.
RC4: Rivest Cipher.
RMA: Return Materials Authorization.
ROM: Read-Only Memory.
RPMB: Replay Protected Memory Block.
RSA: Rivest–Shamir–Adleman (cryptosystem).
SHA: Secure Hash Algorithm.
SKU: Stock Keeping Unit.
SoC: System on Chip.
SSL: Secure Sockets Layer.
SWD: Single Wire Debug.
TCP/IP: Transmission Control Protocol/Internet Protocol.
TCU: Telematics Control Unit.
TLS: Transport LEvel Security.
UART: Universal Asynchronous Receiver/Transmitter.
UFS: Universal Flash Storage.
USB: Universal Serial Bus.
V2I: Vehicle-to-Infrastructure (communication technology).
V2V: Vehicle-to-Vehicle (communication technology).
V2X: Vehicle-to-X (collective term for V2V and V2I).
VIN: Vehicle Identification Number.