Jonathan Malangoni

Design and development of a smart parking device

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

11 May 2018

Metropolia

Author Title Number of Pages Date	Jonathan Malangoni Design and development of a smart parking device 50 pages 11 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Embedded Engineering
Instructors	Keijo Länsikunnas, Senior Lecturer

Recently developed smart parking mobile applications provide their users with valuable savings in time and money, however a significant portion of parking consumers cannot use these applications as they require a smartphone. In order to serve these user groups, a new smart parking system was designed to support a standalone client device. This device can be used to remotely make parking payments without the need for a smartphone, or to display the vehicle's arrival time in free parking areas that require a parking disc. The device can also detect movement and rest in order to automatically end parking when the user's vehicle departs and to activate or deactivate the display when the user's vehicle is parked or driving to conserve battery.

This thesis documents the design and implementation of the embedded system software for the smart parking device and its interface to cloud-based services. Particular focus is given to discussion of secure and dependable service delivery and mobility detection and classification as core feature requirements of the device. Design choices, implementations, and challenges are explored and evaluated, including discussion of topics such as secure and dependable systems, embedded cryptosystems, remote firmware updates, and mobility detection and classification methods.

The smart parking device is currently available and supported for use in all Helsinki city public parking areas. It is the only remote parking payment method currently supported by the city that does not require a smartphone.

Keywords	Internet of Things, smart parking, embedded systems, security, dependability, mobility

Contents

List of Abbreviations

1 Introduction 1

1.1 Project setting and overview 1
1.2 Smart parking device requirements 3

2 Background 5

2.1 Secure and dependable service delivery 5
2.1.1 Attributes of a secure and dependable system 5
2.1.2 Dependability 9
2.1.3 Security 11
2.2 Mobility detection and classification 14
2.2.1 Mobility states 14
2.2.2 Movement detection 15
2.2.3 Risk and cost analysis 16

3 Design and implementation 21

3.1 Secure and dependable service delivery 24
3.1.1 Dependability 24
3.1.2 Security 31
3.2 Mobility detection and classification 36
3.2.1 Accelerometer and device mobility states 36
3.2.2 Initial settings for mobility classification 38

4 Results and analysis 38

4.1 Secure and dependable service delivery 39
4.1.1 Dependability 39
4.1.2 Security 41
4.2 Mobility detection and classification 43

5 Conclusions 45

References 47

List of Abbreviations

3G	Third generation
ADC	Analog-digital converter
API	Application programming interface
BOM	Bill of materials
CA	Certificate authority
CIA	Confidentiality-integrity-availability
CRC	Cyclic redundancy check
DH	Diffie-Hellman
DNS	Domain name system
EC	Elliptic-curve
ECDH	Elliptic-curve Diffie-Hellman
ED	Edwards-curve
EEPROM	Electrically erasable programmable read-only memory
GPIO	General-purpose input/output
GSM	Global System for mobile
I2C	Inter-integrated circuit
IMU	Inertial measurement unit
IP	Internet protocol
LTE	Long-term evolution
MEMS	Microelectromechanical systems
MNO	Mobile network operator
MTBF	Mean time between failures
MTTR	Mean time to repair
MVC	Model-view-controller
NTP	Network time protocol
OLED	Organic light-emitting diode
PKCS	Public key cryptography standard
PKI	Public key infrastructure
PPM	Parts per million
RAM	Random-access memory
REST	Representational state transfer
RNG	Random number generator
RSA	Rivest-Shamir-Adleman
RSSI	Received signal strength indicator
RTC	Real-time clock

SHA	Secure hash algorithm
SIM	Subscriber identity module
UART	Universal asynchronous receiver-transmitter
UI	User interface

1 Introduction

Until recently, public parking in the capital city of Helsinki, Finland required users to locate a payment kiosk, retrieve a ticket, and return to their vehicle every time they needed to reserve or extend their paid parking time. The introduction of smartphone applications helped relieve these disruptions for many consumers, however a significant proportion of parking users are unable to take advantage these new services as they do not own a smartphone or cannot procure one from their employer for making remote parking payments related to work. To bring the convenience of remote parking payments to these users, a new smart parking system was created. This thesis documents the process of designing and developing the embedded software for the system's standalone client device focusing on two of its key feature requirements: 1) secure and dependable service delivery, and; 2) mobility detection and classification.

1.1 Project setting and overview

In the past, users paying for public parking in Helsinki were generally required to make a cash or credit-card payment at a payment kiosk nearby their parking area. In return for their payment, the user would receive a paper ticket indicating their time of arrival and amount of parking time reserved. The user would then need to return to their vehicle and place the ticket on their vehicle's dashboard. If the user needed to extend the amount of time purchased, they would need to return to the kiosk, make an additional payment, and place the new ticket in their vehicle.

For a user, the physical journey to the kiosk and back to their vehicle can be an inconvenience that becomes especially disruptive when it includes a return to the parking area from another place in order to extend parking time. As such, users often opt to pay extra and "overbook" their parking reservation in order to ensure that they will not need to return to the parking area before they are ready to depart.

To help alleviate this burden on the public, Helsinki city opened its parking payment system to selected third-party service providers, allowing them to handle parking payments on behalf of users. This enabled the creation and use of smartphone applications for public parking payments. For users, the ability to pay for parking on their smartphone removed the need to travel to a payment kiosk. Furthermore, users were

able to avoid the need to overpay for parking, either by extending their parking time from the application as needed, or by overbooking a reservation and then shortening it retroactively to end at the time of departure.

The introduction of smartphone applications for remote parking payments was well-received. However, not all public parking users could take advantage of these new services easily. Vehicle fleet managers such as rental car agencies and utility providers that use public parking services as part of their daily work may find it difficult to provision smartphones for all of their drivers. Furthermore, many individual public parking users do not have a smartphone. According to some research, 41-45% of Finns aged 55 or older did not own a smartphone in 2015 [1] [2].

In order to serve these user groups, a new smart parking system was designed to support a standalone client device. This device can be used to remotely make payments for public parking in Helsinki with all of the aforementioned benefits of a smartphone application, but without the need for a smartphone. Additionally, since the device is designed to be left in the user's vehicle, it can be used as a parking disc to display the vehicle's arrival time in limited-time free parking areas that require it.

The smart parking system as a whole consists of a cloud-based service and a standalone device as a client of that service. The cloud-based service consists of back-end and front-end web applications, databases, and other cloud-hosted core infrastructure. The back-end web application is a service that responds to requests from devices through an application programming interface (API) following the representational state transfer (REST) model. The front-end web application services browser-based clients as a user interface (UI) for management of devices, accounts, and payment details, but is not used to make parking reservations.

When a user wishes to reserve paid parking time, they use the device to start a "parking event" of a set duration with the cloud-based service. When they are ready to depart, the user can change the duration of their parking event to last exactly as long as they have been parked by using the device to stop the parking event. This allows the user to pay only for the amount of time they have parked and to do so without needing to leave their vehicle or own a smartphone.

The cloud-based service utilizes an API provided by the city of Helsinki to reserve parking time on behalf of users. City parking enforcers can check the cloud service to determine if a vehicle has paid for parking and what time the vehicle must depart before issuing a penalty.

The smart parking system's cloud-based services and device hardware and software were all designed and implemented simultaneously within a small start-up company focusing on smart parking solutions for the Finnish market. The company's engineering team at this time consisted of four people: an embedded software developer, a hardware engineer, a senior backend web software developer, and a junior full-stack web software developer. No project manager, product manager, or product owner was employed, however the company CEO and a technical advisor fulfilled these roles where needed. Design of the interface and interrelated processes between the device and cloud-based services was shared between the embedded developer and the senior web developer, including the API, device registration process, and security procedures and protocols.

1.2 Smart parking device requirements

At the beginning of the planning and design process, formal and informal requirements for the smart parking device were given to the engineering team. The main requirements for the smart parking device related to its embedded software are summarized as follows:

- Payment mode functionality: The device can act as a client of cloud-based services to deliver core capabilities including starting and stopping paid parking events.
- Disc mode functionality: The device can be used as a digital parking disc to display a time of arrival (rounded up to the nearest half-hour, per local legislation) for parking enforcers to inspect.
- Secure and dependable service delivery: The device must ensure secure and dependable service delivery, including the ability to remotely update firmware in order to address bugs and enhance capabilities with little to no user interaction.
- Mobility detection and classification: The device can automatically sense when it is moving or stationary in order to: A) stop paid parking events when the vehicle departs, and; B) activate/deactivate the display depending on the vehicle's mobility in order to conserve power when used as a parking disc.

- User interface: The device includes a UI with a display and buttons that can be used to deliver service to the user in an intuitive manner.
- Power consumption: The device can be powered by a rechargeable battery with a typical full-charge operational lifetime of 1 week or longer when used as a parking disc, or 1 month or longer when not used as a parking disc.

Further requirements were also given regarding the device's physicality and operating conditions, cost, development time, and manufacturing, which are not discussed in depth here. Specifications of the device requirements may be considered to be loosely defined; for example, while service delivery was required to be secure and dependable, no specification was given to define or measure the security or dependability of service delivery. Likewise, requirements validation and verification were often done on an informal and ad hoc basis.

Two of the above requirements are chosen here as the focal topics for this thesis: 1) secure and dependable service delivery including remote firmware updates, and; 2) mobility detection and classification. Further discussion of these topics will touch upon elements of other main requirements listed as well. Indeed, any combination of these requirements could have been chosen for review, but the selected topics were preferred as they offered both a breadth of relevant research and opportunities for reflection on design and implementation.

The remainder of this thesis is presented as follows. First, the *Background* section explores the selected requirements with related research to familiarize the reader with relevant topics and available design choices. The *Design and implementation* section describes what design choices were made and how their implementations were realized. The *Results and analysis* section reviews the challenges met during the project and critically examines the solutions employed. Finally, the *Conclusions* section completes the thesis with observations and recommendations for similar projects conducted in the future.

2 Background

2.1 Secure and dependable service delivery

This section discusses the selected device feature of secure and dependable delivery of the smart parking system's core services. For the case at hand, the core services being delivered by the device consist of starting and stopping parking events. As the terms "secure" and "dependable" were not extensively defined descriptively or measurably as part of the device requirements, these terms need some further explanation. Therefore, a basic model for defining and integrating attributes of security and dependability is introduced in the discussion below, followed by an evaluation of methods that can be employed to achieve these attributes. Other services provided by the device, e.g. mobility detection/classification and use as a digital parking disc, are related but not required for core service delivery, and thus are not discussed under the model applied here.

2.1.1 Attributes of a secure and dependable system

To help shape discussion here, an integrated conceptual model for secure and dependable systems based on the work of Avižienis et al. [3] is employed to: 1) identify attributes as goals for the system and service delivery; 2) classify faults that lead to errors and possibly failures of the system, and; 3) identify suitable methods of fault tolerance and removal. The model uses the well-known "CIA triad" of information security [4] to specify primary attributes for the principle domain of security, and includes overlapping and additional primary attributes for the principle domain of dependability. These primary attributes of security and dependability are defined in Table 1.

Table 1. Primary attributes of secure and dependable systems, retrieved from [3].

Attribute	**Principle domain(s)**
Confidentiality – Absence of unauthorized disclosure of information	Security
Integrity – Absence of improper system alterations	Security, Dependability
Availability – Readiness for correct service	Security, Dependability

Attribute	Principle domain(s)
Reliability – Continuity of correct service	Dependability
Safety – Absence of catastrophic consequences to the user and the environment	Dependability
Maintainability – Ability to undergo modifications and repairs	Dependability

Some adaptations to the list of attributes above are required here. Firstly, since communications between the device and cloud-based service initiate parking reservations on behalf of an authorized user, authenticity is required as a secondary attribute for service delivery to be secure. Authenticity here can be defined as a special form of integrity that also ensures the originator of a message is who or what they claim to be [3] [5]. Secondly, as this case does not include any safety-critical components, applications, or potential catastrophic failures, the attribute of safety is not of primary concern and will not be discussed further here.

Further definitions according to the model are as follows. Dependability can be defined as a system's ability to deliver its core service in a trusted manner while avoiding unacceptably frequent or severe service failures. A *failure* is defined as the inability of the system to provide its intended service. Failures are caused by *errors*, which are defined as deviations of the system from its correct state. An event that causes an error is defined as a *fault*. In other words, a fault causes an error, which in turn can cause a failure. This failure may then result in a fault manifested in another part of the system through the mechanism of error propagation. It is important to note, however, that not all errors inevitably lead to failures. Furthermore, not all failures are *service failures* that result in incorrect service delivery to other systems or users. A failure of one component of the system may propagate to other components, but this propagation may be stopped by intentional or unintentional system recovery. In short, the end result of a fault depends on how its resulting errors are detected and handed. Faults are classified and grouped in Table 2.

Table 2. Classification of elementary faults, retrieved from [3].

Viewpoint	Classification
Phase of creation or occur-	Development – Occurs during system development, maintenance during use, and generation of procedures to operate or maintain the system

Viewpoint	Classification
rence	Operational – Occurs during service delivery of the use phase
System boundaries	Internal – Originates inside the system boundary
	External – Originates outside the system boundary and propagates errors into the system by interaction or interference
Phenomenological cause	Natural – Caused by natural phenomena without human participation
	Human-made – Results from human actions
Dimension	Hardware – Originates in, or affects, hardware
	Software – Affects software, i.e. programs or data
Objective	Malicious – Introduced by a human with the malicious objective of causing harm to the system
	Non-malicious – Introduced without a malicious objective
Intent	Deliberate – Result of a harmful decision
	Non-deliberate – Introduced without awareness
Capability	Accidental – Introduced inadvertently
	Incompetence – Results from lack of professional competence or from inadequacy of the development organization
Persistence	Permanent – Presence is assumed to be continuous in time
	Transient – Presence is bounded in time

For this case, some classifications of faults are of particular importance for specific attributes and contexts, while other classifications are generally irrelevant. The view is taken that any information security vulnerabilities (note: not exploits) existing in the interface between the device and cloud-based service may be classified as developmental (introduced during design and/or implementation), internal or external (depending on where on the interface the vulnerability is), and permanent (predictably exploitable until corrected). For faults related to dependability, both internal and external faults are significant as the device acts as a client for the external cloud-based service, and service failure could be caused by an internal fault of the device or an external fault in its connectivity to the cloud-based service or the service itself. Service availability is mostly affected by operational faults internal to the device such as system halts, while service reliability is susceptible to external operational faults such as loss of connectivity originating from a mobile network operator (MNO) as internet service provider. For all attributes of the service only software faults are deemed relevant, although this in-

cludes faults originating in hardware as long as their resulting failures are manifested as faults in software.

Further classification of faults for purposes of this discussion is unwarranted, despite their potential use for other goals. Failures and errors can also be classified extensively, however for the case at hand these classifications are not of primary importance and are thus excluded from discussion.

Four conceptual methods for managing faults are identified by the applied model: prevention, tolerance, removal, and forecasting. Fault prevention refers to processes for reducing the number and scope of faults introduced during design and development. Fault tolerance refers to methods for error detection and recovery to avoid system failure in the event of a fault. Fault removal refers to methods for verifying, diagnosing, and correcting faults through testing, maintenance, and software updates. Fault forecasting refers to qualitative and/or quantitative evaluation of the system's behaviour when activating or reacting to a fault.

Depending on the system, certain fault management methods are more applicable to particular attributes than others. In this case, attributes of security are best attained through fault prevention by way of a secure design and implementation, availability and reliability can be served through fault tolerance with proper error detection and handling, and maintainability can be seen as the system's ability to perform in-operation fault removal. Fault forecasting, while a useful and insightful tool, is omitted from this discussion.

The immediately following discussion on secure and dependable service delivery is broken into two parts. The first focuses on the principle domain of dependability and how techniques of fault tolerance and fault removal can be used to attain availability, reliability, and maintainability of service. The second part focuses on ensuring attributes of confidentiality, integrity, and authenticity in the security domain, and reviews cryptographic methods for preventing malicious faults, i.e. attacks and exploits, that may erode these attributes.

2.1.2 Dependability

2.1.2.1 Availability, reliability, and fault tolerance

As the notions of availability and reliability often differ amongst relevant research [6] [7], some expansion on their definitions is warranted here. In this case, availability is defined as the probability that the service delivered will be correct at any given time, often referred to as uptime. This is also known as *inherent* availability, and may be calculated as in Figure 1 using measured values for mean time before failure (MTBF) and mean time to repair (MTTR). Reliability is a related but separate concept referring to the probability that the service delivered will be correct *up to* a given time. In this case, reliability as a function of time (t) may be simply expressed as the exponential distribution shown in Figure 1 below with a given MTBF, or its reciprocal failure rate (λ). [8]

$$A = \frac{MTBF}{MTBF + MTTR}$$

$$R(t) = e^{-t/MTBF} = e^{-\lambda t}$$

Figure 1. Equations expressing availability (A) and reliability (R)

The two attributes may diverge in certain cases. For example, a highly available but unreliable system may be correct 99% of the time but still fail for a short period every hour. A system of high reliability but low availability may be correct for a long and continuous period of time but, when it does fail, it fails spectacularly for an extended period. In this case, techniques for ensuring the system's dependability generally influence availability and reliability simultaneously and to equal effect.

For the smart parking device, the principle method of avoiding breakdown of availability or reliability of service during operation is to use fault tolerance techniques to detect, handle, and recover from faults and errors before they propagate into service failures. Categorical definitions of these techniques are presented in Table 3.

Table 3. Fault tolerance techniques of error detection and system recovery, retrieved from [3].

Strategy	**Technique**
Error detection – Identifies the presence of an error	Concurrent detection – Takes place during normal service delivery
	Preemptive detection – Takes place while normal service delivery is suspended; checks the system for latent errors and dormant faults
Error handling – Eliminates errors from the system state	Rollback – Brings the system back to a saved state that existed prior to error occurrence
	Rollforward – State without detected errors is a new state
	Compensation – The erroneous state contains enough redundancy to enable error to be masked
Fault handling – Prevents faults from being activated again	Diagnosis – Identifies and records the cause of errors in terms of both location and type
	Isolation – Performs physical or logical exclusion of the fault components from further participation in service delivery, i.e. makes the fault dormant
	Reconfiguration – Either switches in spare components or reassigns tasks among non-failed components
	Reinitialization – Checks, updates, and records the new configuration and updates system tables and records

As Table 3 shows, there are many strategies for detection and recovery available. A system reset can be described as rollback combined with reinitialization, whereby the system is restored to its initial power-up state which in most cases can be assumed to be stable and error-free. Likewise, if multiple functional units provide the same service and a service failure occurs in one of these units, compensation can be employed by using another redundant unit for service. If the failure is not intermittent, isolation may also be employed by disabling the failed unit to prevent further errors. Handling of detected internal faults is usually followed by corrective maintenance to remove the fault, which is discussed in the next section.

2.1.2.2 Maintainability and fault removal

Maintainability is an important means for improving other attributes by enabling fault removal during development or during active deployment of service. Fault removal during development can generally be considered as validation and verification testing.

While testing is arguably the most powerful tool a software developer has for identifying and removing faults, it is an exceedingly broad topic that will not be covered here.

Fault removal during use includes four general types of maintenance: preventative and corrective maintenance, which are further generalized as repairs, and adaptive and augmentative maintenance, which may be considered as modifications to the system's configuration. Preventative maintenance aims to identity and remove faults before they manifest as errors in use, while corrective maintenance isolates and removes known faults. Adaptive maintenance makes changes to the system to adjust to its environmental changes, and augmentative maintenance extends the system's functionality. For the case at hand, modification maintenance is sparingly employed and configuration options are too few to warrant further discussion. Also particular to this case is the unimportance of the division between corrective and preventative maintenance types, since the same procedure is used for both operations, namely a remote firmware update.

Recall that as part of the system requirement for secure and dependable service delivery, the device must support a procedure for remote firmware updates in operation. The update procedure requires its own use of fault tolerance techniques in order to attain dependability. These and other strategies employed during software maintenance updates are discussed in the *Design and implementation* section.

2.1.3 Security

The following section on the primary domain of security discusses cryptographic methods for ensuring confidentiality, integrity, and authenticity of information passed between the device and the cloud-based service. As opposed to the previous section on dependability, discussion here is focused on applied techniques rather than the conceptual framework above, and is mainly drawn from additional research.

In the security domain, attributes of confidentiality, integrity, and authenticity of data are often attained through the use of cryptography, including techniques of encryption, hashing, and message authentication codes (MACs) or digital signatures [9]. It is also often the case that two or more of these techniques are combined to achieve one or more attribute. The discussion that follows begins with a review of these techniques and ends with an overview of currently commonplace internet standards and protocols implementing the techniques covered.

2.1.3.1 Confidentiality

Encryption algorithms are used to convert plaintext messages into ciphertext and vice versa to ensure confidentiality during storage and transport. They can generally be separated into two types: asymmetric and symmetric. Asymmetric algorithms use different keys for processes of encryption and decryption, while symmetric algorithms use the same key. Asymmetric cryptography is typically employed using a combination of public and private keys. The terms public and private here refer directly to the need for confidentiality of the keys themselves—public keys may be distributed without regard for confidentiality, while private keys must be kept secret in order to maintain their use.

An important limitation to asymmetric encryption algorithms is that in practice only "one-way" encryption is supported. Assuming only one party holds the private key of a public-private key pair, anyone with the public key may encrypt their messages but the private key holder is the only one able to decrypt them. Where confidentiality or authenticity is required for messages originating from and sent to all parties, multiple asymmetric key pairs must be generated and their public keys distributed—one for each party—or another scheme must be used. Another important limitation especially in the context of embedded computing is that procedures of asymmetric encryption and decryption (and sometimes key generation as well) can take orders of magnitude more processing cycles and therefore power consumption than their symmetric counterparts [10]. This greater resource use of asymmetric algorithms extends also into the amount of program memory required by their implementations, as is shown in later discussion in the *Design and implementation* section.

For these reasons, asymmetric and symmetric algorithms are typically used in tandem via a hybrid scheme, whereby slower asymmetric algorithms are used only to exchange (and sometimes generate) symmetric keys, which in turn are used to achieve high-throughput, low-overhead encryption and decryption of following data. The exchange of symmetric keys typically uses one of two basic methods: asymmetric encryption or Diffie-Hellman (DH) key exchange. In the first method, one party uses a second party's public key to encrypt a pre-generated symmetric key and sends it to the second party, who then uses their private key to decrypt it. In the second method, two parties use their own and the others' public keys to create a shared secret, which in turn is used to create a shared symmetric key. In this case, the symmetric key is never

sent over any medium. Both methods are supported for key exchange in the commonplace Transport Layer Security (TLS) internet protocol [11] [12] [13] used in HTTPS.

2.1.3.2 Integrity and authenticity

Encryption alone may ensure confidentiality, but additional methods are needed to ensure integrity in general and authenticity in particular. Integrity of data can be ensured by applying a cryptographic hash function to the data to create a compressed digest. Assuming the hash function is sufficiently resistant to collision and preimage attacks, the digest serves as a unique imprint of the original data that can be sent to the receiver via a secure channel and independently verified by the receiver to confirm that the full-length data has not been altered. [14] [9]

Authenticity builds upon integrity by ensuring the identity of a message's origin. Without this authentication of identity, a malicious party could perform a "man-in-the-middle" (MITM) attack by posing as a sender's intended destination. In practical terms, authenticity is often realized by combining a hash function with a private key, whereby only a party in possession of the key could produce its output. This keyed hash function ensures integrity via hashing and identity via possession of the private key. While their implementations differ, the basic concept of combining a hash with a private key for authentication purposes is used both by symmetric cryptography in the form of MACs and asymmetric cryptography in the form of digital signatures. MACs can also be combined with (typically symmetric) encryption to perform authenticated encryption (AE), which provides confidentiality, integrity, and authenticity, and is supported by TLS 1.2 [11].

2.1.3.3 Trust

As with encryption, asymmetric authentication methods are "one-way," since the holder of the private key is the only party capable of signing messages. As such, MACs are generally used to ensure authenticity of communications between two trusted parties, while digital signatures are used to help establish that trust by authenticating the public keys distributed during the hybrid key exchange process described above.

This establishment of trust is standardized as part of the common internet X.509 public key infrastructure (PKI) used by TLS and thus HTTPS [11]. X.509 certificates contain the public key and identity of a party, and are digitally signed either by the party itself or by another entity such as a trusted certificate authority (CA). By verifying this signature, authenticity of the certificate's public key and identity can be ensured as long as the verifier trusts the signing entity. For a public key user to support PKI, it must be initialized with one or more "root" certificates containing the assured public key and identity of a CA. Trust in the CA allows the client to trust its own self-signed certificates, certificates of third parties signed by the CA, and further certificates signed by these third parties if the CA had deemed them trustworthy. This certification path enables the client to establish authenticity of the public key and identity of any end-entity that is trusted by a CA. [15] [16]

While TLS and PKI are internet standards and nearly ubiquitously deployed, both possess a number of significant vulnerabilities. Notably, use of TLS 1.0 is currently not recommended due to the proven impact of attacks and increasing availability of exploits. [17] [18] [19]

2.2 Mobility detection and classification

In this section, the selected feature of mobility detection and classification is defined and discussed. Multiple methods for movement detection are identified and analysed according to risk and cost in order to distinguish the optimal set of solutions for the smart parking device.

2.2.1 Mobility states

Mobility detection and classification are used to: 1) automatically stop a parking event when the device's vehicle has driven away during an ongoing parking event in payment mode, 2) automatically turn off the device's display to conserve power when the user's vehicle has driven away in disc mode, and 3) automatically re-activate the device's display and restart the arrival time when the user's vehicle has parked in disc mode. It follows from these intentions that the device's method for detecting mobility should be able to classify between two mobility states: movement and rest.

Using the rest state as a proxy for parking may seem naïve on its face, since a vehicle may be stationary while driving, for example when it is waiting at an intersection or stopped in traffic. However, in the case of the device's application, the only penalty for classifying a vehicle as stationary when it is actually driving is a temporary increase in power consumption due to activating the display in disc mode. This impact can be safely ignored as it is quite insignificant compared to the consequences of other types of classification errors. Such consequences are discussed further below as a part of risk analysis.

Since the classification scheme required is binary, it follows that if the user's vehicle is not moving it must be at rest and vice versa. For the purposes of this application, it can also be assumed that any movement of the device is due to a movement of the vehicle it is in (see the *Design and implementation* section below for further details on this assumption). Therefore, the method used for mobility detection and classification needs only to be able to reliably sense movement of the device to be able to distinguish between movement and rest. A selection of technologies and methods for achieving this goal is reviewed below.

2.2.2 Movement detection

Among the most common types of sensors used for detecting movement in consumer applications are inertial sensors, specifically accelerometers which measure translational acceleration, and gyroscopes which measure angular orientation. These sensor types may be combined in the form of an inertial measurement unit (IMU) to help account for errors experienced when using a single sensor type. Another type of sensor often included in IMUs is the magnetometer, which acts as a compass to provide heading information. If interference and error are disregarded, any change in acceleration, angular orientation, or heading sensed on the device can be attributed to movement. Thus, accelerometers, gyroscopes, and magnetometers are all valid candidate methods for detecting and classifying mobility on the device.

GSM and later generations (3G/LTE) of cellular networks can also be leveraged to estimate mobility. For example, research by Sohn et al. [20] demonstrates that classification of a GSM modem user's mobility as stationary, walking, or driving can be inferred by comparing network trace data including received signal strength indicator (RSSI) values from multiple cell towers in a time series. As an alternative method to traditional

sensor-based approaches, using cellular network trace data is an enticing option as it would require no additional hardware since a 3G modem is already included on the device for internet connectivity.

2.2.3 Risk and cost analysis

2.2.3.1 Risk analysis

Since multiple methods have been identified as potential solutions for detecting and classifying mobility, further comparison is needed to identify which solutions are optimal for the smart parking device. A simple qualitative model for risk assessment was used to identity risks associated with each method and to evaluate the likelihood and consequences of each risk both separately and collectively [21]. In this case, risk events can be categorized as false negatives and false positives when detecting and classifying the mobility states of movement and rest.

Table 4 lists different risk events and their consequences and assigned severity levels. Of all possible risk events, a false positive classification of movement when the vehicle is actually at rest is by far the most impactful. Firstly, it is the only risk event with a meaningful consequence that can occur without the user neglecting their personal responsibility as outlined in the smart parking system's terms of service. Secondly, this risk event could easily result in a penalty being issued to the user which would require additional resources from the user as well as the smart parking system's customer service department to resolve. Following from this, it is clear that any successful mobility detection and classification solution for the device must be highly sensitive to false positive classifications of movement.

Table 4. Risk events and consequences of movement and rest classification errors

Risk event	**Consequences**	**Severity**
False negative movement classification	Payment mode – Parking event will continue after the user has driven away unless it is stopped manually. This may result in over-payment by the user. Note: It is clearly stated in terms of service that it is user's responsibility to stop parking events when they are done parking. Disc mode – Display will not sleep and consume more power than necessary.	Low (2)

Risk event	Consequences	Severity
False positive movement classification	Payment mode – Parking event will end prematurely without user interaction. This may result in a parking fine to the user and subsequent reconciliation process that is expensive for the service provider and highly inconvenient for the customer. Disc mode – Device will reset its arrival time once it senses and classifies itself as at rest.	High (4)
False negative rest classification	Payment mode – None Disc mode – Display will not automatically activate. May result in a parking fine for the user. Note: It is clearly stated in terms of service that it is user's responsibility to check that the correct arrival time is displayed before leaving the vehicle.	Low (2)
False positive rest classification	Payment mode – None Disc mode – Display will automatically activate and consume more power than necessary.	Very low (1)

For the accelerometer and gyroscope sensors, a false positive error in movement classification may occur if the device experiences a shock or vibration when at rest. Depending on the sensitivity of the sensor, this error may be of high probability. The likelihood of this error may be reduced substantially by filtering out low-frequency changes in the sensed phenomena and requiring a sustained period of change in sensed phenomena for an event to be classified as movement.

A magnetometer's ability to easily detect stray magnetic fields like the Earth's can also make the sensor susceptible to changes in its local electro-magnetic field. This results in a high likelihood of false positive classification of moving while at rest, especially for sensors that are highly sensitive to noise and interference. A vehicle's body/engine can be one such source of noise, but this can be compensated for with a user-initiated calibration procedure. However, noise introduced by ambient sources like variable current power lines or nearby vehicles is highly unpredictable, and since the duration of such interference events cannot be assumed to be constrained, efforts to mitigate classification errors by filtering out low-frequency changes or requiring a sustained period of change are not compelling. [22]

Typical consumer-grade magnetometers used in positioning applications are based on the Hall effect and suffer from a higher noise floor than magnetic field sensors based on other technologies [23] [24]. However, within Hall-effect magnetometers as a group,

many designs are more resistive to interference than others. Thus, it is helpful to separate this group into low-noise and high-noise variants for purposes of analysis. It is important to note that this noise resistance does come at a cost in terms of price and power consumption. These costs are discussed further in later sections.

Assessment of the likelihood of classification errors using cellular network data relies on the work of Sohn et al. The authors' classification method using cellular network traces correctly detected periods of rest in 92.5% of ground truth events, with false positives in 4.6% of cases. Driving events were correctly identified for 81.7% of ground truth events, with false positives in 15.7% of cases. [20]

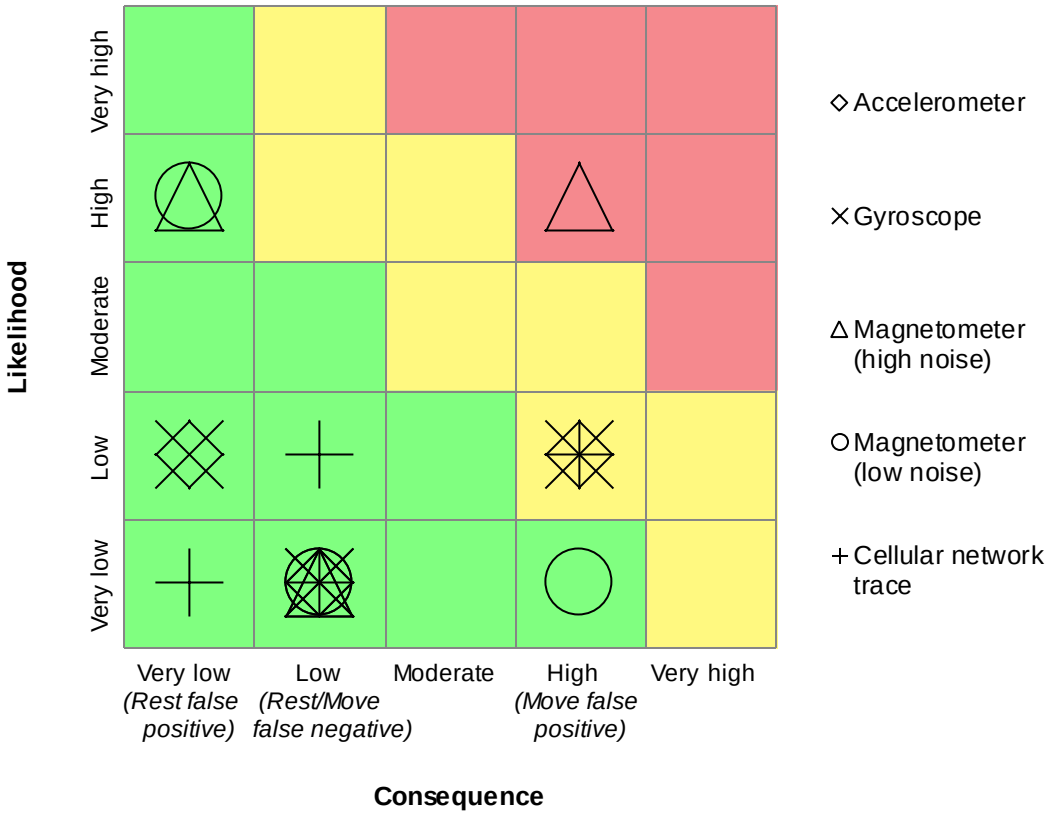

Figure 2. Risk matrix of selected sensor methods for detecting and classifying mobility

Figure 2 presents the likelihoods and consequences of each method for detecting and classifying mobility. It is clear from this perspective that the high consequence of false positive movement classification is the most important determining factor, and the risk of using a high-noise magnetometer is deemed to too large for further consideration. The risk of false positive movement classification associated is also of some concern for all other methods except the high-noise magnetometer, thus use of any of these

methods will require careful planning and prevention approaches to reduce the likelihood of this risk event from occurring.

2.2.3.2 Cost analysis

Many factors other than risk help determine the optimal method for fulfilling any requirement, and in this case costs related to monetary price and power consumption are especially important. To further analyse methods on the basis of these costs, data on average prices and average active current draws were taken from available samples at various electronics parts distributors available to the Finnish market.

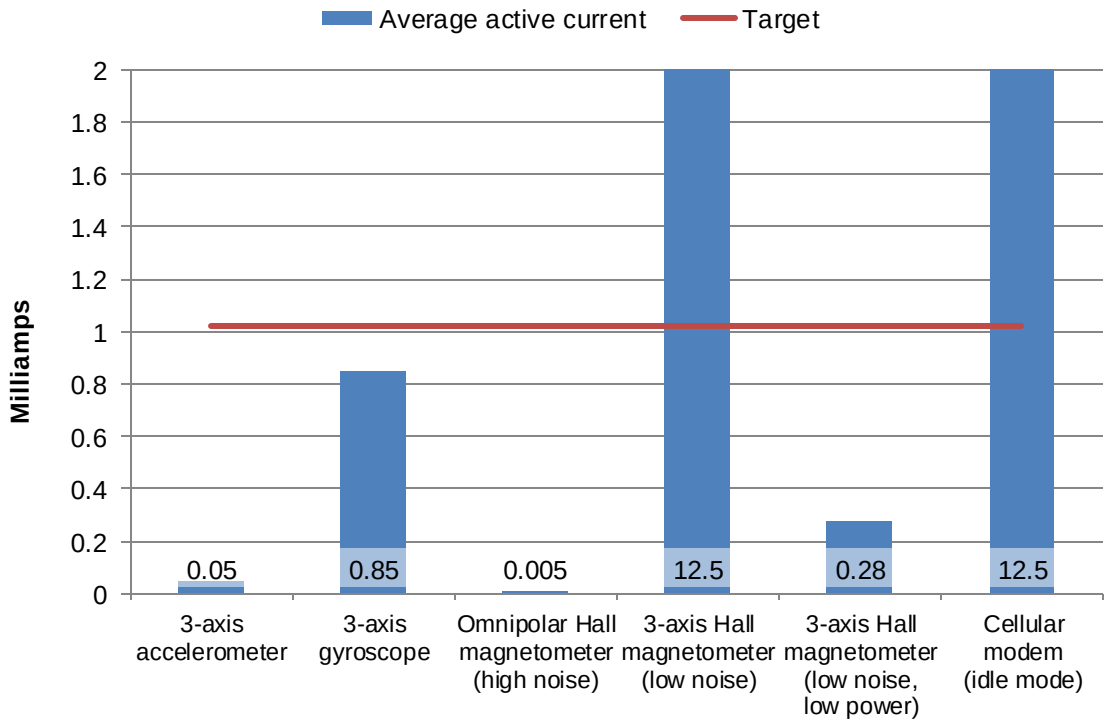

Figure 3. Average active current of selected methods for detecting and classifying mobility vs. target for smart parking device

Figure 3 presents a comparison of selected methods based on average current draw of the method while actively detecting movement or rest. A target average active current draw of 1.02 mA was determined based on requirements for battery life, battery capacity, and current consumption of other device components. As Figure 3 shows, using the device's cellular modem or a low-noise magnetometer not designed for low power consumption would require too much current to fulfil the device's requirement of performing

for at least a week on a full charge. Thus, these two methods cannot be considered further.

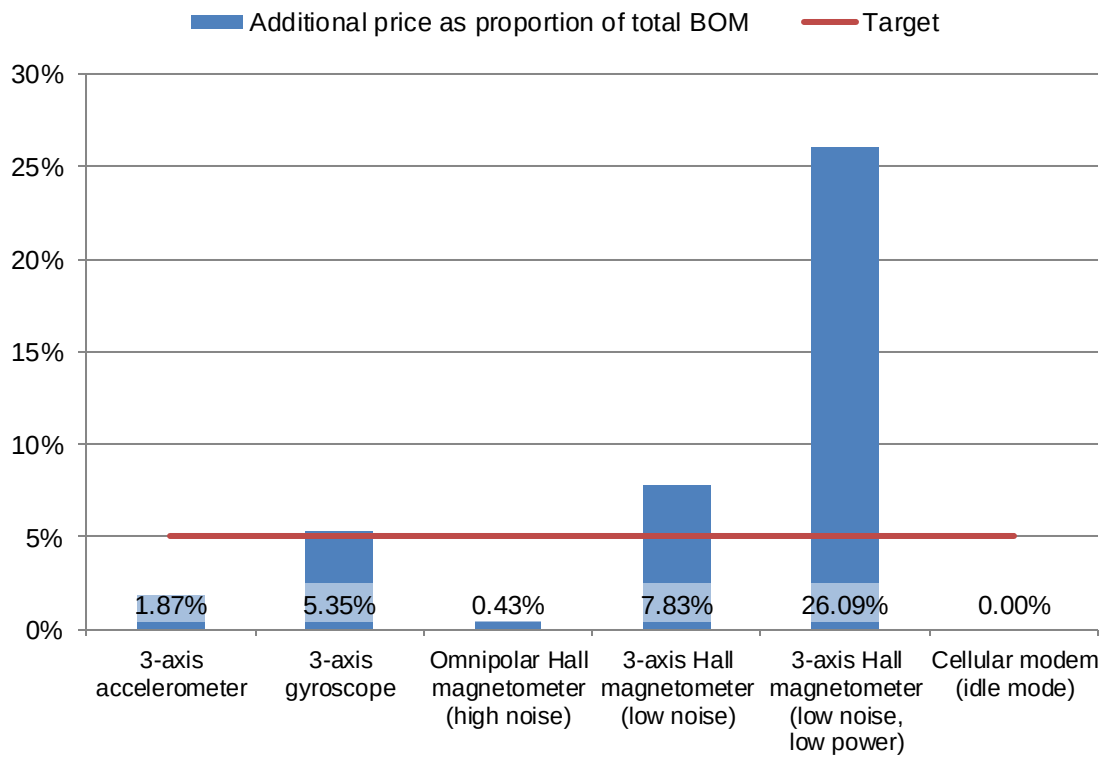

Figure 4. Additional price of selected methods for detecting and classifying mobility as a proportion of total BOM vs. target for smart parking device

Figure 4 presents a comparison of methods based on their average price as a proportion of the total bill of materials (BOM) of the device. A related target price was determined based on sourcing and costs of other device components and a separate target total BOM for the device. The cost of using a low-noise, low-power magnetometer solution would be far higher in terms of price than any other solution, as it would raise the BOM of the device by 20-25% more than all other methods. As such, the low-noise, low-power magnetometer is not considered further.

2.2.3.3 Outcome

After careful consideration of multiple methods for detection and binary classification of mobility, risk and cost analysis resulted in two suitable solutions: a 3-axis accelerometer and a 3-axis gyroscope. While the accelerometer and the gyroscope are similar in terms of risk assessment, the accelerometer clearly outperforms the gyroscope in

terms of cost in price and power consumption. Additionally, research in other application areas shows that a single accelerometer at least performs similarly to, and often outperforms, a single gyroscope when used to detect and classify mobility [25] [26] [27].

3 Design and implementation

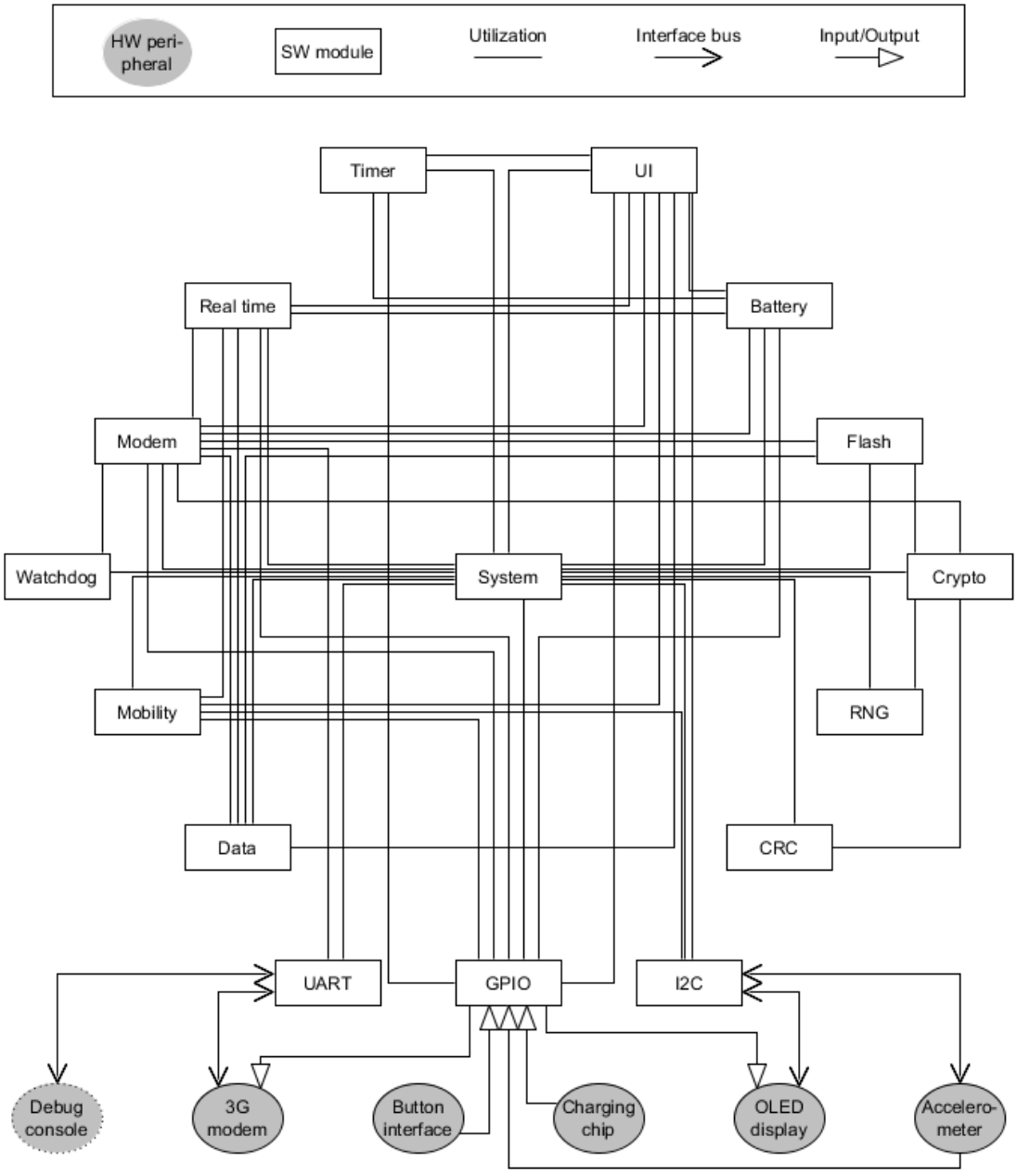

Figure 5. Block diagram of smart parking device software modules and hardware peripherals

Figure 5 shows the logical design and layout of the device's software modules, hardware peripherals, and interfaces. Modules represent both interface implementations and MCU peripheral units, and are the highest-level software constituents of the system. Within each module are lower-level abstractions termed here as components. A component may be an element as rudimentary as a hardware register or as sophisticated as the composite state of a module, and may have its own state machines and constituent components. Table 5 lists each software module and describes their basic functions and components. Note that the software implementations of some modules are complex enough to warrant two levels of abstraction.

Table 5. Description of smart parking device software modules

Module	**Description**
Battery	Analog-to-digital converter (ADC) unit – Controls measurement of battery voltage as a proxy for remaining battery charge
	Battery abstraction – Manages state of battery and charging chip
CRC	Cyclical redundancy check (CRC) unit – Supports cryptography
Crypto	Cryptography utility – Handles keys and primitives, wraps and calls cryptographic library functions
Data	Data utility – Handles user and configuration data storage as emulated electrically erasable programmable read-only memory (EEPROM) with reading, writing, and cycling of flash memory
Flash	Flash memory utility – Reads, writes, and erases program memory
I2C	I2C interface – Connects display and accelerometer peripherals to MCU
GPIO	General-purpose input/output (GPIO) unit – Controls external interrupts from all modules and external interfaces; Reads and debounces external input
	Input abstraction – Manages state of button input
Mobility	Accelerometer peripheral driver – Controls accelerometer functionality for mobility detection and classification
	Mobility abstraction – Manages state of mobility
Modem	3G modem peripheral driver – Controls low-level connectivity processes like power-up, configuration, building commands, and parsing and handling responses
	Network communications abstraction – Controls high-level connectivity processes like network registration, IP address procurement, and service endpoint communication; Manages state of network/internet connectivity

Module	Description
Real time	Real-time clock (RTC) unit – Controls independent timer with external 32.768 kHz oscillator for timekeeping and wakeup functions
	Real time abstraction – Performs time conversions and manages state of time synchronization
RNG	Random number generator (RNG) unit – Supports cryptography
System	System abstraction – Controls core system functionality such as system boot and initialization, clock configuration, stop mode entry and exit, critical section entry and exit, system reset, and the main loop; Manages state of system
Timer	Timer unit – Controls general purpose timers and their callbacks
UART	UART interface – Connects modem to MCU and delivers serial output to external port (serial output active only in development)
UI	OLED display peripheral driver – Controls low-level display processes like power-up, configuration, and glyph drawing
	User interface (UI) abstraction – Controls high-level display processes based on model-view-controller (MVC) design pattern; Manages state of UI
Watchdog	Independent watchdog timer unit – Controls independent watchdog timer to detect and handle system halt failures by resetting the system

The software design follows an event-driven model using finite state machines to represent modules and their constituent components. Events are generally interrupts generated by a module or user interaction whose properties affect the state of one or more components or modules. A resulting state change within one module may then cause a change in state of another module, and so on. Each software module at its highest level of abstraction is designed to operate within the same normalized set of finite states, although the states of their constituent components (e.g. modem network registration state, button press state, mobility classification state, scrolling display state, etc.) are not generally applicable to other components. Figure 6 shows the simplified nature of each module as a finite state machine.

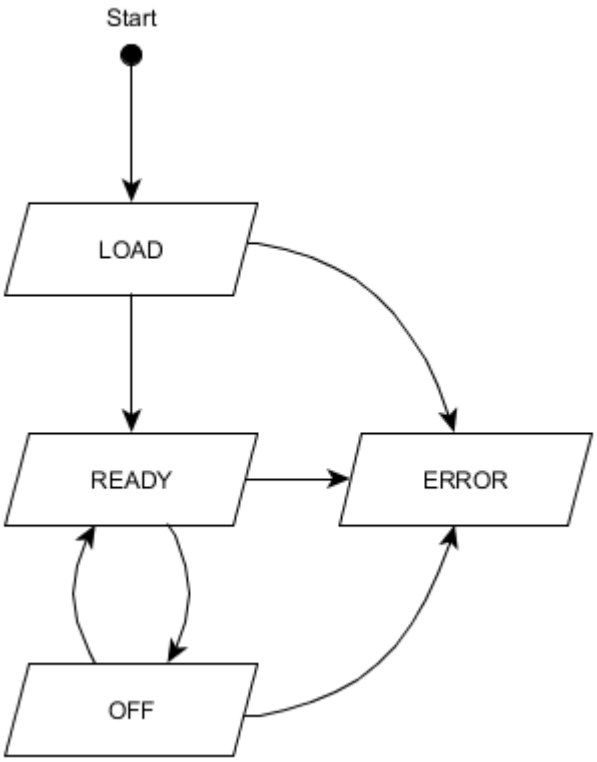

Figure 6. The normalized finite state machine model applied to all software modules

The first time a module is activated, it is initialized in the LOAD state, transitions to the READY (active) state, and from there it shifts between READY and OFF (inactive, minimal power consumption) states. The ERROR state is entered during the fault handling process if a failure to recover from a fault results in a failure of the module to provide service. Fault tolerance techniques of detection and recovery as they apply to dependability of the system are discussed below.

3.1 Secure and dependable service delivery

3.1.1 Dependability

To ensure dependability of the service, multiple strategies of fault tolerance are employed. As a complex system with many modules and components, there are numerous possible faults. Since exhaustive coverage of all possible faults in the system is impractical, efforts of detection and recovery of operational faults are focused on particular components and processes based on the likelihood and consequence of their faults and failures.

The system's approach to error detection is to make a best effort to check all inputs and outputs for errors in order to detect and handle them individually before propagation. All detected errors are logged in the device's user and configuration data space and uploaded to the cloud-based service regularly. Logs are also available on the device display via a special maintenance view in case they cannot be uploaded due to service failure. All boots are also logged with certain characteristics in case a system reset occurs before its contributing errors can be logged.

Specific detected errors are handled depending on their context. The following sections on internal and external fault tolerance review a selection of operational faults related to dependability and detection and recovery methods implemented in order to manage them.

3.1.1.1 Internal fault tolerance

If a module encounters a fault or error that cannot be handled using programmed methods such as module reinitialization, it will typically shift to its error state as part of the fault handling process. This is detected as a service failure of the component by the system module, which is handled by rollback and reinitialization of the system itself to its initial state via system reset. A system reset can be a "soft reset" where no changes are made to the system configuration, or a "factory reset" where the base firmware image is used in place of an updated image in case the updated image contains a persistent fault. Both reset types may be directly initiated by the user via a hardware button connected to the MCU's reset pin (a soft reset requires a single button press, while a factory reset requires a combination button press and hold). This externally-initiated recovery method is highly effective for correcting many kinds of faults and failures that the software does not detect or handle.

One particular internal fault worth mentioning is the system halt. Also called a "hang" or "freeze," a system halt is a complete service failure characterized by a constant (i.e. unresponsive) external system state. Importantly, a system halt may prevent the system from employing typical fault tolerance methods and requires special action for detection and recovery. In this case, error detection and recovery are performed by an independent software watchdog timer. When the timer fails to be refreshed by the system, it automatically handles the fault by performing a rollback to the system's initial

state in the form of a soft reset. Upon reset, the device log is updated to notify that the watchdog was activated.

3.1.1.2 External fault tolerance

Dependable service delivery relies on multiple external factors and therefore is susceptible to multiple external faults. Communications with the service are especially important as they contain many potential external faults that can cause service failure, including failure to connect to a mobile network operator (MNO), DNS failure, or time desynchronization.

Failure to connect to an MNO may come from a number of causes. Due to 100% 3G network coverage by MNOs in Finland [28], faults resulting from lack of geographical network coverage should be extremely rare. However, multi-level garages and underground parking lots often create their own "dead zones" due to interference, and even the most reliable MNOs may experience unscheduled downtime. Such external faults are automatically detected and handled by the modem by reconfiguring and reconnecting itself to the network, or by connecting to a new network when activated in roaming mode. Use of a roaming SIM subscription on the device also ensures that any supported network may be used as a redundant link in a cost-effective manner. When the modem is unable to register to a network or obtain an IP address for contacting the service, the error propagates to the device where error detection and recovery are activated. In such a case the error is assumed to be transient (i.e. service may resume at any time) and there is typically no special mitigation process, so the error is reported to the user and the modem module is rolled back to its initial ready state or rolled forward to its off state.

Since the smart parking service uses dynamically allocated IP addresses for its load balancing servers, DNS is required for the device to connect to the service. Therefore, an external fault in one or more elements of the DNS infrastructure could result in service failure. Such a fault would most likely be limited to a single service provider at once, thus a simple redundancy of DNS servers should suffice for recovery. Initial error detection is straightforward as the modem responds with a specific error code when DNS fails. The system attempts compensation by iterating through a hardcoded list of public DNS servers, configuring each one for use and requesting a domain name trans-

lation. If the error cannot be compensated, the system recovers in the same manner as above by reporting the error and rolling the modem module back or forward.

Service failure may also be caused by time desynchronization due to the inclusion of a timestamp-derived nonce in HMAC message authentication between the device and service. A timestamp-derived nonce was chosen since (epoch) timestamps both ensure against replay attacks and can never be repeated, and because the device has no other reliable way to provide or enforce uniqueness of a nonce. However, this feature of authenticity could affect availability of the service: if the device is unable to keep its RTC synchronized to the service's time within the required deviation period, the service will reject its messages and vice versa.

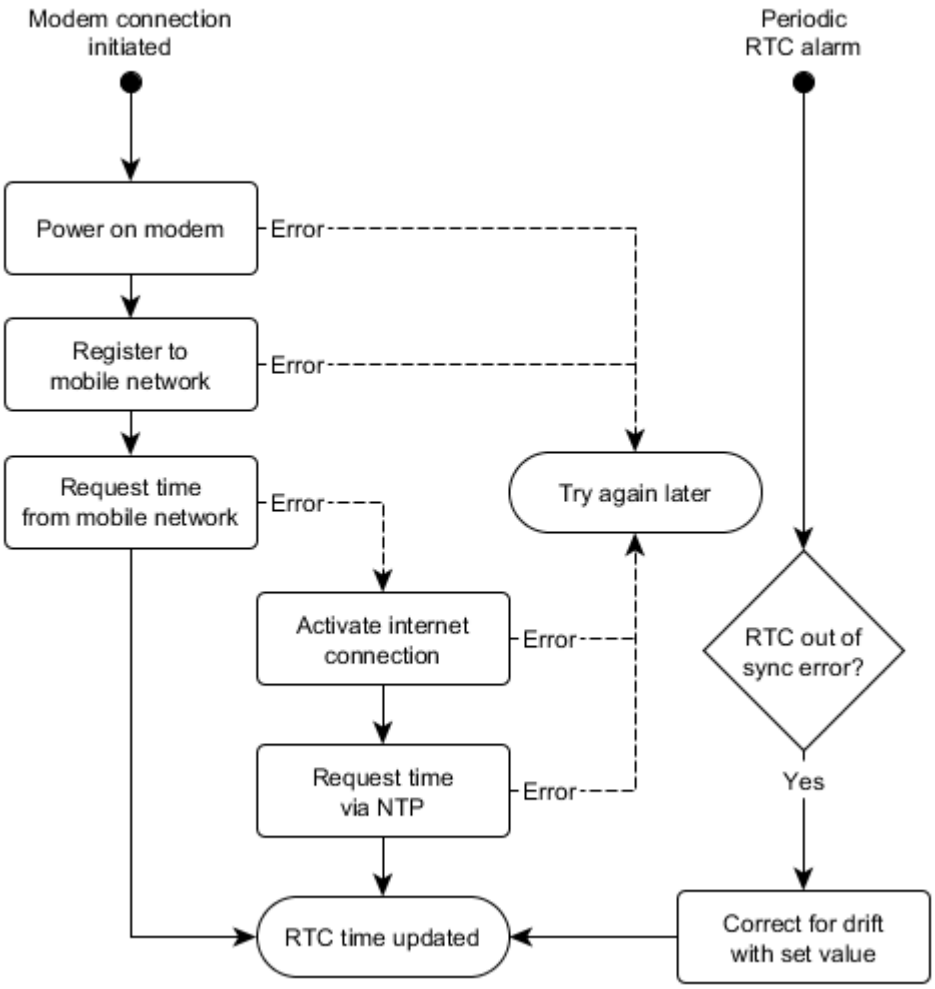

Figure 7. RTC drift error detection and compensation

Time desynchronization is a special case as its root cause is an internal fault of the device's RTC. As the RTC's oscillator drifts due to environmental factors, the device's time becomes out of sync with the service's time. RTC drift on the device was found to be fairly high, measuring an average of 333 PPM at room temperature, or about 1 second every 50 minutes.

RTC drift is handled systematically by first attempting compensation using two diversely redundant resources: the mobile network and the network time protocol (NTP). Figure 7 illustrates the procedure. First, the device requests the current network time from its MNO upon registration. While many MNOs support this feature, the service is not mandatory or guaranteed. If network registration succeeds but network time synchronization fails, the error is handled by attempting compensation again using the NTP service. This requires not only registration on a mobile network, but also internet connectivity and an IP address assignment. If this service fails at any stage, the final compensation method is to simply correct the RTC's time for drift with the tested offset of 1 second for every 50 minutes since the RTC was last synchronized. This recovery method is automatically employed if the RTC has not been synchronized for a certain period of time, which is checked for regularly during a generic alarm handling process.

3.1.1.3 Fault removal and remote firmware updates

The maintainability of the system is derived from its ability to remotely install firmware updates to enhance the service and remove faults. However, this ability brings with it many functions that include potential faults of high consequence, such as erasing application program memory, and setting stack memory and interrupt vectors in operation. Therefore, the design and implementation of the firmware update process must be highly dependable in order to ensure against faults and service failures.

As shown in Figure 8, the device memory is separated into two sections: one that is always write protected, and one that is re-writable. To ensure system resilience and minimize the potential of unrecoverable service failure, a bootloader as well as all functionality required to update the firmware should always be available and write-protected on the device. In this particular case, such functionality includes the ability to register to an MNO, achieve internet connectivity, connect to cloud-based services, and download updates over the air. The resulting amount of program memory needed for this func-

tionality was seen as large enough to warrant preserving a fully-functional application image for this purpose.

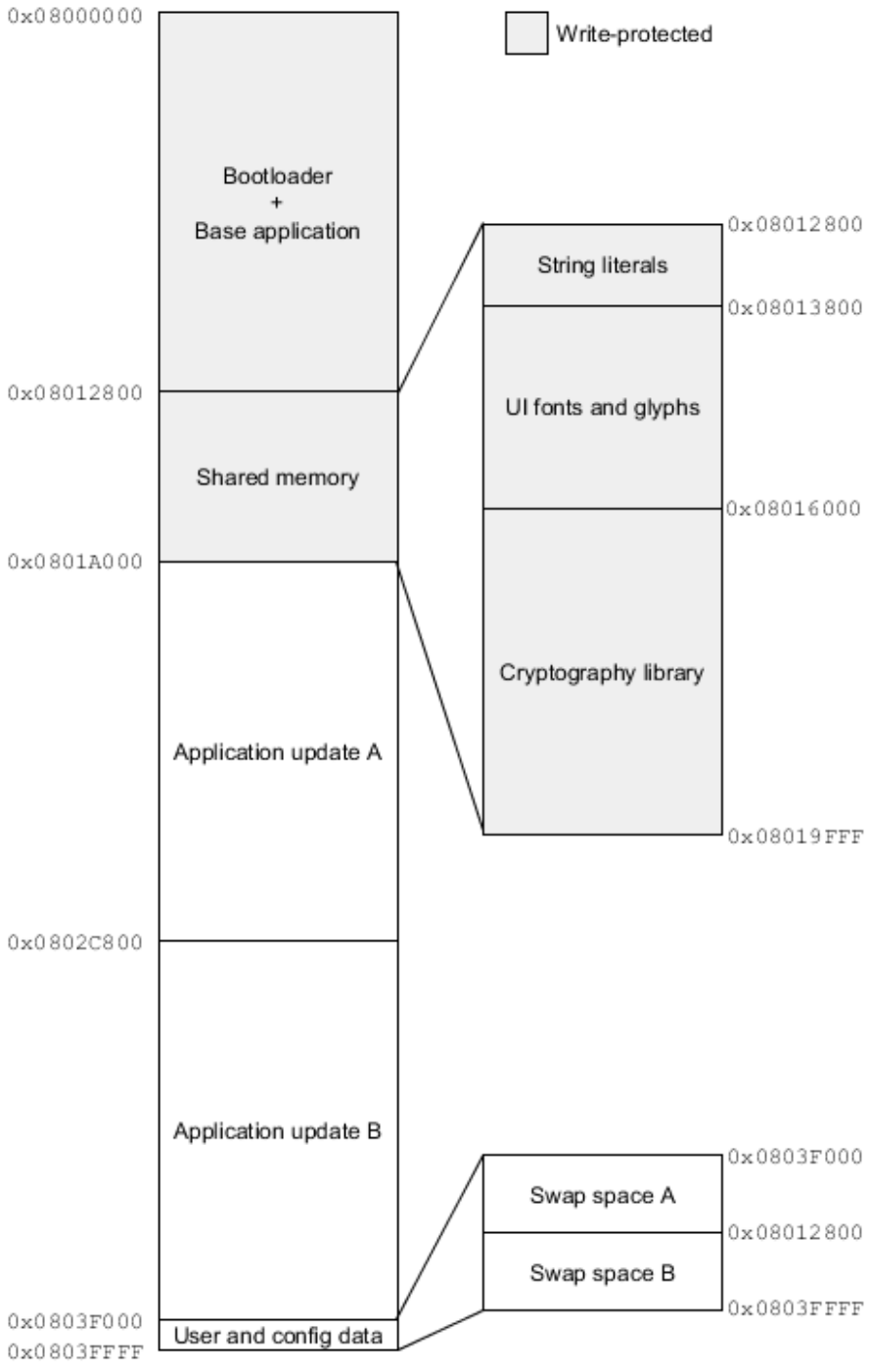

Figure 8. Device memory map

Thus, the first partition in memory includes not only bootloader functionality, but also a fully-functional "base" application image. Although the bootloader and application are

logically separated, the combined memory space is treated as a single block that is always write-protected and never erased. Having two additional application partitions for updates enables the device to upgrade the firmware from a non-base application version as well as rollback to a non-base application version (if available). This allows for a smooth transition between upgrades and minimizes the likelihood of rollbacks to the base image, which may be several versions behind the most recent upgrade image.

An obvious drawback to having three partitions for application data is that their size is more limited compared to using a typical dual-partition implementation. To conserve available space, a dedicated read-only shared partition is used to store constant program memory accessible to all applications. This conserves 30 kB of program memory, which accounts for a reduction of each application partition size by about 29%. The shared partition mostly contains string literals for the UI in 3 languages (Finnish, Swedish, and English), UI font and animation glyphs, and shared library functions which can be used for future updates with no changes.

The highest-address partition in the device memory holds user and configuration data as emulated EEPROM. Since the smallest erasable unit of flash memory on the MCU is a 2048 kB page, the partition is split into two single-page spaces which are swapped in order to safely copy memory before overwriting, as opposed to copying to RAM which would be lost in the event of a power failure.

The update procedure also employs its own fault tolerance techniques in order to ensure dependability. When the firmware update image is downloaded from the service and staged on the modem, its size is checked for sanity and the inactive application memory block is erased. The image is then transferred to the MCU in chunks and each one is written to the inactive application memory block and verified. Once the entire image has been written to the device, its checksum is calculated and compared with an independently generated checksum sent by the service. Once the written image's checksum is validated, the system's configuration data is updated to set the new images partition as active and the system is reset.

Upon reset, the bootloader partially initializes the system and performs a number of checks. First, a generic boot counter which persists across resets and power loss is incremented to record the boot. Once the system has completed all checks and before

jumping to an application image, the generic boot counter is reset. If this counter passes a certain limit it is diagnosed as a persistent fault, a factory reset is performed, and the system and its configuration data are rolled back to using the base application image and configuration.

Assuming the generic boot counter check is passed, the bootloader next logs the cause of the boot, enables the user interface buttons, and checks if an interface button is actively pressed. As mentioned previously, if the user presses the dedicated reset button in combination with an interface button for a duration, the system will perform a factory reset. If no interface button press is registered or the button is not held down long enough, the bootloader continues by retrieving the active image configuration.

At this point, a second boot counter is incremented to record the boot, this time specifying the image being booted. Once the bootloader has jumped to the active application image and the system has completed all preliminary checks, this counter is reset by the application. If this counter passes a certain limit, it is diagnosed within the bootloader as a persistent fault specific to the active image, and instead of performing a factory reset, the system will void the image and rollback to the previous active image.

Finally, any modules initialized during the boot are de-initialized, the stack pointer and interrupt vector table are set to the active partition's memory space, and the bootloader jumps to the new application.

3.1.2 Security

In the *Background* section, multiple techniques for attaining confidentiality, integrity, and authenticity were introduced, including the ubiquitous TLS and X.509 public key infrastructure internet standards. This section identifies important limitations on the device—namely that it cannot fully support PKI or a secure version of TLS—and follows the ad hoc design process of a supplementary embedded cryptosystem used to ensure security of communications between the device and the cloud-based service.

3.1.2.1 Limitations

As discussed in the *Background* section, HTTPS using TLS 1.1 and above can provide confidentiality and authentication of communications between a client and server when properly used. This solution is also conducive to availability as an information security goal since it is widely used and supported, and unlikely to introduce complexities to the device software's implementation that might lead to a loss of service.

While HTTPS would clearly be an optimal solution, unfortunately the modem chosen for the device—the SIMCom SIM808C—does not allow for its proper use. The modem officially supports TLS up to version 1.0, which, as discussed previously, suffers inherent vulnerabilities to multiple attack vectors and as such has been abandoned. Furthermore, testing during development found that there is no working method on the SIM808C to deny self-signed end-entity server certificates issued for the cloud-based service's servers, although the manufacturer's documentation implies that such an option is available. This essentially avoids the authenticity and trust provided by PKI and leaves HTTPS on the device vulnerable to MITM attacks from anyone with a self-signed certificate claiming to be the cloud-based service. Thus, HTTPS on the device cannot be relied upon for confidentiality or authentication. [29]

As a result of the aforementioned limitations, additional cryptographic functionality was required on the device to ensure confidentiality of messages containing sensitive information and to guarantee authenticity of all messages between the device and the cloud-based service to prevent fraud.

3.1.2.2 Cryptography implementation

Multiple third-party libraries were considered to provide a trusted cryptographic implementation for the device. Although full TLS protocol support was a valued goal, known available options such as mbedTLS (formerly PolarSSL), wolfSSL, and axTLS required more program memory than was available. The STM32 cryptography library from STMicroelectronics was chosen due to its memory footprint, availability, specified design and support for the device's MCU, and certification status of many of its algorithms with the US Cryptographic Algorithm Validation Program. The compiled library is optimized for a minimal program memory footprint, and eventually required some 16 kB of

space. All other evaluated options required at least 30 kB of program memory even if they were built with minimal functional support. [30] [31] [32] [33] [34]

3.1.2.3 Ad hoc design of an embedded cryptosystem

While multiple cryptographic methods may be used for authentication, all require a secret or private key. Furthermore, this key must be unique to each device so that if one key is compromised, the others may continue to be used. Also, the device must have the capability to generate and exchange new keys with the cloud-based service in order to maintain availability if its original is compromised or lost by the service.

A typical hybrid scheme of asymmetric cryptography for symmetric key exchange and symmetric cryptography for authentication and encryption of later messages was chosen to reduce bandwidth and power consumption. First, asymmetric cryptography methods were examined. Although the device cryptographic library supports both Rivest-Shamir-Adleman (RSA) asymmetric encryption and elliptic curve (EC) cryptography DH (ECDH) key exchange, ECDH was not considered due to the engineering team's lack of knowledge regarding EC cryptography in general and specific uncertainty regarding if and how the service's web applications might support ECDH. The consequences of this omission are discussed further in the *Results and analysis* section.

RSA (2048-bit key size, PKCS #1 v1.5) was thus chosen as the asymmetric cryptography method supported on the device. The device cryptography library supports RSA public-key encryption, private-key decryption, and digital signature generation and validation, but does not support public-private key pair generation. Besides encrypting the device key for key exchange, the service's public key can also be used by the device to verify the authenticity of messages originating from the service that are digitally signed with the service's private key. This is important if, for example, the device's symmetric key is lost by the service, as it allows a request to the device for a new device key to still be authenticated. The device's on-board random number generator (RNG) peripheral is used to create random number sequences for padding in RSA encryption and for the creation of random symmetric keys.

To summarize thus far, the device must support:

1. Generation of a private symmetric key unique to the device

2. A method of ensuring authenticity for messages sent from the device and the cloud-based service using a device-bound symmetric key
3. Asymmetric encryption of messages containing sensitive information using the service's public key
4. Digital signature verification of certain messages from the service using the service's public key
5. Re-generation and distribution of a new device key upon an authenticated request from the service

A related requirement not included above is that the device must be able to send its initial symmetric key to the cloud-based service in an authenticated and confidential manner to prevent exposure. While public-key encryption ensures confidentiality for the initial key exchange, it does not ensure authenticity. Instead, this requirement is satisfied by a preliminary key distribution procedure managed during the manufacturing and testing stage of the device production process. Furthermore, confidentiality of messages from the service to the device is not presently a baseline requirement, since no sensitive information needs to be sent in this direction. However, an optimal solution satisfying the above requirements would also include this functionality in order to support expansion of the device and service's functionality.

Thus far, solutions for all requirements for security of service delivery have been discussed except the authentication method for messages to and from the device using the device key. Both MAC and AE solutions were considered, with AE initially preferred due to its support for encryption of messages originating from both the device and the cloud-based service. Specific methods of MAC and AE were evaluated based on their program memory footprint [35] compared to a target value representing the maximum amount of memory available for the algorithm and its support functions. The results of this analysis are shown in Figure 9.

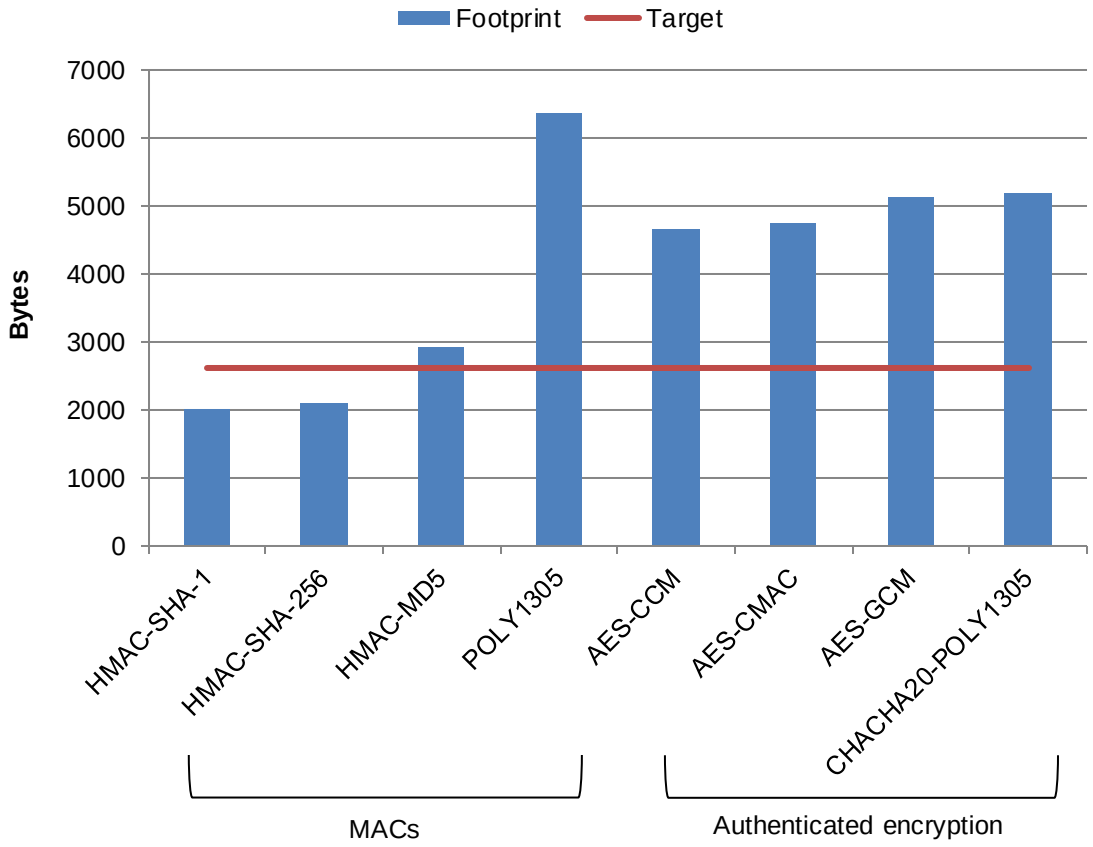

Figure 9. Program memory footprints of selected cryptographic methods vs. target for authentication of messages from smart parking device and service

As Figure 9 shows, HMAC is the only viable authentication method when taking program memory footprint into account. Two suitably-sized HMAC hash variants are available using SHA-1 and SHA-256. Although HMAC does not require a collision-resistant hash function to operate effectively [36], SHA-256 is still recommended for new designs and is the best choice here given that its memory footprint differs from SHA-1 by only 100 bytes.

While the chosen methods for ensuring confidentiality and authenticity fulfil the device's baseline requirements, there are some important limitations to the final design. First, while asymmetric encryption is supported for messages originating from the device, the output size of these messages is fixed (in this case to 256 bytes) regardless of input size. Due to this overhead, messages *not* containing sensitive information do not undergo additional encryption. As a compromise, HTTPS on the modem is still used "as is" for all messages despite its known vulnerabilities, as it is a simple and low-cost way to avoid sending any messages in the clear over HTTP, including those not containing sensitive information. Furthermore, since the chosen authentication method does not

offer encryption, and since the available amount of program memory prevents inclusion of an additional encryption method, the secrecy of messages originating from the cloud-based service cannot be guaranteed in the event of a MITM attack. While confidentiality of these messages is presently deemed unnecessary, this limitation will affect what features the device can support in the future and how they are implemented.

3.2 Mobility detection and classification

In the *Background* section, the challenges of mobility detection and classification for a smart parking device were discussed, multiple solutions were evaluated with respect to risk and cost, and the conclusion was drawn that an accelerometer would be the optimal solution. This section provides an overview of the accelerometer chosen and some details on how it is used by the application to detect and classify mobility of the device.

3.2.1 Accelerometer and device mobility states

The LIS2DH12 3-axis MEMS accelerometer was chosen due to factors such as availability, price, average active current consumption, and the ability to send an interrupt signal on a single line when either movement or absence of movement is detected. The LIS2DH12 is used in low-power mode with a full scale of ±2 g and a sampling frequency of 10 Hz, resulting in a nominal average active current draw of 3 µA. To filter out the gravitational component of acceleration data, the accelerometer's built-in high-pass filter is activated with a 0.2 Hz cutoff frequency for both movement and rest detection. [37]

Since the device was provisioned a single interrupt line between the accelerometer and MCU, the device can only detect a single classification state of movement or rest at a time. The application was designed to shift the accelerometer's detection mode between three distinct states: off, movement detection, and rest detection. When the accelerometer is in the movement detection state, the device's mobility state is at rest. Likewise, when the accelerometer is in the rest detection state, the device's mobility state is moving. When the accelerometer and its detection mode are in the off state, the device's mobility state is unknown and irrelevant.

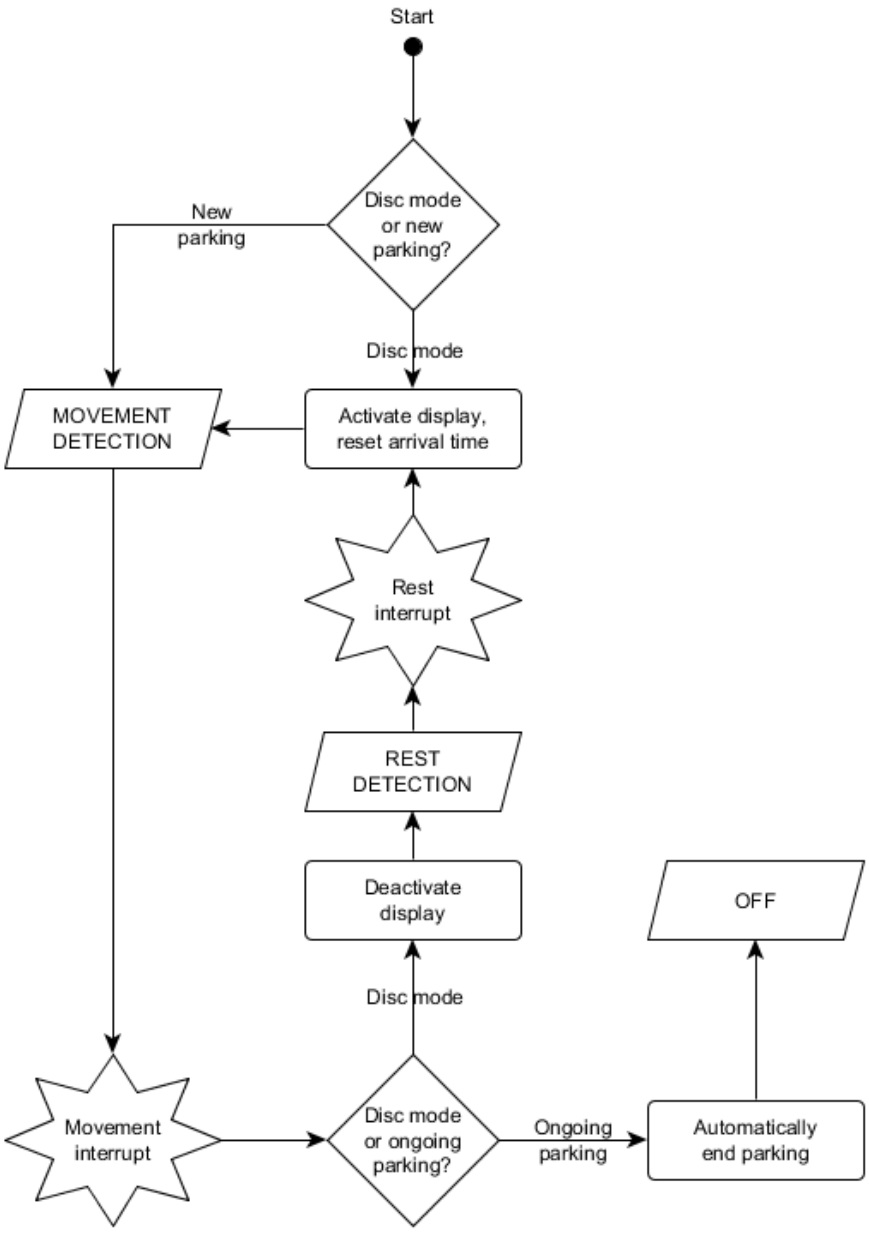

Figure 10. Simplified flow chart of motion detection and classification within smart parking device application context

Figure 10 shows the basic flow of accelerometer motion detection and classification within the context of the application. The accelerometer is first activated in the movement detection state when the application's disc mode is activated or a new parking event has started in payment mode. If disc mode is activated, the accelerometer will cycle between the two active detection states until another mode is entered. If movement detection is used to automatically stop a parking event, the accelerometer and its detection mode will enter the off state after the movement interrupt signal is received.

3.2.2 Initial settings for mobility classification

Initial threshold and duration settings for classification of mobility states were established based on research and field testing. These settings are not dynamically configurable at runtime, but can easily be changed within the source code and pushed to the device in a firmware update.

In the movement detection state, the lower threshold for movement was set at 32 milligravities (mg) or 0.314 m/s^2 sustained over any axis samples across a duration of 3 seconds. In other words, if acceleration on the X, Y, or Z axis is above the threshold continuously for 3 seconds, the accelerometer will send an interrupt signal to the MCU. These values are derived from research on typical acceleration rates of vehicles [38] and field testing of ambient noise when a car is parked, and are set to minimize the likelihood of false positive movement classifications due to shock or noise.

In the rest detection state, the upper threshold for rest detection was set at 64 mg or 0.628 m/s^2 sustained over all axis samples across a duration of 20 seconds. In other words, if acceleration on the X, Y, *and* Z axis are continuously below this threshold for 20 seconds, the accelerometer will send an interrupt signal. These values are derived from field testing of ambient noise when a car is driving, and are set to minimize the likelihood of false negative rest classifications.

4 Results and analysis

Over 2,000 smart parking device units were manufactured, programmed, and tested in 2017 to fulfil an initial order with additional manufacturing runs to follow according to demand. The smart parking system is, as of the time of writing, one of only three available service providers officially supported by Helsinki city for remote parking payments [39], and the only service provider that does not require a smartphone.

Formal testing on key attributes related to the security and dependability of service delivery as well as motion detection and classification was unfortunately not possible as part of this project, as key employees working on the project (including the author) left the company between device manufacturing and distribution phases. However, informal testing related to these topics was done during the development and manufacturing phases. The following section includes the results of these tests, as well as a

critical analysis of the consequences of decisions made during the design and implementation process.

4.1 Secure and dependable service delivery

4.1.1 Dependability

Informal testing with a non-representative sample size uncovered no complete service failures over a period of 6 months, not including early failures detected in the device manufacturing and testing phase. This weakly suggests that the device may exhibit an MTBF of at least 6 months. As there was no downtime experienced, actual availability is even more difficult to estimate. Regardless of the eventual measured outcomes, whether they will be considered satisfactory is impossible to predict as no measureable requirements for availability and reliability were given for the project.

One particularly consequential early failure was experienced during the device manufacturing and testing phase in approximately 1 out of 10 devices. A hardware fault was identified that could potentially prevent the device from connecting to the cloud-based service, effectively blocking delivery of core functionality including remote firmware updates. The fault activated seemingly randomly even after days of proper functionality, and although the system detected the error and attempted fault handling procedures including modem rollback and soft reset as well as system factory reset, these methods did not lead to recovery. As the original hardware fault could not be removed, it fell to the system software to find and implement an effective recovery method that could be deployed as a firmware update.

Fortunately, a bug was found in the system software that, when removed, resulted in full system recovery from the hardware fault. (While the original recovery method was designed to perform a factory reset of the modem if a soft reset failed, it did not perform its task as intended). This fix was included as part of a new firmware version, which was used as the base application image for those devices that had not already been programmed and assembled. However, some hundreds of devices had already left the factory with the dormant fault and no effective method of system recovery. Although the fault could be mitigated with an immediate remote firmware update to those devices, it could always be reactivated with no recovery method if the firmware was rolled back to

the base application image via factory reset. The decision was made to reprogram these devices with the new firmware update as their base application image.

Unfortunately, the affected devices had already been assembled in their casings, which are very difficult to remove without damaging the OLED display elements. Thus, a special update was remotely deployed to each affected device that, when activated, would overwrite its base image and bootloader with the next update. A second update would then be made using the new firmware update supporting recovery from the aforementioned hardware failure. While this method ran the risk of rendering devices unusable should an error or power loss occur during specific times in the update procedure, this consequence was of low severity since the devices had not yet been distributed to end-users and could be manually reprogrammed as a last resort.

Ultimately, the two-part firmware update worked as intended with no failures and all devices shipped with the new recovery method available within their base application. This episode illustrates the importance of fault tolerance as well as fault removal in a dependable system for ensuring availability and reliability of service. Without support for firmware updates, many devices would have been susceptible to complete service failure and thus unsuitable for distribution to end-users.

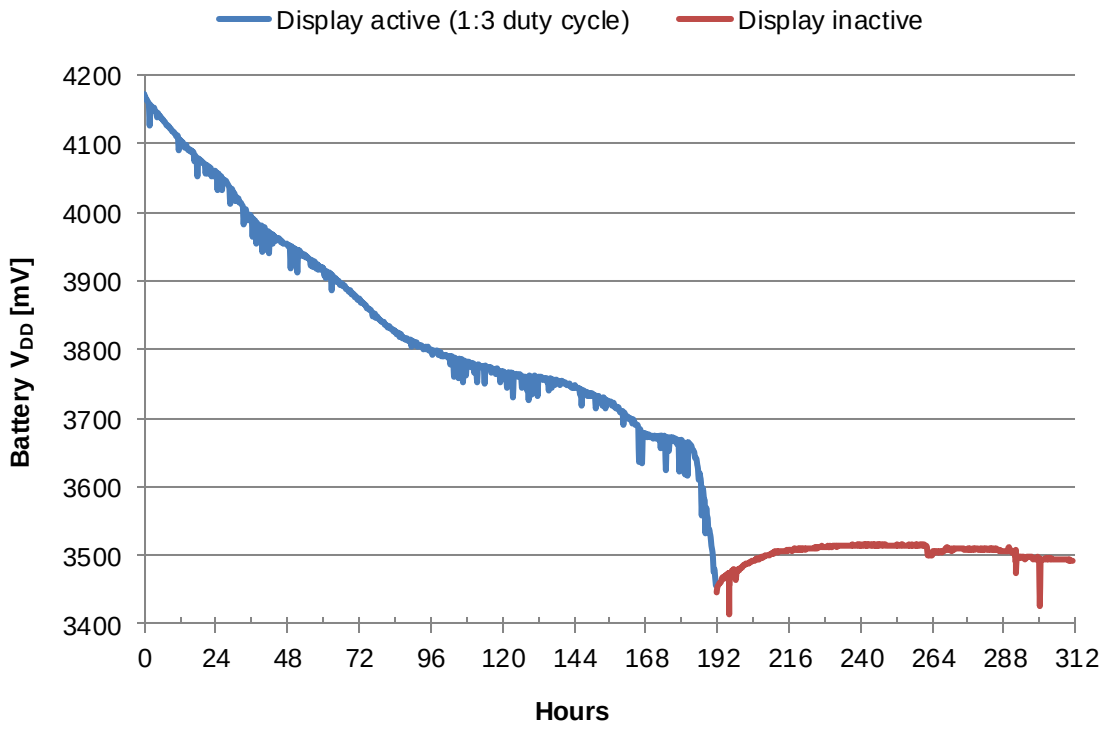

Figure 11. Device battery voltage over time in disc mode (0 hours = full charge)

Finally, informal testing was performed to validate that the device's operational lifetime on a single full battery charge would be at least 1 week when used as a parking disc, or 1 month or longer when not used as a parking disc. As Figure 11 shows, the operational lifetime of the device when used as a parking disc with a 1:3 active display duty cycle (i.e. the display blinks on for 1 second and off for 2 seconds) is around 8 full days. After battery voltage is reduced to 3450 mV, the device enters a low-power state where the display is completely inactive in order to conserve charge. Since the device's step-up converter requires 3400 mV in order to power the modem, this change to a low-power state maintains availability of the cloud-based service for some days. Additional informal testing of the device when not used as a parking disc showed a minimum operational lifetime of over 1 month on a single full battery charge.

4.1.2 Security

As discussed previously, a major challenge to ensuring security of communications between the device and cloud-based services was introduced due to a failure of the modem chosen to fully support a secure version of TLS and to reject self-signed X.509 certificates. Certainly, the first lesson illustrated here is that any system component should be properly tested and validated for key functionalities before being chosen and procured in quantity. Unfortunately, during the ad hoc design of a supplementary embedded cryptosystem intended to mitigate this failure, another important oversight was made: the omission of ECDH-based algorithms from consideration among methods of symmetric key exchange.

Figure 12 revisits a previous comparison of the program memory footprints of various MAC and AE methods, this time including target values based on use of Curve25519 and ED25519 for symmetric key exchange. It should be noted that ED25519 is actually a digital signature and verification scheme, and thus its program memory footprint includes both ECDH and digital signature and verification functionality.

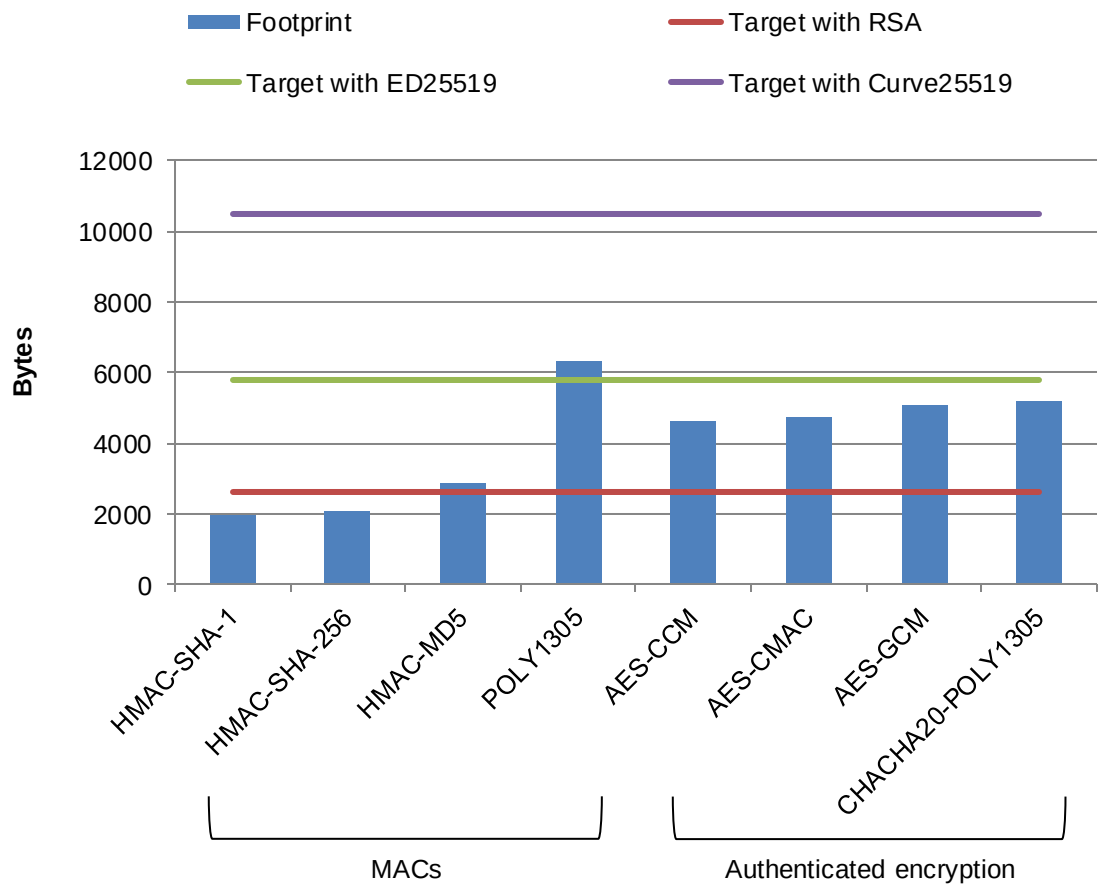

Figure 12. Program memory footprints of selected cryptographic methods for message authentication vs. targets based on choice of asymmetric cryptography

Both implementations provide key exchange functionality while requiring far less program memory than RSA, thus affording ample room for AE methods as well as MACs. Clearly, had these methods been properly understood and included for review during the design phase, authenticated encryption and secure key exchange could have been provided with less program memory than the chosen design.

Nevertheless, in light of the preliminary results of informal dependability testing mentioned above, it can be inferred that the additional security measures undertaken at least did not reduce availability and reliability of service delivery. Regardless of its shortcomings, it is reminded here that the ad hoc design for ensuring confidentiality, integrity, and authenticity of communications between the device and the cloud-based service is implemented in addition to HTTPS with TLS 1.0, and is only provided in order to prevent malicious MITM attacks on the service by actors with the ability to spoof the cloud-based service or a device.

4.2 Mobility detection and classification

Results from informal field testing of mobility detection and classification with a non-representative sample size were generally encouraging. Most notably, no false positive movement classifications were reported. This is important as a false positive movement classification was the only potential risk event with a consequence of high severity. While some false negative movement classifications did occur, changes in the configuration of movement detection and classification in order to prevent these events come with the risk of increasing the probability of false positive movement classifications, and thus are unlikely to be considered for future development unless new circumstances arise. Additionally, some false negatives and false positives in rest classifications were reported. These failings are not considered critical, but may be more amenable to improvement with further testing and configuration changes.

An oversight similar to the one described above was also made during comparison of alternative methods for mobility detection whereby a suitable candidate was omitted from review. Although magnetometers based on the Hall effect were included, magnetoresistive magnetometers were not studied as they were unknown to the engineering team at the time. In recent years, this low-noise, low-power technology has become more prevalent in mobility detection applications driven in part by reduced component costs. Revised risk and cost analyses illustrated in Figures 13, 14, and 15 show that a magnetoresistive magnetometer may be a better choice than an accelerometer for binary mobility classification applications in general, and particularly those that are sensitive to false positives for movement detection.

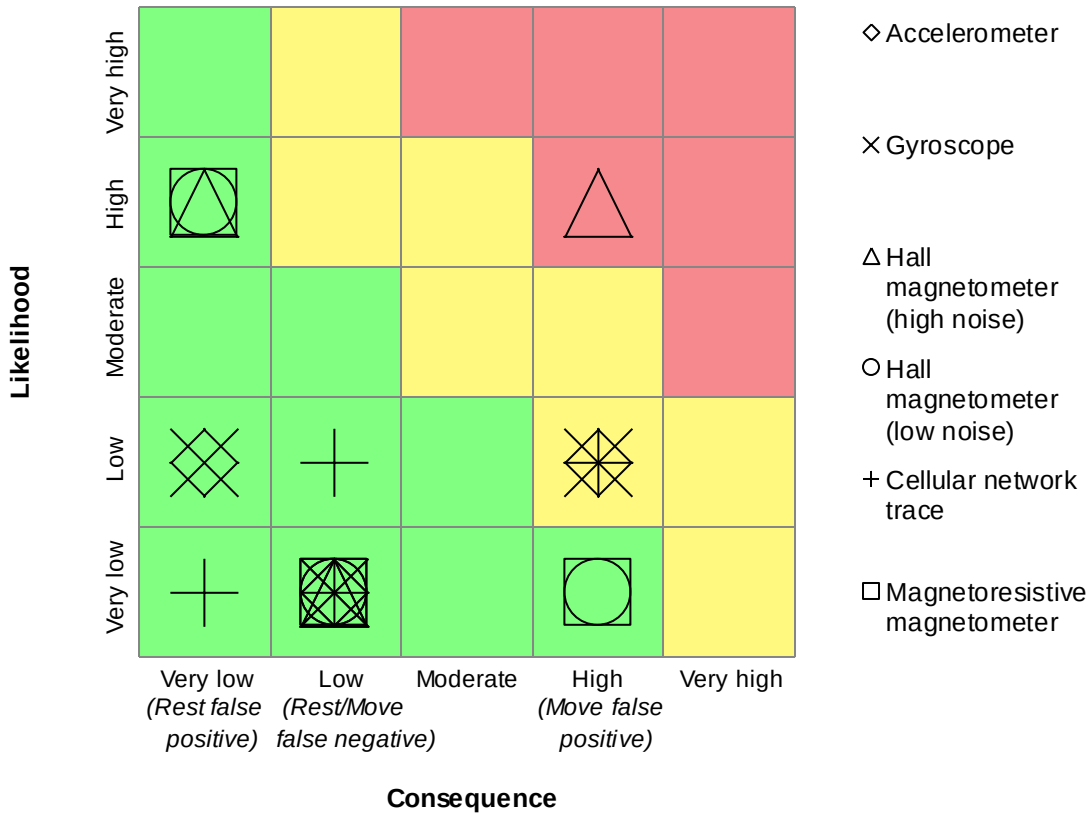

Figure 13. Revised risk matrix including magnetoresistive magnetometer

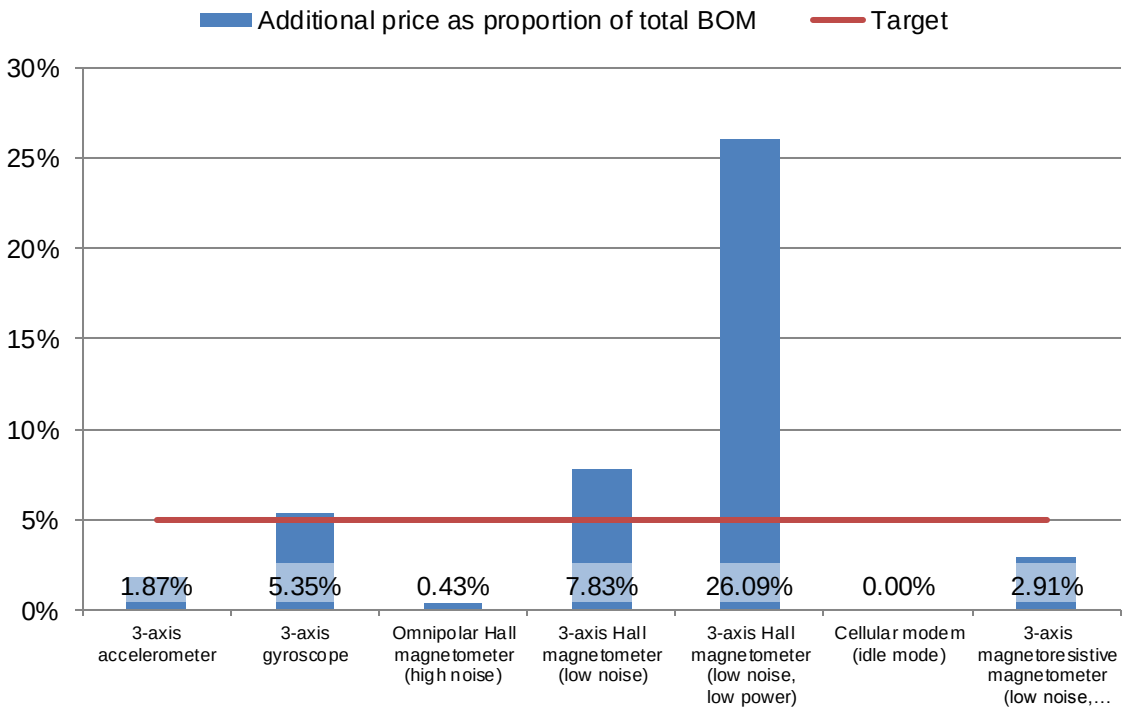

Figure 14. Revised price comparison including magnetoresistive magnetometer

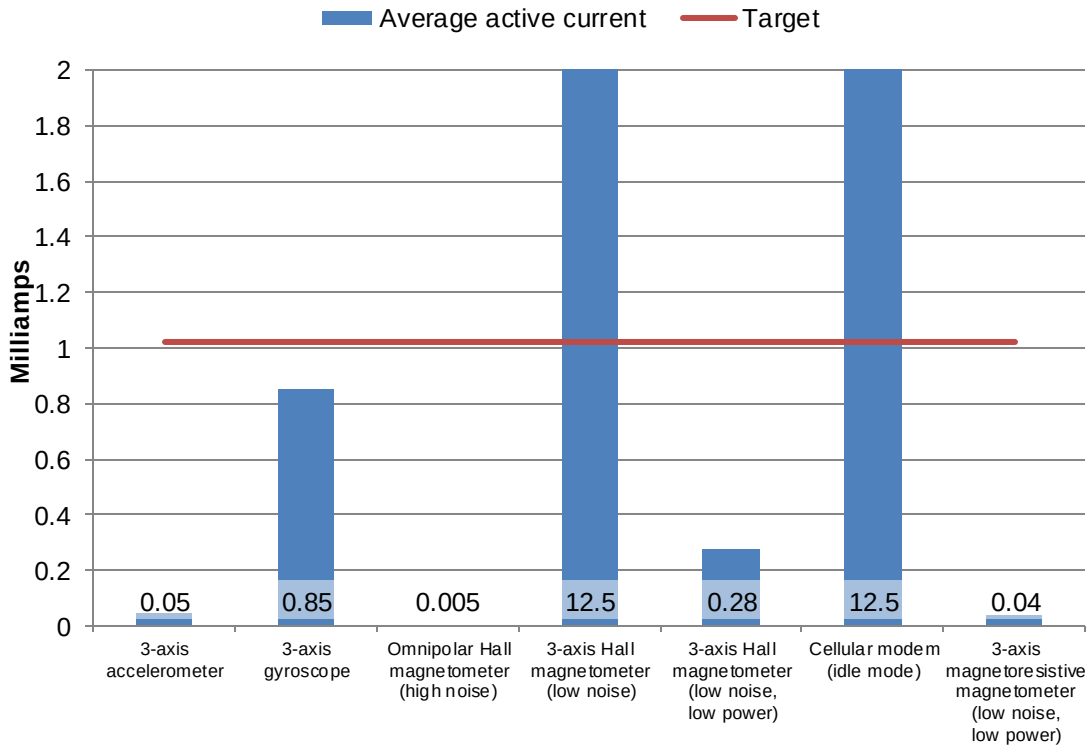

Figure 15. Revised active current comparison including magnetoresistive magnetometer

Regardless of the potential benefits, changing the component and technology used for mobility detection and classification on the smart parking device is infeasible at this stage due to the additional resources required to prototype, test, and implement this design change in hardware and software. However, future projects may benefit from this review if the magnetoresistive magnetometer indeed proves to be a reliable technology for mobility detection and classification.

5 Conclusions

As the above discussions show, the requirements and constraints of a complex embedded system can create very different challenges. Creating a secure channel for communications between the device and the cloud-based service proved to be exceptionally challenging due to the inherent complexity of effectively implementing information security without the use of a complete, pre-packaged cryptosystem. Careful planning, provisioning, and validation testing of security-related components and resources are crucial to providing a secure environment.

Providing dependability of service was also shown to be demanding, although in this case the challenge was not due to improper design and planning but instead caused by an unforeseen transient fault that only affected certain devices. Here, investment in the system's maintainability proved to be essential to avoiding a very costly failure.

Fulfilling the requirements of mobility detection and classification was perhaps a more straightforward case where careful and thorough analysis of potential solutions identified a clearly optimal choice that, when implemented, worked quite effectively. Even though this analysis was not as comprehensive as it could have been, the ultimate result is still perfectly satisfactory, achieving the required functionality while meeting constraints of power consumption, price, and risk acceptability.

Yet, risk and cost analysis alone are not enough to determine if an optimal choice will actually result in a solution that satisfies project requirements. Thorough test implementations and prototyping of any proposed choice are also necessary to uncover discrepancies between assumptions of how a solution is supposed to work and how it actually operates in practice.

As with all applied software engineering projects, design and implementation of an embedded system requires careful balancing of many requirements and restrictions. In this case, program memory, price, and power consumption were all deciding factors in many design decisions. Overprovisioning of low-cost, high-value resources such as program memory may result in the ability to choose more optimal solutions for satisfying project requirements while remaining within other constraints imposed by the system in use.

Clearly, the maintainability of a system is a key attribute that should be ensured and protected as a high priority. Desirable attributes such as confidentiality, integrity, authenticity, availability, and reliability all contribute to a system's ability to provide service securely and dependably. However, as this case shows, maintainability is singularly vital to ensuring that these attributes are preserved in the face of latent faults and failures.

References

[1] Deloitte & Touche, "Mobile Consumer 2015: The Finnish Perspective; The new world of mobile is rising," 2015.

[2] A. Berenguer, J. Goncalves, S. Hosio, D. Ferreira, T. Anagnostopoulos and V. Kostakos, "Are smartphones ubiquitous? An in-depth survey of smartphone adoption by seniors," *IEEE Consumer Electronics Magazine,* vol. 6, no. 1, pp. 104-110, 2017.

[3] A. Avižienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, 2004.

[4] J. McLean, "Security Models and Information Flow," *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy,* pp. 177-186, 1990.

[5] ISO/IEC, "ISO/IEC 27000:2018 (Fifth Edition) Information technology — Security techniques — Information security management systems — Overview and vocabulary," 2018.

[6] U.S. National Institute of Standards and Technology, "FIPS PUB 199: Standards for Security Categorization of Federal Information and Information Systems," 2005.

[7] U.S. National Institute of Standards and Technology, "NIST Special Publication 800-12 Rev 1: An Introduction to Information Security," 2017.

[8] H. P. Barringer, "Availability, Reliability, Maintainability, and Capability," Barringer & Associates, Inc., 1997.

[9] M. Nieles, K. Dempsey and V. Y. Pillitteri, "NIST Special Publication 800-12: An Introduction to Information Security," 2017.

[10] O. K. Jasim Mohammad, S. Abbas, E.-S. M. El-Horbaty and A.-B. M. Salem, "A comparative study between modern encryption algorithms based on cloud computing environment," in *The 8th International Conference for Internet Technology and Secured Transactions*, 2013.

[11] T. Dierks and C. Allen, *RFC5246: The Transport Layer Security (TLS) Protocol Version 1.2,* 2008.

[12] T. Dierks and E. Rescorla, *RFC4346: The Transport Layer Security (TLS) Protocol Version 1.1,* 2006.

[13] T. Dierks and C. Allen, *RFC2246: The TLS Protocol Version 1.0,* 1999.

[14] B. Van Rompay, "Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers," Katholieke Universiteit Leuven, Leuven-Heverlee, Belgium, 2004.

[15] M. Gielesberger, "Alternatives to X.509," in *Proceedings of the Seminars Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 2013.

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, *RFC5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,* 2008.

[17] E. de la Hoz, G. Cochrane, J. M. Moreira-Lemus, R. Paez-Reyes, I. Marsa-Maestre and B. Alarcos, "Detecting and defeating advanced man-in-the-middle attacks against TLS," in *6th International Conference On Cyber Conflict (CyCon 2014)*, Tallinn, 2014.

[18] M. Atighetchi, N. Soule, P. Pal, J. Loyall, A. Sinclair and R. Grant, "Safe Configuration of TLS Connections," in *6th Symposium on Security Analytics and Automation*, 2013.

[19] J. Wang, Y. Yang, L. Chen, G. Yang, Z. Chen and L. Wen, "A Combination of Timing Attack and Statistical Method to Reduce Computational Complexities of SSL/TLS Side-Channel Attacks," in *11th International Conference on Computational Intelligence and Security (CIS)*, 2015.

[20] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. Smith, S. Consolvo, J. Hightower, W. G. Griswold and E. de Lara, "Mobility Detection Using Everyday GSM Traces," in *UbiComp 2006: Ubiquitous Computing*, Springer, Berlin, Heidelberg, 2006, pp. 212-224.

[21] United States Department of Defense, "Department of Defense Risk, Issue, and Opportunity Management Guide for Defense Acquisition Programs," Office of the Deputy Assistant Secretary of Defense for Systems Engineering, Washington, D.C., 2015.

[22] Transportation Research Board of the National Academies, "Magnetometer Sensor Feasibility for Railroad and Highway Equipment Detection," 2006.

[23] J. Valadeiro, S. Cardoso, R. Macedo, A. Guedes, J. Gaspar and P. P. Freitas, "Hybrid Integration of Magnetoresistive Sensors with MEMS as a Strategy to Detect Ultra-Low Magnetic Fields," *Micromachines,* vol. 7, no. 88, 2016.

[24] P. Ripka and M. Janosek, "Advances in Magnetic Field Sensors," *IEEE Sensors*

Journal, vol. 10, no. 6, pp. 1108-1116, 2010.

[25] O. D. Incel, "Analysis of Movement, Orientation and Rotation-Based Sensing for Phone Placement Recognition," *Sensors,* vol. 15, 2015.

[26] A. J. Casson, A. V. Galvez and D. Jarchi, "Gyroscope vs. accelerometer measurements of motion from wrist PPG during physical exercise," *ICT Express,* vol. 2, no. 4, pp. 175-179, 2016.

[27] W. Sousa, E. Souto, J. Rodrigres, P. Sadarc, R. Jalali and K. El-Khatib, "A Comparative Analysis of the Impact of Features on Human Activity Recognition with Smartphone Sensors," *WebMedia '17 Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web,* pp. 397-404, 2017.

[28] European Commission Directorate-General of Communications Networks, Content and Technology, "Broadband Coverage in Europe 2015: Mapping progress towards the coverage objectives of the Digital Agenda," 2016.

[29] Shanghai SIMCom Wireless Solutions Ltd., *SIM800 Series SSL Application Note V1.02,* 2016.

[30] STMicroelectronics, *STM32 Crypto Library User Manual UM1924 Rev 2,* 2015.

[31] STMicroelectronics, *STM32 Cryptographic Library Data Brief,* 2017.

[32] ARM Limited, "Obtaining Code Size - Knowledge Base - mbed TLS (Previously PolarSSL)," 15 June 2017. [Online]. Available: https://tls.mbed.org/kb/how-to/code-size. [Accessed 1 April 2018].

[33] wolfSSL Inc., "wolfSSL Build Sizes - wolfSSL," 16 August 2011. [Online]. Available: https://www.wolfssl.com/wolfssl-build-sizes/. [Accessed 1 April 2018].

[34] C. Hamilton-Rich, "FAQ (axTLS Embedded SSL)," 07 August 2016. [Online]. Available: http://axtls.sourceforge.net/faq.htm. [Accessed 1 April 2018].

[35] STMicroelectronics, *STM32L4 Cryptographic firmware library performance and memory requirements for EWARM,* 2015.

[36] M. Bellare, "New Proofs for NMAC and HMAC: Security without Collision-Resistance," *Journal of Cryptology,* vol. 28, no. 4, pp. 844-878, 2015.

[37] STMicroelectronics, *LIS2DH12 Datasheet, Revision 5,* 2016.

[38] A. Mehar, S. Chandra and S. Velmurugan, "Speed and Acceleration Characteristics of Different Types of Vehicles on Multi-Lane Highways," *European Transport \ Trasporti Europei,* no. 55, 2013.

[39] City of Helsinki, "Payment methods and parking fines | City of Helsinki," 10 April

2018. [Online]. Available: https://www.hel.fi/helsinki/en/maps-and-transport/parking/methods/parking-fine. [Accessed 20 April 2018].