02220 Distributed Systems

Gas payment from the car

Andrea Dittadi
s151496

Cristina García García
s150669

Enrico Cimitan
s151414

Piergiacomo De Marchi
s151546

15th May 2016

Contents

1 Introduction

1.1 Problem Statement

Nowadays, more and more devices are connected to each other and always better ways to improve and accelerate the common normal daily actions are provided to the consumers. The rhythm of life is becoming more and more frenetic and people are less focused on the normal simple actions.

People moved from cash to credit cards and now are moving to mobile payments. Nowadays basically all kinds of payments can be performed by mobile phone, as Android Pay, Apple Pay and Samsung Pay have already showed us. But in the case of a gas station, what if the user could pay without even getting out of the car, simply by indicating the amount of gas she wants?

Such an app could be run in every kind of satellite navigators: the only thing the user should do is ask for the fuel and allow the payment. The satellite navigator will then take care of paying the gas station and of storing the information of the transaction. Storing such information could be helpful for the user to keep track of the cheapest gas station and the weekly fuel consumption.

The system would also make the payments more secure and the process of paying and storing the user operations more user-friendly since no user interaction is required. Moreover, such an automatic system in the future could even become the starting point of a revolution: a completely automatic refueling-system with no user interaction whatsoever.

This system should provide a secure way of payment that considers how the sensitive data is stored, how it should be exchanged, the threats to the system and how to protect the system from them. It should also be robust enough to ensure the proper functioning even if the communication between the devices, or the devices themselves, fail. Finally, the information of all the users is to be stored in a server to enhance the user's experience. It is also highly important that the system is scalable when considering the amount of potential users.

As for security regarding payments, our system will include HCE (Host Card Emulator) and tokenization. Nowadays, people usually pay with credit card in a gas station. Encryption is used to protect the sensitive data so that only authorized users can read it, but the best approach to protect the data is to remove it. With tokenization the user's bank account information is neither stored in the device nor sent. Instead, a token is generated in the cloud and used. This is more secure as it limits the consequences in case an attacker steals the token or gains access to the device.

1.2 Structure of the report

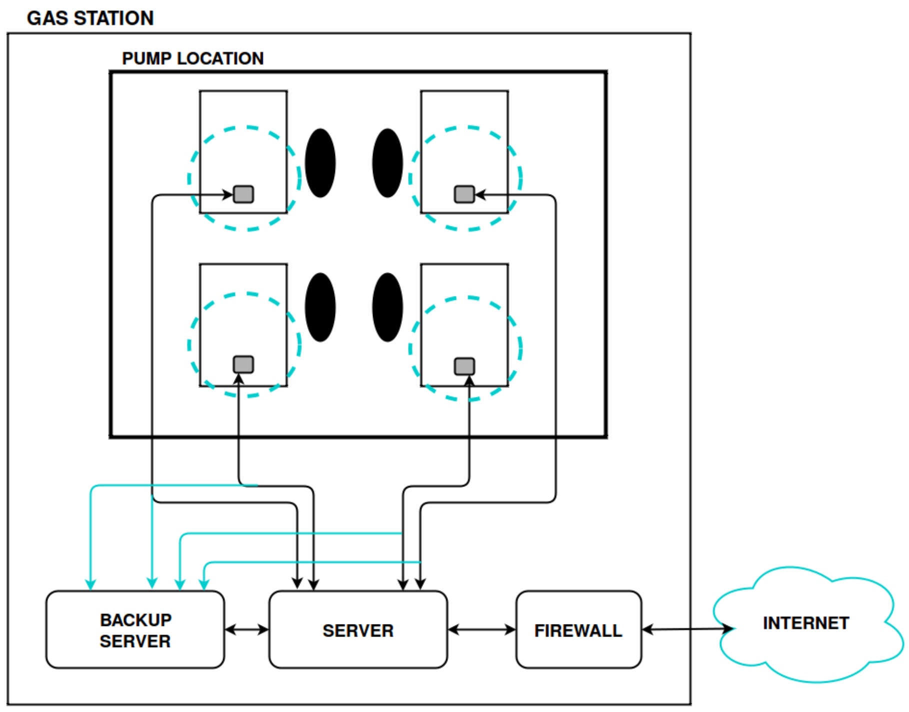

Figure 1.1: Overview of the system.

An overview of the gas station system can be seen in figure 1.1. When the car approaches the pump (the black ellipse in the figure), the satellite navigator will receive the information that is being broadcast. Then it will reply with the same ID and the amount of money to refuel and the communication will start. Thanks to a limitation of the operating field of our routers, we assume that one car cannot receive two different IDs corresponding to two pumps due to the small working distance.

In this report will analyze how the communication between the car and the gas station, as well as with the cloud, should happen in order to build a system that is secure, fault-tolerant and scalable, and how this information should be stored. Security is undoubtedly of foremost importance in this system due to the sensitive information involved. However, addressing this vast topic in a mobile payment system is a huge problem that exceeds the scope of the project. Therefore, the focus has been mainly on the HCE implementation as well as on the threat assessment. The system has been analysed from a high-level perspective and some solutions have been provided for the threats identified.

Enrico Cimitan will work on making the system fault-tolerant. Cristina García García will be responsible for dealing with the scalability issues. Piergiacomo De Marchi and Andrea Dittadi will focus on the problem of how to exchange the information in a secure way between the device, the gas station and the cloud as well as the secure storage of the sensible information.

First, an overview of the system is introduced in chapter 2. Chapter 3 deals with the problem of how to make it fault-tolerant. Chapter 4 addresses the scalability issues. Finally, Chapter 5 will explain in detail the security aspects of the system, and in Chapter 6 we will mention other issues of our system that we did not address.

2 Gas payment

2.1 Overview

The Internet of Things (IoT) has come to make people's life easier, smarter and more efficient. Forecasts show that 50 billion devices will be connected by 2020 and also 90% of cars. We can benefit from this and allow the user to perform the payment without leaving the car, while at the same time making the system more secure without needing the user to worry about it.

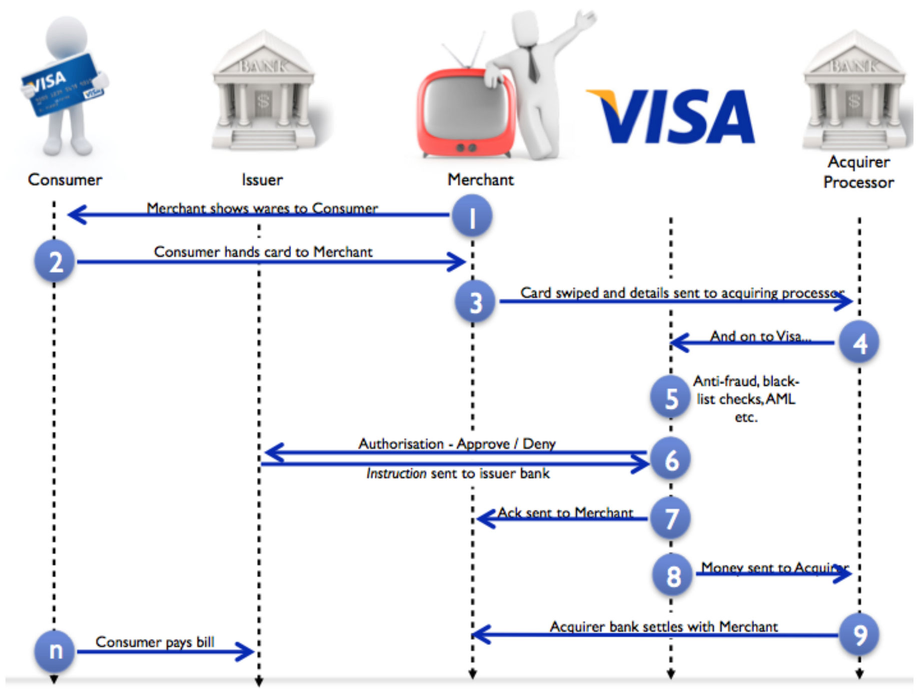
Consumer
Issuer
Merchant
VISA
Acquirer Processor
Merchant shows wares to Consumer
1
2
Consumer hands card to Merchant
3
Card swiped and details sent to acquiring processor
4
And on to Visa...
5
Anti-fraud, black-list checks, AML etc.
Authorisation - Approve / Deny
6
Instruction sent to issuer bank
7
Ack sent to Merchant
8
Money sent to Acquirer
9
Acquirer bank settles with Merchant
n
Consumer pays bill

Figure 2.1: Visa pay flow for a card transaction.

Nowadays, to perform a card transaction the details of the card have to be sent through different entities in order to perform all the controls and verifications, as shown in figure 2.1 (image from [4]). Although this sensitive data is encrypted, security breaches may occur and this information can be exposed. Also, if someone has access to the key used for the encryption, she can also obtain the information. *Tokenization* is a process that replaces this sensitive data for non-sensitive place-holders to reduce the potential data exposure and only access the original value when strictly necessary.

The system will also provide another additional service for the user: it will allow her to track the fuel consumption. Each time the user visits a gas station, the data will be recorded in a server. Storing this information will allow the user to consult which are the cheapest gas stations as well as managing the fuel consumption. The Host Card Emulator (HCE) technology can be used along with tokenization to achieve this service. An overview of the proposed system can be seen in figure 2.2.

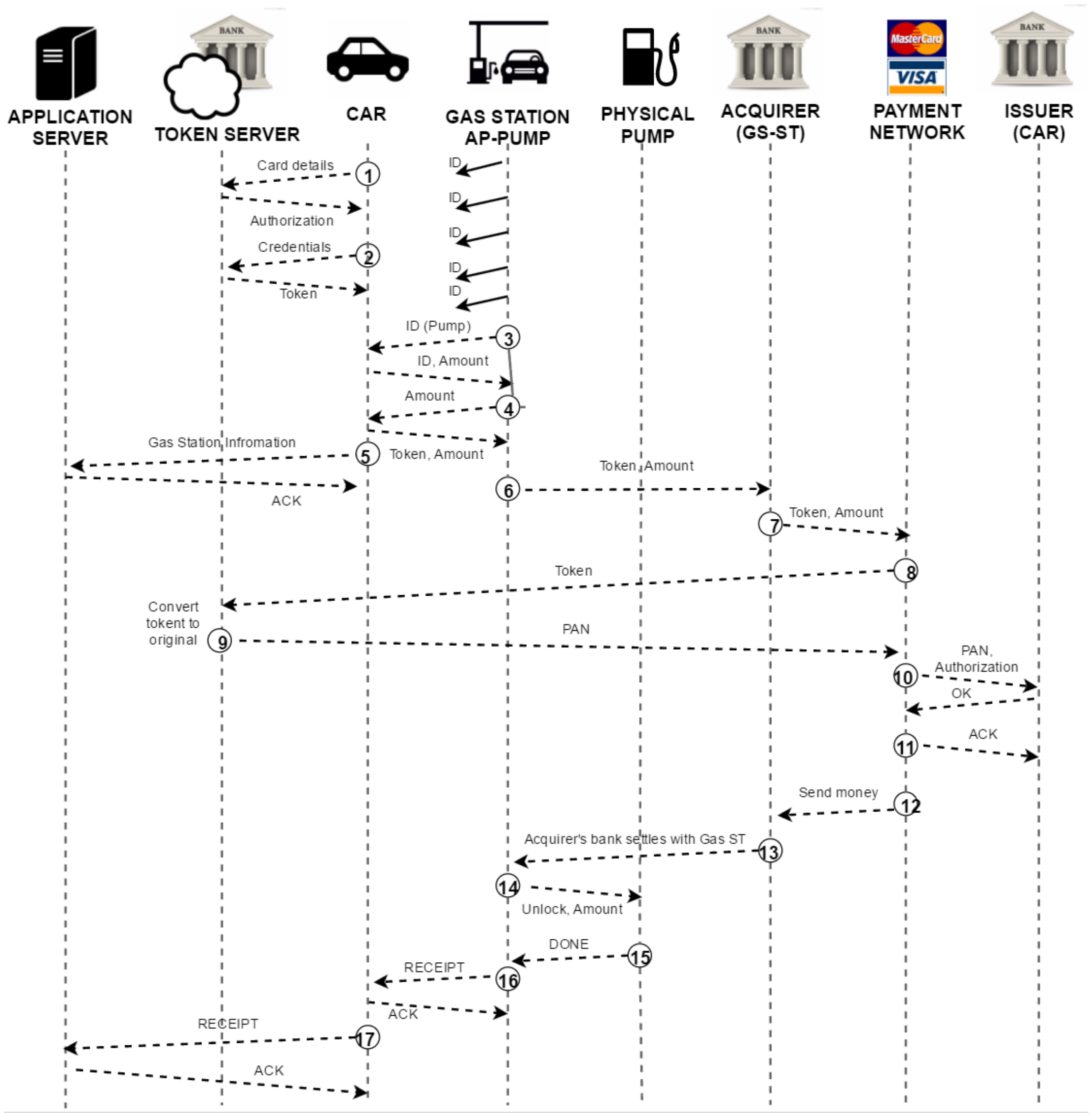

Figure 2.2: Gas payment from the car using tokenization

Note that only the involved parties in the payment process are shown. Our system also consists of an app that can keep track of the fuel consumption, providing useful information for the user such as the cheapest gas stations, as mentioned above. The information exchange between the app and the server can be done by using standard protocols such as HTTPS and therefore it is not shown in the flowchart.

2.2 Distributed architecture

Our system consists of many different parts, as shown in the figure 2.2. The car represents the user of our system, that is the customer of the gas station. It interacts with the first and outermost entity of our system, that is the access point located at the pump. This access point immediately forwards any message to the gas station, that is the central entity of our system. The gas station communicates then with the physical pump and the application server, that are other two internal entities of our system; the first has the purpose of dispensing fuel, the second one of recording the purchases of a customer in order to create a history accessible by the user. The other entities acting in the system are external entities, mainly connected to the payment process, that are not directly managed by our system, but whose channel must be taken in consideration in our discussion.

The interaction between the single entities will be discussed in depth later, but we want to give a general idea of how we have conceived the communication between them. We will identify the entity with the processes they run, in a system perspective, but in general we will use the same name to indicate the hardware associated with them.

car↔router: these two entities are strongly correlated by the fact that they need to be very near.

router↔station & station↔physical pump: in our idea these entities are closer to each other: we will not take this as an assumption, allowing a gas pump to rely on a far station. This could carry latency problems.

station↔application server: this communication is physically decoupled, and we allow the two entities to be far from each other.

Due to the sensible nature of the setting (handling payments and giving a primary importance to the user experience), our system is going to be a synchronous distributed system. This means that each operation will have a time bound to be executed, and the channels are supposed to have a limited amount of introduced delay.

2.3 Assumptions

Some assumptions have been done for the design of the system:

- The car has a satellite navigator connected to the Internet and that it is capable of authenticating the user as well as performing the rest of operations needed to communicate with the different servers. This assumption is reasonable given the large amount of connected cars nowadays.

- The communication between banks as well as the operations that have to be performed are assumed to be already implemented by them and no faults in that part of the system are considered.

- There are only few pumps per gas station. This seems as a fair assumption, as nowadays there are not gas stations with dozens of pumps.

- A physical pump may fail, but it cannot deceive about the amount of fuel given to the user. In other words, the receipt send to the user at the end of the operation reflect the exact amount of fuel that she has received.

3 Fault Tolerance

Any distributed system relies on the communication between a number of devices. If one of these communications fails (either because of the channel or because of a component), the system should be robust enough to keep working correctly. In the case of our system, a money loss of the customer could be the effect of such a crash; therefore fault tolerance gains a primary importance in this setting.

In the following chapter we will address such problems one by one, refining and improving our previous design and protocols, until we reach a satisfying level of reliability for our complete system. We will proceed in a bottom-up fashion, addressing the problems in a smaller scope (i.e. the single pump) and then expanding up to the whole system architecture.

Note that in the following we will assume that the bank systems and channels will not fail; in an implementation of the system, this is a reasonable assumption, since our protocol could not work without the bank servers being reachable and active. In fact the only thing we could do is make sure that we have a at-most-once communication, and just become unable to offer the service if the bank stops responding.

3.1 Router access points

The first problem that we have to address is the communication between the car of the customer and the receiving router that is associated with each pump. The interaction between the two parties starts with the pump router broadcasting a low-range message containing its ID, and the car moving in the range to receive this message. It may happen that the router has crashed, and a car approaches the corresponding pump expecting the interaction to start; and therefore, waiting forever (or, until the customer gets tired of waiting). We could think of many ways of tackling this "omission failure"; the two main ways that do not involve a sensible modification in our overall design are:

- Use more than one router per pump: This approach guarantees that, in case one of the routers of a pump is no longer reachable, the other one can connect to the car and initialize the communication. This approach, however, has some important drawbacks. It causes the car to receive messages from different routers regarding the same pump from which the car has to choose one; this can in principle cause a security problem (because of someone faking to be a router). It can also cause two different vehicles to connect to two access points of the same pump; this would require a mutual exclusion access to the pump administration by the two routers. We can implement such a protocol, but risking more delay in the overall execution, that is something we do not want the customer to experience.

- Monitor and tolerate failure: An alternative approach is to keep the system lightweight, avoiding to add more devices that manage the same pump. Keeping only one router per pump implies that if a router fails its pump is no longer usable by a customer. Since we assume that the station has more than one pump, the failure of one of them is an event that we can tolerate, as long as we have a way to prevent users from trying to access it. This result can be achieved by implementing a broadcast service in the server, that periodically asks each router for its state. If the router does not answer within a certain timeout, than the server flags it as not working, and shows an error message in the display of the pump. (Notice that we are assuming here that a router's capability of broadcasting its ID fails if and only if its capability of answering to the server fails, coherently with our view of the entities as processes)

Because of the reasons explained above, we are going to privilege the second solution. Our system can survive the loss of a pump with a non drastic impact on the user, as long as we notify him of the failure, for example with a message on the display of the pump or with a screen at the entrance of the station that reports the event of a malfunctioning pump. However, we have to be aware that in case of a big number of pumps the messages to verify the aliveness of the access points could be a significant amount; based on the assumptions for our system, though, we can deduce that the amount of pumps will never be so big to cause such a decrease in the system performance.

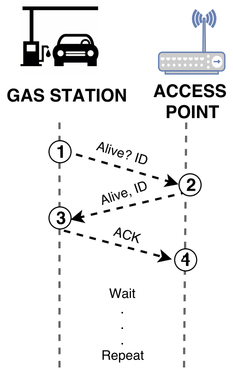

Figure 3.1: The basic protocol to check if an access point is reachable.

3.2 The physical pump

The physical pump is another atomic part of our system that is crucial in the interaction with the customer. It is run by a very simple protocol that is responsible of regulating the distribution of fuel according to a message from the server, and to answer this message with an acknowledgement. We have to consider two different cases of failure of the physical pump: the complete failure or the impossibility to complete the fuel delivery.

If the pump stops working when no customer is being served, then our system should behave as in the case of the access points: notifying the customer in advance that the pump is out of order, and preventing the fuel to be ordered from that pump. This requires, similarly to the previous section, that the system knows whether the pump has crashed. The liveness of the access points was checked by broadcasting a control message; we can either follow the same approach, and keep flooding the network with "ping messages", or delaying the control at payment time.

In this case we choose the second approach. In the previous section the communication was started by the router, so the server had to take the initiative to have confirmation of functionality. Here the server is the entity that starts the communication (thus acting as a client for the pump), therefore it can just check the availability of the pump whenever it is needed. This allows us to save a lot of messages sent, paying as a price the fact that the first customer after the failure will not be aware of it before reaching the pump (because the server has not identified the failure yet).

We have then to handle the case in which the pump is not able to deliver the fuel. This includes the case of the fuel supply ending, the case of the pump failure occurring immediately having confirmed its liveness to the server, and the failure occurring during the fuel dispensing. To handle this case, it's sufficient to have a timeout for the server waiting for a response from the pump. In case the timeout expires, the server deduces that the pump is no longer working, and can issue a refund ticket for the undispensed amount (if the pump sends periodical status updates), or for the whole payment.

Note that we are not handling the case of a wrong amount of fuel dispensed without the pump being aware of it; this is a reasonable assumption, since such an event is very rare and anyway not handled by the current existing systems.

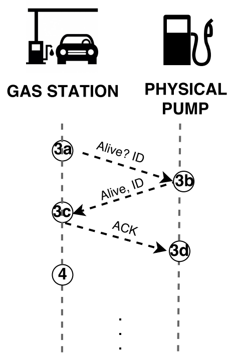

Figure 3.2: The basic protocol to make sure that a pump is reachable

3.3 The station server

The design of the intra-station architecture of our system is based on a thin client/fat server approach. This means that almost all the computations, decisions and outside interactions are managed by this central entity that is therefore a potential bottleneck of the system. Being the number of pumps usually very low, this server embodies a problem for fault tolerance rather than for scalability: if the server fails, this means that no pump will be able to sell and dispense fuel.

To handle failures of the station server and keep the system up and running, we should use more than one server. We have to decide, though, how to make these servers behave towards one another and towards the pumps. One possible way is to use the second server as a backup server that handles the requests only if the first and main server has crashed. As a second choice, we can keep them active at the same time, but this would rise problems on mutual exclusion in the handling of a pump. Therefore, we are choosing the first option, and keep the second server as an alternative choice to be chosen only in case a certain numbers of requests to the main server has failed consecutively.

But what if the station server fails when the payment has been processed but the pump has not been unlocked yet? To handle this case, and exploiting the fact of having a central enterprise server that records all the purchases, we can think of a simple additional step for the protocol: when the car sends the token to the station, it also sends a message to the enterprise server, recording the basic information on the station and the payment. In this way, in case the station server fails in this critical instant, the customer can prove its presence in the station, even if the receipt has not been issued before the crash.

3.4 The enterprise server and the cloud database

As mentioned before, we want to be able to record the purchases of single customers, to be able to provide them with some statistic data, for example on their average monthly consumption. This server provides also an additional level of confidence in the system for the customer, that can prove its purchase as explained before. These services have to be handled by a central enterprise server, that could also be used for the administration of the fuel price and other wide scope operations. Such a server is clearly a bottleneck of the system: every station needs to communicate with it, therefore (on the side of a scalability problem) we have a node whose crash would mean that its function will be lost until its fixing.

To prevent the failure of this system we could think of having more replicas of the database; this makes it harder to keep them consistent (requiring reliable multicast communication to be sure that every database is recording the same actions) and to give mutual access to them, but helps the fault tolerance.

Note that the car should have timeout for the ACK from the enterprise server; otherwise the waiting time can be unlimited, if the server is not going to answer because of a failure.

4 Gas Manager

Many mobile apps have been developed for fuel tracking, such as “My Cars” for Android or “Fuel Monitor” for Apple . They allow the user to track when, where and how much gas she buys, as well as the costs. Some of these apps have thousands of users, but all share the same problem: the user has to manually input the data.

But what about a system that could have all the advantages of those apps without any troubles? The satellite navigator will take care of storing the necessary information in the server. This information will provide the user with the necessary data to keep track of the consumption or to know which are the cheapest gas stations and it could also be used by other apps to provide further services. Nevertheless, how to implement an API for this is out of the scope of the project.

4.1 Information exchange

Chapter 2 presented an overview of the system. The basic flowchart considering the communication with the server can be seen in figure 4.1.

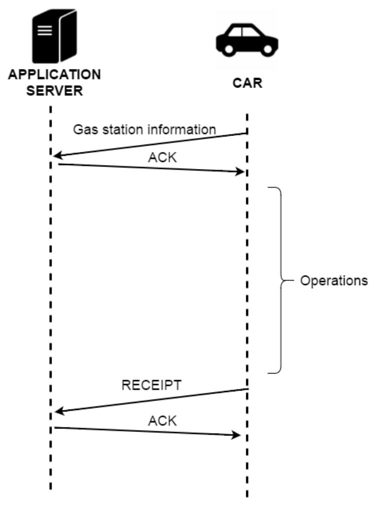

Figure 4.1: Information exchange - basic flowchart

First, the car will send the information related with the gas station once it has sent the token: the name of the gas station, the date and the amount of money that the user wants to pay. The name of the gas station is sent in the first iteration between the gas station and the car whereas the satellite navigator can provide the time. This information will be stored in the server and an ID corresponding to the user is also sent to store the data in the relevant entry in the database. The purpose of recording this information before finishing the operations is to log it in case something fails in the gas station system and no receipt is sent. This way, the user can proof that his request has not been satisfied by correlating it with the information that the banks store relating the payment.

If the system is working properly, the user will receive a receipt in the end confirming the amount of fuel that she has received, as well as the confirmation of the amount of money charged. The car will forward this information to the server to be stored so that it can calculate the volume price. This information may be used after to compare different gas stations to inform the user of the cheapest ones.

The values that are stored in the database are:

- **Name of the gas station.**
- **Date and time.**
- **Amount of money.**
- **Amount of fuel.**
- **Volume price.** Calculated based on the previous values.

4.1.1 Protocol

HTTPS is used for the information exchange between the car and the server, which is the standard for the communication between apps and servers. The **User Agent** header will contain the name of the app.

4.2 Scalability issues

There were approximately 1.2 billion cars in the world by 2015, the number increasing by millions each month. And the forecasts show that a high proportion of them will be connected in the years to come. Therefore, scalability is a main issue when designing a system that will allow users to log some data from their car every time they go to a gas station. So not only the information stored about each user will increase, but also the number of them. In order to handle this, scalability must be considered in the development of the system. Trying to scale a system that was not been designed for it is a very hard task.

One definition of “scalability” states that it is the “system’s capacity to uphold the same performance under heavier volumes” [8]. Some principles to bear in mind when designing a scalable system are:

- No machine should have neither need complete information.
- Decisions should be based on local information.
- The failure of one machine must not ruin the results.
- A global shared clock should be avoided.

When looking at out system, all these properties are feasible.

If the capacity is not enough to handle the system's requirements, two strategies can be used to overcome the problem [7]:

- **Scale up: vertical.** Adding resources to a single node will directly increase the amount of work that it can handles. This can be done, for instance, by adding CPUs or memory to a computer.
- **Scale out: horizontal.** The other approach is to add more nodes to a system rather than improving the capacity of the existing ones, *e.g.,* by adding a new computer.

The advantage of vertical scaling is that also the speed of the system will increase, but is also limited by physical constraints. Horizontal scaling has the advantage that more machines can be added when needed. However, a way to distribute the request has to be set in the latter case to decide which machine should deal with a request. Both techniques are usually employed.

4.3 Possible architectures

Presumably the system will not be composed of only one machine, specially if the intention is to make the service scalable. It is important to choose a suitable architecture in order to be able to scale the system in the future without needing to modify the underlying structure.

There are two main architectural styles: client-server and peer-to-peer (P2P).

- **Client-server:** there are two classes of processes depending on whether they request a service (client) or they implement it (server).
- **Peer-to-Peer:** in this model all the processes have similar roles and work together to fulfill a task (peers).

Th P2P architecture is much easier to scale since it has been designed to exploit the resources of all the machines: all the peers can both provide or consume services to/from other nodes. However, the security decreases considerably, as no authentication is taken. In addition, the client-server model is also much more stable since the time when the peers are disconnected cannot be controlled in the P2P model.

The client-server approach is best suited for this system but there are still different possibilities depending on whether the processing and data management are split in different machines and how this separation is done.

4.3.1 Single server

The simplest architecture is based on using **one single server**. While it is very easy to set up, it is almost impossible to scale.

4.3.2 Application and Database servers

The next step is to split into two the roles: one machine will be the **application server**, which handles the requests from the user, and another one will be the **database sever**, located in a private network, which also increases the security. Vertical scaling can be used in this architecture for each component. Even if the capacity is increased compared to the former architecture, it is still not enough if the number of users is substantial.

4.3.3 Load balancer

Load Balancer (Reverse Proxie) can be used to distribute the load for multiple servers. This enables horizontal scaling as more servers can be added to the infrastructure. However, there is still a scalability problem if considering the database server, as all the application servers would need to access it. It is also needed a load balancing algorithm:

- **Random:** this method picks a random number to distribute load across the servers. It is not very effective as the load is not equally distributed among the servers.
- **Round Robin:** the connection requests are sent in order. It can work well for some configurations, but is not the most effective if the servers are not equal in processing or connection speed.
- **Weighted Round Robin:** it is an improvement over Round Robin. The number of connections that each machine receives is proportionate to a weight that the administrator can define. This way, if one server can handle more load than other, the load balancer will know.
- **Least Connections:** the requests are sent to the servers based on the number of current connections. This method works better when the servers have similar capabilities. For instance, consider two servers, A, with a limit of 150 connections, and B, limited to 500: if A has 140 connections and B is handling 200, server A

will be selected even if it is closer to reach its maximum capacity than the other one.

- **Weighted Least Connections:** similar to Least Connections but capacity of each server is taken into consideration. It works best when the servers have different capabilities. If not, it will behave as the Least Connection method.

4.3.4 Scaling the database

Other architectures are possible when considering a scalable database. Again, the simpler one is to use one **single server**, but it is not very efficient.

Sharding is a technique whereby the size of the database is reduced by holding the rows of a database table separately. This follows the principle of *horizontal partitioning*. Then, the database is divided in *aggregates*, that combine data that's usually accessed together. For instance, it could be divided based in the location: Danish and German consumers, which will also make easier where to direct the query. Figure 4.2 displays some examples. The main drawback of this technique is its *complexity*.

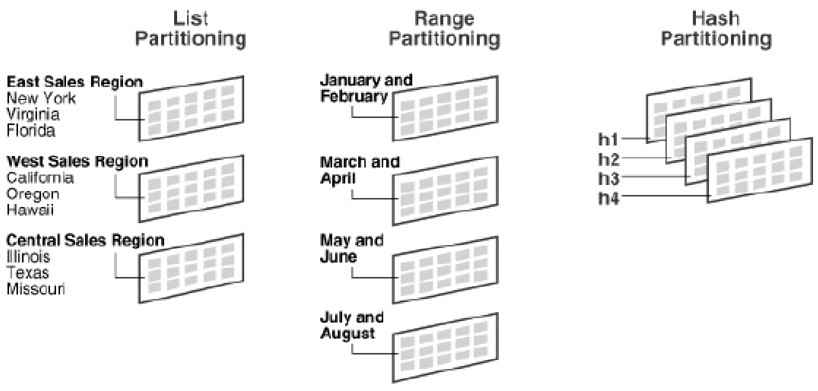

Figure 4.2: Sharding examples using different criterias.

Master-Slave Database Replication can be another solution. One of the databases, considered the master, is used only for updates (write) while one or more slave nodes can deal with reading operations. However, to make the system consistent, each time a change is made the master database has to replicate the information to the slaves. This approach is recommended when the system performs many reads compared to writes. To make the system tolerant to failures, one of the principles stated in 4.2, one of the slaves should be able to substitute the master in case that it crashes.

Peer-to-Peer Replication. Instead of having one master, every node has read and write access, which solves the failure problem. However, consistency here is an issue.

4.4 Scalability of the application system

Taking into consideration the different approaches, the designed system will be a combination between the load balancer and the master-slave database. Both are needed to allow a horizontal scaling in the future, and the master-slave approach has been chosen taking into consideration that an average user does not refuel the car more than once a week. Therefore, even if the number of users is huge, we believe that this can be feasible. However this assumption cannot be made for the read operations: we have to assume that the number of reads performed can be much higher than for the write operations. The *sharding* technique seems also a good solution for handling scalability. However the complexity associated makes it only realistic for systems with strong availability requirements, whereas other techniques may be more suitable for the system here considered.

The system architecture is shown in 4.3. The approach of using a load balancer as well as the implementation of master-slave databases allow to scale out the system easily as the number of clients increases.

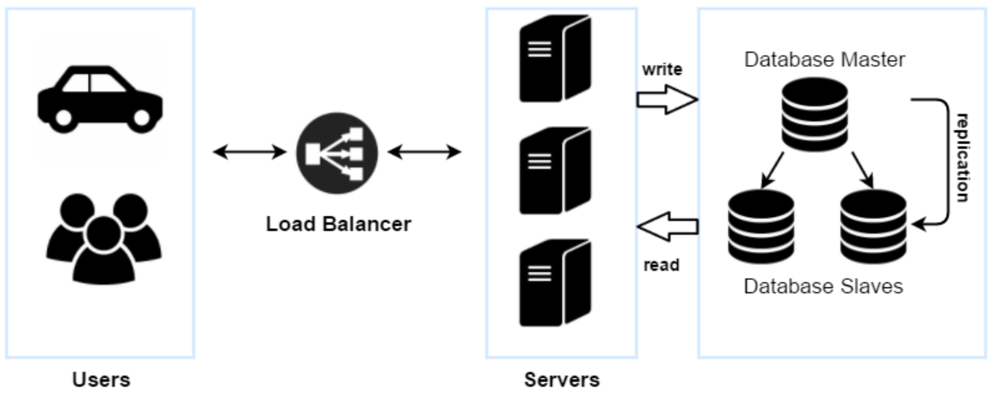

Figure 4.3: Architecture.

When considering a system with a Load Balancer, a load balancing algorithm has to be selected. For this system, the most appropriate approach seems to be the *Weighted Least Connections*. If the capabilities of the servers are similar, *Least Connections* could also be used. However, the former option seems more appropriate as we cannot be certain of which are going to be the capabilities of the equipment. This approach is the one that makes more efficient use of the machines available, which is directly related to scalability.

4.5 Scalability in the gas station

We have not considered the scalability property a problem for the gas station. The reason for this is related with one of the assumptions made in 2.3: there are only few pumps per gas station.

There are some massive gas stations, mainly in the US, with almost 100 fueling locations [1]. However, this is not generally the practice. Usually, the amount of fueling locations can be reduced to 4-8. Even if the number is a bit bigger, we believe that there are not real scalability issues for this scenario, and therefore this problem has not been considered to be as relevant as the other ones that have been addressed.

[1] Buc-ee's opens massive new location in Baytown. http://abc13.com/shopping/buc-ees-opens-in-baytown/440483/

5 Security

There are plenty of vulnerabilities and threats for a system such as the one under consideration, that involves so many parties. A complete vulnerability assessment and risk analysis are a hard task to perform, and a suitable approach could consist of threat assessment, vulnerability assessment, risk analysis (using frameworks such as CORAS or Octave), countermeasure and re-evaluation.

However, this is out of the scope of this project. We will instead focus mainly on the threat assessment of the system, and propose some general solutions for the identified threats. Furthermore, for the same reason, we will not go into details of the attack surfaces and the security solution implementations, but we will analyse the system from a high-level perspective.

5.1 Threats

Below is a list of the threats we identified. Each of them is further explained in the next section, along with a proposed solution for it.

- First authentication of the car to the token server provider (HCE vulnerabilities)
- Storage of tokens in the car's satellite navigator
- Man-in-the-middle (MITM) and eavesdropping. In the MITM attack, an attacker intercepts, possibly alters, and forwards messages between two parties who believe they are directly communicating with each other. A MITM attack where no message alteration occurs is called eavesdropping. In this case the attacker makes independent connections with the two parties and relays messages between them to make them believe they are talking directly to each other.

  The MITM (both in the form of manipulation and in the form of eavesdropping) vulnerabilities in our system can be exploited to access sensitive information. These attacks can take place:

  - between the car and the gas station's access point;
  - between the car and the token service provider;
  - between the gas station's server and the acquirer's bank;
  - between the gas station's server and the physical pump;
  - between the car and the application server;
  - between the application server and the database server.

Similarly to other issues addressed in the previous chapters, we are assuming that the communications among Acquirer's Bank, Payment Network and Issuer Bank are kept safe by the Card Network companies that provide the tokenization services.

- Replay attacks. In general, because of performance issues, the tokens could be reusable and an attacker could send them many times to perform different payments.
- The software running on the gas station server has to be tamper-resistant. By tampering, an attacker can gain control over some aspect of the software with an unauthorized modification that alters the software's code and behaviour. In our case, an attacker could tamper with the software by connecting to the gas station with a laptop. Ways of tampering could be installing rootkits and backdoors, disabling security monitoring, malicious code injection for data theft, altering communication between cars, pumps and the server itself. The purposes of this could be to get free fuel or steal sensitive information from communications with cars.
- A satellite navigator can be associated with many credit cards and a car can be used by many different people so a way to protect a CC stored on the system is needed. Someone inside the car could look at your PIN code while you type it (sol: we could use an addition small screen on the driver's side, or authenticate through the smartphone)
- Gas station's server and the backup server have to be tamper-resistent against malicious entities
- Distributed Denial of Service (DDoS) attacks:
  - DDoS to the server through the access points
  - DDoS to the car that is refueling
  - DDoS to the server through the Internet
- Confidentiality, integrity and availability of the application servers and the database servers

5.2 Solutions

5.2.1 First authentication

To avoid all the technical and business complexities of card credentials stored on devices in Secure Elements (SE), financial institutions are looking to move card credential data to the cloud. With Android's support for Host Card Emulation in the KitKat OS, cards in the cloud are no longer pie in the sky. Moving sensitive card and personal data out of a phone's secure element into the cloud solves business problems by reducing the number of actors and barriers to integration to existing systems.

The security of the system is also improved since it is possible, though not easy, to break into a Secure Element, given the right tools (such as lasers and spectrometers and a certain amount of time). Research in fact shows that there is no attempt to reduce side-channel emissions on a standard ARM CPU such as shielding to reduce emanations or hide power consumption patterns and this causes an SE to be vulnerable to this kind of hardware attacks.

With this knowledge we have therefore decided to adopt Host Card Emulation in our device in order to move all the sensitive information of a credit card and the details of its owner to the cloud. On the other hand, this leads to weaknesses during the initialization phase in which a user has to enroll/register his bank account and his CC with the cloud-based server that provides the wallet application.

To solve this problem, the Smart Card Alliance suggests two solutions: Cloud Storage without Tokenization and Cloud Storage with Tokenization [2]. However, since the former is not considered secure, we decided to go for the latter.

Cloud Storage with Tokenization consists of the following steps:

1. The issuer offering the mobile payment service guides the customer through an enrollment/registration process that incorporates strong authentication methods.
2. The customer's mobile app is provisioned with payment tokens.
3. At time of payment, the customer's mobile payment app provides the tokenized payment credentials to the merchant's POS. The customer may be prompted to enter a PIN specific to the use of the mobile payment app.
4. The merchant routes the transaction to the acquirer, and the transaction is ultimately received by the issuer (over the payment network) for authorization. The issuer (or an entity acting as the token vault on behalf of the issuer) authorizes the transaction after verifying the token and identifying the associated payment credentials.

Notice that this solution also need to consider how to securely provision the payment token to the device. We are going to deal with this in the MITM attacks section.

Furthermore, this solution doesn't reduce the risk of credential exposure due to malware on the device, but reduces the impact of possible exposure by replacing the static payment credential with a token of much reduced scope. To make everything more secure we also customize our tokens in order to reduce their validity. The tokens should be:

- Valid only for transactions not exceeding 1000 DKK (which we assume is enough to fully refuel a car)
- Limited to a single use

- Valid only for a limited time

Thanks to these considerations an attacker with superuser rights on the device will manage to steal only few tokens for a small amount of money that must be used within a short time.

5.2.2 Secure storage of the tokens in the navigator

HCE and tokenization solve the problem of locally storing and sending the PAN (Primary Account Number) to the merchant making the communication between satellite navigator and Gas Station more secure. On the other hand, the storage of these tokens in an electronic device can be really dangerous and is considered one of the two weakest points of HCE (the other one is the transmission of the tokens between the token service provider and the navigator).

To avoid annoying delays and long waiting times during the payment phase, tokens are provided in advance and are stored inside the device for future use. This common technique is on the one hand very user-friendly, fast and reliable, but on the other hand leads to security problems.

Nowadays applications don't address the problem of securely storing the tokens and just rely on the OS's security and on the fact that tokens are usually valid only for one transaction and only for payments below a certain threshold. On the other hand, we think our system is dealing with a great amount of money (on average for a person 900 DKK per week; 432000 DKK per year) and therefore we decided to find a solution.

In order to guarantee confidentiality and integrity of the tokens stored, we store them into a hardware tamper-resistant SE (Secure Element), much harder to break compared to the OS and the memory of a system.

Secure Elements that are compliant with the Global Platform standard, are "personalized" with unique keys (colloquially known as *card-manager keys*) required to perform administrative operations. These keys are not known to the owner of the device (in our case the satellite navigator) or even the operating system on the mobile device; they are managed by a third-party called "trusted services-manager" (TSM).

When the TSM is installing a payment application and configuring that application with its own set of secrets, the commands sent to the SE for performing those steps are encrypted. The Global Platform specification defines a "secure messaging" protocol describing the exact steps for establishing an **authenticated & encrypted** link between TSM and secure element. This protocol – or more accurately series of protocols since there are multiple variants – is designed to ensure that even when TSM and SE are not in physical proximity, as in the case of provisioning a payment application over the Internet (like in our case), sensitive information such as payment keys are delivered only to the designated SE and not recoverable by anyone else.

While it is true that the commands sent by TSM are visible to the host operating system, where they can be intercepted or even modified by our hypothetical adversary who has attained root privileges, the secure messaging protocol ensures that no useful information is learned by mounting such an attack an no altered data is accepted by the SE [5]. To sum up, the storage of the tokens in a SE requires much more effort for an attacker to retrieve them. To break the SE he would probably need of expensive hardware instrumentation as lasers and spectrometers. Naturally adopting a solution that makes use of an SE will rise the cost of the product, but the relation benefits/costs will still be convenient.

5.2.3 MITM attacks

Since there are many entities that are communicating in the system remotely, MITM attacks are maybe the most probable kind of attack that can take place. An attacker, in fact, can be really interested in eavesdropping the communication in order to steal some tokens and use them at a later point. By manipulating the packets transmitted to the gas station from a car, he could also pretend to have paid an huge amount of money and hence have a great amount of fuel, when instead no money has been paid.

Between the car and the gas station's access point. We limit the range of the access points to a few meters, by limiting the power of the transmitted signal, so that an attacker has to be close to the pump in order to perform a MITM.

We also try and prevent MITM attacks from the car's point of view, by means of one-way authentication, where the car authenticates the gas station before performing any transaction with sensitive data. For this purpose, all gas stations would need a public key certificate released by a Certificate Authority (CA). A public key certificate is an electronic document that is in general used to prove ownership of a public key. It includes information about the key and its owner's identity, and the digital signature of the CA that has verified the certificate's contents are correct. The car inspects the signature, and if it is valid then it knows the associated key can be safely used to communicate with the gas station.
Naturally, the gas station's CA must be certified by a Certificate Authorities to be trustworthy.

Between the car and the token service provider. This is maybe the weakest channel of all the system. The first-authentication phase plus all the tokens provided by the token service provider take place via this communication channel. Since the network we are analysing is the Internet and hence is a **non** reliable one, we must ensure authentication and encryption. Naturally, we are going to use the standards set by the Payment Networks and the APIs provided, but for sure we need of a strong encryption algorithm as well as a reliable digital signature system.
The Payment Card Industry (PCI), that is the standardization authority for mobile payments suggests a list of P2PES (Point-to-Point Encryption Solutions) for the exchange of the messages during the first-authentication phase and for the subsequent tokens exchange phase [3]. Visa also suggests its own standard: Visa's PCI-based Cardholder Information Security Program (CISP) [6]. Since it is very difficult to argue which one

is the most secure, we decided on Visa's one for a possible implementation only because the documentation is really clear and easily understandable with respect to the PCI standards.
Visa hence provides two APIs. The first one is used to enroll a new PAN on the Visa's secure cloud in order to receive a new unique ID number associated with the bank account related to that PAN. The second API instead is used to provide a freshly generated token upon a request received carrying the unique ID had during the enrollment of the PAN with the cloud.
The relevant characteristics of the updated protocols are depicted in Figs. 5.1 and 5.2.

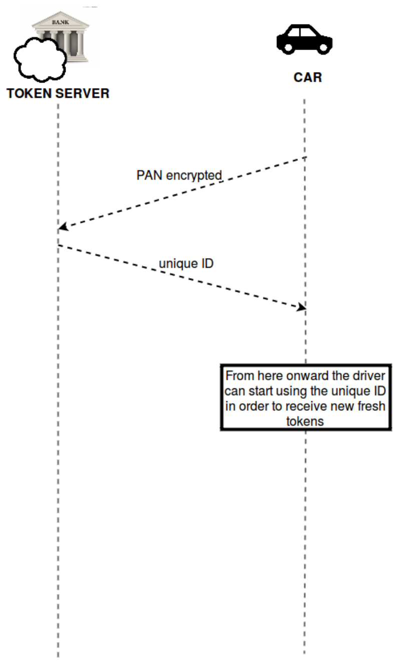
TOKEN SERVER
CAR
PAN encrypted
unique ID
From here onward the driver can start using the unique ID in order to receive new fresh tokens

Figure 5.1: Enrolment of the driver's PAN and reception of the unique ID.

Between the gas station's server and the acquirer's bank. This kind of communication is very similar to the previous one. It is carried on over the Internet network and consists on sending tokens from the gas station to the acquirer's bank and receive their confirmations. The first-authentication phase is not performed over this channel, but a malicious entity could again steal or manipulate the tokens on this communication channel. The The tokens require naturally to be encrypted and the authentication and non-repudiation of what has been sent is also this time very important. To perform the transaction Visa provides another API that makes use again of the same E2EES as before [6].

Between the gas station's server and the physical pump. The connection between the gas station's server and the physical pumps maybe is the one that, sur-

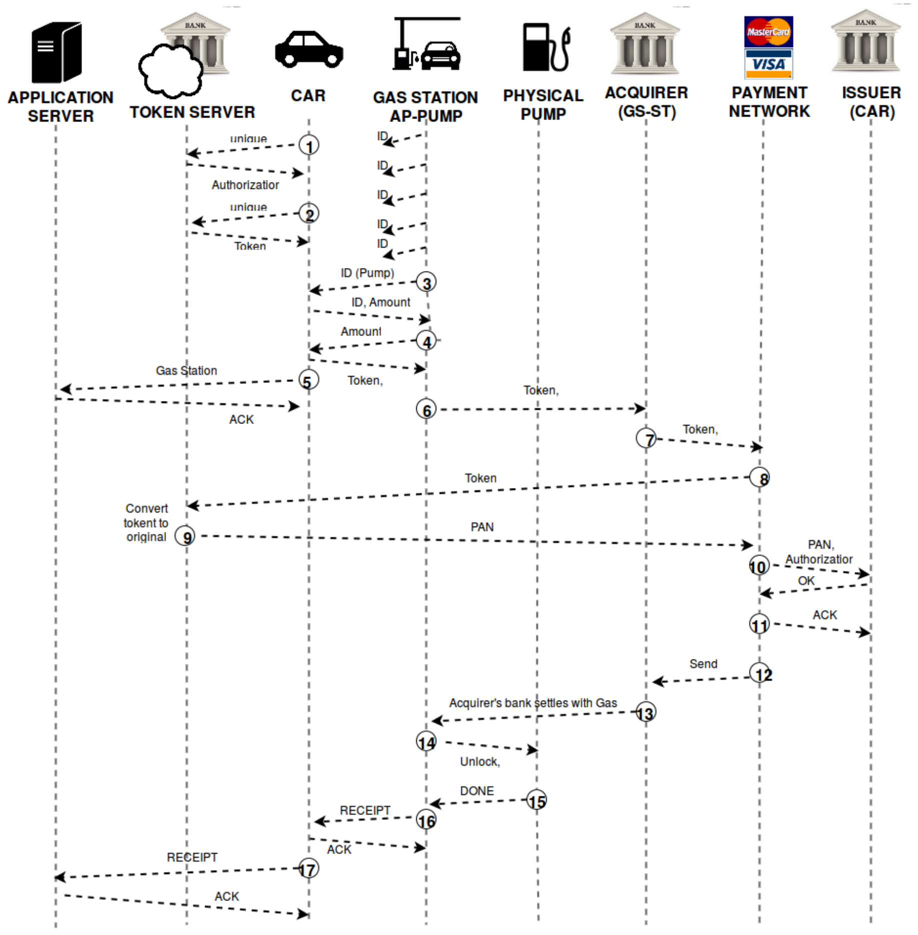

Figure 5.2: System's protocol with the unique ID usage.

prisingly, could require the biggest effort during the implementation phase. This time we cannot use APIs provided by someone else, but instead we have to implement our own protocol and take adequate precautions for the communication. The goal of an attacker who performs a MITM in this channel could be having fuel without paying, or just paying less than what he is suppose to pay. The channel could also be used by an attacker to have a more direct way of hacking the gas station servers, in fact up to now we have never discussed about using a firewall between the pumps and the gas station servers.

The first solution to the problem is of course adding a firewall in the between of the mentioned parties. Furthermore, there is no need to have a wireless communication in here, therefore a cabled one in this case can be more secure if the cables are kept in a secure place not accessible from the outside. After this a secure protocol of communication is also needed, such as TLS 1.2.

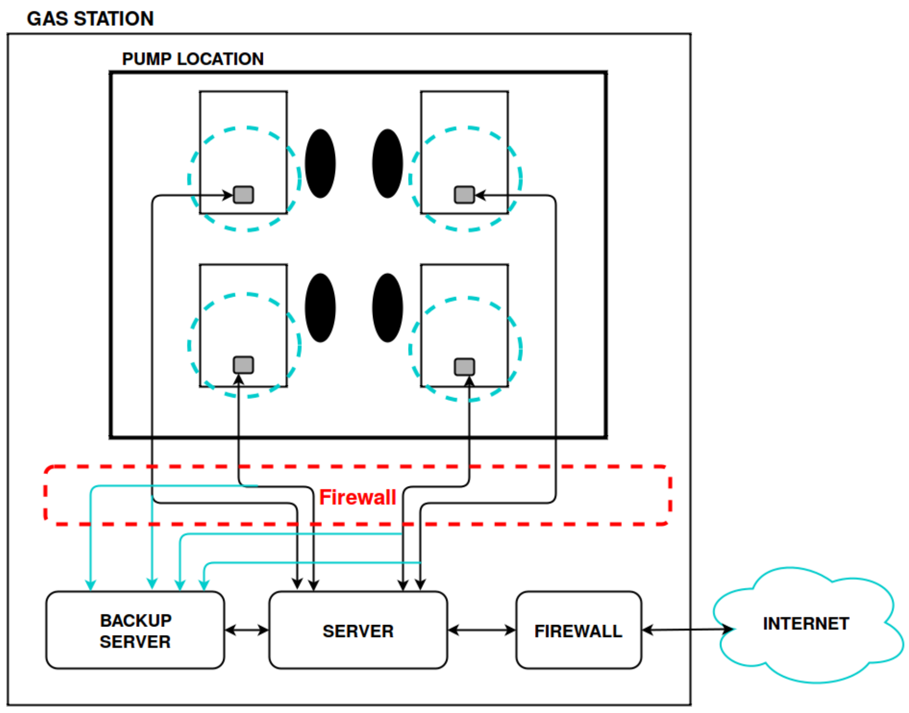

Figure 5.3: Overview of the gas station part of the system.

Between the car and the application server. This communication is used by the satellite navigator on the car to store on the cloud all the activities performed with the car: the amount of fuel payed, the cheapest and nearest gas station, the last time the car's oil was changed and so on. In this case there isn't money exchanged between the parties, but still many personal and sensible user's data are stored on the cloud and is important to protect them with encryption and authentication so, again, we thought that TLS 1.2 fits well the problem, but we didn't go into details.

Between the application server and the database server. The communication between the application server and the database server and in turn the Master DB server and the slave ones (since we decided to use the Master-Slave Database Replication technique) is a pretty weak point, particularly it increases with the scaling of the system. To protect the queries shared we need again of a strong encryption algorithm and a a reliable authentication method. Since the database servers are going to be more than one, and new servers can also be added during time, we thought that a two-way-authentication is fundamental. In our vision the Master DB server needs to know all the CAs of the slave DB servers in advance and the slaves in turn only need to know the Master's one. Again we thought TLS 1.2 can be a good protocol to be used, but we didn't investigate the problem.

5.2.4 Replay attacks

A replay attack could occur in our system when exchanging sensitive information. All communications containing tokens are vulnerable – i.e. when the tokens are first sent to the car, when a token is sent from the car to the gas station's server for payment, and when it is passed along by all the parties involved until it reaches the token server. However, because of how our token system is designed, a malicious third party that gets hold of a token is not able to reuse it, since each token can only be used for one payment.

On the other hand, if the malicious attacker manages to intercept the packets carrying the token, drops them and collect the information in order to perform another payment by himself the problem could persist, but it doesn't. First of all, packets are carried via TLS as stated before hence an hacker must be able to decrypt the information and this isn't an easy job. Secondly, even if the attacker manages to decrypt the information and gets the token plus the unique ID of the victim, he has only a little time to perform the payment since we designed our tokens to expire in a short amount of time. In the worst scenario the hacker can steal the sensible information and perform the payment, but still the maximum amount of money he can use is limited by a threshold that is an intrinsic feature of the tokenization payments and can't be tricked. Moreover, stealing a token is an end in itself, in fact having a token doesn't mean having the PAN and it is exactly this the advantage of the tokenization mechanism.

5.2.5 Tamper resistance of the gas station's server

Although there are no provably secure software anti-tampering methods, effective anti-tamper protection can be implemented, both internally and externally. External anti-tampering methods usually monitor the software to detect tampering, like malware scanners or anti-virus software. Internal anti-tampering is implemented as specific code within a software that detects tampering as it happens – examples of this are runtime integrity checks, encryption or obfuscation to prevent tampering and reverse engineering. An alternative way of dealing with tampering is to build tamper-tolerant software. There are also generic software packages that make programs tamper-resistant, but these can be contrasted with semi-generic attacking tools.

5.2.6 Multi-user protection

Authentication of a user in the car in order to gain access to a specific credit card, can be performed in a number of different ways. Of the three factors that are usually considered in a authentication system – something you know, something you have, something you are – we choose the former, which is the simplest and most common one.

A classic PIN code can be inserted using a specific device on the driver's side, and for this kind of application this can be considered secure enough. A biometric system such as a fingerprint reader could be an alternative, being more expensive and complex but also more secure and user-friendly. A further step could be to integrate the two methods above, getting a two-factor authentication with "something you know" and "something you are". This would however make authentication more complicated and time-consuming for the user, and we believe this level of security is not necessary in this scenario.

Even though this interaction with the user represents an additional source of vulnerability, where even a simple car passenger could become a threat, it is necessary if we want to allow, as stated in the previous sections, data for multiple credit cards stored in a single car.

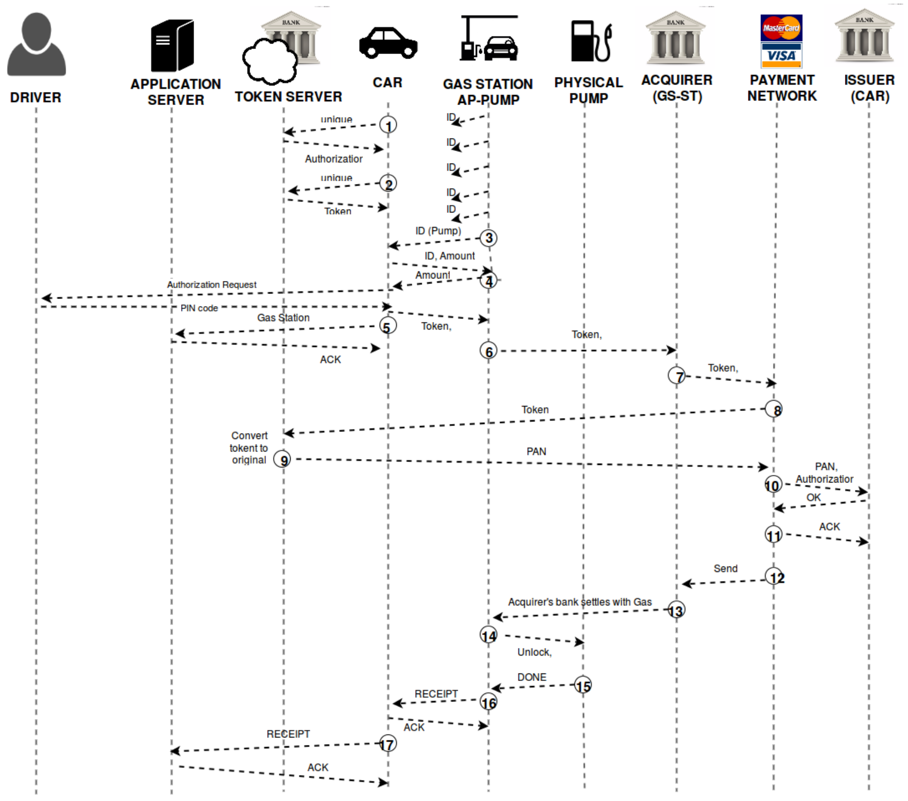

Figure 5.4: System's protocol with the user's interaction.

5.2.7 Gas station's server and backup server

The first line of defense for these two servers are the two firewalls that can be seen in the previous diagram and that separate them from the outside world. Naturally this is not enough and other countermeasures as digital certificates, security communication protocols, anti-injection mechanism, anti ransomware and Role-Based-Access-Control (RBAC) are needed.
There are plenty of information on literature and on the Web on how to protect servers in different contexts and even more different solutions so we are not going to treat this problem in this report. Must also be said that, nowadays, when you buy a server you are

almost always provided with a Security Manager with it hence protecting a server has become much more easier now. Refer to the consultation of CISCO Security Manager for more details [1].

5.2.8 DDoS attacks

DDoS attacks are meant to disrupt normal service. An attacker could perform this kind of attack in order to target the gas station, or to disrupt operations when paying. Since there can be only a few devices connected to each access point, the traffic to an access point or to the server cannot be very large from the gas station side. On the other hand, on the Internet side, a DDoS attack could be more substantial.

DDoS to the server through the access points. By limiting the DHCP range of each access point to a few IPs, we basically remove a source of DDoS attack, and it also makes things easier when the access point has multiple car signals and has to establish a connection with one of them.

DDoS to the car that is refueling. We think that an attacker could be interested to disrupt the communication between the car and a pump. By placing a transmitter in proximity of the pump a DDos to the car could be performed. To avoid this the satellite navigator must have an efficient firewall that filter the channel. We won't go into details of this.

DDoS to the server through the Internet A DDos performed by the Internet to the gas station is probably the most dangerous since the attacker can make use of great resources as Bots and BotNets. Many entities could be interested in make the gas station system goes down, first of all its competitors. As we already discussed the solution proposed is to use a good firewall and great policies on it to block such attacks. Figure 5.3 shows where the firewall is placed.

5.2.9 Application servers and database servers

To protect the Application server and the DB servers the same consideration done for the gas station's server and the backup one are still valid. It must be considered that the communication between Master DB server and its slaves are almost all SQL queries hence extremely sophisticated anti-SQL-injection functions must be implemented on the filters that filter the traffic between the DB servers.

6 Other properties

In this chapter we will address other properties that characterize a distributed system. We haven't analysed them in depth because we think they do not represent major aspects of our system. We will here go rapidly through them, pointing out the issues that they could present.

6.1 Openness

This characteristic only applies in our system if the information stored in the database is available for other apps and services. This may be a possibility, but it is not the main objective and therefore it has not been addressed.

Concerning the possibility of future extensions of the system, our setting does not preclude such a condition, as long as the assumptions on which it bases are not contradicted.

6.2 Concurrency

Several clients may want to access a shared resource at the same time and this should be considered when implementing the system. However, we believe that the other three properties assessed in this project (failure handling, scalability and security) are more crucial and hence they have been addressed.

6.3 Transparency

Even if the **transparency** property was not addressed specifically, the system's design fulfill many of the transparencies that can be considered:

- **Access:** to hide how a resource is accessed and the different data representation.
- **Location:** the system works without knowledge of the physical or network location of the resources.
- **Relocation:** in case a resource is moved to another location
- **Failure:** the system hides the failure and recovery of a resource.
- **Scaling:** the expansion of the system does not affect its structure or the application algorithms.

6.4 Heterogeneity

The heterogeneity challenge is not a problem since we have designed how the information is exchanged by using standard communication protocols. Thus, the system can operate irrespective of the operation system or the hardware that is being used.

7 Conclusion

In this report we outlined and analyzed the design of a system for paying for fuel from the car at a gas station. Our system consists of a satellite navigator with Internet connection, that communicates with the gas station through access points placed at each pump, a gas station server managing fuel requests and payments, and a cloud service for storage of user data. The payment is secured by the use of HCE (Appendix B) and tokenization (Appendix A).

After describing the system and the main assumptions underlying its design and analysis, we worked towards the goal of making the system fault-tolerant by fixing the design of individual components and the communication between them as necessary. Then we outlined common problems and design paradigms to make systems scalable, and addressed the scalability issues in particular regarding the application for users. Regarding security, we spotted and discussed the vulnerabilities of the system, and informally proposed solutions in order to make the system secure. We also briefly mentioned other issues that were not addressed in this report, and how they relate to our system.

Distributed systems is a wide subject and our approach was to divide the tasks and work on them thoroughly, identifying the problems that may affect the system and trying to provide solutions tailored to them. We believe we have addressed the main issues, and that our final design is a sufficiently robust distributed system that could face and withstand a big part of the possible malicious or accidental threats. This project has been useful to increase our knowledge of the different properties described herein as well as the process to design a distributed system.

Appendix A Tokenization

At its root, a token is something with low value representing something with high value; just as a casino chip represents cash. In electronic payments, tokenization is being used to reduce security risks inherent in the collection and transfer of highly sensitive data such as credit card Personal Account Numbers (PAN). In the mobile payments world, it is the EMVCo Payment Tokenization Specification that sets the framework for tokenizing contactless payments.

In the EMVCo specification, the PAN is replaced it with another PAN-like number by the tokenization system. In a typical scheme, the last four digits of the PAN are not tokenized for continuity so banks and merchants can identify customers for actions such as returns and loyalty programs. It is the tokenized pseudo-PAN that is sent from the POS to the issuer for transaction authorization. The issuer uses the tokenization system to de-tokenize the pseudo-PAN and match it to the real account data. The pseudo-PAN cannot be reversed or de-tokenized by any other entity other than the trusted tokenization system. Tokens can be used with all Cardholder Verification Methods (CVM) but are limited in scope, for example to a single merchant, and duration by an expiry value. Tokens are required to have their own BIN ranges that identify the pseudo-PAN as a token to the payment system and can help identify contactless cloud transactions. In a pseudo-PAN tokenization scheme, PCI-DSS risk on merchants is minimized because the tokens have limited use and the PAN data cannot be reverse engineered. Merchants can also use tokenization to secure their private label card data. For payment tokens, issuers will perform Identification and Verification (ID&V) steps to make sure the token is mapped to a valid and authorized PAN. Tokenization specifications are multi-part solutions that also include risk management measures based on endpoint "fingerprints," account activity, and the depth of identification and verification processing on the issuer side.

Appendix B
HCE

What does tokenization have to do with Host Card Emulation (HCE)?

Host Card Emulation, placing card data in the cloud, is the hot topic now that Visa and MasterCard are putting working together on HCE specifications. To avoid all the technical and business complexities of card credentials stored on devices in secure elements, financial institutions are looking to move card credential data to the cloud. With Android's support for Host Card Emulation in the KitKat OS, cards in the cloud are no longer pie in the sky. Moving sensitive card and personal data out of a phone's secure element into the cloud solves business problems by reducing the number of actors and barriers to integration to existing systems. However, passing encrypted card data every time a user wants to transact is simply not feasible from both a security and user experience point of view. To enable secure cloud-based mobile payments, HCE uses multiple techniques to ensure the security of sensitive information without burdening the user experience. Tokenization has been one of the main security measures being considered to make HCE cloud-based mobile payment transactions secure. Others include limited use keys, account replenishment, and risk assessment scores.

Bibliography

[1] Cisco security manager 4.10. http://www.cisco.com/c/en/us/td/docs/security/security_management/cisco_security_manager/security_manager/410/installation/guide/IG/inserver.html.

[2] Host card emulation (hce) 101. Master's thesis.

[3] Pci p2pes. https://www.pcisecuritystandards.org/pci_security/.

[4] Reflections on visa's high-availability payment network. http://abhishek-tiwari.com/post/reflections-on-visa-s-high-availability-payment-processing-infrastructure.

[5] Taking android out of the tcb. https://randomoracle.wordpress.com/2014/04/21/hce-vs-embedded-secure-element-taking-android-out-of-tcb-part-iv/.

[6] Visa p2pes. https://developer.visa.com/products/vts/reference#vts.

[7] Benjamin Erb. Concurrent programming for scalable web architectures. Master's thesis, Ulm University.

[8] Chris Ford et. al. Patterns for performance and operability: Building and testing enterprise software.