Utilities: EcoGes

Network and Computer Security

Alameda

Group 41

92475 Henrique Cavaco
92513 Mafalda Ferreira
105458 João Moreira

Master's Degree in Computer Science and Engineering

Instituto Superior Técnico

2022/2023

1. Business Context

This project aims to develop a new system for the electricity provider, EcoGes, which allows clients to monitor the energy cost of household appliances and the energy production of solar panels. The system is also used to calculate the monthly invoices for consumed energy and taxes and manage the energy plan (flat or bi-hourly rate).

2. Infrastructure Overview

2.1. Design

The EcoGes application concerns a six-tier system:

- A **public website** for clients to access and update their information.
- A **client** that accesses the website to view and update personal information, add new appliances and solar panels, and access energy consumption, production and invoices.
- A **backoffice** where employees can manage the system according to their role in the company: account manager or energy manager.
- A **Role-Based Access Control** entity (**RBAC**) that receives requests from the **backoffice** regarding the employee's access to the client's data according to their role.
- An **admin machine** for employees to access the **backoffice**.
- A **database** to store persistent information regarding the data of clients and employees.

2.2. Technologies

The system was developed in Java using Maven as a build automation tool. The communication between servers and clients was implemented using gRPC, which supports high-performance remote procedure calls. Protobuf was used to serialize structured data in cross-platforms, and the system was tested using JUnit framework. The database server is MySQL.

2.3. Machines and Network Architecture

The system is deployed using a **DMZ** topology with a **single firewall**, providing an additional protection layer between the internal and external networks. The **webserver** is placed in a **DMZ**, and the **firewall** acts as a reverse proxy that intercepts requests from the external network and redirects them to the public website. Additionally, the **firewall** is also used to filter the requests exchanged between the two internal networks and the DMZ, as depicted in **Figure 1**. For simplicity, we placed the **backoffice** and the **RBAC** entity in the same machine (**internal machine**). Due to a limitation on the Virtual Box, we could not add more than four interfaces to the **firewall machine**. Thus, we were required to place the **internal** and **admin machines** within the same subnet. Since messages exchanged between these entities do not trespass through the **main firewall machine**, we configured an additional **firewall** in the **internal machine**.

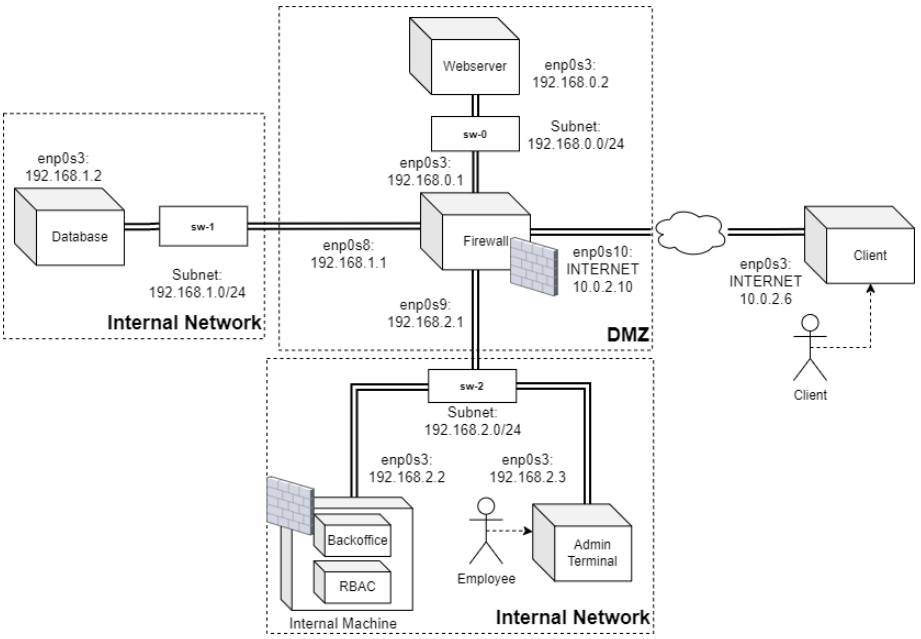

Figure 1. Machines and network architecture.

The system is split into the following:

- VM1: An **external client** machine that can open connections to port 8000 of the **webserver** (VM3).
- VM2: A **firewall** that adopts a white-listing approach, denying all requests by default:
  - Accepts all previously established connections.
  - Redirects all connections from the **external network** (VM1) with a source port higher than 1023 and destination port 8000 to the **public website** (VM3).
  - Accepts new requests from the **webserver** (VM3) and **backoffice** (VM4) with a source port higher than 1023 and if destined to port 3306 of the **database** (VM6).
  - Accepts new requests from the **backoffice** (VM4) with a source port higher than 1023 and if destined to port 8000 of the **webserver** (VM3)
  - All other traffic is rejected.
- VM3: A public **webserver machine** that receives connections from the **external network** (VM1) and **internal machines** (VM5) to port 8000 and opens connections to port 3306 of the **database** server (VM6).
- VM4: An **internal machine** that receives connections from the **admin machine** (VM5) to port 8001 of the **backoffice** and opens connections from the **backoffice** to port 3306 of the **database** server (VM6). Both **backoffice** and **RBAC** communicate inside the same machine. The machine has a local firewall configuration:
  - Accepts all previously established connections.
  - Accepts new requests from the **backoffice** with a source port higher than 1023 and if destined to port 3006 of the **database** server (VM6).
  - Accepts new requests from the **backoffice** with a source port higher than 1023 and if destined to port 8000 of the **webserver** (VM3).
  - Accepts new requests from the **admin machine** with a source port higher than 1023 and if destined to port 8001 of the **backoffice**.
  - Accepts new requests from the **backoffice** on localhost with a source port higher than 1023 and if destined to port 8002 of the **RBAC** on localhost.
  - All other traffic is rejected

- VM5: An **admin machine** that opens connections to port 8001 of the **backoffice** (VM4).
- VM6: A **database server** machine that receives connections to port 3306 from the internal **backoffice** (VM4) and the **public website** (VM3).

3. Secure Communications

We use TLS as an additional layer to secure all communication channels, providing **confidentiality**, **integrity**, **authentication**, and **non-repudiation**. To test if everything was working as expected, we used Wireshark to capture the *TLS Handshake* and verify that messages were being encrypted. The communication concerns the following entity pairs:

- client - webserver
- admin - backoffice
- webserver - database
- backoffice - database
- backoffice - RBAC
- backoffice - webserver

3.1. Distribution of keys

We have a local **C**ertification **A**uthority (**CA**) with its own private key and certificate, which acts as an authority that signs all generated certificates. **Table 1** depicts how these are stored and distributed, described as follows:

- All servers, i.e., **webserver**, **backoffice**, **database**, and **RBAC**, have their private key and a certificate signed by the **CA**. For each entity, except the **database**, the key and certificate are stored in a Java Keystore with password protection.
- On the **database** side, the server has access to its private key, certificate, and the **CA's** certificate, which are stored on disk. The remaining entities also have access to the **CA's** certificate, stored in the Truststore (also a Java Keystore).
- All the entities that communicate with any server must have access to their certificate: the **admin** and **client** have access to the **backoffice** and **webserver** certificate, respectively, and the **backoffice** has access to the **webserver** and **RBAC** certificates. All these certificates are stored in a Truststore.

The generation of the keys and certificates and the signing of the certificates by the **CA** is accomplished using a script. The private keys should be generated within the machine, and the certificates should be distributed later using, for example, a physical device.

Entity	Keystore	Truststore	On Disk
Webserver	webserver.key, webserver.crt	ca.crt	-
RBAC	rbac.key, rbac.crt	ca.crt	-
Backoffice	backoffice.key, backoffice.crt	ca.crt, webserver.crt, rbac.crt	-
Client	-	ca.crt, webserver.crt	-
Admin	-	ca.crt, backoffice.crt	-
Database	-	-	db.key, db.crt, ca.crt

Table 1. Distribution of keys and certificates.

4. Security Challenge

The challenge concerns two main aspects: (i) If the user account is attacked, the attacker should not be able to access **personal** and **energy data**. (ii) The client's **personal data** is

separated from the **consumption data** using cryptography. **Energy managers** can only access **consumption data**, and **account managers** can only access **personal data**. To solve this, we must satisfy the following requirements where the system must ensure that:

- **R1:** The client data is confidential, and an attacker within its account cannot access it.
- **R2:** The employees can only see client information according to their role/department.
- **R3:** The access to the master secret keys that separate the client's personal data from energy data is confidential, i.e., only the **webserver** can access them.
- **R4:** The temporary secret keys used to decrypt the client's data in the **backoffice** are only shared with authenticated and authorized parties.
- **R5:** The sharing of the temporary secret keys is confidential, i.e., only the **backoffice** department that requested the key can access them.
- **R6:** The shared temporary secret keys can only be used once.
- **R7**: The authenticity and integrity of the ticket that represents the permission granted for a department to access a certain data compartment are assured.

5. Proposed Solution

5.1. Trust Assumptions

The trust relationships are described as the following:

- **Fully Trusted:** the **RBAC** and **backoffice** are trusted due to the security reinforcement of the firewall and the isolation of these machines, i.e., no one should be able to access them within the internal network.
- **Partially Trusted:** the **backoffice** departments (account and energy management) are partially trusted. Although they are isolated, we must verify the department's certificate whenever they require access to the temporary secret keys to decrypt client data. The **database** is partially trusted since we must encrypt every table on disk. However, we fully trust the **database** to securely hold its private key, inaccessible to any other entity. Also, we fully trust the **database** to store the wrapped master and temporary secret keys, which need to be always available. The **admin** machines are considered to be partially trusted. Although they are isolated from the external network, we must assume that there is a possibility of the existence of a malicious employee within the company. The **webserver** is partially trusted since it is placed within a DMZ and, although protected with a firewall, we must consider that the machine can become a target of an external attacker. Despite this, we fully trust the **webserver** to manage the master and temporary secret keys that will be used to encrypt and decrypt the client's data. We assume the server is always available and never fails.
- **Untrusted:** the **client** is untrusted since anyone can represent this entity.

Additionally, we assume the password used in the Java Keystore files to store the private keys is never exposed. This is important for the **webserver**, since its private key is required when managing the secret keys. Also, the private key of the database is never exposed.

5.2. Attacker Model

The attacker is considered to be an entity that:

- Can gain access to the client's account credentials or even remote access to the client's account and may try to impersonate the client.
- Can be an employee with malicious intentions but cannot access the client's data.

- Can be an attacker inside the organization that accesses the employee's machines but cannot gain access to the employee's credentials and cannot access client data.
- Can intercept the communication channels and try to disclose information.
- Cannot acknowledge the password of the Java Keystores used to store private keys
- Cannot access the private key of the database.
- Can access the database machine, access the schema information, and retrieve data from the database tables.
- Can intercept messages in the communication channels.
- Can intercept messages from the backoffice to the webserver to acquire the request that contains the RBAC ticket, and try to re-send or tamper with the ticket information.

5.3. Data Obfuscation

The **confidentiality** of a client's data has a data **obfuscation mechanism** in place to disallow attackers that access their credentials or have access to their machine to be able to see sensitive information. When clients request to read **Personal data or Energy consumption/production** data, they are presented with the whole panel **obfuscated** (data represented by a sequence of **asterisks** and the last three digits of each piece of information so that the owner of the data can be able to recognize it). For the **Personal data**, since these are values that are not updated often, columns in the client's table are created to store these values in an obfuscated manner. For the **Energy consumption/production data**, to pass these values as obfuscated to the user, the **webserver** retrieves the original values from the **database** and only then obfuscates the data to respond to the user's request, this is because Energy values are updated often and depend on a lot of tables' information.

5.4. Data Separation and Access Control

The client data is separated into **compartments** using cryptography. Each existing **backoffice** department, **account management,** and **energy management** can only access **personal information** and **energy compartments** respectively. Each entity involved in the data access flow (**webserver**, **RBAC**, **account** and **energy management**) has a private key and a certificate signed by the local **CA**, stored in a Java Keystore with password protection. Every entity has the **CA** certificate stored in a Truststore. It is important to note that, ideally, it would be better to have a separate CA, Department CA, to sign the department's certificates. Additionally, the **webserver** also has the **RBAC** certificate in its Truststore. Similarly to the secure channel protocol, the keys and certificates are generated using a script. The private keys should be generated within the machine, and the certificates should be distributed later using, for example, a physical device.

Access Control. The access control is managed by the **Role-Based Access Control** entity (**RBAC**) that manages the access permissions of each **backoffice** department for a given data **compartment**. Thus, upon every admin request to the **backoffice**, the latter calls the **RBAC** system to retrieve the authorization response. The **RBAC** generates a *Ticket* that includes the user that issued the request, the role and the data **compartment** to access, the timestamp when the request was issued, and the timestamp until the access is valid. The response includes the *Ticket* and its signature by encrypting the digest with the private key.

Data Separation. Each data **compartment** is encrypted with a distinct **master secret key**, one for each **compartment**. The encryption is accomplished using AES encryption with CBC

mode, and each **database** table contains a unique IV used for the encryption. The **webserver** generates and manages these keys, not stored on the server's disk but in the **database**. Before storing them, they are wrapped using the **webserver's** public key. Hence, they can only be fully accessed by the **webserver** by unwrapping them with its private key after retrieving them from the **database**, ensuring **confidentiality**. The master keys are used to encrypt and decrypt client data upon client read and write requests for a **compartment**. On the other hand, sharing secret keys with the **backoffice** requires a more complex solution by using one-time-use **temporary secret keys**, as explained further. Overall, the data is **confidential** and can only be accessed by entities with access to the secret key.

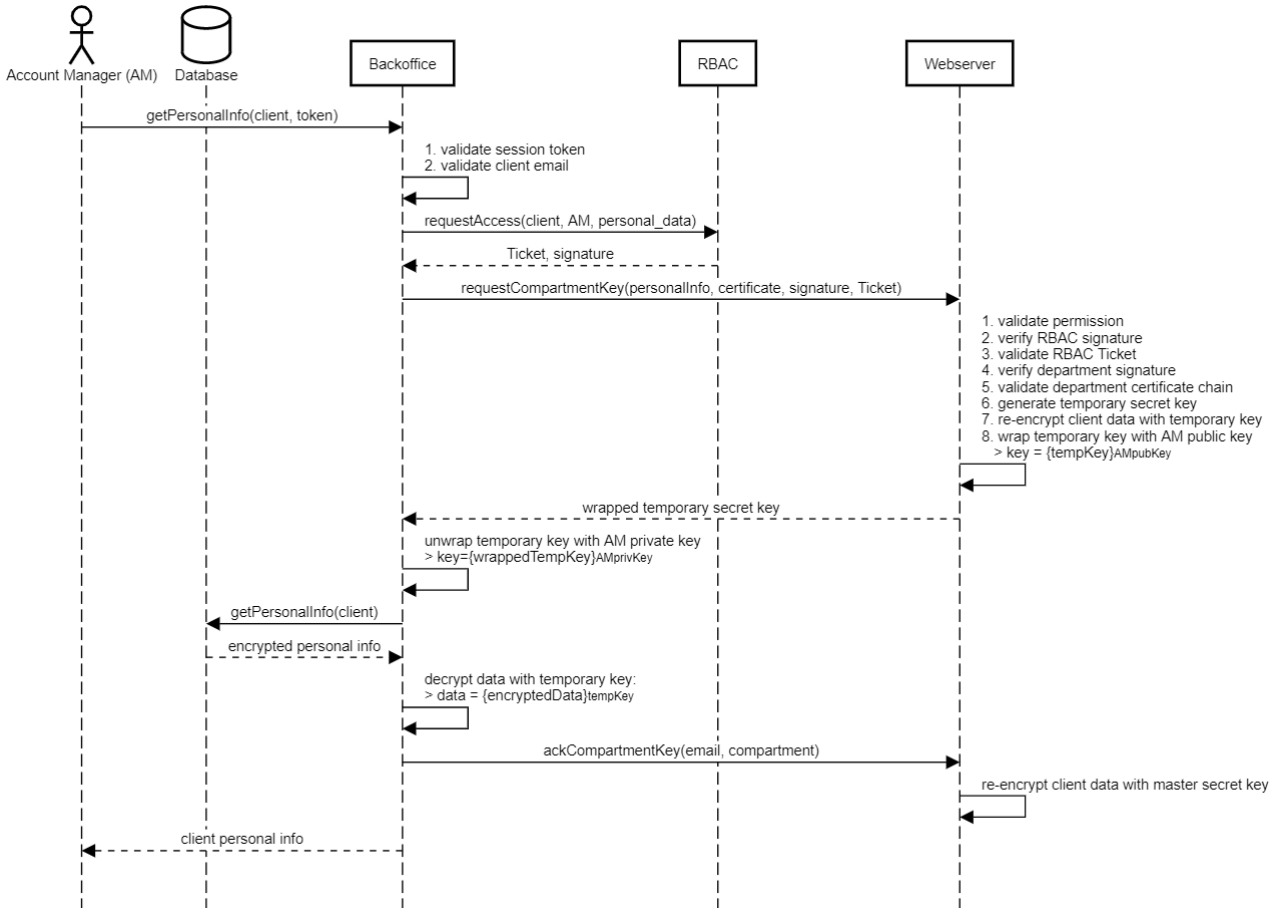

Figure 2. Workflow of internal requests throughout the system.

As depicted in **Figure 2**, when an employee wants to access the client's data, the department sends a `requestAccess` to **RBAC**, and, if access is granted, the **RBAC** responds with a *Ticket* and its signature. Then, the **backoffice** sends a `requestCompartmentKey` request to the **webserver** to retrieve the secret key that allows the decryption of the data. The request contains (i) the department certificate, (ii) the signature of the request, (iii) the **RBAC** ticket, and (iv) the **RBAC** signature of the ticket. The **webserver** (i) verifies the signature of the request, (ii) validates the department's certificate chain using the CA's certificate, (iii) verifies the signature of the ticket using the **RBAC's** public key retrieved from its certificate, and (iv) validates the ticket information. Hence, we ensure **authenticity** and **integrity** by verifying the signatures and the certificate.

If all verifications are successful, the **webserver** generates a **temporary secret key**, re-encrypts the specified data with the key, and sends the key wrapped with the department's public key retrieved from its certificate. Hence, only the department can

unwrap it with its private key, ensuring **confidentiality**. Once the department decrypts the data, it sends an acknowledgment message to the **webserver**, which discards the temporary key and re-encrypts the data again with the master secret key of the specific compartment. While an acknowledge message is not sent, this temporary key is stored in the **database**, wrapped with the **webserver's** public key, to ensure it is not lost and data can still be accessed in case the **backoffice** fails before sending the acknowledgment. Finally, the **backoffice** returns the data to the **employee**. Furthermore, every time data is re-encrypted, a new IV is generated for the corresponding tables, which is used in the AES encryption.

6. Results

Overall, we built a solution as closely as planned. Unfortunately, we were not able to implement full protection for the private key of the **database**, such as encrypting it with a passphare and, as a result, it is stored on disk in plain-text. This is because the key is required to be directly accessible from its location by the MySQL server, in order to enable TLS. However, the access to the directory where the key is stored requires root permissions. This is not ideal but reduces the exposure of the key.

Regarding the network, only the firewall was used to reinforce network security. However, one possible enhancement would be the implementation of a **NIDS** to monitor suspicious activity or policy violations.

Regarding the security challenge, we did not implement more complex authentication mechanisms besides the user's password challenge due to the project's complexity and since it was not in the scope of our challenge. Nevertheless, we believe it would be ideal to reinforce the security by implementing additional authentication mechanisms, such as Two-Factor Authentication (**2FA**), using a mobile device application to authenticate. Clients could use this as a way to retrieve *deobfuscated* data if desired. Secondly, we determined that the **webserver** managed the secret keys. However, the server's availability cannot be guaranteed due to the risk of external attacks. Therefore, we believe it would be better to have a separate, trusted entity point to handle these keys for added security.

Finally, we use certificates to implement secure channels and solve the security challenge. However, we did not consider certificate revocation due to the lack of time and since it was not required. Yet, this could be achieved using a **C**ertificate **R**evocation **L**ist (**CLR**) that would be shared and updated among the servers.

On a final note, to improve availability and protect against data loss, it would be ideal to set up replicas of the **webserver**, **backoffice**, and **databases**. This was not considered due to already having a large number of machines.

6.1. Requirements

For each requirement, we can conclude the following:

- **R1:** Satisfied. The client's personal and energy information is obfuscated when sent to their device, preventing an attacker's ability to see it if they can access the account.
- **R2:** Satisfied. Before accessing the client information, the backoffice needs to contact the RBAC to request permission for the current action. Access will be denied if an employee role does not contain permission to access a certain compartment.
- **R3:** Satisfied. The webserver stores the master secret keys in the database by wrapping them with its public key, and only the webserver can unwrap them with the private key.

- **R4:** Satisfied. The temporary secret keys are only shared with departments that present a valid RBAC ticket and a valid certificate chain, signed by the CA.
- **R5:** Satisfied. The webserver wraps the temporary secret keys with the department's public key so only the department can unwrap them with its private key.
- **R6:** Satisfied. The webserver discards the temporary secret keys after they are used by a certain department and re-encrypts the accessed data with the master secret key.
- **R7:** Satisfied. The webserver verifies the ticket's information and signature with the RBAC public key retrieved from the certificate stored in the Truststore to ensure that the ticket was not tampered with and comes from the legitimate RBAC.

6.2. Main Implementation Choices

For the data compartment encryption, we choose symmetric over asymmetric keys since they are shorter and the operations are less time-consuming. Due to the reduced size, we could wrap them with the public keys without losing information. Each table contains an IV used in AES encryption with CBC mode. This is fundamental since using distinct IVs ensures that identical plaintext blocks will not result in identical ciphertext blocks making data more resistant to known-plain-text attacks [2]. Additionally, with CBC, decryption can be parallelized [2].

To simplify the process of retrieving the secret keys, it was best for the **webserver** to manage them. However, when sharing secret keys with the **backoffice**, we determined that generating temporary, one-time-use keys would be more secure. Otherwise, if the master key was shared, an attacker could later access every client's data related to a particular compartment, which is a risk we want to avoid even if it requires such a complex solution. Also, to request the temporary keys, it was necessary to include the signature and the department's certificate in the request. This was done to validate its entity and prevent **impersonation**, ensuring the request came from a legitimate department.

In designing the **RBAC**, the signature of the **RBAC** response played a crucial role in ensuring the authenticity of the granted access, making sure no one except our **RBAC** could provide valid access. At the same time, timestamps included in the ticket were used to ensure that the same ticket could only be used within time constraints.

6.3. Additional security mechanisms

Regarding the user's passwords, the system only accepts **strong passwords** (minimum 10 letters, 1 upper case, 1 lower case, and 1 special character). For storage, the servers store the **digest** of the **password** along with a unique **salt** per client. This ensures that passwords are not stored in plain text and reinforces security against **rainbow attacks** [1].

Lastly, to secure the system against common web attacks, we used **prepared statements** resilient against **SQL injection**, where code and data are processed separately.

7. Conclusion

While it may not be possible to achieve complete security, we believe our solution effectively addresses the main security concerns of protecting access to client data and compartmentalizing it through the use of encryption. For future work, two possible enhancements would be implementing **2FA** as an additional authentication layer and designing a separate entity to handle the compartment keys.

8. References

[1] Thom, S. (2021, January 31). Rainbow Tables & Why To Add Salt [Webpage]. Retrieved from https://dev.to/salothom/rainbow-tables-why-to-add-salt-45l9.
[2] Ubiq Security. (n.d.). ECB vs. CBC: Block Cipher Mode Differences [Webpage]. Retrieved from https://www.ubiqsecurity.com/ecb-vs-cbc-block-cipher-mode-differences/