
UNIVERSITY OF TRENTO

Department of Information Engineering and Computer Science

Advanced Programming of Cryptographic Methods

# Esse3-Bootleg

## Project report

Riccardo Gennaro
riccardo.gennaro@studenti.unitn.it

Giacomo Checchini
giacomo.cecchini@studenti.unitn.it

Mattia Narducci
mattia.narducci@studenti.unitn.it

December 22, 2023

# Contents

# 1 Description

The goal of this project consists of the development of a JavaEE web app that can be used by a teacher to assign grades to his or her students. The project also aims at deploying such web applications on a dockerized infrastructure that will provide both data-at-rest and data-in-transit security.

# 2 Requirements

## 2.1 Functional Requirements

Following, the functional requirements are divided per user story and listed.

### 2.1.1 Login

**User story**: as an unauthenticated user, I want to be able to log in.
**Requirements**:

- The system must allow users to log into their account by entering their `username` and `password`.
- The system must notify users when the credentials are not correct.
- Upon login, the system must redirect users to the dashboard assigned to their `role`.

### 2.1.2 Student Dashboard

**User story**: as an authenticated user with a student role, I want to be able to see my dashboard.
**Requirements**:

- The system must allow students to see their `name`, `surname`, and `matriculation` number.
- The system must also allow students to see the courses in which they are `enrolled` and their assigned `grades`.

### 2.1.3 Teacher Dashboard

**User story**: as an authenticated user with a teacher role, I want to be able to see my dashboard and assign `grades` to students.
**Requirements**:

- The system must allow teachers to see their `name`, `surname`, and `id` number.
- The system must allow teachers to select their private key from their file system.
- The system must allow teachers to insert the student matriculation and assigned grades through a form.
- The system must notify the teacher whether the grade assignment was successful or not.

### 2.1.4 Teacher Dashboard

**User story**: as an authenticated user with an admin role, I want to be able to see my dashboard and add students.
**Requirements**:

- The system must allow admins to input username, password, first name, and last name to add a new student.

## 2.2 Security Requirements

**Communication requirements**:

- communication between the `application server` and the `client` must be protected by TLSv1.3 and use one of the following cipher suites for data encryption
  - `TLS_AES_256_GCM_SHA384`
  - `TLS_CHACHA20_POLY1305_SHA256`
  - `TLS_AES_128_GCM_SHA256`
- communication between the `application` server and the `database` must be protected by TLSv1.3 and use one of the following cipher suites for data encryption
  - `TLS_AES_256_GCM_SHA384`
  - `TLS_CHACHA20_POLY1305_SHA256`
  - `TLS_AES_128_GCM_SHA256`
- The only TLS version to be accepted during negotiation in every TLS-protected communication is version 1.3.

**Authentication requirements**:

- client-server authentication is ensured by TLSv1.3 using Curve25519. The server certificate is self-signed. For this reason, even if the server authenticates to the client, the browser will raise a warning.
- the database certificate is self-signed. The database is authenticated by the application server.
- the required password length in user credentials is greater than 12.
- The passwords must be hashed and salted using Bcrypt with a work factor of 13.
- the operation of grade assignation performed by the teacher role must be signed using `ECDSA` with standard NIST curve `P-256`, and the inputted challenge must be hashed using `SHA-256`.
- The input challenge is composed of the JSON message and a timestamp used by the server to verify at what time the challenge was signed.
- access to the web application is granted by SFA login.

**Authorization requirements**:

- unauthenticated users can access only the login service.
- authenticated users with role student can access only servlets StudentServlet and LoginServlet.
- authenticated users with role teacher can access only servlets TeacherServlet, AssignGradeServlet, and LoginServlet.
- authenticated users with role admin can access only servlets AdminServlet, AddStudentServlet, and LoginServlet.
- access control is managed by Java filters, and the filtering condition is based on the authentication status and assigned role.

**Session Management requirements**:

- the client session has a set timeout of 5 minutes.
- browser script must not access cookies.

- cookies must be sent over HTTPS connections only.
- the session ID (JSESSIONID) must be transmitted in the TLS handshake and not in the cookies or the URL.

**Stored cryptography requirements**:

- data-at-rest in the database must be encrypted using AES-128 in CTR (counter) mode.
- keys POSIX file permissions must be set to 600 and ownership should be assigned to the database admin user.
- database key-management system is to be used.

# 3 Technical Details

## 3.1 Software Architecture

To develop this project, the MVC design pattern with some parts of the JavaEE (or J2EE) design pattern was adopted. This design pattern is composed of different parts that put together provide a modular architecture that encourages code reutilization. Following, the multiple parts of this architecture are explained.

### 3.1.1 Controller Layer

The controller layer is the layer where the business logic is implemented. In particular, this layer communicates with the model layer to request data writing and reading, and with the view layer to render the front-end interface. The client manipulates the web application via this layer. The business logic and view manipulation are implemented inside the servlets, while the access to the model layer is mediated by the managers.

### 3.1.2 Model Layer

The model layer is the layer that communicates with the database. It establishes the connection and launches queries against the data source. The communication with the database is managed by the enterprise beans, the tables are represented by the entities, while the data are communicated to the controller layer through the Data Transfer Objects (DTOs) that are assembled starting from the entities by a DTO factory.

### 3.1.3 View Layer

The view layer uses the data provided by the controller layer to render the web view. In particular, the web page is served thanks to the Java Server Pages (JSPs), which are HTML pages with Java code injected in them. JSPs are compiled into servlets during compilation.

## 3.2 Implementation

Following we describe the technological stack, the dependencies, and the used libraries.

### Technological stack

To develop this web application, an open-source tech stack was used. Below, we list the used technologies during development

- `Java 17.0.9 with Java Enterprise Edition v.9.1`: Java enterprise extends the set of Java API to enable the creation of large-scale, enterprise-level applications.
- `Javascript on Brave Browser 120.1.61.104`: Javascript was used to perform client side operations. In particular, the Web Crypto API was used to sign the JSON used for the grade assignments.
- `Wildfly v.30.0.0.Final`: an application server that supports and implements JavaEE API such as Java Transaction API (JTA), Java Persistence API (JPA), JavaServer Pages API (JSP), and Enterprise Java Beans (EJB).

- `MariaDB v.11.2.2`: MariaDB is a popular open-source database that is a fork of MySQL. Firstly we considered using PostgreSQL since we already had some experience in its setup. Unfortunately, this database didn't natively implement Transparent Data Encryption (TDE) yet. For this reason, we decided to migrate to MariaDB which already had a native implementation of this technology since version 10.1.
- `Docker v.24.0.7 and docker-compose v.2.23.3`: this architecture is deployed by a docker-compose file. In particular, Wildfly is pulled from `quay.io/wildfly/wildfly:30.0.0.Final-jdk17`, while MariaDB is pulled from `mariadb:latest`. To ease development and testing the network will be delivered in host mode.
- `Apache Maven 4.0.0-alpha-8`: to build the Java project and to manage dependencies, Maven was used.

### Dependency and libraries

The Java codebase presents multiple dependencies. Firstly, we imported the `jakarta.jakartaee-api v.9.1.0` and `jakarta.jakartaee-web-api v.9.1.0` to be able to use the Java enterprise API. To log some operations, `jboss-logging v.3.5.3.Final` was imported alongside to `hibernate-core v.6.3.1.Final` to manipulate the database via an Object Relational Mapping (ORM) framework. Furthermore, to serialize and deserialize JSON messages, Google `gson v.2.10.1` was used.
To use Bcrypt to hash and salt the user's passwords, `spring-security-crypto v.6.2.0` was imported. Finally, as the last dependency, `bcprov-jdk18on v.1.77` from `org.bouncycastle` was used to verify the signature that the client computes when requesting a grade assignation operation.
Regarding the Javascript script for computing the signature, Web Crypto API was used.

## 3.3 Code structure & configuration

Describing every file in the project would not be manageable given the high number of Java classes, configuration files, JSP, and XML files. For this reason, for the Java files, we will limit our discussion to the package level, making exceptions for the files that contain code related to security. Note that details regarding the methods and algorithms used can be found in chapter 4. Furthermore, a discussion on the implementation of the signature made with Javascript will be carried out. Finally, to describe where we implemented TLS and TDE, configuration files will be discussed.

### The web application codebase

The codebase is divided into the following packages and directories:

- `it.unitn.disi.advprog.gennaro.adv_prog_project.beans`: in this package, the enterprise beans are defined. These beans manipulate the database using Hibernate Query Language (HQL). In this package, method `it.unitn.disi.advprog.gennaro.adv_prog_project.beans .UserAccountBean.getUserAccountByCredentials(Java.lang.String, Java.lang.String)` is used to retrieve user details based on its credentials. Since passwords are hashed and salted with Bcrypt, the user entity is returned only if the check between the provided password and the stored hash is successful.
- `it.unitn.disi.advprog.gennaro.adv_prog_project.entities`: here the Object Relational Mapping (ORM) is implemented. This mapping is done via the Java Persistence API by specifying the POJO to be an @Entity. Furthermore, the table column names and other characteristics are mapped thanks to various Java annotations, like relation type, if the field is or is not a key, etc.
- `it.unitn.disi.advprog.gennaro.adv_prog_project.dto` and `.dtoAssembler`: this is just an abstraction layer implemented to avoid the usage of the entities as a way to carry around information. Entities are complex objects, and since we don't want to instantiate those objects

or serialize and send them every time we need the result of a query, we use Data Transfer Objects (DTO). These DTOs are serialized POJOs with getter, setters, equals, hashCode, and toString methods. Servlets and JSPs access data via these objects.

- `it.unitn.disi.advprog.gennaro.adv_prog_project.managers`: managers are simple stateless beans that decouple the access of the enterprise beans used for communication with the DB from the business logic in the servlets.
- `it.unitn.disi.advprog.gennaro.adv_prog_project.servlet`: in the servlets, we specify the business logic of the application. We have divided this package into multiple packages separated by use case scenarios.
- `it.unitn.disi.advprog.gennaro.adv_prog_project.filters`: filters and filter chains are defined in this package to handle user authorization. Filters are Java classes that are put in front of the target servlet of an HTTP request. For example, to access the services of the AdminServlet the request must satisfy the checks put in place by the AuthFilter, i.e. the user must be logged, and the AdminFilter, i.e. the role of the user must be admin. If these checks are not satisfied, the request is redirected to the LoginServlet. The mapping and chains of these filters can be found in file `src/main/webapp/WEB-INF/web.xml`.
- `src/main/webapp/restricted`: in this directory the JSPs can be found. JSPs are described in subsection 3.1.3.
- `src/main/webapp/js/signature_functions.js`: this is the file in which the Web Crypto API is used to compute the signatures for the grade assignation operation.
- `src/main/webapp/WEB-INF/web.xml`: other than the servlet and filter mapping, at the beginning of this file, rules for the session security configuration can be found.

Now, we analyze the files that we think are more related to the security requirements listed in the previous section.

#### AddStudentServlet and UserAccountBean

AddStudentServlet accepts POST requests for adding users with student roles. In particular, before requesting the entity addition of the new student user account, the password provided by the admin at the moment of registration is hashed and salted using the Bcrypt hashing function following OWASP recommendations at [6]. To check the credentials validity, in class `UserAccountBean` we check that the provided password matches the hash stored in the database.

#### AssignGradeServlet and signature_functions.js

To sign the JSON message crafted by the client using `signature_functions.js` script we produce a challenge string that is a concatenation of the JSON fields. The JSON also includes a timestamp. To provide message integrity, the challenge is signed using Web Crypto API `SubtleCrypto:sign()` method using ECDSA with curve P-256. We could have used the Java Cryptographic Architecture (JCA) to sign the message, but that would have required the development of a client-side Java application, and that would have been too time-consuming.

To generate the keys for the teacher signature we used the `openssl` commands that can be found in file `$PROJECT_HOME/teacher_test_signatures/commands.sh`. The generated keys are in `Privacy-Enhanced Mail (PEM)` format, but this API doesn't accept PEM keys. For this reason, the private key is converted to `PKCS#8` format.

The JSON is sent to the server and the verification is carried out by AddStudentServlet. What took us a while to recognize while trying to make the verification work is that method `SubtleCrypto:sign()` outputs an ECDSA signature in `IEEE P1363` format, while the verification implemented in java accepts only signatures in `ASN.1` format. To address this issue we implemented a simple function to convert the first format into the second.

### Wildfly and MariaDB configuration

The infrastructure is composed of a database and an application server. In this section, we will highlight the pieces of configuration files that enable client-server and server-database TLS, other than MariaDB TDE.

#### TLS configuration

TLSv1.3 for client-server is configured in file `$PROJECT_HOME/services/application_server/configuration/standalone.xml`.Following, the TLSv1.3 setup

```
<tls>
    <key-stores>
        <key-store name="demoKeyStore">
            <credential-reference clear-text="reallyStrongPassword"/>
            <implementation type="JKS"/>
            <file path="certificates/server.keystore" relative-to="jboss
                .server.config.dir"/>
        </key-store>
    </key-stores>
    <key-managers>
        <key-manager name="demoKeyManager" key-store="demoKeyStore">
            <credential-reference clear-text="reallyStrongPassword"/>
        </key-manager>
    </key-managers>
    <server-ssl-contexts>
        <server-ssl-context name="demoSSLContext"
            cipher-suite-names="TLS_AES_256_GCM_SHA384:
TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256"
            protocols="TLSv1.3"
            key-manager="demoKeyManager"
        />
    </server-ssl-contexts>
</tls>
```

The Java KeyStore (JKS), is stored in file `$PROJECT_HOME/services/application_server/configuration/certificates` and the commands used are in file `commands.sh`.

Also, the XML code that enables TLSv1.3 for database-server connection is located in the `standalone.xml` file. In particular, to enable the secure connection, `Java DataBase Connector (JDBC)` connection URL was to be modified to integrate the attributes specifying what certificate Wildfly should verify against MariaDB[4].

```
<datasource jndi-name="java:jboss/datasources/mariadb" pool-name="
    mariadbPool">
    <connection-url>
        jdbc:mariadb://localhost:3306/universityDatabase?sslMode=verify-
            full&amp;serverSslCert=/opt/jboss/wildfly/standalone/
            configuration/certificates/server-cert.pem
    </connection-url>
    <driver>mariadb</driver>
    <security user-name="wildfly" password="wildfly"/>
</datasource>
```

Furthermore, MariaDB needs configuration of its private key, certificate authority certificate, and its certificate. This is done in file `$PROJECT_HOME/services/mariadb/mysql/conf.d/server.cnf` under group `[mariadb]` while TLS certificate and keys for mariadb are in directory `$PROJECT_HOME/services/mariadb/mysql/encryption/tls`.

```
ssl-ca=/etc/mysql/conf.d/encryption/tls/ca.pem
```

```
ssl-cert=/etc/mysql/conf.d/encryption/tls/server-cert.pem
ssl-key=/etc/mysql/conf.d/encryption/tls/server-key.pem

tls_version = TLSv1.3
ssl_cipher =
    TLS_CHACHA20_POLY1305_SHA256
    :TLS_AES_128_GCM_SHA256
    :TLS_AES_256_GCM_SHA384
```

### TDE configuration

Like TLS, MariaDB TDE is enabled in file `$PROJECT_HOME/services/mariadb/mysql/conf.d/server.cnf`, and keys are stored in directory `$PROJECT_HOME/services/mariadb/mysql/encryption`.

```
#File Key Management Plugin
plugin_load_add=file_key_management
file_key_management = ON
file_key_management_encryption_algorithm=aes_ctr
loose_file_key_management_filename = /etc/mysql/conf.d/encryption/
   keyfile.enc
loose_file_key_management_filekey = FILE:/etc/mysql/conf.d/encryption/
   keyfile.key

# InnoDB/XtraDB Encryption Setup
innodb_default_encryption_key_id = 1
innodb_encrypt_tables = ON
innodb_encrypt_log = ON
innodb_encryption_threads = 4

# Aria Encryption Setup
aria_encrypt_tables = ON

# Temp & Log Encryption
encrypt-tmp-disk-tables = 1
encrypt-tmp-files = 1
encrypt_binlog = ON
```

Even if we have specified the encryption of the Aria tables, in our implementation we only use the InnoDB engine. Furthermore, encryption is performed at the tablespace level and not per single table. The used encryption algorithms are `AES_128_CTR` for the data tables, and `AES_128_GCM` for temporary files. This is specified with variable `file_key_management_encryption_algorithm`. Despite the name of the variable, this sets the encryption algorithm for the tablespaces, not the algorithm that MariaDB uses to decrypt the key file[3].

# 4 Security Considerations

In this part of the document, we will discuss about key generation, TLS, Transparent Data Encryption (TDE), password hashing and salting, and signatures.

## 4.1 Key generation

Almost all keys and certificates were produced with OpenSSL. The exception is made for the server that uses Java KeyStores that we produced using Java Keytool.

To generate a Certificate Authority (CA) and a certificate signed by it using OpenSSL for the TLS handshake between Wildfly and MariaDB, we started by running

```
openssl ecparam -genkey -name prime256v1 -out ca-key.pem
```

this command outputs a key pair for our new CA. In particular, the outputted keys are generated by subcommand `ecparam` that ensures that such keys are EC keys. Furthermore, the parameter `prime256v1` specifies that the elliptic curve to use is NIST standard curve P-256. We decided to use this curve not only because it is widely employed, but also because it is approved by guideline `NIST SP 800-186` section 3.1, and represents a good balance between cryptographic strength and efficiency[2].

```
openssl req -new -key ca-key.pem -out ca-csr.pem
```

Here we generate a certificate signing request using subcommand `req`. We are then asked to input the certificate attributes like the organization, the Common Name (CN), etc.

```
openssl x509 -req -in ca-csr.pem -signkey ca-key.pem -out ca-cert.pem
```

We then self-sign the CA certificate by giving as input the Certificate Signing Request (CSR) and the CA private key. We now have the CA certificate.

```
openssl ecparam -genkey -name prime256v1 -out server-key.pem
```

We generate another certificate key pair for out MariaDB server as we did for the CA.

```
openssl req -new -key server-key.pem -out server-csr.pem
```

Here we generate a CSR starting from the new key pair.

```
openssl x509 -req -in server-csr.pem -CA ca-cert.pem -CAkey ca-key.pem
                -CAcreateserial -out server-cert.pem
```

And in the end, we signed the MariaDB certificate using the previously generated CA.
For MariaDB, we generated a CA because the TLS configuration requires a CA certificate to be passed as a variable as per documentation.

About Wildfly, a certificate authority was not necessary. We used the following command to generate a self-signed certificate and EC keys using again curve P-256.

```
keytool -genkeypair -alias localhost -keyalg EC -groupname secp256r1 -sigalg
       SHA256withECDSA -validity 365 -keystore server.keystore -dname
    "cn=localhost,o=Acme,c=IT" -keypass reallyStrongPassword -storepass
                          reallyStrongPassword
```

This command generates a data structure, the JKS, that encapsulates key pairs and certificates. Furthermore, the keystore is protected by a password.

## 4.2 Transport Layer Security (TLS)

Both TLS setups use the default cipher suite for TLSv1.3. The configurations have been tested using `testssl v.3.0.8`, and the results of such tests are in file folder `$PROJECT_HOME/testsslResults`. Neither one of the configurations suffers from the tested vulnerabilities. The only warnings are thrown by the fact that the certificates are self-signed or signed by a self-signed CA.

Furthermore, following the guidelines given to us during the course, the selected cipher suites do not make use of RSA for key exchange (see the 2018 oracle BB attack), and the key pairs were generated using elliptic curves instead of RSA to avoid the risk of low entropy in the generated key.

Also, as shown by the tests, version downgrade is not possible since the only offered version is v.1.3.

Finally, regarding data-in-transit encryption, the communication is encrypted using AES in Galois Counter Mode with key sizes of 128 or 256 bits, or ChaCha20 with a key size of 256 bits.

## 4.3 Transparent Data Encryption (TDE)

Transparent Data Encryption is a technology employed in databases to guarantee data-at-rest encryption with minimal performance impact. The storage of a database is usually divided into logical containers used to organize and store tables, indexes, etc. These containers are called tablespaces, and their specification differs from one RDBMS to another. In our project, we decided to encrypt the database at the tablespace level since we didn't have reasons to encrypt single columns with different policies.

TDE requires multiple keys to work. Every tablespace has a header that contains its tablespace key. A tablespace key is a data-encrypting key that is used to encrypt every piece of data that is contained inside a tablespace. To encrypt the tablespaces key we use a key-encrypting-key called master key. Such master key is strongly recommended to be stored outside the database and preferably inside a vault server.
In MariaDB, in particular, talking about the InnoDB engine, the tablespace keys are encrypted using AES in CBC mode with 128, 192, or 256 bits of key length, while data is encrypted using CTR mode for tables and permanent logs, and GCM for temporary files and logs.

The reason AES is used in CTR mode for tablespace encryption is that this mode allows good performances with random memory access operations since it does not operate on a ciphertext feedback basis.

The reason why we chose to use TDE to secure data-at-rest in our database is that the alternatives that we weighted were not appealing to our situation: the first alternative was to encrypt the whole disk, while the second was to encrypt at the attribute level every single query.

Encrypting the disk has its advantages since if the disk is not mounted in the OS, the data are protected by encryption. Nonetheless, this security mechanism is not useful in the case a malicious actor logs into the system where this given disk is mounted, as all data will be unencrypted, contrary to TDE, where the dataspaces are always encrypted. Furthermore, the delivery of a dockerized MariaDB database with encrypted volumes would have added another layer of complexity to the project.

Encrypting single attributes of a query, and in general, handling the data-at-rest cryptography from the Java application would have added significant computational overhead since for every read/write access to the persistence layer, non-optimized cryptographic operations would have been executed. Furthermore, encrypting only the attributes of the tables of a database leaves a significant amount of metadata (like indexes, headers, and logs) in plaintext.

In the end, we have chosen TDE for the fact that it decouples the cryptography operations from our application code, and it guarantees a good balance between security and computational overhead.

## 4.4 Password hashing & salting

To protect the confidentiality of the user's passwords, we decided to implement password hashing and salting following OWASP recommendations (as seen in subsubsection 3.3). In particular, we decided to use Bcrypt with a cost factor of 13[1]. Password hashing is a security technique used to protect passwords in case of exfiltration or unauthorized access. If a malicious actor tries to access the password of a given user, the only thing it will find is a hash.

Unfortunately, hashing alone does not ensure uniqueness: if a password is equal, also the generated hash will be. An attacker could leverage this to understand that salting is not employed and try to break common credentials using rainbow tables attacks[2], also focusing on repeating hashes.

To solve this problem, we use salt to append a securely random-generated value that is appended to the end of the password before it is hashed. This way, launching a successful rainbow table attack becomes virtually impossible, since the rainbow table should contain also the hashes of randomly salted passwords. In our case, Bcrypt securely handles salt generation, but, in any other case, one should put particular care in selecting a PRNG for salt generation as using a predictable generator would compromise the security of the salting operation.

## 4.5 Signature for grade assignation

We decided to make the grade assignation a signed operation not only as an exercise for integrating Java JCA with Javascript Web Crypto API but also because implementing this ensures a higher level of integrity and accountability for a particularly sensitive operation that is assigning grades to students. As stated before in subsubsection 3.3, the used public key cryptographic algorithm was ECDSA with curve P-256 as this algorithm is considered to be a good trade-off between computation cost and security. For example, ECDSA provides a higher security strength (for a given key length) than RSA but does not add performance overhead. ECDSA P-256 is as performant as RSA 2048 while providing security strength that is comparable to RSA 3072, with 128-bit security strength[5]. We are aware that using Edwards Curves in EdDSA algorithms should be preferred in modern applications given that they are easier to implement. An incorrect implementation could introduce a vulnerability to side-channel attacks, see section 3.0 in [1]. Unfortunately, Javascript Web Crypto API only implements support for `RSASSA-PKCS1-v1_5`, `RSA-PSS`, `ECDSA`, and `HMAC`, and not for algorithms that use Edwards Curves.

[1] The cost factor is a parameter that determines the computational cost of the hash function, making it harder and time-consuming for attackers to perform brute-force or dictionary attacks on hashed passwords.

[2] A rainbow table is a precomputed table containing password hash values for commonly used credentials strings. If an attacker acquires access to a list of password hashes, they can rapidly crack all passwords associated with it.

# 5 Known Limitations

## 5.1 Wildfly admin console with no TLS

Even if it was not a requirement for this project, enabling TLS on the admin console of the application server is fundamental since when an admin does log in, its credentials are sent in cleartext.

## 5.2 MariaDB and Wildfly in development configuration

MariaDB and Wildfly use default users: (root, root)[1] for MariaDB and used for connecting our IDE to the database, and (admin, admin) for Wildfly. An exception is made for the user (wildfly, wildfly) in MariaDB that was created with the requirement of using TLS-protected communications. For this and other reasons in this section, this infrastructure is not to be considered production-ready.

## 5.3 Key Management with No Vault Integration

In the TDE implementation, we opted not to incorporate a dedicated key vault for managing encryption keys. Specifically, we did not utilize specialized plugins such as HashiCorp Vault. The decision to forego these plugins was driven by the complexity involved in seamlessly integrating them into the project and the associated time constraints. Instead, encryption keys were managed by adjusting file permissions and ownership. While this approach may provide a basic level of security, it falls short in comparison to the robust key management solutions offered by dedicated vaults, exposing potential vulnerabilities in key handling.

## 5.4 No Mutual TLS between WildFly and MariaDB

One notable limitation pertains to the lack of a mutual Transport Layer Security (mTLS) connection between the WildFly application server and the MariaDB database. Despite our efforts, we encountered challenges in finding comprehensive documentation on establishing mutual TLS for this specific combination. Numerous attempts were made to resolve the issue, but even if we were able to understand how to configure the JDBC driver in Wildfly, we were not able to find satisfying documentation to enable mTLS in MariaDB with JDBC Connector/J successfully. Consequently, WildFly will not partake in the TLS authentication process. On the other hand, to access the database as user wildfly (that is the dedicated user for wildfly to use while connecting to MariaDB) a pair ¡username, password¿ is still required since anonymous sessions are not permitted.

## 5.5 Security Against Replay Attacks for Digital Signature

In the digital signature feature employed for teacher-assigned grades, a potential vulnerability exists in the form of replay attacks. To mitigate this risk, one proposed enhancement is the implementation of a hash table as a session object. Within this data structure, received messages are stored temporarily and automatically removed from the table after a set period, in our case 10 seconds. The rationale behind this approach lies in replay attacks, where an adversary intercepts and maliciously retransmits a valid message. By utilizing a hash table, each incoming message is hashed and stored along with its timestamp. Subsequent messages with the same content can be cross-referenced against the hash table to check for duplicates. The effectiveness of this mechanism hinges on the assumption that replay attacks typically occur within a short timeframe and this method implements a race condition that

[1] this notation means (username, password)

will make replay attacks harder to execute. The 10-second window strikes a balance between providing adequate protection against replay attacks and allowing legitimate messages to be processed without unnecessary delays. Additionally, the use of a hash table as an in-memory data structure facilitates efficient lookups and minimizes the impact on system performance.

## 5.6 Already Distibuted Keys

The digital signature operation uses key pairs that we consider to be already distributed. This is because we felt that implementing a system for key distribution would have been outside of the scope of this project. Furthermore, it is in our interest to explore the possibility of managing such keys via the use of RFID or other physical tokens for future works.

# 6 Installation and Execution

Firstly, you'll need to have installed `docker` and `docker-compose`. Check that you don't have any process listening on ports 3306, 8080, 8443, and 9991. Change the directory into `$PROJECT_HOME`, and run command `docker-compose up --remove-orphans`. Now, Wildfly should already have in deployment the project web application archive, with runtime name `adv_prog_project_war.war`, while MariaDB is running with the test values defined in file
`services/mariadb/docker-entrypoint-initdb.d/init.sql`.

To access the web application visit the link `https://0.0.0.0:8443/adv_prog_project_war/`.

You can also connect to URL `http://localhost:9990/console/index.html` with credentials (admin, admin). From here access `Deplyments > adv_prog_project_war.war` and click on the value of the field Context Root (it's highlighted in blue).

From here you can test the functionalities of the three roles:

- **log as student** using test credentials (user1, password1). From here you see the student dashboard with the grades. Use this dashboard to check if a given grade has been modified, or you can always log in to MariaDB with credentials (root,root) and query for the student table.
- **log as teacher** using test credentials (jane_smith, pass456). Here you find a simple dashboard, a button to import the private key used to sign the operation of grade assignation, and a POST form to assign a grade to a given student. Existing students who are enrolled in the course of the test teacher are IDs 1 and 3. You can use these to test the functionality. You can add other students by logging in as an admin user but no functionality for enrollment is provided.
- **log as admin** using credentials (admin, admin). Here you can submit the form to add a new student. IDs are automatically assigned incrementally.

Furthermore, you can check the data-at-rest encryption by logging in to the MariaDB docker using command `docker exec -it project-mariadb-1 bash`. From here change the directory into `/var/lib/mysql/universityDatabase` and choose a .ibd file to test. Run `apt update` and `apt install -y binutils`. Now, for example, take enrollments.ibd and run `strings enrollments.ibd`. The table should be encrypted.

The web archive that is deployed is the result of the last project build since the docker volume of Wildfly for deployment is bound to the project war output directory (as you can see in the docker-compose.yml). So, if you want to re-compile from source you just have to rebuild the project and rebuild the artifacts.

Alternatively, you can build from source code and deploy it remotely. Doing so requires some setup. For this, we suggest using IDE IntellijIDEA, as it is the same one we used for developing the web application. Firstly, you'll need to run the docker-compose file as before. Successively, you'll need to open the project in IntellijIDEA as a maven project. Then, create a configuration for the JBoss/Wildfly server in remote mode. Set up management port as 9990, username and password as (admin, admin). Furthermore, put as host `0.0.0.0` and as port `8443`. In the "Before launch" section put "Build project" and Build artifacts (select both when prompted). In the tab "Deployment" select artifact "adv_prog_project:war". Finally, apply the changes and run the application.

If you find this part unclear, you can follow the tutorial at `https://dzone.com/articles/deploy-to-wildfly-and-docker-from-intellij-using-m` in the section "Configure a Remote Wildfly Server in IntelliJ".

We have tried to make the installation and deployment as simple as we could. If you face any problem, please do not hesitate to contact us.

# Bibliography


[1] Mehmet Adalier and Antara Teknik. "Efficient and Secure Elliptic Curve Cryptography Implementation of Curve P-256". In: 2015. URL: https://api.semanticscholar.org/CorpusID:9303272.

[2] Lily Chen et al. "Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters". In: (Feb. 2023). URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf.

[3] MariaDB. *File Key Management Encryption Plugin*. https://mariadb.com/kb/en/file-key-management-encryption-plugin/. Accessed 2023-12-19. MariaDB, 2023.

[4] MariaDB. *TLS Connections with MariaDB Connector/J*. https://mariadb.com/docs/server/connect/programming-languages/java/tls/. Accessed 2023-12-19. MariaDB, 2023.

[5] Zachary Miller and Chandan Kundapur. *How to evaluate and use ECDSA certificates in AWS Certificate Manager*. https://aws.amazon.com/blogs/security/how-to-evaluate-and-use-ecdsa-certificates-in-aws-certificate-manager/. Accessed 2023-12-21. Amazon Web Services, Inc., 2022.

[6] OWASP. *Password Storage Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html. Accessed 2023-12-19. OWASP, 2023.