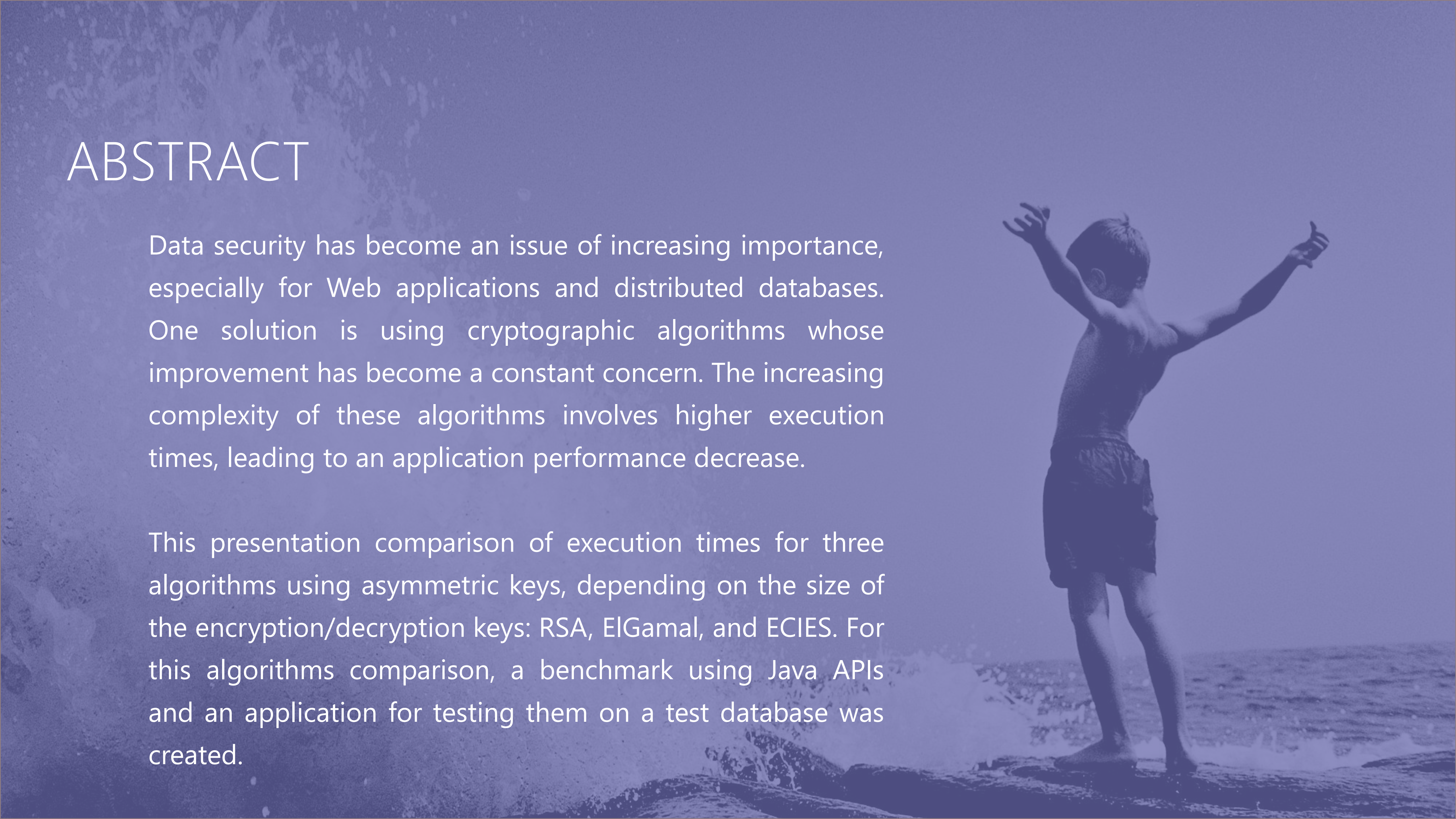
CASE STUDY

# DATABASE ENCRYPTION USING ASYMMETRIC KEYS

By
Nigel Fernandes (8868)
Happy Varughese Cherian (8870)
Upmanyu Jha (8875)
Natasha Lobo (8881)



## ABSTRACT

Data security has become an issue of increasing importance, especially for Web applications and distributed databases. One solution is using cryptographic algorithms whose improvement has become a constant concern. The increasing complexity of these algorithms involves higher execution times, leading to an application performance decrease.

This presentation comparison of execution times for three algorithms using asymmetric keys, depending on the size of the encryption/decryption keys: RSA, ElGamal, and ECIES. For this algorithms comparison, a benchmark using Java APIs and an application for testing them on a test database was created.

## INTRODUCTION

- ❑ Data can be encrypted at the storage level, a very suitable technique for protecting files and folders.
- ❑ However, the storage subsystem is not related to the database schema and the user, therefore, cannot implement security policies (privileges).
- ❑ The level of encryption is determined by the granularity of the system files, therefore non-confidential data is also encrypted.
- ❑ Moreover, some logs or temporary files may include unencrypted data blocks.
- ❑ Encryption at the database level provides more flexibility in terms of granularity of encryption and can map the database schema.
- ❑ Data can be encrypted at the table level, row level or column level.
- ❑ This type of encryption affects the indexes and need special retrieving queries using indexes that can work with such encrypted data.
- ❑ Specific algorithms for database encryption use:
  - Homomorphic encryption
  - Encryption preserving the database schema
  - Fast comparison encryption
- ❑ This presentation presents a comparative study between three asymmetric keys encryption algorithms used in database domain: Rivest-Shamir-Adleman (RSA), ElGamal and Elliptic Curve Integrated Encryption Scheme (ECIES).

## BRIEF OVERVIEW

According to the Internet Security Glossary (RFC 2828, published in 2000), a cryptographic system is defined as ”a set of cryptographic algorithms together with the key management processes that support the use of the algorithms in some application context”

1 . But encrypting data can decrease DBMS’s performance due to consumption of resources such as CPU or memory

At the same time, the storage space increases because of the encrypted data padding.

Padding is a technique used to encrypt messages of given size that the block cipher would not be able to decrypt otherwise.

However, there are cases in which queries and indexes are not significantly affected by encrypting data but these cases are rare and, generally, the encryption of data involves at least a decrease of performances for retrieval queries.

Along with the migration of data in the cloud, encryption has become a very important security service. In recent years dedicated servers are used to encrypt/decrypt data and this leads to a smaller influence on performance.

# ALGORITHMS IMPLEMENTATION

- ❖ The algorithms were implemented using the API provided by BouncyCastle, which offers a wide range of algorithms for the Java programming language.
- ❖ The studied algorithms are RSA, ElGamal, and ECIES.
- ❖ Each algorithm was run in a separate thread in the global benchmark.
- ❖ Also, a function that creates a number of threads equal to the number of tested algorithms was used in order to obtain data needed for drawing the charts showing the processing time for each algorithm.
- ❖ This means that measurements have been made running the algorithms in parallel.

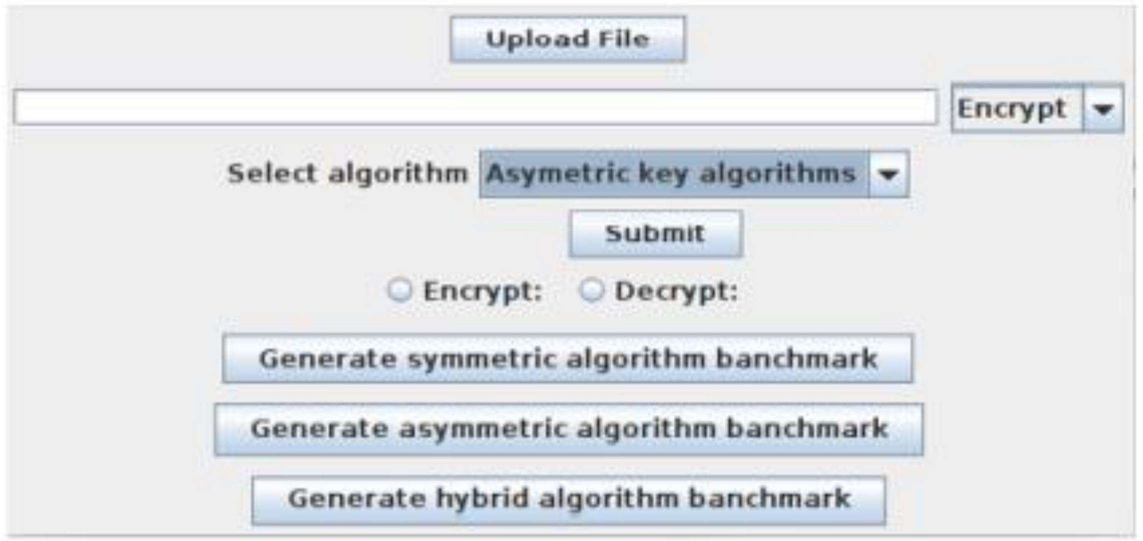

Upload File
Encrypt
Select algorithm
Asymetric key algorithms
Submit
Encrypt:
Decrypt:
Generate symmetric algorithm banchmark
Generate asymmetric algorithm banchmark
Generate hybrid algorithm banchmark

Figure 1: Interface for the benchmark used to test the algorithms

- ❖ Algorithms’ testing was done using several test scenarios for encryption and decryption of strings with length 10, 20 and 30 characters using 192, 256 and 1 024 bits length asymmetric keys for encryption/decryption.
- ❖ Time was measured by performing 10 runs on strings of the same length using equal length asymmetric keys

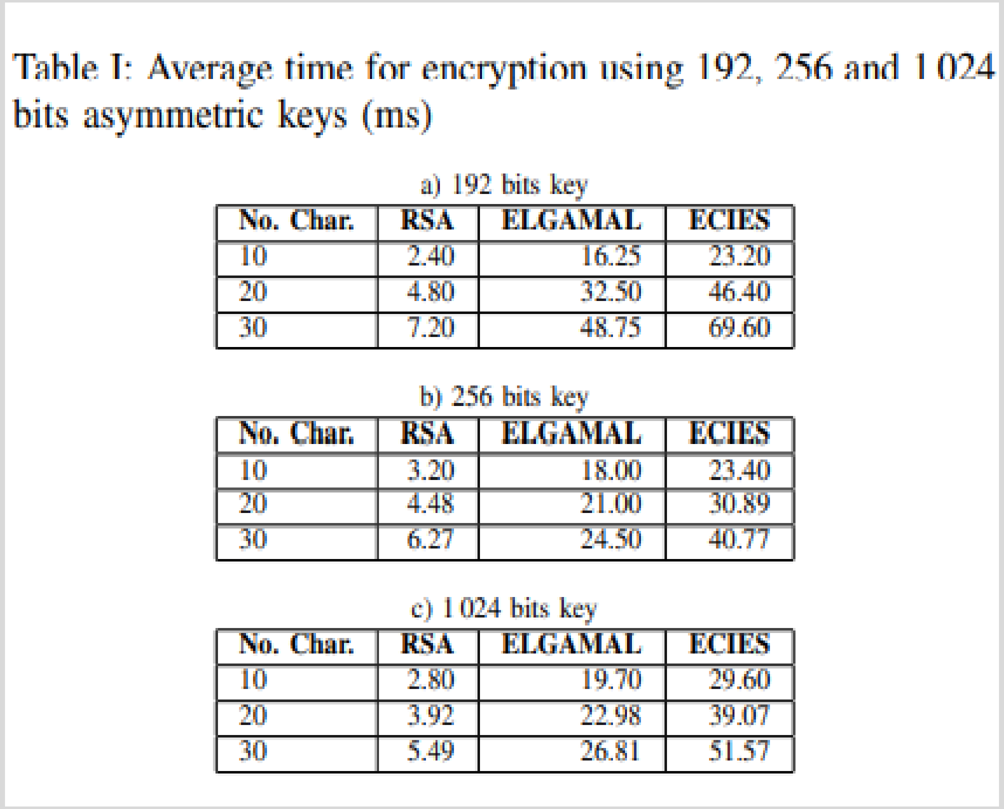


Table I: Average time for encryption using 192, 256 and 1 024 bits asymmetric keys (ms)

a) 192 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 2.40 | 16.25 | 23.20 |
| 20 | 4.80 | 32.50 | 46.40 |
| 30 | 7.20 | 48.75 | 69.60 |

b) 256 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 3.20 | 18.00 | 23.40 |
| 20 | 4.48 | 21.00 | 30.89 |
| 30 | 6.27 | 24.50 | 40.77 |

c) 1 024 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 2.80 | 19.70 | 29.60 |
| 20 | 3.92 | 22.98 | 39.07 |
| 30 | 5.49 | 26.81 | 51.57 |

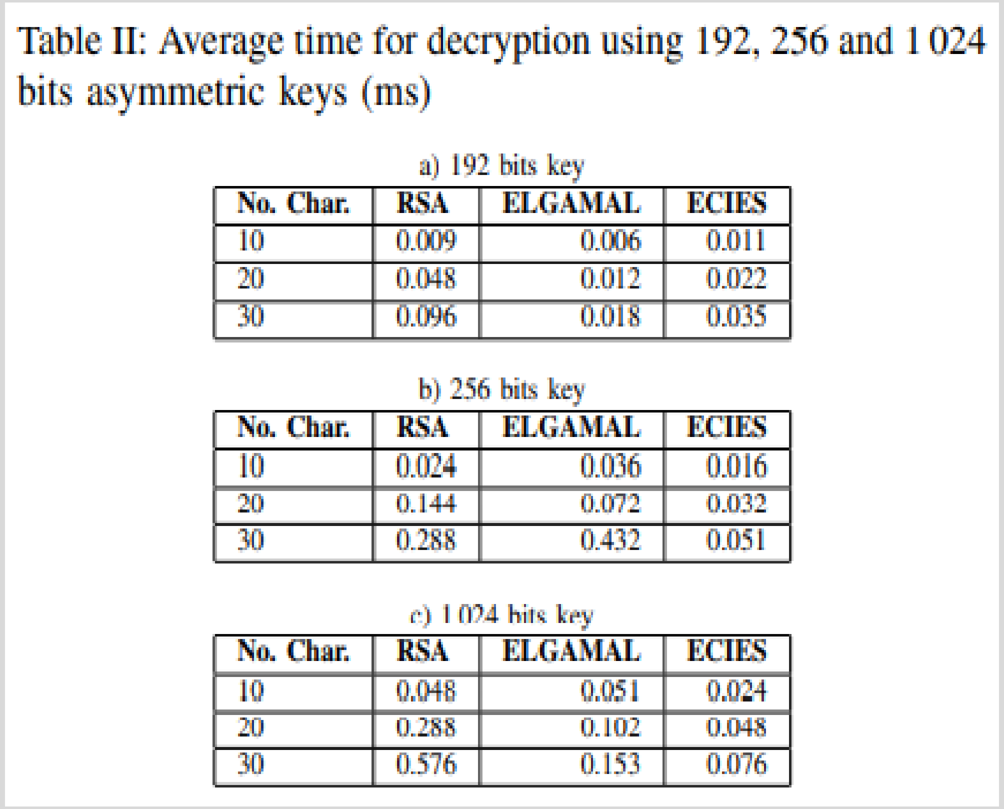


Table II: Average time for decryption using 192, 256 and 1 024 bits asymmetric keys (ms)

a) 192 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 0.009 | 0.006 | 0.011 |
| 20 | 0.048 | 0.012 | 0.022 |
| 30 | 0.096 | 0.018 | 0.035 |

b) 256 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 0.024 | 0.036 | 0.016 |
| 20 | 0.144 | 0.072 | 0.032 |
| 30 | 0.288 | 0.432 | 0.051 |

c) 1 024 bits key

| No. Char. | RSA | ELGAMAL | ECIES |
|---|---|---|---|
| 10 | 0.048 | 0.051 | 0.024 |
| 20 | 0.288 | 0.102 | 0.048 |
| 30 | 0.576 | 0.153 | 0.076 |

The average time in milliseconds obtained when encrypting strings of 10, 20 and 30 characters is shown in Table I.

The average time in milliseconds obtained when decrypting strings of 10, 20 and 30 characters is shown in Table II.

The analysis of the test results shows that:

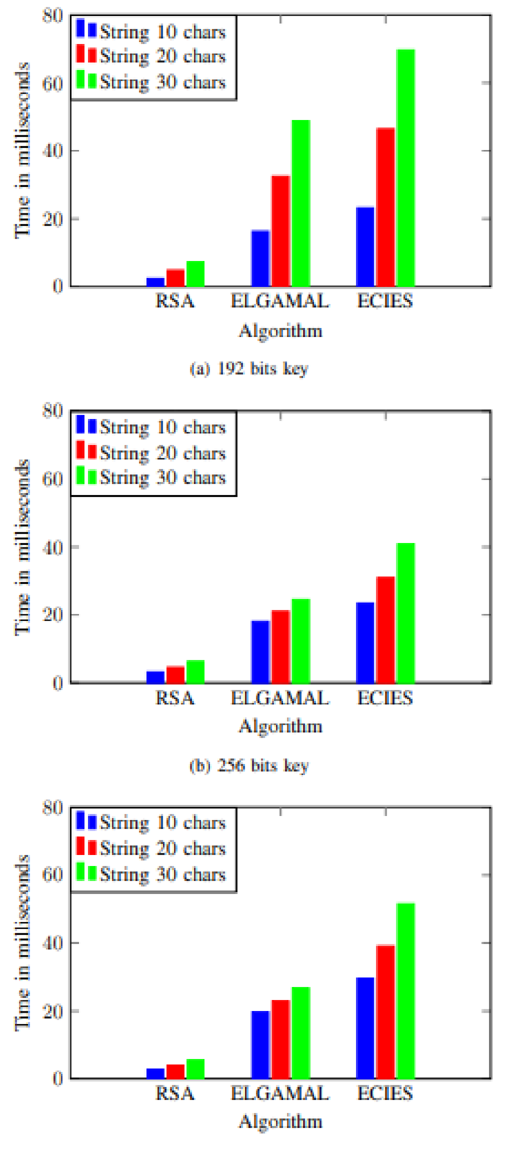
• Encryption time increases with the number of characters encrypted. The biggest encryption time is for algorithm ECIES and it is about 10 times larger than for algorithm RSA; ElGamal has an encryption time 7 times bigger than RSA for the same number of characters.

• The algorithms have encryption time of the same order of magnitude considering the length of the encryption key.

• Decryption time is much lower than encryption time for all algorithms (tens of times lower).

• For RSA, encryption time varies very little with the number of characters; the greatest variation is for ECIES algorithm where the time encryption varies inversely with the number of characters encrypted, using the same size of the encryption key.

• The decryption time increases with the size of the decryption key for all three algorithms, biggest for ECIES and smallest for RSA.

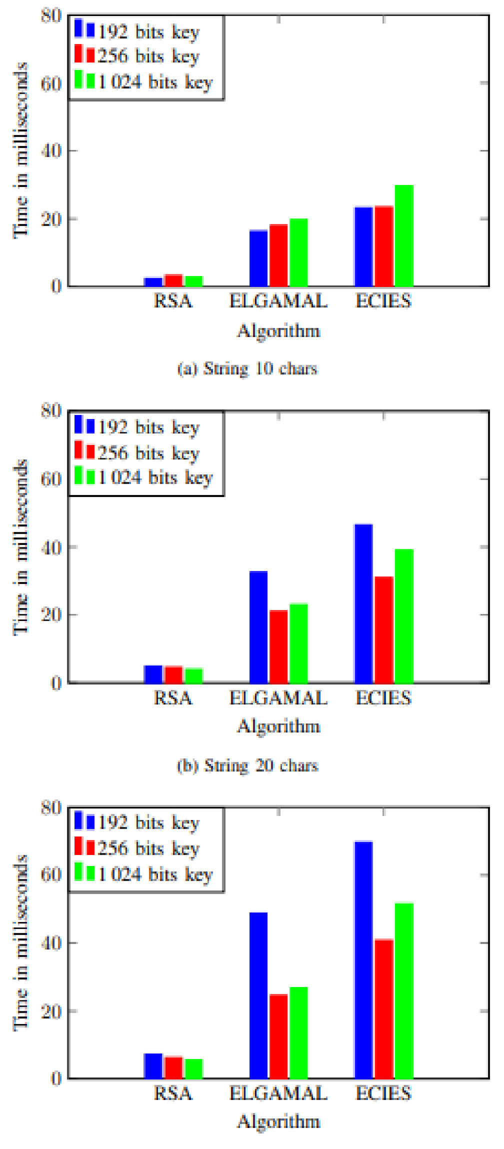


(a) 192 bits key

(b) 256 bits key

(c) 1 024 bits key

Figure 2: Average time for encryption using 192, 256 and 1024 bits asymmetric keys (ms)

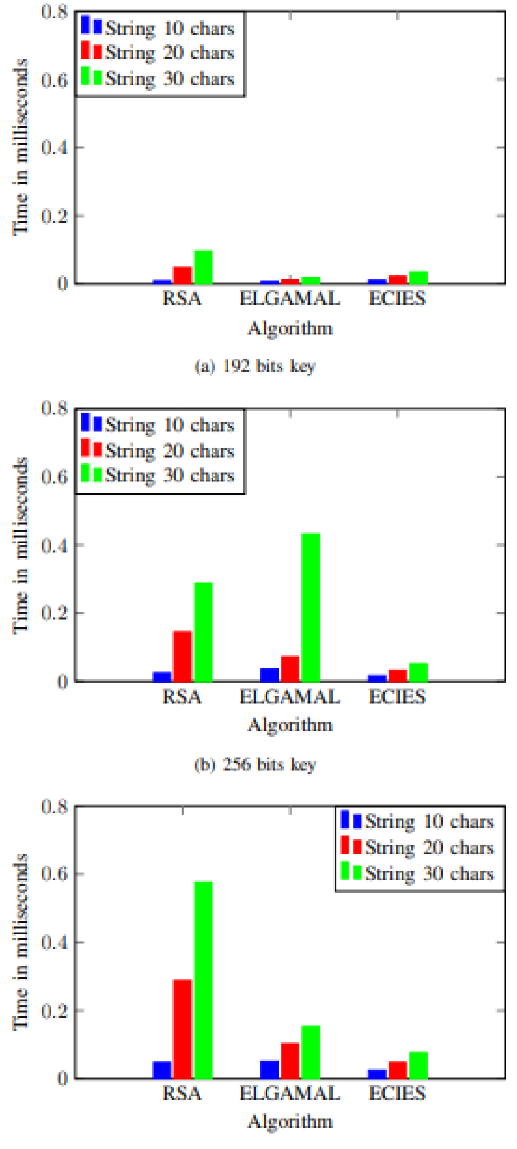


(a) String 10 chars

(b) String 20 chars

(c) String 30 chars

Figure 3: Average time for encrypting 10, 20 and 30 characters strings (ms)

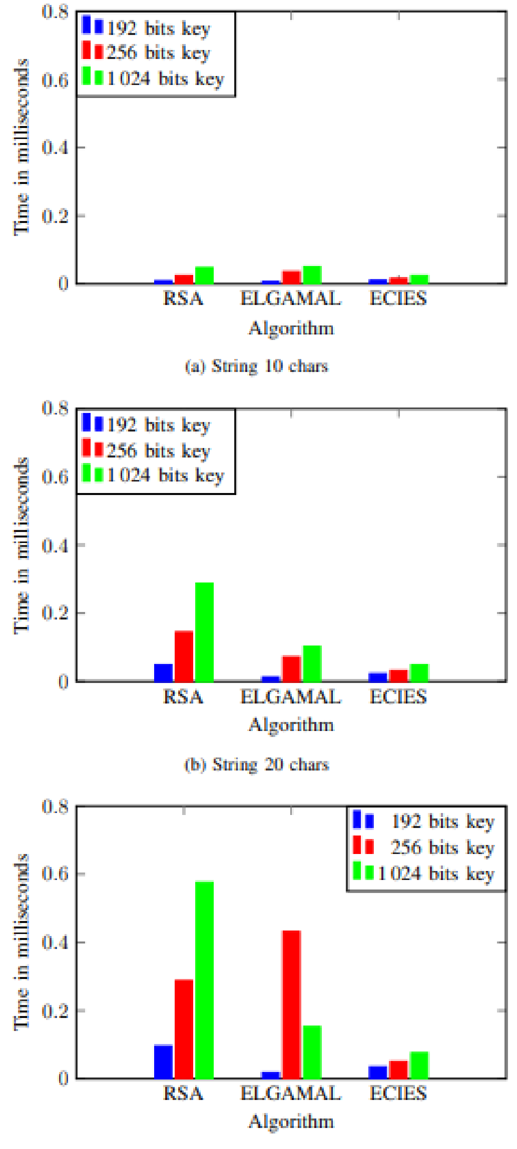


(a) 192 bits key

(b) 256 bits key

(c) 1 024 bits key

Figure 4: Average time for decryption using 192, 256 and 1024 bits asymmetric keys (ms)



(a) String 10 chars

(b) String 20 chars

(c) String 30 chars

Figure 5: Average time for decrypting 10, 20 and 30 characters strings (ms)

# ARCHITECTURE DIAGRAM

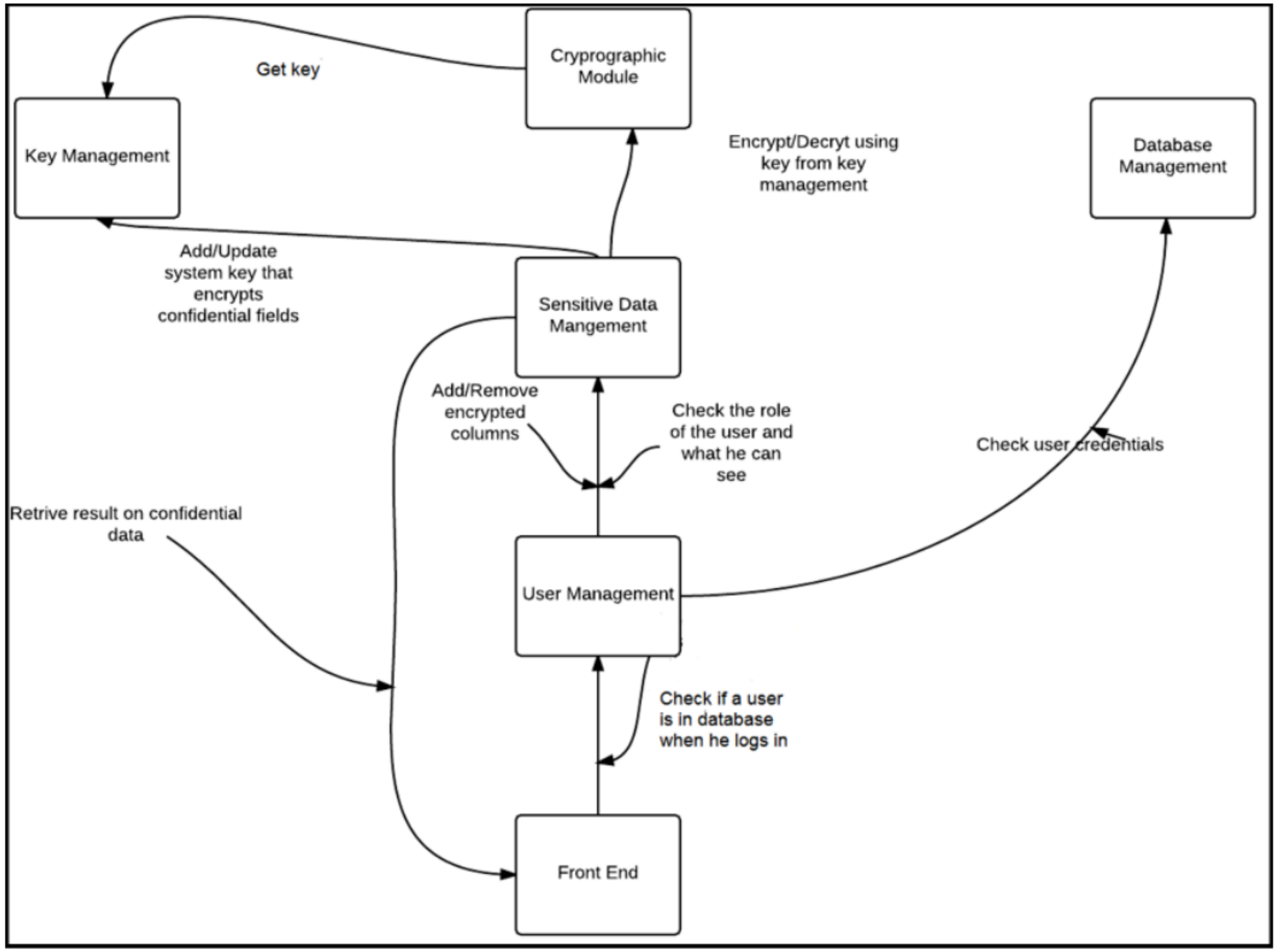

Cryprographic Module
Get key
Key Management
Encrypt/Decryt using key from key management
Database Management
Add/Update system key that encrypts confidential fields
Sensitive Data Mangement
Add/Remove encrypted columns
Check the role of the user and what he can see
Check user credentials
Retrive result on confidential data
User Management
Check if a user is in database when he logs in
Front End

Figure 6: Application diagram

## IMPLEMENTATION DETAILS

A Java application was developed to test the algorithms on a database.

The technologies used for developing the application are:
Java 16,
Java SWT for the graphic part and
MySQL 8.0.28 as database engine.

The test application allows a user, depending on the assigned role, to read data from specific columns in standard or encrypted format.
If the assigned role contains the appropriate privileges, users can read, write and can execute filtering, comparison and aggregation operations on the encrypted columns.

For the tests, two database schemes were used:
a database schema for the encrypted data and another for application logic.

The application consists of several sub-modules:

- Operational database module - for running SQL queries.
- Confidential data module - for read/write operations on encrypted columns.
- Cryptographic module - for data encryption/decryption;
- Key management module - for the management of encryption/decryption keys.
- Users and roles management module - designed to manage users and roles assigned to them.
- Front-end module - contains the graphical user interface. Users can log in and use the functionalities of the application.

The main menu has the following basic features: navigation through the database, report generation and setting of confidential columns (only for manager role).

A user with administrator privileges will be able to generate graphs showing algorithms performances and then, depending on the result, can choose the algorithm and the asymmetric key for the cryptographic operations on confidential columns in the database.

Figure 6 shows the application diagram and the connections between modules.

To implement the application functionalities, besides the tables containing the encrypted and unencrypted data, the following additional tables were created:

- DB USERS - contains user’s name and password for all application accounts.
- DB ROLES - contains possible user roles.
- USER ROLE TABLE - contains the association between users and all its roles.
- ENCRYPTED COLUMNS - contains table names and column names that contain encrypted data.

There are four possible roles for a user:

- ADMINISTRATOR - permission to add/delete columns in the ENCRYPTED COLUMNS table.
- HR MANAGER - permission to generate reports on confidential data (can see decrypted values of encrypted columns such as salary or commission of an employee).
- GENERAL MANAGER - permission to access all data.
- HR EMPLOYEE - permission to generate reports and filter confidential data but without access to decrypted atomic data (but can see aggregated values such as average salary per department and so on).

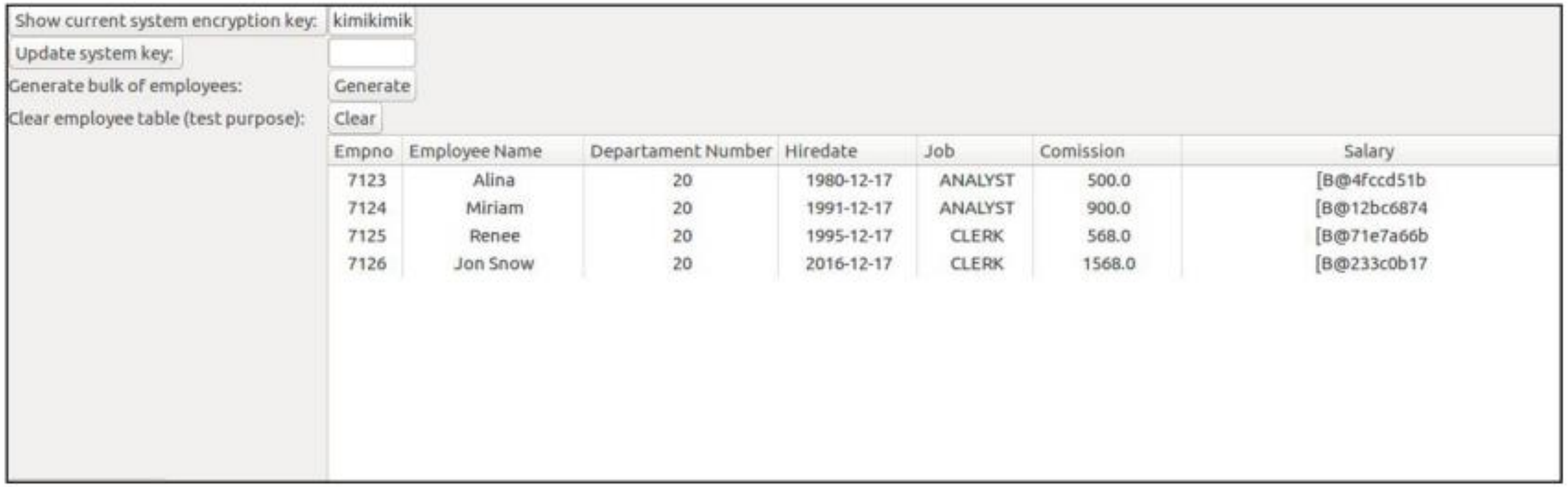
In Figure 7 following application features are shown:

- View system encryption key.
- Changing the system encryption key.
- Generate table entries for Employees table.
- Delete data in the table (for testing purposes).

The SELECT queries use a decryption function for retrieving, comparing and filtering the data (Figure 8)

```
SELECT *
FROM EMPLOYEES
WHERE
   decrypt (system_key , SALARY).toInteger ()> 3000;
```

Figure 8: The Decryption function used in a SQL query



Show current system encryption key: kimikimik
Update system key:
Generate bulk of employees: Generate
Clear employee table (test purpose): Clear

| Empno | Employee Name | Departament Number | Hiredate | Job | Comission | Salary |
|---|---|---|---|---|---|---|
| 7123 | Alina | 20 | 1980-12-17 | ANALYST | 500.0 | [B@4fccd51b |
| 7124 | Miriam | 20 | 1991-12-17 | ANALYST | 900.0 | [B@12bc6874 |
| 7125 | Renee | 20 | 1995-12-17 | CLERK | 568.0 | [B@71e7a66b |
| 7126 | Jon Snow | 20 | 2016-12-17 | CLERK | 1568.0 | [B@233c0b17 |

Figure 7: Application features

## CONCLUSION:

- ❑ Several criteria should considered when choosing an encryption algorithm suitable for a database.
- ❑ The most important criterion is the level of security.
- ❑ An algorithm with fewer security breaches may involve a higher encryption/decryption time, depending on the size of the encryption key.
- ❑ This leads to significant decreases in performances or on-line applications that work with large volumes of data or a large number of users acting in parallel.
- ❑ Encryption also involves additional consumption of resources on the server, and the data storage space for encrypted data will be larger due to the padding.
- ❑ Controlling data privacy can be achieved by assigning roles to the users, roles for both the unencrypted and encrypted data. Another important aspect is the key management because decryption can be made only with a valid key at the time of encryption.
- ❑ There is also the risk of losing the data if the key administration is flawed, especially for older archives. In this case, the whole database is compromised.