Exercise 1: Library for Matrix Operations

Microsoft’s Simple Encrypted Arithmetic Library (SEAL)\(^1\) is a publicly available homomorphic encryption library. It can be found and downloaded at [http://sealcrypto.codeplex.com/](http://sealcrypto.codeplex.com/).

Implement a library `13b_MS-SEAL.h` supporting matrix operations using homomorphic encryption on basis of the MS SEAL.

**due date:** 24.01.2021 (EOB)
**no. of students:** 2
**deliverables:**
1. Implementation (including source code(s))
2. Documentation (max. 10 pages)
3. Presentation (10 – max. 15 minutes)

---

`13b_MS-SEAL.h`

```c
#include <float.h>
#include <stdbool.h>

typedef struct {
  double *entries;
  unsigned int width;
  unsigned int height;
} matrix;

/*
 Initialize new matrix:
 - reserve memory only
 */
matrix initMatrix(unsigned int width, unsigned int height);

/*
 Initialize new matrix:
 - reserve memory
 - set any value to 0
 */
matrix initMatrixZero(unsigned int width, unsigned int height);

/*
 Initialize new matrix:
 - reserve memory
 - set any value to random number
 */
matrix initMatrixRand(unsigned int width, unsigned int height);
```

/ copy a matrix and return its copy
* /
matrix copyMatrix(matrix toCopy);

/*
destroy matrix
– free memory
– set any remaining value to NULL
*/
void freeMatrix(matrix toDestroy);

/*
return entry at position (xPos, yPos), DBL_MAX in case of error
*/
double getEntryAt(matrix a, unsigned int xPos, unsigned int yPos);

/*
set entry at position (xPos, yPos)
return true in case of success, false otherwise
*/
bool setEntryAt(matrix a, unsigned int xPos, unsigned int yPos, double value);

/*
print matrix to stdout
*/
void prettyPrint(matrix a);

/*
add two matrices
return:
– result in newly created matrix
– matrix of size 0 in case of error
*/
matrix addMatrix(matrix a, matrix b);

/*
subtract two matrices:
return: "a - b"
– result in newly created matrix
– matrix of size 0 in case of error
*/
matrix subMatrix(matrix a, matrix b);

/*
multiply two matrices
return: "a * b"
– result in newly created matrix
– matrix of size 0 in case of error
*/
matrix multMatrix(matrix a, matrix b);

/*
transpose matrix
return: "a^T"
*/
matrix transposeMatrix(matrix a);

/*
return determinant of matrix, DBL_MAX in case of error
*/
double determinante(matrix a); // simple algorithm
double detQuick(matrix a); // optional: more efficient algorithm