```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MODULO 37

int char_to_int(char c) {
    if (c == ' ') return 26;
    if (c >= 'A' && c <= 'Z') return c - 'A';
    if (c >= '0' && c <= '9') return 27 + (c - '0');
    return 0;
}

char int_to_char(int x) {
    if (x == 26) return ' ';
    if (x >= 0 && x <= 25) return 'A' + x;
    if (x >= 27 && x <= 36) return '0' + (x - 27);
    return '?';
}

int mod_inverse(int a, int m) {
    int t = 0, new_t = 1;
    int r = m, new_r = a;
    while (new_r != 0) {
        int quotient = r / new_r;
        int temp = new_t;
        new_t = t - quotient * new_t;
        t = temp;
        temp = new_r;
        new_r = r - quotient * new_r;
        r = temp;
    }
    if (r > 1) return -1;
    if (t < 0) t += m;
    return t;
}

int determinant(int** matrix, int n) {
    if (n == 1) return matrix[0][0] % MODULO;
    if (n == 2) return (matrix[0][0] * matrix[1][1] - matrix[0][1] *
matrix[1][0]) % MODULO;
    int det = 0;
    int** submatrix = malloc((n - 1) * sizeof(int*));
    for (int i = 0; i < n - 1; i++)
        submatrix[i] = malloc((n - 1) * sizeof(int));
    for (int i = 0; i < n; i++) {
        int subi = 0;
        for (int j = 1; j < n; j++) {
            int subj = 0;
            for (int k = 0; k < n; k++) {
                if (k == i) continue;
                submatrix[subi][subj++] = matrix[j][k];
            }
            subi++;
        }
        int sign = (i % 2 == 0) ? 1 : -1;
        det = (det + sign * matrix[0][i] * determinant(submatrix, n - 1))
% MODULO;
    }
    for (int i = 0; i < n - 1; i++) free(submatrix[i]);
    free(submatrix);
    if (det < 0) det += MODULO;
    return det;
}

void adjoint(int** matrix, int** adj, int n) {
    if (n == 1) {
        adj[0][0] = 1;
        return;
    }
    int** temp = malloc((n - 1) * sizeof(int*));
    for (int i = 0; i < n - 1; i++)
        temp[i] = malloc((n - 1) * sizeof(int));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int subi = 0;
            for (int x = 0; x < n; x++) {
                if (x == i) continue;
                int subj = 0;
                for (int y = 0; y < n; y++) {
                    if (y == j) continue;
                    temp[subi][subj++] = matrix[x][y];
                }
                subi++;
            }
            int sign = ((i + j) % 2 == 0) ? 1 : -1;
            adj[j][i] = (sign * determinant(temp, n - 1)) % MODULO;
            if (adj[j][i] < 0) adj[j][i] += MODULO;
        }
    }
    for (int i = 0; i < n - 1; i++) free(temp[i]);
    free(temp);
}

int** inverse_matrix(int** matrix, int n) {
    int det = determinant(matrix, n);
    int det_inv = mod_inverse(det, MODULO);
    if (det_inv == -1) return NULL;
    int** adj = malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
        adj[i] = malloc(n * sizeof(int));
    adjoint(matrix, adj, n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            adj[i][j] = (adj[i][j] * det_inv) % MODULO;
    return adj;
}

void read_key_matrix(int** matrix, int n) {
    printf("Entrez la matrice cle (%d x %d), entiers entre 0 et 36 :\n",
n, n);
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            scanf("%d", &matrix[i][j]);
```

```c
}

void multiply_matrix_vector(int** matrix, int* vector, int* result, int
n) {
    for (int i = 0; i < n; ++i) {
        result[i] = 0;
        for (int j = 0; j < n; ++j)
            result[i] += matrix[i][j] * vector[j];
        result[i] %= MODULO;
    }
}

void strip_newline(char* str) {
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n')
        str[len - 1] = '\0';
}

void clear_input() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

int main() {
    int n, choix;
    printf("Taille de la matrice cle (n x n) : ");
    scanf("%d", &n);

    int** key = malloc(n * sizeof(int*));
    for (int i = 0; i < n; ++i)
        key[i] = malloc(n * sizeof(int));
    read_key_matrix(key, n);

    do {
        printf("\n=== MENU ===\n");
        printf("1. Chiffrer un message\n");
        printf("2. Dechiffrer un message\n");
        printf("3. Quitter\n");
        printf("Votre choix : ");
        scanf("%d", &choix);
        clear_input();

        if (choix == 1 || choix == 2) {
            char message[10000];
            printf("Entrez le message en MAJUSCULES (lettres, espaces,
chiffres) :\n");
            fgets(message, sizeof(message), stdin);
            strip_newline(message);

            int len = strlen(message);
            int padded_len = len + (n - len % n) % n;

            int* numeric_msg = malloc(padded_len * sizeof(int));
            for (int i = 0; i < len; ++i)
                numeric_msg[i] = char_to_int(toupper(message[i]));
            for (int i = len; i < padded_len; ++i)
                numeric_msg[i] = char_to_int(' ');
            char* result_msg = malloc((padded_len + 1) * sizeof(char));
            int* block = malloc(n * sizeof(int));
            int* result = malloc(n * sizeof(int));

            int** matrix_used = key;
            if (choix == 2) {
                matrix_used = inverse_matrix(key, n);
                if (matrix_used == NULL) {
                    printf("La matrice cle n'a pas d'inverse.\n");
                    free(numeric_msg); free(result_msg); free(block);
free(result);
                    continue;
                }
            }

            for (int i = 0; i < padded_len; i += n) {
                for (int j = 0; j < n; ++j)
                    block[j] = numeric_msg[i + j];
                multiply_matrix_vector(matrix_used, block, result, n);
                for (int j = 0; j < n; ++j)
                    result_msg[i + j] = int_to_char(result[j]);
            }
            result_msg[padded_len] = '\0';

            if (choix == 1)
                printf("Message chiffre :\n%s\n", result_msg);
            else
                printf("Message decrypte :\n%s\n", result_msg);

            if (choix == 2) {
                for (int i = 0; i < n; ++i) free(matrix_used[i]);
                free(matrix_used);
            }

            free(numeric_msg);
            free(result_msg);
            free(block);
            free(result);
        }

    } while (choix != 3);

    for (int i = 0; i < n; ++i)
        free(key[i]);
    free(key);

    printf("Programme termine.\n");
    return 0;
}
```