```c
/* DUKH Attack
 * COMP10002 Foundations of Algorithms, Semester 1, 2021
 * Skeleton code written by Shaanan Cohney, April 2021
 *
 * Full Name: Thomas Choi
 * Student Number: 1202247
 * Date: 17.04.2021
 */

/****** Include libraries ******/

#include <stdio.h>
#include <stdlib.h>
/* Do NOT use the following two libraries in stage 1! */
#include <string.h>
#include <ctype.h>

/* Provides functions AES_encrypt and AES_decrypt (see the assignment spec) */
#include "aes.h"
/* Provides functions to submit your work for each stage.
 * See the definitions in a1grader.h, they are all available to use.
 * But don't submit your stages more than once... that's weird! */
#include "a1grader.h"

/****** Definitions of constants ******/

#define BOOK_LENGTH 1284         /* The maximum length of a cipher book */
#define MAX_MSG_LENGTH 1024      /* The maximum length of an encrypted message
*/
#define BLOCKSIZE 16             /* The length of a block (key, output) */
#define N_TIMESTEPS 20           /* number of timesteps */
#define N_OUTPUT_BLOCKS 2        /* number of output blocks */

// TODO Add your own #defines here, if needed
#define LHS 1
#define RHS 0
#define OUTPUTS_9 0
#define OUTPUTS_10 1
#define START_OF_TIMESTEP 11

/****** Type definitions ******/
/* Recall that these are merely aliases, or shortcuts to their underlying types.
 * For example, block_t can be used in place of an array, length 16 (BLOCKSIZE)
 * of unsigned char, and vice versa. */

typedef char book_t[BOOK_LENGTH];     /* A cipherbook (1284 bytes) */
typedef unsigned char byte_t;         /* A byte (8 bits) */
typedef byte_t block_t[BLOCKSIZE];    /* A cipher bitset (block) (16 bytes) */
typedef byte_t msg_t[MAX_MSG_LENGTH]; /* An encrypted message (l bytes) */

// TODO Add your own type definitions here, if needed



/****** Function Prototypes ******
 * There are more functions defined in aes.h and grader.h */
// Scaffold

int read_hex_line(byte_t output[], int max_count, char *last_char);

```

```
 60   // Hint: Variables passed by pointers should be modified in your stages'
implementation!
 61
 62   void stage0(msg_t ciphertext, int *ciphertext_length,
 63               block_t outputs[N_OUTPUT_BLOCKS], block_t timesteps[N_TIMESTEPS],
 64               book_t cipherbook);
 65   void stage1(book_t cipherbook, int *book_len);
 66   void stage2(byte_t codebook[], int book_len, block_t outputs[N_OUTPUT_BLOCKS],
 67               block_t timesteps[N_TIMESTEPS], block_t key2);
 68   void stage3(block_t key2, block_t outputs[N_OUTPUT_BLOCKS],
 69               block_t timesteps[N_TIMESTEPS], byte_t key1[], int cipher_length);
 70   void stage4(byte_t key1[], byte_t ciphertext[], int cipher_length,
 71               byte_t plaintext[]);
 72
 73   // TODO: Put your own function prototypes here! Recommended: separate into
stages.
 74   // Stage 1:
 75   int is_alphanumeric(char ch);
 76
 77   // Stage 2:
 78   void xor_byte_array(byte_t x[], byte_t y[], byte_t xor[], int length);
 79   void find_eq1_one_side(block_t timesteps[], block_t key, block_t outputs[],
 80                          block_t result, int side);
 81   int is_equal(block_t block1, block_t block2);
 82   void copy_blocks(block_t dest, block_t src);
 83
 84   // Stage 3:
 85   void find_initial_state(block_t outputs[], block_t timesteps[],
 86                           block_t key2, block_t states[]);
 87   void ran_number_gen(block_t generated_outputs[], block_t states[], block_t time
steps[],
 88                       block_t key2, int num_of_gen_outputs);
 89   void copy_gen_outputs(byte_t key1[], block_t generated_outputs[], int ciphertex
t_length);
 90
 91   /* The main function of the program */
 92   // It is strongly suggested you do NOT modify this function.
 93   int main(int argc, char *argv[])
 94   {
 95       //// Stage 0
 96       /* These will store our input from the input file */
 97       msg_t ciphertext;                  // encrypted message, to be decrypted in
the attack
 98       int ciphertext_length = 0;         // length of the encrypted message
 99       book_t cipherbook;                 // book used to make key k2
100       block_t timesteps[N_TIMESTEPS];    // timesteps used to generate outputs
(hex)
101       block_t outputs[N_OUTPUT_BLOCKS];  // outputs from the random number
generator (hex)
102
103       // Run your stage 0 code
104       stage0(ciphertext, &ciphertext_length, outputs, timesteps, cipherbook);
105       // And submit the results.  Don't delete this...
106       submit_stage0(ciphertext_length, ciphertext, outputs, timesteps, cipherbook)
;
107
108       //// Stage 1
109       int book_len = 0;    // length of the cipher book after having removed
punctuation
110       stage1(cipherbook, &book_len);
```

```
111       submit_stage1(cipherbook, book_len);
112
113       //// Stage 2
114       block_t key2;         // the key k2 (hexadecimal)
115       stage2((byte_t *) cipherbook, book_len, outputs, timesteps, key2);
116       submit_stage2(key2);
117
118       //// Stage 3
119       byte_t key1[MAX_MSG_LENGTH];        // the key k2 (hexadecimal)
120       stage3(key2, outputs, timesteps, key1, ciphertext_length);
121       submit_stage3(key1);
122
123       //// Stage 4
124       byte_t plaintext[MAX_MSG_LENGTH];  // the plaintext output
125       stage4(key1, ciphertext, ciphertext_length, plaintext);
126       submit_stage4(plaintext);
127
128       return 0;
129   }
130
131   /********* Scaffold Functions *********/
132
133   /* Reads a line in from stdin, converting pairs of hexadecimal (0-F) chars to
134    * byte_t (0-255), storing the result into the output array,
135    * stopping after max_count values are read, or a newline is read.
136    *
137    * Returns the number of *bytes* read.
138    * The last char read in from stdin is stored in the value pointed to by
last_char.
139    * If you don't need to know what last_char is, set that argument to NULL
140    */
141   int read_hex_line(byte_t output[], int max_count, char *last_char)
142   {
143       char hex[2];
144       int count;
145       for (count = 0; count < max_count; count++)
146       {
147           /* Consider the first character of the hex */
148           hex[0] = getchar();
149           if (hex[0] == '\n')
150           {
151               if (last_char)
152               {
153                   *last_char = hex[0];
154               }
155               break;
156           }
157           /* Now the second */
158           hex[1] = getchar();
159           if (last_char)
160           {
161               *last_char = hex[0];
162           }
163           if (hex[1] == '\n')
164           {
165               break;
166           }
167
168           /* Convert this hex into an int and store it */
169           output[count] = hex_to_int(hex); // (defined in aes.h)
```

```
170         }
171
172         return count - 1;
173     }
174
175     /********* Stage 0 Functions *********/
176     // read the input file
177     void stage0(msg_t ciphertext, int *ciphertext_length, block_t outputs[N_OUTPUT_
BLOCKS],
178                 block_t timesteps[N_TIMESTEPS], book_t cipherbook)
179     {
180         // TODO: Implement stage 0!
181         int i, ch;
182         scanf("%d\n", ciphertext_length);
183
184         read_hex_line(ciphertext, *ciphertext_length, NULL);
185         ch = getchar(); // remove the newline between each line
186
187         for (i = 0; i < N_OUTPUT_BLOCKS; i++)
188         {
189             read_hex_line(outputs[i], BLOCKSIZE, NULL);
190         }
191         ch = getchar(); // remove the newline between each line
192
193         for (i = 0; i < N_TIMESTEPS; i++)
194         {
195             read_hex_line(timesteps[i], BLOCKSIZE, NULL);
196         }
197         ch = getchar(); // remove the newline between each line
198
199         for (i = 0; i < BOOK_LENGTH; i++)
200         {
201             cipherbook[i] = getchar();
202         }
203
204         /* !! Submission Instructions !! Store your results in the variables:
205          *      ciphertext, ciphertext_length, outputs, timesteps, cipherbook
206          * These are passed to submit_stage0 for some useful output and submission.
*/
207     }
208
209     // TODO: Add functions here, if needed.
210
211     /********* Stage 1 Functions *********/
212     // Reminder: you *cannot* use string.h or ctype.h for this stage!
213
214     // strip punctuation in the cipherbook
215     void stage1(book_t cipherbook, int *book_len)
216     {
217         // TODO: Implement stage 1!
218         int i;
219         for (i = 0; i < BOOK_LENGTH; i++)
220         {
221
222             if (is_alphanumeric(cipherbook[i]))
223             {
224                 cipherbook[*book_len] = cipherbook[i];
225                 *book_len += 1;
226             }
227
```

```
228         }
229 
230         /* !! Submission Instructions !! Store your results in the variables:
231          *      cipherbook, book_len
232          * These are passed to submit_stage1 for some useful output and submission.
*/
233     }
234 
235     // TODO: Add functions here, if needed.
236 
237     // check if the char is alphanumeric
238     int is_alphanumeric(char ch)
239     {
240 
241         if ( (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && c
h <= '9') )
242         {
243             return 1;
244         }
245 
246         return 0;
247     }
248 
249     /********* Stage 2 Functions *********/
250 
251     // guess the key k2
252     void stage2(byte_t codebook[], int book_len, block_t outputs[N_OUTPUT_BLOCKS],
253                 block_t timesteps[N_TIMESTEPS], block_t key2)
254     {
255         // TODO: Implement stage 2!
256         int i;
257         byte_t *key_ptr;
258 
259         for (i = 0; i < book_len; i += 16)
260         {
261             key_ptr = codebook + i;
262 
263             block_t eq1_right, eq1_left;
264             find_eq1_one_side(timesteps, key_ptr, outputs, eq1_right, RHS);
265             find_eq1_one_side(timesteps, key_ptr, outputs, eq1_left, LHS);
266 
267             if (is_equal(eq1_left, eq1_right))
268             {
269                 copy_blocks(key2, key_ptr);
270             }
271 
272         }
273 
274         /* !! Submission Instructions !! Store your results in the variable:
275          *      key2
276          * These will be passed to submit_stage2 to let you see some useful output!
*/
277     }
278 
279     // TODO: Add functions here, if needed.
280 
281     // calculate the xor of two blocks and save to block_t xor
282     void xor_byte_array(byte_t x[], byte_t y[], byte_t xor[], int length)
283     {
284         int i;
```

Type text here

```
285
286         for (i = 0; i < length; i++)
287         {
288             xor[i] = x[i] ^ y[i];
289         }
290
291     }
292
293     // calculate the single side of equation 1
294     void find_eq1_one_side(block_t timesteps[], block_t key, block_t outputs[],
295                            block_t result, int side)
296     {
297         block_t aes_encrypt_output;
298
299         if (side == LHS)
300         {
301             // calculate the left hand side of equation 1
302
303             block_t aes_decrypt_output, decrypt_encrypt_xor;
304
305             AES_decrypt(outputs[OUTPUTS_10], key, aes_decrypt_output);
306             AES_encrypt(timesteps[10], key, aes_encrypt_output);
307
308             xor_byte_array(aes_decrypt_output, aes_encrypt_output, decrypt_encrypt_
xor, BLOCKSIZE);
309
310             AES_decrypt(decrypt_encrypt_xor, key, result);
311
312         }
313         else
314         {
315             // calculate the right hand side of equation 1
316
317             AES_encrypt(timesteps[9], key, aes_encrypt_output);
318             xor_byte_array(outputs[OUTPUTS_9], aes_encrypt_output, result,
BLOCKSIZE);
319
320         }
321
322     }
323
324     // check if two blocks are equal
325     int is_equal(block_t block1, block_t block2)
326     {
327         int i;
328
329         for (i = 0; i < BLOCKSIZE; i++)
330         {
331
332             if (block1[i] != block2[i])
333             {
334                 return 0;
335             }
336
337         }
338
339         return 1;
340
341     }
342
```

```
343   // copy the block from src to dest
344   void copy_blocks(block_t dest, block_t src)
345   {
346       int i;
347
348       for (i = 0; i < BLOCKSIZE; i++)
349       {
350           dest[i] = src[i];
351       }
352
353   }
354   /********** Stage 3 Functions *********/
355
356   // generate the key k1
357   void stage3(block_t key2, block_t outputs[N_OUTPUT_BLOCKS],
358               block_t timesteps[N_TIMESTEPS], byte_t key1[], int
ciphertext_length)
359   {
360       // TODO: Implement stage 3!
361
362       // ciphertext_length bytes of output, each output is 16-bytes in length
363       int num_of_gen_outputs = ciphertext_length / BLOCKSIZE;
364
365       // states has num_of_gen_outputs+1 because of the extra initial state
366       block_t states[num_of_gen_outputs+1], generated_outputs[num_of_gen_outputs];

367
368       find_initial_state(outputs, timesteps, key2, states);
369
370       ran_number_gen(generated_outputs, states, timesteps, key2, num_of_gen_outpu
ts);
371
372       copy_gen_outputs(key1, generated_outputs, ciphertext_length);
373
374       /* !! Submission Instructions !! Store your results in the variable:
375        *      key1
376        * These will be passed to submit_stage3 to let you see some useful output!
*/
377   }
378
379   // TODO: Add functions here, if needed.
380
381   // calculate the initial state of the generator
382   void find_initial_state(block_t outputs[], block_t timesteps[], block_t key2, bl
ock_t states[])
383   {
384       block_t aes_encrypt_output, output10_encrypt_xor;
385
386       AES_encrypt(timesteps[10], key2, aes_encrypt_output);
387       xor_byte_array(outputs[OUTPUTS_10], aes_encrypt_output,
output10_encrypt_xor, BLOCKSIZE);
388
389       AES_encrypt(output10_encrypt_xor, key2, states[0]);  // initial state is
stored in states[0]
390   }
391
392   // implement the random number generator and generate output of length
num_of_gen_outputs
393   void ran_number_gen(block_t generated_outputs[], block_t states[], block_t time
steps[],
```

```
394                         block_t key2, int num_of_gen_outputs)
395   {
396       block_t intermediates[num_of_gen_outputs];
397       int i;
398
399       for (i = 0; i < num_of_gen_outputs; i++)
400       {
401           block_t intermediate_state_xor, output_intermediate_xor;
402
403           AES_encrypt(timesteps[START_OF_TIMESTEP+i], key2, intermediates[i]);
404           xor_byte_array(intermediates[i], states[i], intermediate_state_xor, BLO
CKSIZE);
405
406           AES_encrypt(intermediate_state_xor, key2, generated_outputs[i]);
407           xor_byte_array(generated_outputs[i], intermediates[i],
408                          output_intermediate_xor, BLOCKSIZE);
409
410           AES_encrypt(output_intermediate_xor, key2, states[i+1]);
411
412       }
413
414   }
415
416   // copy the generated outputs to k1
417   void copy_gen_outputs(byte_t key1[], block_t generated_outputs[], int ciphertex
t_length)
418   {
419       int i, row = 0, col = 0;
420
421       for (i = 0; i < ciphertext_length; i++)
422       {
423
424           if (col % BLOCKSIZE == 0 && col != 0)
425           {
426               col = 0;
427               row++;
428           }
429
430           key1[i] = generated_outputs[row][col];
431           col++;
432
433       }
434
435   }
436   /********* Stage 4 Functions *********/
437
438   // decrypte the original message
439   void stage4(byte_t key1[], byte_t ciphertext[], int cipher_length, byte_t plain
text[])
440   {
441       // TODO: Implement stage 4!
442       xor_byte_array(ciphertext, key1, plaintext, cipher_length);
443
444       /* !! Submission Instructions !! Store your results in the variable:
445        *      plaintext
446        * These will be passed to submit_stage4 to let you see some useful output!
*/
447   }
448
449   // TODO: Add functions here, if needed.
```

```
450
451   /********* END OF ASSIGNMENT! *********/
452   /* If you would like to try the bonus stage, attempt it in a new file, bonus.c
*/
453   // Feel free to write a comment to the marker or the lecturer below...
454
455   // algorithms are awesome
```