```c
/** crypto.c : functions for encryption and decryption

  AUTHOR:   Brendan Dolan-Gavitt
  CLASS:    CS6262
  DATE:     9/15/2009
  LICENSE:  GPL Version 3 or higher

**/

#include <unistd.h>
#include <stdio.h>
#include <ctype.h>
#include "crypto.h"

/** get_key : derive a key based on a password input by the user
 *    Arguments: none
 *    Returns: pointer to an allocated buffer containing the key
 */
void * get_key() {
    char *pass;
    void *digest;

    digest = malloc(MD5_DIGEST_SIZE);
```

This is the get_key function

```c
    pass = getpass("Password: ");

    // It seems like this should have some way of checking
    // for failure, but the function returns void :p
    gcry_md_hash_buffer(GCRY_MD_MD5, digest, pass, strlen(pass));

    return digest;
}

/** encrypt_block: encrypts one block of the input file
 *                 and writes it to the output file
 *    Arguments:
 *      infile: the input file
 *      outfile: the output file
 *      hciph: libgcrypt cipher handle
 *      ciph: the libgcrypt cipher to use
 *    Returns: 0 on success, 1 on failure
 */
int encrypt_block(FILE *infile, FILE *outfile,
                  gcry_cipher_hd_t hciph, int ciph) {
    gcry_error_t err;
    enc_block_t out;
    unsigned char buf[BLOCK_SIZE];
    int nchars, ivlen;
    void *iv;

    // Zero the buffer so we get padding "for free"
    memset(buf, 0, BLOCK_SIZE);

    // Get a 16-byte random IV
    iv = gcry_random_bytes(16, GCRY_STRONG_RANDOM);

    // We may not use the entire IV; some ciphers have a smaller
    // blocksize
    ivlen = gcry_cipher_get_algo_blklen(ciph);
    err = gcry_cipher_setiv(hciph, iv, ivlen);
    CHECK_ERROR("Error setting IV");

    memcpy(out.iv, iv, 16);

    out.size = fread(buf, 1, BLOCK_SIZE, infile);
    if (out.size < BLOCK_SIZE && ferror(infile)) {
        fprintf(stderr, "File read error, aborting.\n");
        return 1;
    }

    // If we didn't get any data, its probably EOF
    if (out.size == 0) {
        if (feof(infile)) {
            return 0;
        }
        else {
            fprintf(stderr, "WARNING: read 0 bytes, but end "
                            "of file not reached.\n");
            return 1;
        }
    }

    err = gcry_cipher_encrypt(hciph, out.crybuf, BLOCK_SIZE,
                              buf, BLOCK_SIZE);
    CHECK_ERROR("Error encrypting block");

    nchars = fwrite(&out, 1, sizeof(enc_block_t), outfile);
    fprintf(stderr, "read %d bytes, wrote %d bytes\n", out.size, nchars);

    return 0;
}
```

```
/** decrypt_block: decrypts one block of the input file
 *                 and writes it to the output file
 *   Arguments:
 *     infile: the input file
 *     outfile: the output file
 *     hciph: libgcrypt cipher handle
 *     ciph: the libgcrypt cipher to use
 *   Returns: 0 on success, 1 on failure
 */
int decrypt_block(FILE *infile, FILE *outfile,
                  gcry_cipher_hd_t hciph, int ciph) {
    gcry_error_t err;
    enc_block_t in;
    int nchars, ivlen;
    unsigned char buf[BLOCK_SIZE];

    nchars = fread(&in, sizeof(in), 1, infile);
    if(!nchars) {
        if(feof(infile)) {
            // All good, we're done
            return 0;
        }
        else {
            fprintf(stderr, "File␣read␣error,␣aborting.\n");
            return 1;
        }
    }

    // We may not use the entire IV; some ciphers have a smaller
    // blocksize
    ivlen = gcry_cipher_get_algo_blklen(ciph);
    err = gcry_cipher_setiv(hciph, in.iv, ivlen);
    CHECK_ERROR("Error␣setting␣IV");

    err = gcry_cipher_decrypt(hciph, buf, BLOCK_SIZE,
                              in.crybuf, BLOCK_SIZE);
    CHECK_ERROR("Error␣decrypting␣block");

    nchars = fwrite(buf, 1, in.size, outfile);
    fprintf(stderr, "read␣%d␣bytes,␣wrote␣%d␣bytes\n", BLOCK_SIZE, nchars);

    return 0;
}

/** encrypt_file: encrypts a whole file, block by block, using a key
 *                provided by the user
 *    Arguments:
 *       infile: the input file
 *       outfile: the output file
 *       cipher: the libgcrypt cipher to use
 *    Returns: 0 on success, 1 on failure
 */
int encrypt_file(FILE *infile, FILE *outfile, int cipher) {
    gcry_cipher_hd_t hciph;
    gcry_error_t err;
    void *key;
    size_t keysize = 0;

    // Use CBC mode to avoid security issues with ECB
    err = gcry_cipher_open(&hciph, cipher, GCRY_CIPHER_MODE_CBC,
                           GCRY_CIPHER_SECURE);
    CHECK_ERROR("Error␣opening␣cipher");

    // Get the appropriate key size for this cipher
    // We will truncate the key if necessary, and if there
    // isn't enough key material, we must abort
    keysize = gcry_cipher_get_algo_keylen(cipher);
    if(!keysize) {
        fprintf(stderr, "Error␣getting␣key␣length␣for␣cipher.\n");
        return 1;
    }

    if (keysize > MD5_DIGEST_SIZE) {
        fprintf(stderr, "Error:␣requested␣algorithm␣requires␣"
                        "more␣key␣material␣than␣available.\n");
        return 1;
    }

    key = get_key();

    err = gcry_cipher_setkey (hciph, key, keysize);
    CHECK_ERROR("Error␣setting␣key");

    // Encrypt block by block
    while (!feof(infile)) {
        err = encrypt_block(infile, outfile, hciph, cipher);
        if (err) {
            fprintf(stderr, "Error:␣failed␣to␣encrypt␣block\n");
            return 1;
        }
    }
    // Allocated by get_key, we need to free it here
    free(key);

    return 0;
}

/** encrypt_file: decrypts a whole file, block by block, using a key
 *                provided by the user
 *   Arguments:
 *     infile: the input file
 *     outfile: the output file
 *     cipher: the libgcrypt cipher to use
 *   Returns: 0 on success, 1 on failure
 */
int decrypt_file(FILE *infile, FILE *outfile, int cipher) {
    gcry_cipher_hd_t hciph;
    gcry_error_t err;
    void *key;
    size_t keysize = 0;

    // Use CBC mode to avoid security issues with ECB
    err = gcry_cipher_open(&hciph, cipher, GCRY_CIPHER_MODE_CBC,
                           GCRY_CIPHER_SECURE);
    CHECK_ERROR("Error␣opening␣cipher");

    keysize = gcry_cipher_get_algo_keylen(cipher);
    if(!keysize) {
        fprintf(stderr, "Error␣getting␣key␣length␣for␣cipher.\n");
        return 1;
    }

    if (keysize > MD5_DIGEST_SIZE) {
        fprintf(stderr, "Error:␣requested␣algorithm␣requires␣"
                        "more␣key␣material␣than␣available.\n");
        return 1;
    }

    key = get_key();

    err = gcry_cipher_setkey (hciph, key, keysize);
    CHECK_ERROR("Error␣setting␣key");

    // Decrypt block by block
    while (!feof(infile)) {
        err = decrypt_block(infile, outfile, hciph, cipher);
        if (err) {
            fprintf(stderr, "Error:␣failed␣to␣decrypt␣block\n");
            return 1;
        }
    }

    // Allocated by get_key, we need to free it here
    free(key);

    return 0;
}
```