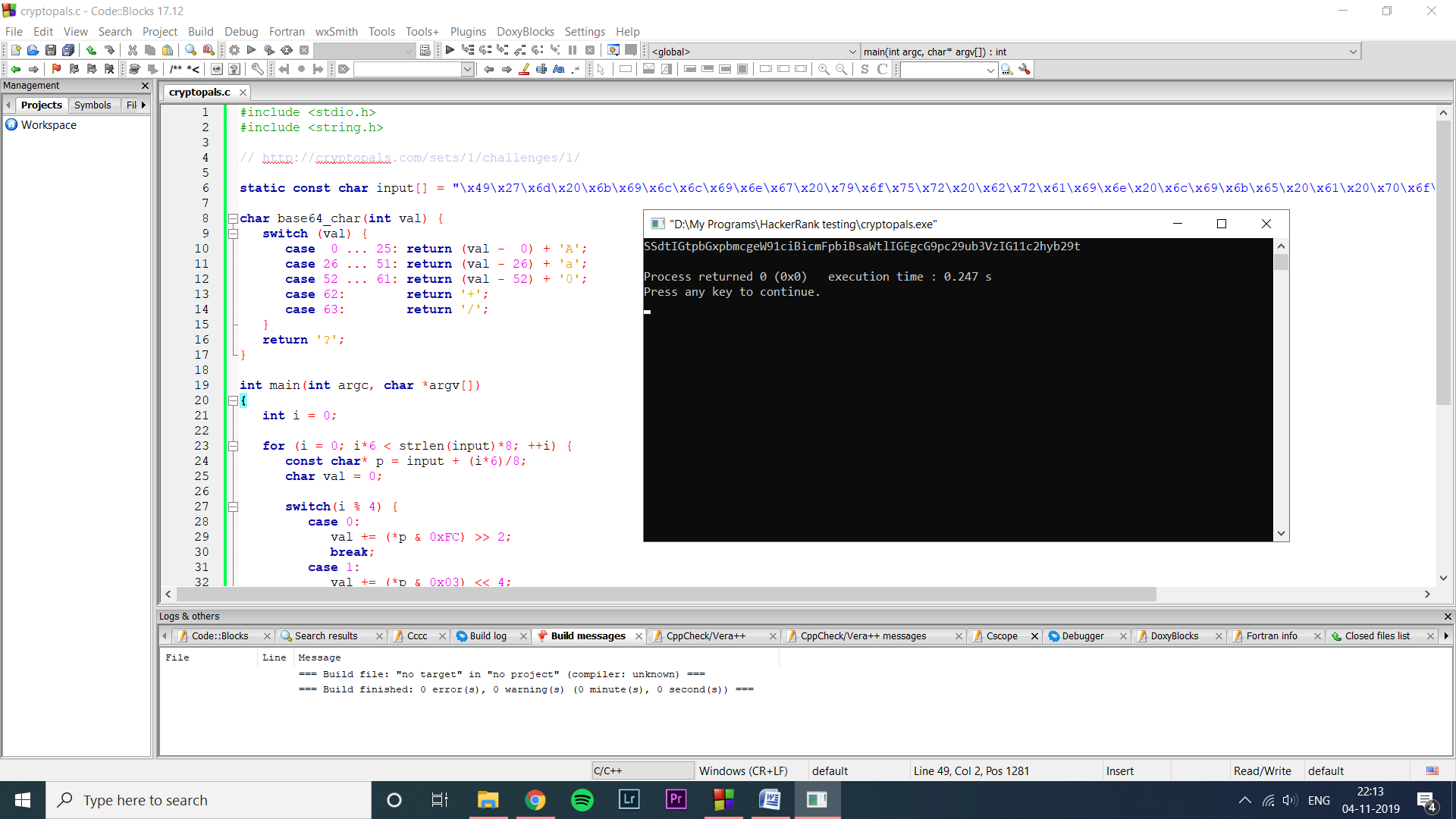
Challenge 1

Convert hex to base64

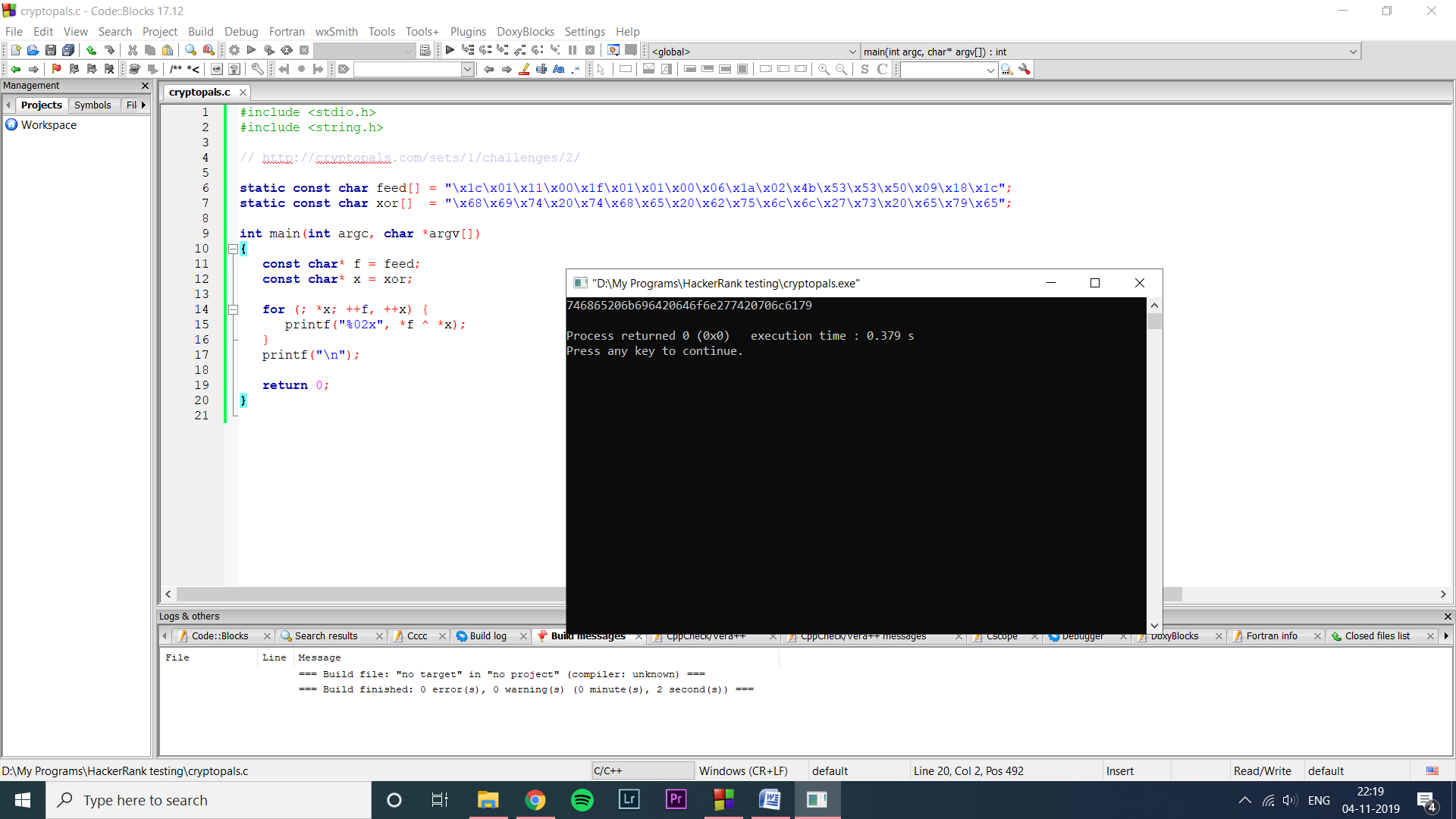
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/1/ |
|  |  |
|  | static const char input[] = "\x49\x27\x6d\x20\x6b\x69\x6c\x6c\x69\x6e\x67\x20\x79\x6f\x75\x72\x20\x62\x72\x61\x69\x6e\x20\x6c\x69\x6b\x65\x20\x61\x20\x70\x6f\x69\x73\x6f\x6e\x6f\x75\x73\x20\x6d\x75\x73\x68\x72\x6f\x6f\x6d"; |
|  |  |
|  | char base64\_char(int val) { |
|  | switch (val) { |
|  | case 0 ... 25: return (val - 0) + 'A'; |
|  | case 26 ... 51: return (val - 26) + 'a'; |
|  | case 52 ... 61: return (val - 52) + '0'; |
|  | case 62: return '+'; |
|  | case 63: return '/'; |
|  | } |
|  | return '?'; |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | int i = 0; |
|  |  |
|  | for (i = 0; i\*6 < strlen(input)\*8; ++i) { |
|  | const char\* p = input + (i\*6)/8; |
|  | char val = 0; |
|  |  |
|  | switch(i % 4) { |
|  | case 0: |
|  | val += (\*p & 0xFC) >> 2; |
|  | break; |
|  | case 1: |
|  | val += (\*p & 0x03) << 4; |
|  | val += \*(p+1) >> 4; |
|  | break; |
|  | case 2: |
|  | val += (\*p & 0x0F) << 2; |
|  | val += \*(p+1) >> 6; |
|  | break; |
|  | case 3: |
|  | val += (\*p & 0x3f); |
|  | val += \*(p+1) >> 8; |
|  | break; |
|  | } |
|  | printf("%c", base64\_char(val)); |
|  | } |
|  | printf("\n"); |
|  |  |
|  | return 0; |
|  | } |



Challenge 2

Fixed XOR

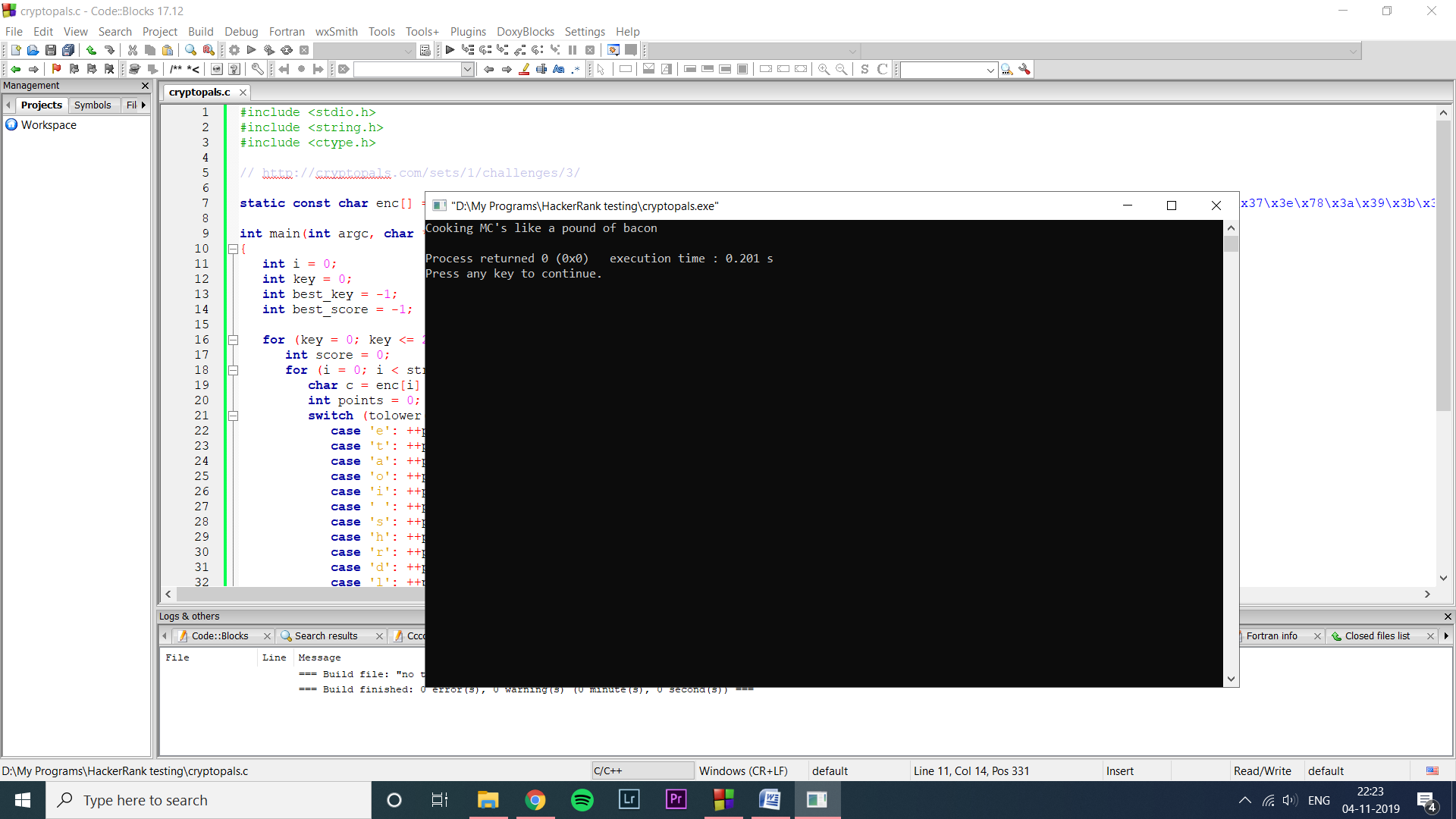
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/2/ |
|  |  |
|  | static const char feed[] = "\x1c\x01\x11\x00\x1f\x01\x01\x00\x06\x1a\x02\x4b\x53\x53\x50\x09\x18\x1c"; |
|  | static const char xor[] = "\x68\x69\x74\x20\x74\x68\x65\x20\x62\x75\x6c\x6c\x27\x73\x20\x65\x79\x65"; |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | const char\* f = feed; |
|  | const char\* x = xor; |
|  |  |
|  | for (; \*x; ++f, ++x) { |
|  | printf("%02x", \*f ^ \*x); |
|  | } |
|  | printf("\n"); |
|  |  |
|  | return 0; |
|  | } |



Challenge 3

Single-byte XOR cipher

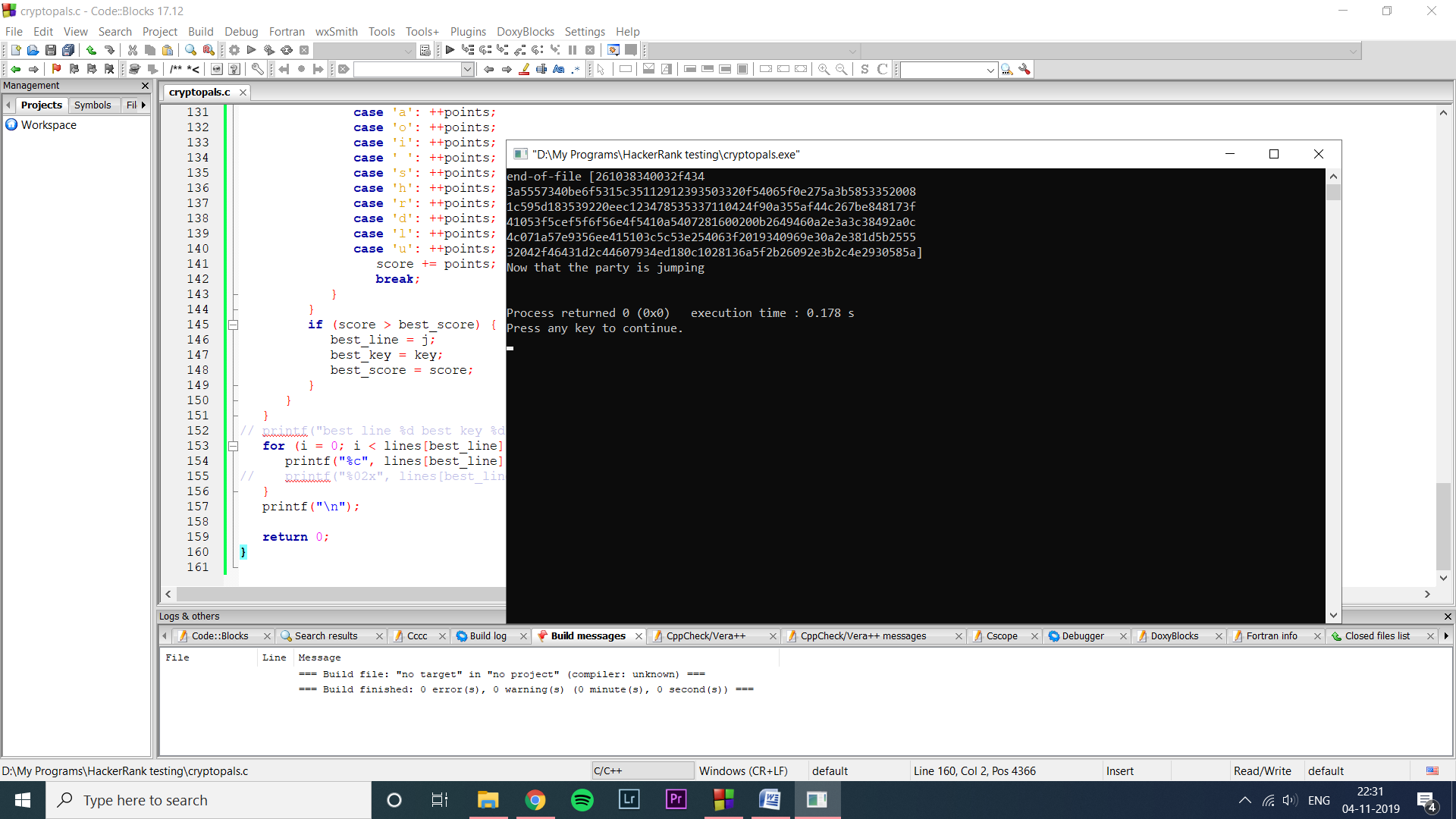
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/3/ |
|  |  |
|  | static const char enc[] = "\x1b\x37\x37\x33\x31\x36\x3f\x78\x15\x1b\x7f\x2b\x78\x34\x31\x33\x3d\x78\x39\x78\x28\x37\x2d\x36\x3c\x78\x37\x3e\x78\x3a\x39\x3b\x37\x36"; |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | int i = 0; |
|  | int key = 0; |
|  | int best\_key = -1; |
|  | int best\_score = -1; |
|  |  |
|  | for (key = 0; key <= 255; ++key) { |
|  | int score = 0; |
|  | for (i = 0; i < strlen(enc); ++i) { |
|  | char c = enc[i] ^ (char)key; |
|  | int points = 0; |
|  | switch (tolower(c)) { |
|  | case 'e': ++points; |
|  | case 't': ++points; |
|  | case 'a': ++points; |
|  | case 'o': ++points; |
|  | case 'i': ++points; |
|  | case ' ': ++points; |
|  | case 's': ++points; |
|  | case 'h': ++points; |
|  | case 'r': ++points; |
|  | case 'd': ++points; |
|  | case 'l': ++points; |
|  | case 'u': ++points; |
|  | score += points; |
|  | break; |
|  | } |
|  | } |
|  | if (score > best\_score) { |
|  | best\_key = key; |
|  | best\_score = score; |
|  | } |
|  | } |
|  | for (i = 0; i < strlen(enc); ++i) { |
|  | printf("%c", enc[i] ^ (char)best\_key); |
|  | } |
|  | printf("\n"); |
|  | return 0; |
|  | } |



Challenge 4

Detect single-character XOR

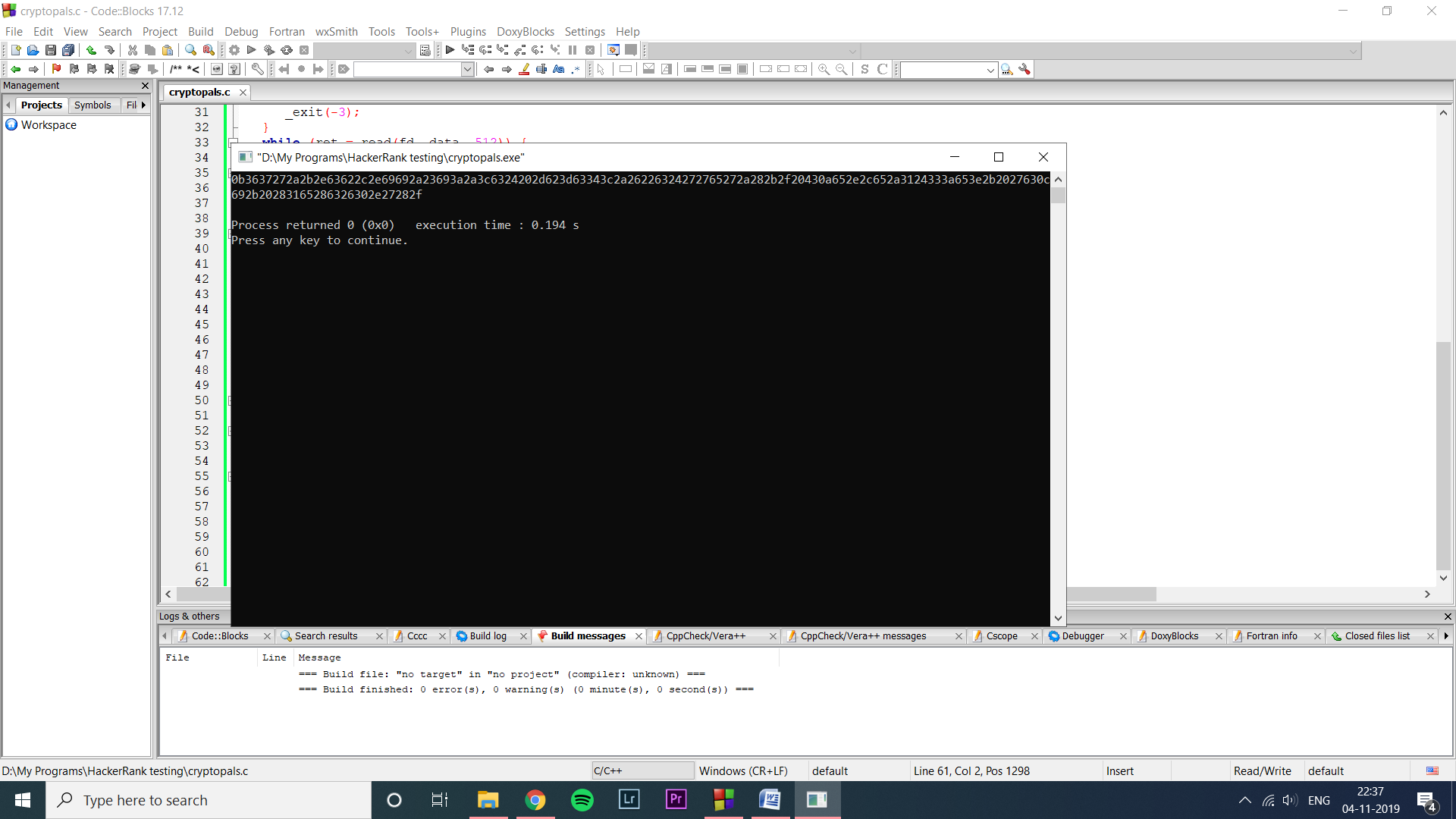
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <unistd.h> |
|  | #include <stdlib.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/4/ |
|  |  |
|  | struct one\_line { |
|  | unsigned char\* pos; |
|  | size\_t len; |
|  | }; |
|  |  |
|  | unsigned char hex\_ascii\_to\_bin(unsigned char c) { |
|  | switch (c) { |
|  | case '0' ... '9' : return c - '0'; |
|  | case 'a' ... 'f' : return c - 'a' + 10; |
|  | case 'A' ... 'F' : return c - 'A' + 10; |
|  | } |
|  | return '\0'; |
|  | } |
|  |  |
|  | struct one\_line\* read\_file(const char\* filename) |
|  | { |
|  | unsigned char\* data = NULL; |
|  | ssize\_t total = 0; |
|  | size\_t filesize = 0; |
|  | size\_t lines = 0; |
|  | struct one\_line\* all\_lines = NULL; |
|  |  |
|  | { // get file size |
|  | struct stat info; |
|  | if (stat(filename, &info)) { |
|  | printf("failed to get file info [%s]: %m\n", filename); |
|  | \_exit(-1); |
|  | } |
|  | filesize = info.st\_size; |
|  | } |
|  | { // allocate buffer |
|  | data = (char\*)malloc(filesize); |
|  | if (!data) { |
|  | printf("failed to allocate buffer\n"); |
|  | \_exit(-2); |
|  | } |
|  | } |
|  | { // read file to buffer |
|  | int fd = open(filename, O\_RDONLY); |
|  | if (0 > fd) { |
|  | printf("failed to open file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | while (total < (ssize\_t)filesize) { |
|  | ssize\_t now = read(fd, &data[total], filesize - total); |
|  | if (0 > now) { |
|  | printf("error while reading from file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | if (0 == now) { |
|  | printf("end-of-file [%s]\n"); |
|  | break; |
|  | } |
|  | total += now; |
|  | } |
|  | close(fd); |
|  | } |
|  | { // count lines |
|  | char\* p = data; |
|  | for (lines = 0; (p = strchr(p, '\n')); ++lines, ++p) {} |
|  | } |
|  | { // allocate Lines buffer |
|  | all\_lines = (struct one\_line\*)malloc(sizeof(struct one\_line) \* (lines + 1)); |
|  | if (!all\_lines) { |
|  | printf("failed to allocate lines\n"); |
|  | \_exit(-3); |
|  | } |
|  | } |
|  | { // store lines |
|  | size\_t i = 0; |
|  | unsigned char c = 0; |
|  | size\_t pos = 0; |
|  | struct one\_line\* this\_line = all\_lines; |
|  | this\_line->pos = data; |
|  |  |
|  | // walk the buffer |
|  | for (i = 0, pos = 0; i < filesize; ++i) { |
|  | // on newline, end this line, start a new one |
|  | if ('\n' == data[i]) { |
|  | this\_line->len = &data[pos] - this\_line->pos; |
|  | ++this\_line; |
|  | this\_line->pos = &data[pos]; |
|  | continue; |
|  | } |
|  | // otherwise convert ascii hex to bin hex |
|  | if (i % 2) { |
|  | c += hex\_ascii\_to\_bin(data[i]); |
|  | data[pos] = c; |
|  | ++pos; |
|  | continue; |
|  | } |
|  | c = hex\_ascii\_to\_bin(data[i]) << 4; |
|  | } |
|  | this\_line->pos = 0; // terminator |
|  | this\_line->len = 0; // |
|  | } |
|  | return all\_lines; |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | int i = 0; |
|  | int j = 0; |
|  | int best\_line = -1; |
|  | int best\_key = -1; |
|  | int best\_score = -1; |
|  | int best\_input = -1; |
|  | struct one\_line\* lines = read\_file("4.txt"); |
|  |  |
|  | for (j = 0; (lines[j].pos && lines[j].len); ++j) { |
|  | int key = 0; |
|  | for (key = 0; key <= 255; ++key) { |
|  | int score = 0; |
|  | for (i = 0; i < lines[j].len; ++i) { |
|  | char c = lines[j].pos[i] ^ (char)key; |
|  | int points = 0; |
|  | switch (tolower(c)) { |
|  | case 'e': ++points; |
|  | case 't': ++points; |
|  | case 'a': ++points; |
|  | case 'o': ++points; |
|  | case 'i': ++points; |
|  | case ' ': ++points; |
|  | case 's': ++points; |
|  | case 'h': ++points; |
|  | case 'r': ++points; |
|  | case 'd': ++points; |
|  | case 'l': ++points; |
|  | case 'u': ++points; |
|  | score += points; |
|  | break; |
|  | } |
|  | } |
|  | if (score > best\_score) { |
|  | best\_line = j; |
|  | best\_key = key; |
|  | best\_score = score; |
|  | } |
|  | } |
|  | } |
|  | // printf("best line %d best key %d\n", best\_line, best\_key); |
|  | for (i = 0; i < lines[best\_line].len; ++i) { |
|  | printf("%c", lines[best\_line].pos[i] ^ (char)best\_key); |
|  | // printf("%02x", lines[best\_line].pos[i]); |
|  | } |
|  | printf("\n"); |
|  |  |
|  | return 0; |
|  | } |



Challenge 5

Implement repeating-key XOR

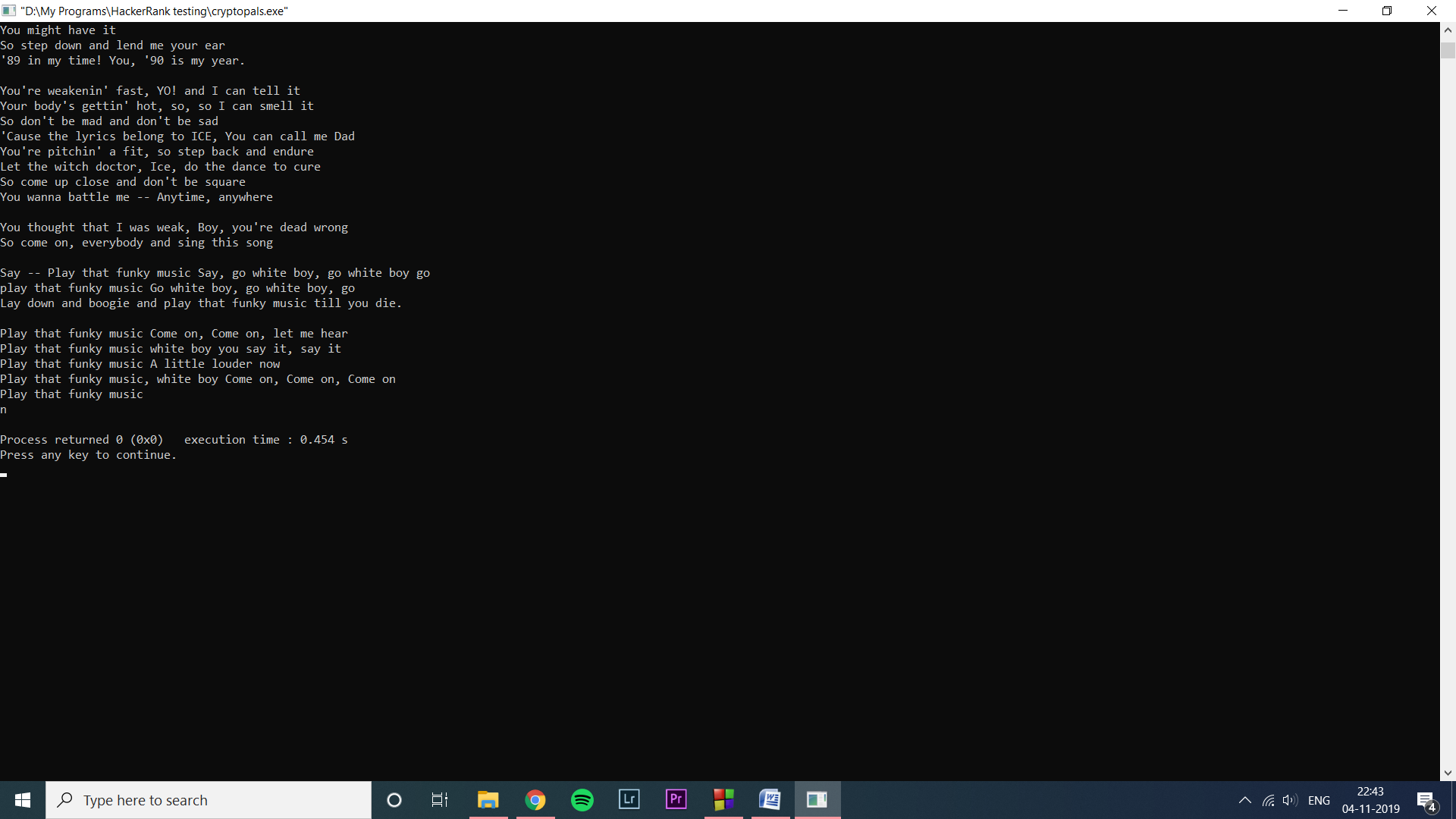
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <unistd.h> |
|  | #include <stdlib.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/5/ |
|  |  |
|  | const char const key[] = "ICE"; |
|  |  |
|  | void read\_and\_encrypt\_file(const char\* filename) |
|  | { |
|  | unsigned char\* data = NULL; |
|  | int fd = -1; |
|  | size\_t key\_index = 0; |
|  | ssize\_t ret = 0; |
|  |  |
|  | // allocate buffer |
|  | data = (char\*)malloc(512); |
|  | if (!data) { |
|  | printf("failed to allocate buffer\n"); |
|  | \_exit(-2); |
|  | } |
|  | // read file to buffer |
|  | fd = open(filename, O\_RDONLY); |
|  | if (0 > fd) { |
|  | printf("failed to open file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | while (ret = read(fd, data, 512)) { |
|  | int i = 0; |
|  | if (0 > ret) { |
|  | printf("error while reading from file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | for (i = 0; i < ret; ++i) { |
|  | printf("%02x", data[i] ^ key[key\_index]); |
|  | ++key\_index; |
|  | key\_index %= strlen(key); |
|  | } |
|  | } |
|  | close(fd); |
|  | printf("\n"); |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | const char\* file = NULL; |
|  | if (2 <= argc) { |
|  | file = argv[1]; |
|  | } |
|  | else { |
|  | file = "5.txt"; |
|  | } |
|  | read\_and\_encrypt\_file(file); |
|  |  |
|  | return 0; |
|  | } |



Challenge 6

Break repeating-key XOR

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <unistd.h> |
|  | #include <stdlib.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/6/ |
|  |  |
|  | unsigned char hex\_ascii\_to\_bin(unsigned char c) { |
|  | switch (c) { |
|  | case '0' ... '9' : return c - '0'; |
|  | case 'a' ... 'f' : return c - 'a' + 10; |
|  | case 'A' ... 'F' : return c - 'A' + 10; |
|  | } |
|  | return '\0'; |
|  | } |
|  |  |
|  | unsigned char\* read\_file(const char\* filename, size\_t\* len) |
|  | { |
|  | unsigned char\* data = NULL; |
|  | ssize\_t total = 0; |
|  | size\_t filesize = 0; |
|  |  |
|  | { // get file size |
|  | struct stat info; |
|  | if (stat(filename, &info)) { |
|  | printf("failed to get file info [%s]: %m\n", filename); |
|  | \_exit(-1); |
|  | } |
|  | filesize = info.st\_size; |
|  | } |
|  | { // allocate buffer |
|  | data = (char\*)malloc(filesize); |
|  | if (!data) { |
|  | printf("failed to allocate buffer\n"); |
|  | \_exit(-2); |
|  | } |
|  | } |
|  | { // read file to buffer |
|  | int fd = open(filename, O\_RDONLY); |
|  | if (0 > fd) { |
|  | printf("failed to open file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | while (total < (ssize\_t)filesize) { |
|  | ssize\_t now = read(fd, &data[total], filesize - total); |
|  | if (0 > now) { |
|  | printf("error while reading from file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | if (0 == now) { |
|  | printf("end-of-file [%s]\n"); |
|  | break; |
|  | } |
|  | total += now; |
|  | } |
|  | \*len = (size\_t)total; |
|  | close(fd); |
|  | } |
|  | return data; |
|  | } |
|  |  |
|  | char char\_to\_base64(int val) { |
|  | switch (val) { |
|  | case 'A' ... 'Z': return (val + 0) - 'A'; |
|  | case 'a' ... 'z': return (val + 26) - 'a'; |
|  | case '0' ... '9': return (val + 52) - '0'; |
|  | case '+': return 62; |
|  | case '/': return 63; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | void base64\_to\_bin(unsigned char\* data, size\_t \*len) |
|  | { |
|  | size\_t i = 0; |
|  | size\_t j = 0; |
|  | unsigned int c = 0; |
|  | size\_t shift = 3; |
|  |  |
|  | // walk the buffer |
|  | for (i = 0; i < \*len; ++i) { |
|  | // skip newlines |
|  | if ('\n' == data[i]) { |
|  | continue; |
|  | } |
|  | // convert base64 to bin hex |
|  | c += char\_to\_base64(data[i]) << (6 \* shift); |
|  | if (0 == shift) { |
|  | data[j++] = c >> (2 \* 8); |
|  | data[j++] = c >> (1 \* 8); |
|  | data[j++] = c >> (0 \* 8); |
|  | shift = 3; |
|  | c = 0; |
|  | } else { |
|  | --shift; |
|  | } |
|  | } |
|  | \*len = j; |
|  | } |
|  |  |
|  | size\_t hamming\_distance(const unsigned char\* a, const unsigned char\* b, size\_t len) |
|  | { |
|  | size\_t dist = 0; |
|  | size\_t i = 0; |
|  | for (i = 0; i < len; ++i) { |
|  | unsigned char c = a[i] ^ b[i]; |
|  | while (c) { |
|  | if (c & 0x01) { ++dist; } |
|  | c = (c & 0xff) >> 1; |
|  | } |
|  | } |
|  | return dist; |
|  | } |
|  |  |
|  | size\_t find\_best\_key\_size(unsigned char\* data, size\_t data\_len) |
|  | { |
|  | size\_t best\_size = 0; |
|  | size\_t best\_score = 1000000; |
|  | int key = 0; |
|  |  |
|  | for (key = 2; key <= 40; ++key) { |
|  | int i = 0; |
|  | size\_t score = 0; |
|  | for (i = 0; i < 10; ++i) { // do the test twice |
|  | score += hamming\_distance(&data[i\*key], &data[(i+1)\*key], key); |
|  | } |
|  | score = score \* 10000 / key; // normalize |
|  | //printf("score for key %d is %d\n", key, score); |
|  | if (score < best\_score) { |
|  | best\_score = score; |
|  | best\_size = key; |
|  | } |
|  | } |
|  | //printf("found best key length %d with score %d\n", best\_size, best\_score); |
|  | return best\_size; |
|  | } |
|  |  |
|  | int score\_for\_char(char c) |
|  | { |
|  | int points = 0; |
|  | switch (tolower(c)) { |
|  | case 'e': ++points; |
|  | case 't': ++points; |
|  | case 'a': ++points; |
|  | case 'o': ++points; |
|  | case 'i': ++points; |
|  | case ' ': ++points; |
|  | case 's': ++points; |
|  | case 'h': ++points; |
|  | case 'r': ++points; |
|  | case 'd': ++points; |
|  | case 'l': ++points; |
|  | case 'u': ++points; |
|  | break; |
|  | // case 0x80 ... 0xff: points = -1; |
|  | } |
|  | return points; |
|  | } |
|  |  |
|  | void calculate\_key(const unsigned char\* data, size\_t data\_len, unsigned char\* key, size\_t key\_len) |
|  | { |
|  | int i = 0; |
|  |  |
|  | for (i = 0; i < key\_len; ++i) { |
|  | // calculate single byte at 'i' |
|  | unsigned char best\_byte = 0; |
|  | int best\_score = 0; |
|  | int byte = 0; |
|  | for (byte = 0; byte <= 255; ++byte) { |
|  | int j = 0; |
|  | int score = 0; |
|  | for (j = i; j < data\_len; j += key\_len) { |
|  | score += score\_for\_char(data[j] ^ (unsigned char)byte); |
|  | } |
|  | if (score > best\_score) { |
|  | best\_score = score; |
|  | best\_byte = byte; |
|  | } |
|  | } |
|  | key[i] = best\_byte; |
|  | //printf("best byte for %d is 0x%02x with score %d\n", i, best\_byte, best\_score); |
|  | } |
|  | } |
|  |  |
|  | void decrypt\_repeated\_key(const unsigned char\* data, size\_t data\_len, unsigned char\* key, size\_t key\_size) |
|  | { |
|  | size\_t i = 0; |
|  | size\_t k = 0; |
|  | for (i = 0; i < data\_len; ++i) { |
|  | printf("%c", data[i] ^ key[k]); |
|  | ++k; |
|  | k %= key\_size; |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | size\_t data\_len = 0; |
|  | unsigned char\* data = NULL; |
|  | size\_t key\_size = 0; |
|  | unsigned char\* key = NULL; |
|  |  |
|  | { // test Hamming distance |
|  | const char a[] = "this is a test"; |
|  | const char b[] = "wokka wokka!!!"; |
|  | size\_t dist = hamming\_distance(a, b, strlen(a)); |
|  | if (37 != dist) { |
|  | printf("Hamming #define istance %d is not 37\n", dist); |
|  | \_exit(-1); |
|  | } |
|  | } |
|  |  |
|  | // read file to buffer |
|  | data = read\_file("6.txt", &data\_len); |
|  |  |
|  | // convert base64 to binary data |
|  | base64\_to\_bin(data, &data\_len); |
|  |  |
|  | // find best key size |
|  | key\_size = find\_best\_key\_size(data, data\_len); |
|  |  |
|  | // calculate key |
|  | key = (unsigned char\*)malloc(key\_size); |
|  | calculate\_key(data, data\_len, key, key\_size); |
|  |  |
|  | // decrypt |
|  | decrypt\_repeated\_key(data, data\_len, key, key\_size); |
|  |  |
|  | printf("\n"); |
|  |  |
|  | // TODO free buffers |
|  | return 0; |
|  | } |



Challenge 7

AES in ECB mode

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <unistd.h> |
|  | #include <stdlib.h> |
|  |  |
|  | #include <polarssl/aes.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/7/ |
|  |  |
|  | unsigned char\* read\_file(const char\* filename, size\_t\* len) |
|  | { |
|  | unsigned char\* data = NULL; |
|  | ssize\_t total = 0; |
|  | size\_t filesize = 0; |
|  |  |
|  | { // get file size |
|  | struct stat info; |
|  | if (stat(filename, &info)) { |
|  | printf("failed to get file info [%s]: %m\n", filename); |
|  | \_exit(-1); |
|  | } |
|  | filesize = info.st\_size; |
|  | } |
|  | { // allocate buffer |
|  | data = (char\*)malloc(filesize); |
|  | if (!data) { |
|  | printf("failed to allocate buffer\n"); |
|  | \_exit(-2); |
|  | } |
|  | } |
|  | { // read file to buffer |
|  | int fd = open(filename, O\_RDONLY); |
|  | if (0 > fd) { |
|  | printf("failed to open file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | while (total < (ssize\_t)filesize) { |
|  | ssize\_t now = read(fd, &data[total], filesize - total); |
|  | if (0 > now) { |
|  | printf("error while reading from file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | if (0 == now) { |
|  | printf("end-of-file [%s]\n"); |
|  | break; |
|  | } |
|  | total += now; |
|  | } |
|  | \*len = (size\_t)total; |
|  | close(fd); |
|  | } |
|  | return data; |
|  | } |
|  |  |
|  | char char\_to\_base64(int val) { |
|  | switch (val) { |
|  | case 'A' ... 'Z': return (val + 0) - 'A'; |
|  | case 'a' ... 'z': return (val + 26) - 'a'; |
|  | case '0' ... '9': return (val + 52) - '0'; |
|  | case '+': return 62; |
|  | case '/': return 63; |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | void base64\_to\_bin(unsigned char\* data, size\_t \*len) |
|  | { |
|  | size\_t i = 0; |
|  | size\_t j = 0; |
|  | unsigned int c = 0; |
|  | size\_t shift = 3; |
|  |  |
|  | // walk the buffer |
|  | for (i = 0; i < \*len; ++i) { |
|  | // skip newlines |
|  | if ('\n' == data[i]) { |
|  | continue; |
|  | } |
|  | // convert base64 to bin hex |
|  | c += char\_to\_base64(data[i]) << (6 \* shift); |
|  | if (0 == shift) { |
|  | data[j++] = c >> (2 \* 8); |
|  | data[j++] = c >> (1 \* 8); |
|  | data[j++] = c >> (0 \* 8); |
|  | shift = 3; |
|  | c = 0; |
|  | } else { |
|  | --shift; |
|  | } |
|  | } |
|  | \*len = j; |
|  | } |
|  |  |
|  | void decrypt\_aes\_ecb(unsigned char\* data, size\_t data\_len, unsigned char\* key, size\_t key\_len) |
|  | { |
|  | int ret = 0; |
|  | aes\_context ctx; |
|  | unsigned char decr[16]; |
|  | int i = 0; |
|  |  |
|  | ret = aes\_setkey\_dec(&ctx, key, key\_len\*8); // length in bits |
|  | if (ret) { |
|  | printf("invalid AES key length %d\n", key\_len); |
|  | \_exit(-20); |
|  | } |
|  | for (i = 0; i < data\_len; i += 16) { |
|  | int j = 0; |
|  | ret = aes\_crypt\_ecb(&ctx, AES\_DECRYPT, &data[i], decr); |
|  | if (ret) { |
|  | printf("failed to decrypt\n"); |
|  | \_exit(-21); |
|  | } |
|  | for (j = 0; j < 16; ++j) { |
|  | printf("%c", decr[j]); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | size\_t data\_len = 0; |
|  | unsigned char\* data = NULL; |
|  | unsigned char key[] = "YELLOW SUBMARINE"; |
|  |  |
|  | // read file to buffer |
|  | data = read\_file("7.txt", &data\_len); |
|  |  |
|  | // convert base64 to binary data |
|  | base64\_to\_bin(data, &data\_len); |
|  |  |
|  | // decrypt |
|  | decrypt\_aes\_ecb(data, data\_len, key, sizeof(key)-1 ); |
|  |  |
|  | printf("\n"); |
|  |  |
|  | // TODO free buffers |
|  | return 0; |
|  | } |

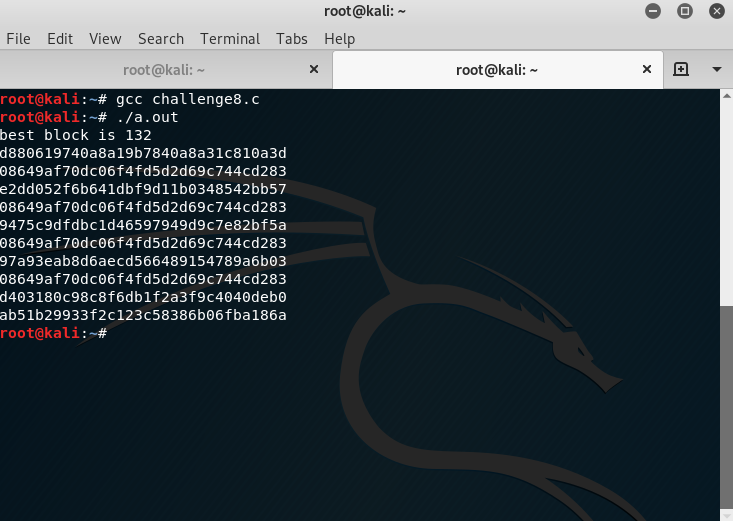
Decrypted text

I'm back and I'm ringin' the bell A rockin' on the mike while the fly girls yell In ecstasy in the back of me Well that's my DJ Deshay cuttin' all them Z's Hittin' hard and the girlies goin' crazy Vanilla's on the mike, man I'm not lazy. I'm lettin' my drug kick in It controls my mouth and I begin To just let it flow, let my concepts go My posse's to the side yellin', Go Vanilla Go! Smooth 'cause that's the way I will be And if you don't give a damn, then Why you starin' at me So get off 'cause I control the stage There's no dissin' allowed I'm in my own phase The girlies sa y they love me and that is ok And I can dance better than any kid n' play Stage 2 -- Yea the one ya' wanna listen to It's off my head so let the beat play through So I can funk it up and make it sound good 1-2-3 Yo -- Knock on some wood For good luck, I like my rhymes atrocious Supercalafragilisticexpialidocious I'm an effect and that you can bet I can take a fly girl and make her wet. I'm like Samson -- Samson to Delilah There's no denyin', You can try to hang But you'll keep tryin' to get my style Over and over, practice makes perfect But not if you're a loafer. You'll get nowhere, no place, no time, no girls Soon -- Oh my God, homebody, you probably eat Spaghetti with a spoon! Come on and say it! VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino Intoxicating so you stagger like a wino So punks stop trying and girl stop cryin' Vanilla Ice is sellin' and you people are buyin' 'Cause why the freaks are jockin' like Crazy Glue Movin' and groovin' trying to sing along All through the ghetto groovin' this here song Now you're amazed by the VIP posse. Steppin' so hard like a German Nazi Startled by the bases hittin' ground There's no trippin' on mine, I'm just gettin' down Sparkamatic, I'm hangin' tight like a fanatic You trapped me once and I thought that You might have it So step down and lend me your ear '89 in my time! You, '90 is my year. You're weakenin' fast, YO! and I can tell it Your body's gettin' hot, so, so I can smell it So don't be mad and don't be sad 'Cause the lyrics belong to ICE, You can call me Dad You're pitchin' a fit, so step back and endure Let the witch doctor, Ice, do the dance to cure So come up close and don't be square You wanna battle me -- Anytime, anywhere You thought that I was weak, Boy, you're dead wrong So come on, everybody and sing this song Say -- Play that funky music Say, go white boy, go white boy go play that funky music Go white boy, go white boy, go Lay down and boogie and play that funky music till you die. Play that funky music Come on, Come on, let me hear Play that funky music white boy you say it, say it Play that funky music A little louder now Play that funky music, white boy Come on, Come on, Come on Play that funky music

Challenge 8

Detect AES in ECB mode

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <ctype.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <unistd.h> |
|  | #include <stdlib.h> |
|  |  |
|  | // http://cryptopals.com/sets/1/challenges/8/ |
|  |  |
|  | struct block { |
|  | unsigned char\* pos; |
|  | size\_t len; |
|  | }; |
|  |  |
|  | unsigned char hex\_ascii\_to\_bin(unsigned char c) { |
|  | switch (c) { |
|  | case '0' ... '9' : return c - '0'; |
|  | case 'a' ... 'f' : return c - 'a' + 10; |
|  | case 'A' ... 'F' : return c - 'A' + 10; |
|  | } |
|  | return '\0'; |
|  | } |
|  |  |
|  | unsigned char\* read\_file(const char\* filename, size\_t\* len) |
|  | { |
|  | unsigned char\* data = NULL; |
|  | ssize\_t total = 0; |
|  | size\_t filesize = 0; |
|  |  |
|  | { // get file size |
|  | struct stat info; |
|  | if (stat(filename, &info)) { |
|  | printf("failed to get file info [%s]: %m\n", filename); |
|  | \_exit(-1); |
|  | } |
|  | filesize = info.st\_size; |
|  | } |
|  | { // allocate buffer |
|  | data = (char\*)malloc(filesize); |
|  | if (!data) { |
|  | printf("failed to allocate buffer\n"); |
|  | \_exit(-2); |
|  | } |
|  | } |
|  | { // read file to buffer |
|  | int fd = open(filename, O\_RDONLY); |
|  | if (0 > fd) { |
|  | printf("failed to open file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | while (total < (ssize\_t)filesize) { |
|  | ssize\_t now = read(fd, &data[total], filesize - total); |
|  | if (0 > now) { |
|  | printf("error while reading from file [%s]: %m\n", filename); |
|  | \_exit(-3); |
|  | } |
|  | if (0 == now) { |
|  | printf("end-of-file\n"); |
|  | break; |
|  | } |
|  | total += now; |
|  | } |
|  | \*len = (size\_t)total; |
|  | close(fd); |
|  | } |
|  | return data; |
|  | } |
|  |  |
|  | struct block\* hex\_to\_bin\_blocks(unsigned char\* data, size\_t len) |
|  | { |
|  | size\_t lines = 0; |
|  | struct block\* blocks = NULL; |
|  |  |
|  | { // count lines |
|  | unsigned char\* p = data; |
|  | for (lines = 0; p < (data + len); ++lines, ++p) { |
|  | p = strchr(p, '\n'); |
|  | } |
|  | } |
|  | { // allocate blocks buffer |
|  | blocks = (struct block\*)malloc(sizeof(struct block) \* (lines + 1)); |
|  | if (!blocks) { |
|  | printf("failed to allocate blocks\n"); |
|  | \_exit(-3); |
|  | } |
|  | } |
|  | { // store lines |
|  | size\_t i = 0; |
|  | unsigned char c = 0; |
|  | size\_t pos = 0; |
|  | struct block\* this = blocks; |
|  | this->pos = data; |
|  |  |
|  | // walk the buffer |
|  | lines = 0; |
|  | for (i = 0, pos = 0; i < len; ++i) { |
|  | // on newline, end this block, start a new one |
|  | if ('\n' == data[i]) { |
|  | //printf(" <-- len %d\n", i); |
|  | ++lines; |
|  | this->len = &data[pos] - this->pos; |
|  | ++this; |
|  | this->pos = &data[pos]; |
|  | continue; |
|  | } |
|  | //printf("%02x", data[i]); |
|  | // otherwise convert ascii hex to bin hex |
|  | if ((i - lines) % 2) { |
|  | c += hex\_ascii\_to\_bin(data[i]); |
|  | data[pos] = c; |
|  | //printf("%02x", data[pos]); |
|  | ++pos; |
|  | continue; |
|  | } |
|  | c = hex\_ascii\_to\_bin(data[i]) << 4; |
|  | } |
|  | this->pos = 0; // terminator |
|  | this->len = 0; // |
|  | } |
|  | return blocks; |
|  | } |
|  |  |
|  | int find\_best\_aes\_block(struct block\* blocks) |
|  | { |
|  | int best\_block = -1; |
|  | int best\_score = -1; |
|  | int i = 0; |
|  |  |
|  | for (i = 0; (blocks[i].pos && blocks[i].len); ++i) { |
|  | int score = 0; |
|  | int j = 0; |
|  | for (j = 0; j < blocks[i].len; j += 16) { |
|  | int k = 0; |
|  | for (k = j + 16; k < blocks[i].len; k += 16) { |
|  | if (0 == memcmp(&blocks[i].pos[j], &blocks[i].pos[k], 16)) { |
|  | ++score; |
|  | } |
|  | } |
|  | } |
|  | if (score > best\_score) { |
|  | //printf("score %d is better than %d for %d\n", score, best\_score, i); |
|  | best\_score = score; |
|  | best\_block = i; |
|  | } |
|  | } |
|  |  |
|  | return best\_block; |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | size\_t data\_len = 0; |
|  | unsigned char\* data = NULL; |
|  | struct block\* blocks = NULL; |
|  | int best = -1; |
|  |  |
|  | // read file to buffer |
|  | data = read\_file("8.txt", &data\_len); |
|  |  |
|  | // convert hex to binary data blocks |
|  | blocks = hex\_to\_bin\_blocks(data, data\_len); |
|  |  |
|  | // find best block |
|  | best = find\_best\_aes\_block(blocks); |
|  | printf("best block is %d\n", best); |
|  |  |
|  | // print it in blocks of 16 byte |
|  | { |
|  | int i = 0; |
|  | for (i = 0; i < blocks[best].len; ++i) { |
|  | printf("%02x", blocks[best].pos[i]); |
|  | if (15 == i % 16) { |
|  | printf("\n"); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // TODO free buffers |
|  | return 0; |
|  | } |



Challenge 1 out of 5

PART -2

# **Challenge 1:Hash - Message Digest 5**

#### **Statement**

Crack the following hash :

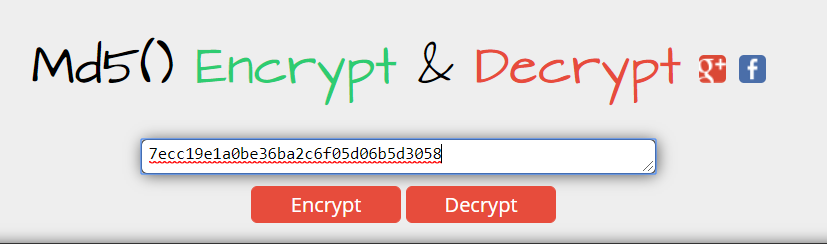
7ecc19e1a0be36ba2c6f05d06b5d3058

**Solution:**

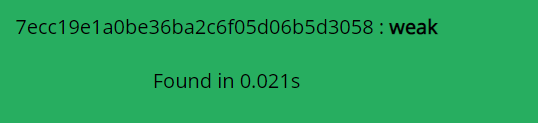
To get the hash password I used online tool named as Md5() Encrypt &Decrypt

Link of the tool : <https://md5decrypt.net/en/>

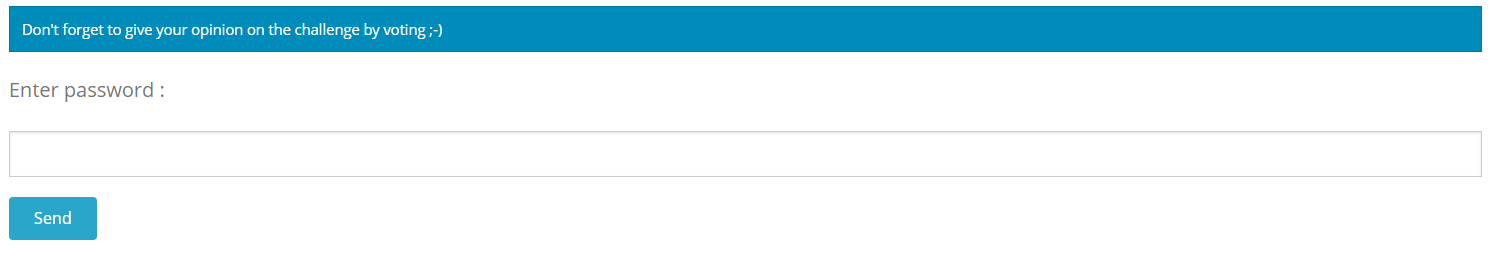
Step 1: Copy the hash you want get the password



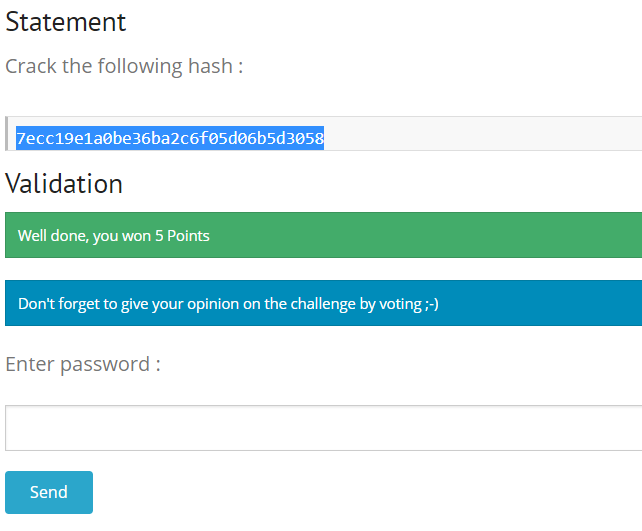
Step 2: Then Click on Decrypt. After Decrypting screen showed



Step 3:Enter the password : weak



Result



# Challenge 2 out of 5

# **2.Hash - SHA-2**

#### **Statement**

This hash was stolen during a session interception on a critical application, errors may have occurred during transmission. No crack attempt has resulted so far; hash format seems unknown. Find the corresponding plaintext.

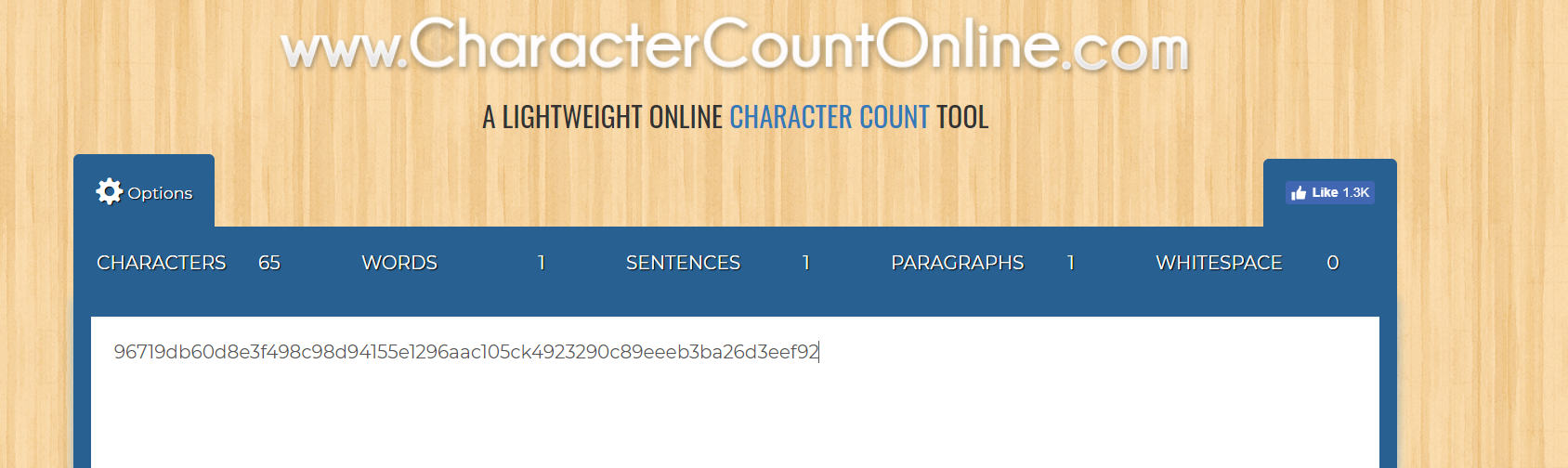
96719db60d8e3f498c98d94155e1296aac105ck4923290c89eeeb3ba26d3eef92

The answer is the SHA-1 of this password.

Solution:

Since the given key hash have errors let’s identify the errors.

Calculate the length of hash



Total no of characters are 65 to nearest SHA-256 and hexadecimal string 64 character long

The characters that sha-256 include ABCDEFGH

So if see the hash 96719db60d8e3f498c98d94155e1296aac105ck4923290c89eeeb3ba26d3eef92

The hash has “k” extra so now corrected hash –

96719db60d8e3f498c98d94155e1296aac105c4923290c89eeeb3ba26d3eef92

Now go to kali linux terminal

Type the command- **hash-identifier**



Let try for hash-256



Since the password is SHA-1.

Encrypt “4dM1n” in SHA-1 by encrypter

After encryption we get “**a7c9d5a37201c08c5b7b156173bea5ec2063edf9”**

**Now enter this as password:**

