CAA 24-25

Exercise Sheet on Randomness
Solutions

1 Code Review: Randomness Generation

1. Sometimes, the generated value contains a lot of zeros. One can also notice some patters in the generated bits that repeat to often.

2. The problem in the code is that g is not a generator of $\mathbb{Z}_p^*$. In fact, it has order 673, which is really low. This means that g^x can take only 673 different values. Another (less critical) problem is that the seed is too small.

3. In sage, it is simple to find a generator of $\mathbb{Z}_p^*$ with the following code

```
F = Integers(p)
print(F.multiplicative_generator())
```

   which returns 11. Replacing the value of g by 11 fixes the problem.

   To fix the seed problem, you can replace the seed generation by `secrets.randbelow(p)`.

2 RDSEED

The following code queries a random seed. We do 10 tries to RDSEED until we fail our program.

```
#include <stdint.h>
#include <stdio.h>

//Code from official documentation
int rdseed32_step (uint32_t *seed)
{
  unsigned char ok;

  asm volatile ("rdseed %0; setc %1"
    : "=r" (*seed), "=qm" (ok));

  return (int) ok;
}

int main(){
        uint32_t val = 0;
```

```
        uint8_t tries = 10; //We do 10 tries before failing
        while (!rdseed32_step(&val)){
                tries --;
                if (tries == 0)
                        return 1;
        }
        printf("%u\n", val);
        return 0;
}
```

3 Implement Diffie-Hellman using OpenSSL

```
#include <string.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/dh.h>
#include <openssl/err.h>
#include <openssl/pem.h>


/**
 * Print openSSL error messages
 */
void handleErrors(void)
{
        ERR_print_errors_fp(stderr);
        abort();
}

/**
 * Encode <length> bits of <input> using base64. Allocates memory for output.
 */
char* base64_encode(const unsigned char *input, int length)
{
        BIO *bmem, *b64;
        BUF_MEM *bptr;

        b64 = BIO_new(BIO_f_base64());
        bmem = BIO_new(BIO_s_mem());
        b64 = BIO_push(b64, bmem);
        BIO_write(b64, input, length);
        BIO_flush(b64);
        BIO_get_mem_ptr(b64, &bptr);

        char *buff = (char *)malloc(bptr->length);
        memcpy(buff, bptr->data, bptr->length-1);
        buff[bptr->length-1] = 0;

        BIO_free_all(b64);
        return buff;
}


/**
```

```
43  * Computes the DH secret using the private key pair <keyPair>, the received
        public part <peerPublicKey>.
44  * The result is stored into <*pSecret> (memory is allocated) and this secret
        has length <*secretLength>
45  */
46 void computeSecret(EVP_PKEY *keyPair, EVP_PKEY *peerPublicKey, unsigned char **
        pSecret, size_t *secretLength){
47          EVP_PKEY_CTX *ctx; //context
48          /* Put key pair*/
49          if(!(ctx = EVP_PKEY_CTX_new(keyPair, NULL))) handleErrors();
50
51          if (EVP_PKEY_derive_init(ctx) !=  1){
52                  handleErrors();
53          }
54          if (EVP_PKEY_derive_set_peer(ctx, peerPublicKey) != 1){
55                  handleErrors();
56          }
57
58          /* Determine buffer length */
59          if (EVP_PKEY_derive(ctx, NULL, secretLength) != 1){
60                  handleErrors();
61          }
62
63          *pSecret = OPENSSL_malloc(*secretLength);
64
65          if (!*pSecret){
66                  printf("Malloc error\n");
67                  handleErrors();
68          }
69
70          if (EVP_PKEY_derive(ctx, *pSecret, secretLength) != 1){
71                  handleErrors();
72          }
73          EVP_PKEY_CTX_free(ctx);
74          /* Shared secret is secretLength bytes written to buffer secret */
75 }
76
77
78 /**
79  * Generates DH keys. The private part (g, g^x) is stored in <*privateKeyPair>
        and the public part in a file named <publicKeyFileName> in PEM format.
80  * The context <ctx> has to be provided.
81  */
82 void generateKeys(EVP_PKEY_CTX* ctx, EVP_PKEY** privateKeyPair, const char*
        publicKeyFileName){
83          if(NULL == (*privateKeyPair = EVP_PKEY_new()))
84                  handleErrors();
85          if(1 != EVP_PKEY_keygen(ctx, privateKeyPair))
86                  handleErrors();
87          FILE *file = fopen(publicKeyFileName, "w");
88          if(!file){
89                  fprintf(stderr, "error opening file %s\n", publicKeyFileName);
90                  abort();
91          }
92          PEM_write_PUBKEY(file, *privateKeyPair);
93          fclose(file);
94 }
```

```
95
96  /**
97   * Loads the public key <publicKey> given in PEM format from the file <
        publicKeyFilename>
98   */
99  void receivePublicKey(EVP_PKEY** publicKey, const char* publicKeyFileName){
100         FILE* file = fopen(publicKeyFileName, "r");
101         *publicKey = PEM_read_PUBKEY(file, NULL, 0, NULL);
102         fclose(file);
103 }
104
105 /**
106  * Given two derived DH secrets, checks that they are equal and prints their
        base64.
107  * Alice's secret is <aliceSecret> and has <aliceSecretLength> bytes
108  * Bob's secret is <bobSecret> and has <bobSecretLength> bytes
109  */
110 void testSecrets(const unsigned char* aliceSecret, const unsigned char*
        bobSecret, const size_t aliceSecretLength, const size_t bobSecretLength){
111         if(aliceSecretLength != bobSecretLength)
112                 fprintf(stderr, "missmatch in secret lengths\n");
113         for (int i = 0; i < aliceSecretLength; ++i){
114                 if (aliceSecret[i] != bobSecret[i]){
115                         fprintf(stderr, "error in the keys at index %d : %d %d\
        n", i, aliceSecret[i], bobSecret[i]);
116                         abort();
117                 }
118         }
119
120         char* b64_string = base64_encode(aliceSecret, aliceSecretLength);
121         fprintf(stderr, "%s\n", b64_string );
122         free(b64_string);
123 }
124
125
126 int main(){
127         //OpenSSl init
128
129         //Load the human readable error strings for libcrypto
130         ERR_load_crypto_strings();
131         // Load all digest and cipher algorithms
132         OpenSSL_add_all_algorithms();
133         // Load config file, and other important initialisation
134         CONF_modules_load(NULL, NULL, 0);
135
136         EVP_PKEY * params;
137         if(NULL == (params = EVP_PKEY_new()))
138                 handleErrors();
139         // Use built-in parameters
140         DH* values = DH_get_2048_256();
141         if(1 != EVP_PKEY_set1_DH(params,values))
142                 handleErrors();
143
144         // Create context for the key generation
145         EVP_PKEY_CTX *ctx;
146         if(!(ctx = EVP_PKEY_CTX_new(params, NULL)))
147                 handleErrors();
```

```
148
149         // Generate new keys
150         EVP_PKEY *alicekey, *bobkey;
151         if(1 != EVP_PKEY_keygen_init(ctx))
152                 handleErrors();
153         generateKeys(ctx, &alicekey, "alice.pub");
154         generateKeys(ctx, &bobkey, "bob.pub");
155
156
157         //Receive public keys
158         EVP_PKEY *bobPublicKey, *alicePublicKey;
159         receivePublicKey(&alicePublicKey, "alice.pub");
160         receivePublicKey(&bobPublicKey, "bob.pub");
161
162         unsigned char *bobSecret, *aliceSecret;
163         size_t aliceSecretLength, bobSecretLength;
164         computeSecret(alicekey, bobPublicKey, &aliceSecret, &aliceSecretLength)
    ; //Alice side
165         computeSecret(bobkey, alicePublicKey, &bobSecret, &bobSecretLength); //
    Bob side
166
167         testSecrets(aliceSecret, bobSecret, aliceSecretLength, bobSecretLength)
    ;
168
169         //freeing
170         OPENSSL_free(bobSecret);
171         OPENSSL_free(aliceSecret);
172         EVP_PKEY_free(alicekey);
173         EVP_PKEY_free(bobkey);
174         EVP_PKEY_free(alicePublicKey);
175         EVP_PKEY_free(bobPublicKey);
176         EVP_PKEY_CTX_free(ctx);
177         DH_free(values);
178         EVP_PKEY_free(params);
179         //openSSL cleanup
180
181         //Removes all digests and ciphers
182         EVP_cleanup();
183         CRYPTO_cleanup_all_ex_data();
184         //Remove error strings
185         ERR_free_strings();
186         return 0;
187 }
```