Cybersecurity - Homework 5

Vlad Turno (1835365)

November 27, 2025

1) Introduction

In cryptography, a DRBG (deterministic random bit generator) is an algorithm designed to produce a sequence of bits apparently random, but generated in a deterministic way (for example, by defining a starting condition called seed). Given the same seed and parameters, a DRBG should always produce the same sequence of bits, and this is useful in applications where the "randomness" is needed to be reproducible (but seem not from an external perspective). A CS-PRNG (cryptographically secure pseudo-random number generator) is a PRNG (pseudo-random number generator) designed for cryptanalysis and therefore secure in a cryptographic context. A regular PRNG could be fast but predictable (it could reveal patterns or some structure over time) but a CS-PRNG is designed to be resilient by producing numbers difficult to predict even by knowing part of the output or some internal state of the number generator.

2) Implementation

I'm now gonna implement two cryptographically secure pseudo-random number generators for binary strings using OpenSSL library within a C program. One will be a deterministic random bit generator (DRBG) using SHA-256, and the other a traditional CS-PRNG based on a secure hash function. The program will calculate the cost in terms of time elapsed and memory usage for the two generators. The two generated bit sequences, having size 1024 bytes, will be later compared in terms of bit value occurrences. Source code and outcome below:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/sha.h>
#include <time.h>

// SHA-256 hashing function
void sha256(const unsigned char *input, size_t length, unsigned char *output) {
    SHA256_CTX sha256_ctx;
    SHA256_Init(&sha256_ctx);
    SHA256_Update(&sha256_ctx, input, length);
    SHA256_Final(output, &sha256_ctx);
}

// DRBG (deterministic random bit generator)
typedef struct {
    unsigned char state[SHA256_DIGEST_LENGTH];  // internal state
    unsigned char output[SHA256_DIGEST_LENGTH]; // output buffer
} DRBG;

void drbg_init(DRBG *drbg, const unsigned char *seed, size_t seed_len) {
    sha256(seed, seed_len, drbg->state);
}

void drbg_generate(DRBG *drbg) {
    sha256(drbg->state, SHA256_DIGEST_LENGTH, drbg->state);
    memcpy(drbg->output, drbg->state, SHA256_DIGEST_LENGTH);
}

void drbg_get_bits(DRBG *drbg, unsigned char *output, size_t num_bits) {
    size_t full_bytes = num_bits / 8;
    size_t remaining_bits = num_bits % 8;
```

```
33
34      for (size_t i = 0; i < full_bytes; i++) {
35          output[i] = drbg->output[i];
36      }
37
38      if (remaining_bits > 0) {
39          unsigned char last_byte = drbg->output[full_bytes];
40          output[full_bytes] = last_byte & ((1 << remaining_bits) - 1);  // mask
                remaining bits
41      }
42  }
43
44  // CS-PRNG (cryptographically secure PRNG)
45  typedef struct {
46      unsigned char state[SHA256_DIGEST_LENGTH];  // Internal state
47  } CS_PRNG;
48
49  void cs_prng_init(CS_PRNG *prng, const unsigned char *seed, size_t seed_len) {
50      sha256(seed, seed_len, prng->state);
51  }
52
53  void cs_prng_generate(CS_PRNG *prng, unsigned char *output, size_t output_len) {
54      size_t bytes_generated = 0;
55      unsigned char hash_output[SHA256_DIGEST_LENGTH];
56
57      while (bytes_generated < output_len) {
58          sha256(prng->state, SHA256_DIGEST_LENGTH, hash_output);  // Generate new
                hash
59          size_t bytes_to_copy = (output_len - bytes_generated < SHA256_DIGEST_LENGTH
                )
60                                     ? output_len - bytes_generated
61                                     : SHA256_DIGEST_LENGTH;
62          memcpy(output + bytes_generated, hash_output, bytes_to_copy);
63          bytes_generated += bytes_to_copy;
64          memcpy(prng->state, hash_output, SHA256_DIGEST_LENGTH);  // Update the
                state
65      }
66  }
67
68  // Utility function to print binary string for visualization
69  void print_binary_string(const unsigned char *data, size_t length) {
70      for (size_t i = 0; i < length; i++) {
71          for (int j = 7; j >= 0; j--) {
72              printf("%d", (data[i] >> j) & 1);
73          }
74      }
75      printf("\n");
76  }
77
78  // main function to compare time and space of both PRNGs
79  int main() {
80      unsigned char seed[] = "random_seed";  // seed for both PRNGs
81      size_t seed_len = strlen((char *)seed);
82
83      DRBG drbg;
84      drbg_init(&drbg, seed, seed_len);
85
86      unsigned char drbg_output[1024]; // 1 MB of random bits
87      clock_t start_time = clock();
88      for (int i = 0; i < 1024 / SHA256_DIGEST_LENGTH; i++) {
89          drbg_generate(&drbg);
90          drbg_get_bits(&drbg, drbg_output + i * SHA256_DIGEST_LENGTH,
                SHA256_DIGEST_LENGTH);
91      }
92      clock_t end_time = clock();
93
94      // DRBG generated output
95      // printf("DRBG Output:\n");
96      // print_binary_string(drbg_output, 1024);  // print output as binary
97
98      // DRBG file write
99      FILE *file = fopen("drbg.bin", "wb");  // open file for writing in binary mode
100     if (file == NULL) {
101         perror("Error opening file for writing");
102         return 1;
103     }
```

```c

    size_t written = fwrite(drbg_output, 1, sizeof(drbg_output), file);
    if (written != sizeof(drbg_output)) {
        perror("Error writing to file");
        fclose(file);
        return 1;
    }

    fclose(file);

    // DRBG time stats
    double drbg_time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Time taken for DRBG to generate 1024 random bits: %.6f seconds\n",
        drbg_time_taken);

    // DRBG space usage
    printf("DRBG internal state size: %lu bytes\n", sizeof(drbg.state));
    printf("DRBG output buffer size: %lu bytes\n", sizeof(drbg.output));

    CS_PRNG prng;
    cs_prng_init(&prng, seed, seed_len);

    unsigned char prng_output[1024]; // 1 MB of random bits
    start_time = clock();
    cs_prng_generate(&prng, prng_output, sizeof(prng_output));
    end_time = clock();

    // CS-PRNG generated output
    // printf("CS-PRNG Output:\n");
    // print_binary_string(prng_output, 1024);  // print output as binary

    // CS-PRNG file write
    file = fopen("cs-prng.bin", "wb");  // open file for writing in binary mode
    if (file == NULL) {
        perror("Error opening file for writing");
        return 1;
    }

    written = fwrite(prng_output, 1, sizeof(prng_output), file);
    if (written != sizeof(prng_output)) {
        perror("Error writing to file");
        fclose(file);
        return 1;
    }

    fclose(file);

    // CS-PRNG time stats
    double prng_time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Time taken for CS-PRNG to generate 1024 random bits: %.6f seconds\n",
        prng_time_taken);

    // CS-PRNG space usage
    printf("CS-PRNG internal state size: %lu bytes\n", sizeof(prng.state));

    return 0;
}
```

Listing 1: prng.c

```
$ gcc -o prng prng.c -lssl -lcrypto
$ ./prng

Time taken for DRBG to generate 1024 random bits: 0.000027 seconds
DRBG internal state size: 32 bytes
DRBG output buffer size: 32 bytes
Time taken for CS-PRNG to generate 1024 random bits: 0.000026 seconds
CS-PRNG internal state size: 32 bytes
```

3) Comparison

The two binary sequences generated are now stored in a binary file each and with a C program we now count the number of 0s and 1s in each sequence to highlight the differences in the outcome (source code and outcomes below):

```c
#include <stdio.h>
#include <stdlib.h>

void count_bits(const char *filename, int *count_zeros, int *count_ones) {
    FILE *file = fopen(filename, "rb");
    if (file == NULL) {
        perror("Error opening file");
        exit(1);
    }

    unsigned char byte;
    while (fread(&byte, sizeof(unsigned char), 1, file)) {
        for (int i = 7; i >= 0; i--) {
            if ((byte >> i) & 1) {
                (*count_ones)++;
            } else {
                (*count_zeros)++;
            }
        }
    }

    fclose(file);
}

int main() {
    const char *file1 = "cs-prng.bin";  // Replace with actual file paths
    const char *file2 = "drbg.bin";  // Replace with actual file paths

    int zeros1 = 0, ones1 = 0;
    int zeros2 = 0, ones2 = 0;

    count_bits(file1, &zeros1, &ones1);
    count_bits(file2, &zeros2, &ones2);

    printf("%s: Zeros = %d, Ones = %d\n", file1, zeros1, ones1);
    printf("%s: Zeros = %d, Ones = %d\n", file2, zeros2, ones2);

    return 0;
}
```

Listing 2: counter.c

```
$ gcc -o counter counter.c
$ ./counter

cs-prng.bin: Zeros = 4129, Ones = 4063
drbg.bin:    Zeros = 6556, Ones = 1636
```

4) Broadening

The previous analysis can be broadened to include some more different DRBG constructions to compare in terms of time performance and space usage. There is a variety of deterministic random bit generators that can be used, like HASH-based and HMAC-based implemented using SHA-256 or AES-CTR using the advanced encryption standard in the counter mode.