Assignment 4 — Secure Programming 2020

Part 1 — 24%

In this part, you are asked to analyse a few function to understand what they do. The functions perform simple arithmetic and bitwise operations on their inputs. We do not ask you to describe these operations, but to explain what the outcome is. For example, consider the function:

```
int32_t example1(int32_t a) {
  return (a^0xFFFFFFFF)+1;
}
```

Saying that `example1` calculates the exclusive or of the input with the number `0xFFFFFFFF` and adds one is technically correct, but is not the expected answer, and will not give any marks. The correct answer is that the function computes the two's complement of the input. (Or any equivalent description.)

Similarly, for `example2` below, the expected answer is that the function returns bit `b` of `a`. Saying that it shifts the number 1 by `b` bits to the left and returns the result of anding that with `a` is not sufficient.

```
uint32_t example2(uint32_t a, uint32_t b) {
  return (1<<b)&a;
}
```

For all questions, assume that signed numbers are represented using two's complement. Moreover, contrary to the C standard, for this assignment the behaviour of integer overflow is well defined to wrap around. For example, `MAX_INT32+1` results in `MIN_INT32`.

Question 1 — 6%

```
int32_t f1(int32_t a) {
  return a & -a;
}
```

Question 2 — 6%

```
int32_t f2(int32_t a) {
  return (a | -a)>>31;
}
```

Question 3 — 6%

```
uint32_t f3(uint32_t a, uint32_t b, int32_t c, int32_t d) {
  c ^= d;
  c = (c | -c) >> 31;
  return (a & ~c) | (b & c);
}
```

Question 4 — 6%

```
uint8_t f4(uint8_t a, uint8_t b) {
  uint8_t c;
  c = (a & b) << 1; a ^= b;
  b = (a & c) << 1; a ^= c;
  c = (a & b) << 1; a ^= b;
  b = (a & c) << 1; a ^= c;
  c = (a & b) << 1; a ^= b;
  b = (a & c) << 1; a ^= c;
  c = (a & b) << 1; a ^= b;
  b = (a & c) << 1; a ^= c;
  return a;
}
```

Part 2 — 76% (70% for COMP SCI 7307)

In this part you are asked to implement a library that performs basic arithmetic with integers of arbitrary lengths, called *bignums*.

The library provides one abstract data structure `bn_t`, which is a pointer to the (externally abstract) `struct bn`. The functions operating on this structure are described below. This interface is defined in the file `bn.h`. The file `bn.c` provides an implementation of `bn_alloc()` and `bn_toString()`. It also includes the skeletons of the other five functions (`bn_free()`, `bn_add()`, `bn_sub()`, `bn_mul()`, and `bn_fromString()`), which you should implement.

- **`bn_t bn_alloc(void)`** Allocates a new bignum and initialises it to (positive) 0. Returns `NULL` if not enough resources are available.
- **`void bn_free(bn_t bn)`** Frees all the resources associated with the bignum `bn`.
- **`int bn_add(bn_t result, bn_t a, bn_t b)`** Adds the bignums `a` and `b` and places the sum in `result`. Returns 0 if completed successfully and -1 otherwise. Note that `result` may be the same bignum as `a` or `b`.
- **`int bn_sub(bn_t result, bn_t a, bn_t b)`** Subtracts the bignum `b` from `a` and places the difference in `result`. Returns 0 if completed successfully and -1 otherwise. Note that `result` may be the same bignum as `a` or `b`.
- **`int bn_mul(bn_t result, bn_t a, bn_t b)`** Multiplies the bignums `a` and `b` and places the product in `result`. Returns 0 if completed successfully and -1 otherwise. Note that `result` may be the same bignum as `a` or `b`.
- **`int bn_fromString(bn_t bn, const char *str)`** Read the decimal number in the string `str` and stores its value in `bn`. Returns 0 if completed successfully and -1 otherwise.
- **`int bn_toString(bn_t bn, char *buf, int buflen)`** Converts a bignum `bn` to a string. including the terminating `NUL` character. If `buflen` is large enough to store the converted string, the function stores the string in `buf` and returns 0. Otherwise, `buf` is not changed and the return value is the number of characters required to store the string representation, The function returns a negative number in case of error.

`struct bn`

The file `bn.c` defines the bignum structure `struct bn` as:

```
struct bn {
  int bn_len;
  int bn_size;
  int bn_sign;
```

```
  uint16_t *bn_data;
};
```

The fields of the structure are:

- `bn_len` The length (number of digits) of the number represented by the structure.
- `bn_data` An array that stores the values of the digits of the number.
- `bn_sign` The sign of the number. The value of the sign is 1 for positive numbers and -1 for negative numbers.
- `bn_size` The number of digits allocated in `bn_data`.

The number is reprsented as a sequence of *digits* in radix (base) $2^{16} = 65536$. That is, if s is the value of `bn_sign`, l the value of `bn_len`, and d_i the value of the i^{th} element in `bn_data` (i.e. `bn_data[i]`), then the value represented by the structure is given by: $s \cdot \sum_{i=0}^{l-1} (2^{16})^i d_i$.

Use Example

The example below shows a program that uses the library to calculate the first 100 elements of the Fibonacci sequence.

```
#include <stdio.h>
#include "bn.h"

int main() {
  char buf[1000];
  bn_t a = bn_alloc();
  bn_t b = bn_alloc();

  bn_fromString(a, "0");
  bn_fromString(b, "1");

  for (int i = 0; i < 100; i++) {
    bn_toString(a, buf, sizeof(buf));
    printf("%2d: %s\n", i, buf);
    bn_add(a, a, b);
    bn_t t = a;
    a = b;
    b = t;
  }
```

Part 3 — 6% (COMP SCI 7307 only)

Note: This part is only for Masters students taking COMP SCI 7307.

Use a subset of the C language to implement the function `uint32_t mul(uint16_t a, uint16_t b)`, which calculates the product of `a` and `b`. You are only allowed to use the following subset of the language:

- Variables of types `uint16_t` and `uint32_t`.
- The binary operators `+`, `&`, `|`, `^`, `==`.
- The ternary operator `?:`.
- The assignment operator `=`, as well as its combination with other allowed operators, e.g. `+=`, `&=`, etc.
- The `return` statement.

Submission Instructions

You should submit a `.tar` or a `.tgz` archive. The archive should contain a single directory, whose name is your student a-number. In that directory, we expect to find four items:

- A text file named `info.txt`, which contains two lines. The first line is your name, and the second is your student a-number.
- A PDF document named `assignment4.pdf`, which contains the answers to Questions 1–4. The file should be typed-up, i.e. scanning a handwritten paper is not acceptable. Also, the file must be a PDF document. Text or Word documents are not acceptable, even if you changed the extension to `.pdf`. The file should also include the answer to Part 3, if you did that.
- A folder named `part2` with the implementation of your bignum library. Typing `make` in this folder should build `libbn.so`, which contains an implementation of all of the required functions. Note that we plan to look at your code, so please make sure it is readable.

Soundness Test: We will provide a soundness test script to test your submission. If your submission does not pass a soundness test it will not be marked and will be graded as zero.