

Problem Set 6
Course **Security Engineering**
(Winter Term 2018)

Bauhaus-Universität Weimar, Chair of Media Security

Prof. Dr. Stefan Lucks, Nathalie Dittrich

URL: <http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/>

Due Date: 18 Jan 2019, 01:30 PM, via email to nathalie.jolanthe.dittrich@uni-weimar.de.

Goals: Practicing more with **gnatprove** and getting familiar with Ada tasks.

Task 1 – Introduction (No Credits)

Inform yourselves about tasks in Ada, e.g., from the lecture slides, from Chapter 19 in John English, and/or from the Ada Wiki book.

Task 2 – Mini-Project – Sorting with gnatprove (4 Credits)

Inform yourselves about Selection Sort. Then, implement the following package **Sorting**.

```
1 package Sorting is
2     type Natural_Array is array(Natural range <>) of Natural;
3
4     procedure Selection_Sort(A: in out Natural_Array);
5 end Sorting;
```

Write a sufficient test suite for your implementation with either **AUnit** or **testgen**. Add correct Pre-/Post-conditions and loop invariants to successfully prove its correctness with **gnatprove** in **prove** mode. You can add helper functions to the specification.

Task 3 – Mini Project – Parallel Sum (4 Credits)

Implement the following package specification **Parallel_Algorithms**, that computes the sum of all array elements in parallel. Use at least two tasks:

```
1 generic
2     type Item_Type is private;
3     with function "+"(Left: Item_Type; Right: Item_Type) return Item_Type;
4 package Parallel_Algorithms is
5
6     type Array_Type is array(Natural range <>) of Item_Type;
7     type Array_Access_Type is access all Array_Type;
8
9     procedure Parallel_Sum(Input: Array_Access_Type; Result: out Item_Type);
10
11 end Parallel_Algorithms;
```

Write a sufficient test suite for your implementation with either **AUnit** or **testgen**.

Task 4 – Mini Project – Concurrent Sorting (6 Credits)

Inform yourselves about Merge Sort.

- a) Implement the following package `Parallel_Algorithms` for realizing Merge Sort in a parallel manner. Use at least two tasks.

```
1 generic
2   type Item_Type is private;
3   with function "="(Left: Item_Type; Right: Item_Type) return Boolean;
4   with function "<"(Left: Item_Type; Right: Item_Type) return Boolean is <>;
5 package Parallel_Algorithms is
6
7   type Array_Type is array(Natural range <>) of Item_Type;
8   type Array_Access_Type is access all Array_Type;
9
10  procedure Parallel_Merge_Sort(Input: Array_Access_Type;
11                               Result: Array_Access_Type);
12
13 end Parallel_Algorithms;
```

- b) Inform yourselves about parsing CommandLine arguments in Ada. For example, read AdaCore Gems 138 and 139. Then, implement a wrapping program that calls your implementation from a). A call to your program should look as follows:

```
1 $ ./parallel_merge_sort -i <in_file_path> -o <out_file_path> -t <max_time>
```

where `<in_file_path>` denotes the relative path to a text file containing one integer per line whose contents will be sorted and stored in a text file under the relative path of `<out_file_path>`. The program shall print the usage and do nothing if no such in-file exists or if the out-file already exists; `<max_time>` specifies the maximum run time of your program. Pressing `q` should stop your program immediately. Write a sufficient test suite for your implementation with either `AUnit` or `testgen`.

Task 5 – Mini-Project – Meet-in-the-Middle Attack (Two People) (6 Credits)

XTEA is a small block cipher which encrypts 64-bit plaintexts in 64 rounds under a 128-bit key. Let Mini-XTEA denote a reduced version of XTEA with the difference that Mini-XTEA employs only a 64-bit key K , which consists of two 32-bit words $K[0]$, and $K[1]$. Furthermore, Mini-XTEA has a simpler key schedule: each of the first 32 rounds use only the first key word, $K[0]$, and each of the latter 32 rounds only $K[1]$.

An adversary has eavesdropped the following three plaintext-ciphertext pairs:

#	Plaintext	Ciphertext
1	(16#53616665#, 16#20616e64#)	(16#328cebe1#, 16#b84934dc#)
2	(16#53656375#, 16#72652020#)	(16#e68b802c#, 16#bcdd2ca3#)
3	(16#536f6674#, 16#77617265#)	(16#39943672#, 16#c3db80ec#)

Implement the following Ada specification for mounting a meet-in-the-middle attack to recover the secret key K that was used to encrypt the pairs. Your implementation should use at least two tasks that test key candidates in parallel.

```

1 with Mini_XTEA; use Mini_XTEA;
2
3 package MITM is
4     procedure Run_MITM(Plaintexts:    State_Array_Type;
5                       Ciphertexts:    State_Array_Type;
6                       Key:            out Key_Type;
7                       Has_Found_Key:  out Boolean);
8 end MITM;

```

```

1 with Interfaces;
2
3 use Interfaces;
4
5 package Mini_XTEA is
6     subtype Word_Type is Unsigned_32;
7
8     type Side          is (Left, Right);
9     type Key_Type      is array(0..1) of Word_Type;
10    type State_Type    is array(Side'Range) of Word_Type;
11    type State_Array_Type is array(Natural range <>) of State_Type;
12    type State_Array_Access_Type is not null access all State_Array_Type;
13
14    NUM_ROUNDS:        constant Positive := 64;
15    DELTA_CONSTANT:   constant Word_Type := 16#9E3779B9#;
16
17    function Decrypt(Ciphertext: State_Type; Key: Key_Type) return State_Type;
18    function Decrypt_Second_Half(Ciphertext: State_Type; Key: Word_Type)
19        return State_Type;
20
21    function Encrypt(Plaintext: State_Type; Key: Key_Type) return State_Type;
22    function Encrypt_First_Half(Plaintext: State_Type; Key: Word_Type)
23        return State_Type;
24 end Mini_XTEA;

```

Write a sufficient test suite for your implementation with either **AUnit** or **testgen**.