RSA implementation in MIPS Assembler Language

Jannik Franz, Tobias Ruck

May 8, 2015

1.1 Algorithm

1.1.1 Introduction

Generally the algorithm implemented follows along the suggested exercise and approach. This section will focus on explaining how the different parts were implemented. Later overall constraints created by using such methods will be explained.

1.1.2 Implementation

Prime Generation

To generate prime numbers the "sieve of eratosthenes" is used. Generally this algorithm says whether any number up to n is a prime or not. The limit n has to be predefined before sieving. For the implementation $n = 1500$. This means that 1500 bytes have to be allocated which mark whether or not a certain number is a prime or not.

```
signed char primes[MAX_NUMBERS];
memset(primes, 1, sizeof(primes));
for (i = 2; i < MAX_NUMBERS; i++)
        if (primes[i])
                for (j = i; j * i < MAX_NUMBERS; j++)
                        primes[j * i] ^= primes[j * i];
```

The first line allocates the space for up to MAX_NUMBERS bytes. MAX_NUMBERS is 1500 in this case. The second line sets the value of all the adresses reserved to 1. The first loops goes over every number from 2 to MAX_NUMBERS. 0 and 1 are skipped since they are both not considered prime and not plausible for the algorithm because it strokes out multiples of previously processed numbers from the array marking them as non-primes as they are multiples. Next up is making sure to only take multiples of numbers currently considered prime to optimize. The inner loop calculates multiples of the current number and crosses them out from the array via an exclusive-OR operation.

1.2 Constraints