Project Report

Name: Kemele Muhammed Endris M.No,: 160753

Integer Pseudo-random Generator

Describe your integer pseudorandom generator?

A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state, which includes a truly random seed.[1]

We created two integer pseudo-random generators, namely the fast pseudo-random number generator, fprng() and secure pseudo-random number generator, spring().

Fast pseudo-random number generator

The fast pseudo-random number generator, fprng(), is used to generate a random number with good statistical properties by using lfsr() and maj5 stream cipher. It first generates a random initial state of 19 bits and a key of 64 bits using the C rand() function by seeding the current system time. Then the initial state of 19 bits is shifted using the LFSR with the polynomial below generating the length of random number needed bits times. Then the output of LFSR is encrypted using the Maj5 stream cipher, which gives us statistically good random integer number.

```
void fprng(uint8_t* num, int length)
{
  srand ((unsigned int)time(NULL));
//generate state
//Generate key
 uint64_t poly = 0x80027; // x^19 + x^18 + x^17 + x^14 + 1 ;
 lfsr(&poly, &state, 19, length, msg);
 maj5_encrypt(msg, key, num, length);
}
```

Secure Pseudo-random Number Generator

Secure pseudo-random generator, sprng(), is used to generate a random number with good statistical properties by using BunnyCBC block cipher encryption. It first generates three random numbers, namely 24 bit initial vector, 24 bit key and n bits of msg, using a linux strong random numbers in /dev/random file. /dev/random is a special file that serves as a random number generator or as a pseudorandom

number generator. It allows access to environmental noise collected from device drivers and other sources.[3]

Then it encrypts the randomly generated message using BunntCBC_encrypt function by randomly generated key and initial vector. The output of the encrypted message is our secure random integer number.

Prime Pseudo-random Generator

Describe your prime pseudorandom generator?

The prime pseudo-random number generator uses the fast random number generator. It simply generates the random integer number using frand() then change the last bit of the random number to 1 to increase the probability of getting prime number (by increasing the probability of the number being odd). Then using BN_is_prime() function of OpenSSL, it checks whether the generated number is prime or not. If NO, generate again, till it gets the prime number. *This function could have generated a better/strong prime number by testing whether its half is a coprime with it, but it takes longer time and resource.*

```
void primeNum(uint8_t* num, int length)
{
    BIGNUM *n = BN_new();
    int byteLength = length / 8 + ((length % 8 != 0) ? 1 : 0); //length in byte
    do{  fprng(num , length);     //generate a number
       num[byteLength-1] |= 0x01;  /* Last bit as 1 */
       n = BN_bin2bn((const unsigned char *) num, byteLength , NULL);
       is_p = BN_is_prime(n,10,NULL, NULL, NULL) ;
       if (is_p ==  1){
          printf("n = %s is prime!\n", BN_bn2dec(n)) ;      exit=1;
          }
     }while(exit == 0);
}
```

Generating RSA keys

Describe how you generated the RSA keys?

We generate the keys as follows.

1) Choose two large random prime numbers P and Q of similar length.
2) Compute N = P x Q. N is the modulus for both the Public and Private keys.
3) PSI = (P-1)(Q-1) , PSI is also called the Euler's totient function.
4) Choose an integer E, such that 1 < E < PSI, making sure that E and PSI are co-prime. E is the Public key exponent.
5) Calculate D = E-1 (mod PSI) , normally using Extended Euclidean algorithm. D is the Private key exponent.

The prime number generator takes from 60-200 seconds to generate the 512 bit prime number for RSA key. In this range of time it checks between 105051 – 283274 integers whether they are primes or not.

Symmetric Keys

Describe how you generated the symmetric keys?

The symmetric keys are generated using secure random number generator, sprng(). Specifically if the cipher suit is Bunny, we generate 24 bit secure random key (excluding all 1's and all 0's, by do-while) and 64 bit secure random key for Maj5 and All5 stream cipher suit (excluding all 1's and all 0's, by do-while).

Protocol Dome Flaws

In the protocol proposed in the course there may be dome flawes: please elaborate?

Brute-force attacks are feasible on the symmetric cipher; our current use of RSA could be vulnerable to timing attacks too.

References

1. http://en.wikipedia.org/wiki/Pseudorandom_number_generator
2. http://www.codeproject.com/Articles/421656/RSA-Library-with-Private-Key-Encryption-in-Csharp
3. http://en.wikipedia.org/wiki//dev/random