EHN 410

Practical 3 Lecture

Tapfuma Chanaiwa

University of Pretoria

May 8, 2020

Outline

GMP library

- C library for arbitrary precision arithmetic on integers, rational numbers and floating point numbers
- Open source
- Optimized for different processors
- Provides interfaces for a range of mathematical operations
- For your RSA implementation, you will use the integer data type `mpz_t`
- Documentation available at: http://gmplib.org

Useful GMP functions

Initialisation, assignment and conversion functions :

- `void mpz_init(mpz_t integer)`
- `void mpz_clear(mpz_t integer)`
- `void mpz_set_ui(mpz_t rop, unsigned long int op)`
- `void mpz_set_st(mpz_t rop, char* str, int base)`
- `unsigned long int mpz_get_ui(mpz_t 0p)`
- `char* mpz_get_str(char* str, int base, mpz_t op)`

Useful GMP functions

Arithmetic and exponential functions:

- `void mpz_add(mpz_t rop, mpz_t op1, mpz_t op2)`
- `void mpz_sub(mpz_t rop, mpz_t op1, mpz_t op2)`
- `void mpz_ui(mpz_t rop, mpz_t op1,unsigned long int op2)`
- `void mpz_mul(mpz_t rop, mpz_t op1, mpz_t op2)`
- `void mpz_pown(mpz_t rop, mpz_t base, mpz_t exp,mpz_t mod)`

Useful GMP functions

Comparison and number theoretic functions:

- `void mpz_cmp (mpz_t op1, mpz_t op2)`
- `void mpz_gcd (mpz_t op1, unsigned long int op2)`
- `void mpz_nextprime (mpz_t rop, mpz_t op)`
- `void mpz_gcd (mpz_t rop, mpz_t op1, mpz_t op2)`

RC4

- Symmetric stream cipher
- Variable key length 1-256 bytes (8-2048 bits)
- Is based on random permutations
- Very large period (10^{100}) which makes it ideal for random number generation
- Very fast and simple algorithm (throughput of 45Mbps on a Pentium II)
- Commonly used in SSL/TLS, WEP and WPA

RC4 implementation

- Your RC4 key should be 16 bytes (128 bits) in length
- **Encryption**: XOR each byte generated from the keystream with a byte from the plaintext
- **Decryption**: XOR the keystream with the ciphertext
- More details on RC4 implementation (key scheduling algorithm) on pages 61-64 of Stallings
- For the practical, an executable called **rc4** must be developed. This will be used to encrypt/decrypt a message using RC4.

Implementation Notes: RC4

Your **rc4** program should contain the following functionality:

- Structure to store your RC4 context:
  `rc4ctx_t`
- Initialisation of RC4 with a variable key length:
  `void rc4_init (rc4ctx_t* rc4c, U8* key, int keylen)`
- Retrieve a byte that can be XOR'ed with the plaintext to provide the ciphertext:
  `U8 rc4_getbyte (rc4ctx_t* rc4c)`

Note: `U8` is of type `unsigned char`

RSA

- Public key cryptographic algorithm
- Developed at MIT in 1977, by Ron Rivest, Adi Shamir and Leonard Adelman
- Used for encryption/decryption and digital signatures.
- Chapter 3 of Stallings contains detailed information regarding the RSA algorithm.
- For the practical, RSA will be used to encrypt/decrypt the 128-bit RC4 key

RSA implementation

Three programs must be implemented:

- **rsakeygen** - the key generator for RSA
- **rsaencrypt** - encryption algorithm that encrypts 16-byte RC4 key with RSA public key.
- **rsadecrypt** - decryption algorithm that decrypts 16-byte RC4 key with RSA private key

where:

- Public key: $KU = \{e, n\}$
- Private key: $KR = \{d, n\}$

RSA implementation

rsakeygen

- Should be able to generate keys of different bit lengths, $n = pq$ should have the same bit lengths as your key.
- To achieve this, p and q should each be half the key-length where both p and q should be randomly generated prime numbers.

Note: e is commonly chosen as 3, 17 or 65537 as it has a better chance of a coprime with $\Phi(n)$

Implementation Notes: RSA

Make use of the following for your RSA implementation:

- Store a prime of size bits in p. Uses the RNG based on RC4:
  `void getprime(mpz_t p, int bits)`
- Structure to store all RSA information:
  `rsactx_t`
- Initialisation of the RSA structure:
  `void rsa_init(rsactx_t* rsactx)`
- Free memory allocated to the RSA structure:
  `void rsa_clean(rsactx_t* rsactx)`

Random Number Generation (RNG)

- RC4 should be used for random number generation for p and q in RSA.
- For testing purposes the RNG should be initialised to the same key (e.g. $0x0123456789ABCDEF$).
- In practice, a random initialisation value such as system time, for instance, would be selected.
- For the demo, ensure that you include functionality to specify the initialisation key.

Implementation Notes: RNG

Make use of RC4 to generate random numbers:

- Create a random seed for RC4:
  `void rseed (U8* key, int keylen)`
- Get the next random byte: `U8 rrand();`

Hint: Store the start of the random number generator in a global variable (i.e. a `rc4ctx_t` structure)

Encryption procedure

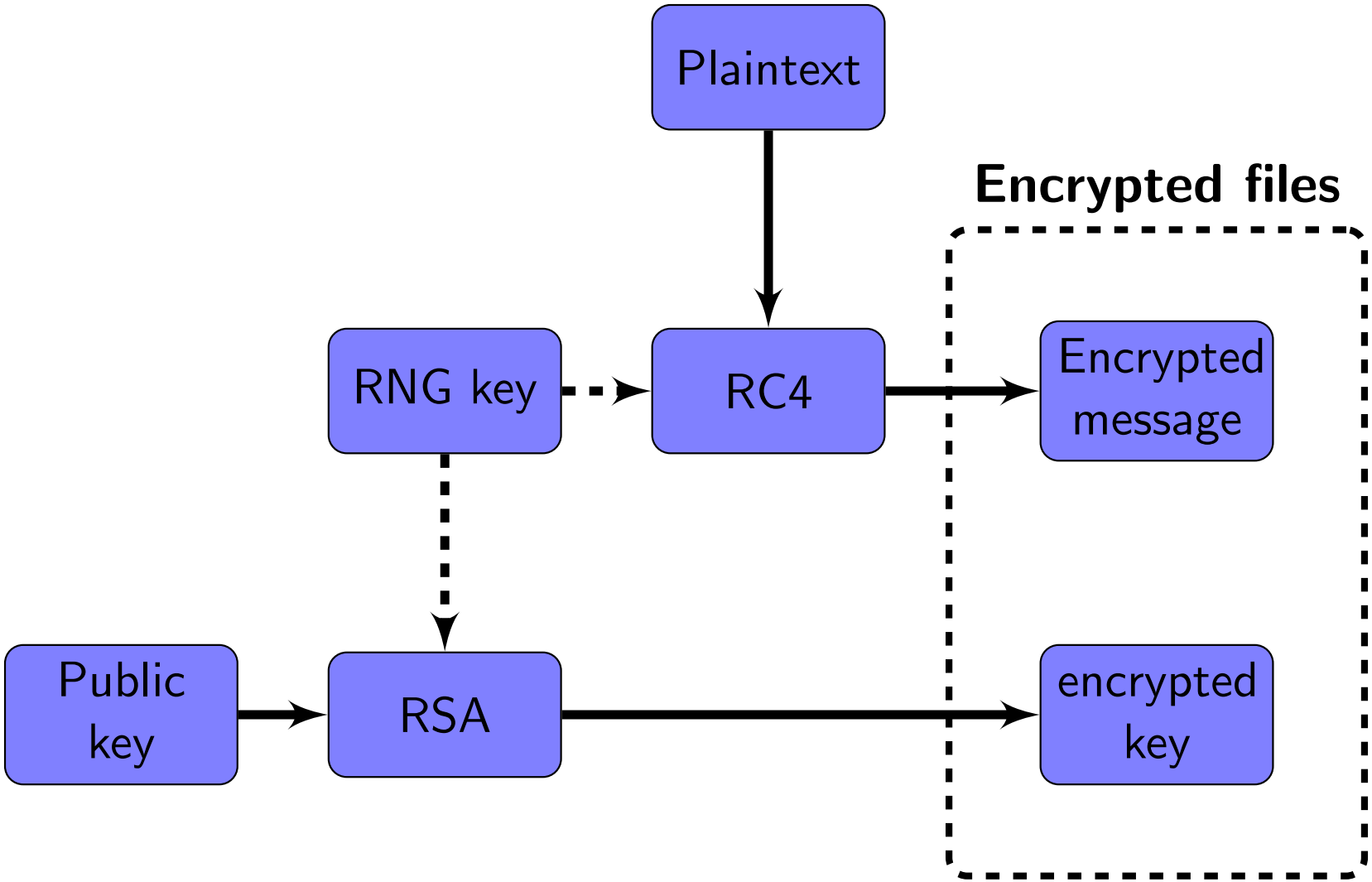

Figure: RSA encryption procedure

Files and RC4 key

- Public/Private RSA keys are stored as base 10 numbers (i.e. x_{10}) in a text file.
- RC4 password/key is a 16 character string. If shorter than 16, make use of zero padding
- See the practical guide for more information on files and keys

Deliverables

Implementation

- Implement RC4 from first principles.
- Implement RSA from first principles.
- Generate a public/private key pair using RSA, to encrypt RC4 key.
- Use the GMP library(http://gmplib.org/) for large numbers and complex modulus computations in RSA.
- Encrypt/decrypt a message using RC4.
- Doxygen documentation.

Deliverables

Demo

- Date: Monday, 15 June 2020 at 12pm (S.A.S.T)
- Your practical will be marked in absentia.
- 4 source code files are expected: **rc4, rsakeygen, rsaencrypt** and **rsadecrypt**

Submission

Demo

- Doxygen documentation (as a PDF).
- Code and makefile
- "ReadMe" file containing usage instructions.
- Upload all the zip file to the AMS site. Be sure to upload to your correct group upload slot. You can check the google sheet for your group number.
- Upload all as a ZIP file :

  EHN410-G*XX*-P2

  where:
  - GXX your group number.
  - P2 refers to practical 2.