PUBLIC-KEY CRYPTOGRAPHY - THE RSA CRYPTOSYSTEM

Nichita CARAGHENOV, Cristian CARTOFEANU, Daniel MACRINICI,
scientific supervisor Mihail KULEV

Technical University of Moldova

Abstract: *This paper's main goal is to deliver an accurate image on the concepts of data security and cryptography, by presenting the theory and by developing the application of a fundamental public-key cryptosystem used nowadays, namely the RSA cryptosystem. To prove the usefulness and power of public-key cryptography from a practical standpoint, a software application has been designed, that implements every algorithm inherent to the process of securing data (primes generation, key generation, encryption and decryption).*

Keywords: *Cryptography, RSA, public-key, encryption, decryption, factorization, application.*

1. Introduction

Public-key algorithms are most often based on the computational complexity of "hard" problems, often from number theory. For example, the hardness of RSA is related to the integer factorization problem, while Diffie–Hellman and DSA are related to the discrete logarithm problem. More recently, elliptic curve cryptography has developed in which security is based on number theoretic problems involving elliptic curves. Because of the difficulty of the underlying problems, most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly hybrid cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician, had developed an equivalent system in 1973, but it was classified until 1997[1].

The RSA algorithm involves three steps: key generation, encryption and decryption. RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers p and q.

For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.

2. Compute $n = p \cdot q$.

n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.

3. Compute $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1)$, where φ is Euler's totient function.
4. Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$; i.e., e and $\varphi(n)$ are coprime.

e is released as the public key exponent. e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $2^{16}+1= 65537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.

5. Determine d as $d-1\equiv e \ (mod\ \varphi(n))$, i.e., d is the multiplicative inverse of $e \ (modulo\ \varphi(n))$.

This is more clearly stated as solve for d given $d \cdot e\equiv 1 \ (mod\ \varphi(n))$. This is often computed using the extended Euclidean algorithm.d is kept as the private key exponent.By construction, $d \cdot e \equiv 1 \ (mod\ \varphi(n))$. The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\varphi(n)$ must also be kept secret because they can be used to calculate d [2].

An example of real life use of encryption and decryption of a message over a communication channel is presented below [3].

Encryption:

Alice transmits her public key (n, e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice.

He first turns M into an integer m, such that $0 \leq m < n$ by using an agreed-upon reversible protocol known asa padding scheme. He then computes the ciphertext c corresponding to $c \equiv m{\bullet}e \ (mod\ n)$.

This can be done quickly using the method of exponentiation by squaring. Bob then transmits c to Alice.

Decryption:

Alice can recover m from c by using her private key exponent d via computing

$m = c{\bullet}d \ (mod\ n)$ [6].

2. Application description

To prove the usefulness and power of public-key cryptography from a practical standpoint, a software application has been designed, named RSA Cryptosystem Application which allows the user to fully understand, appreciate and successfully use the public-key cryptography encryption of data, employing the RSA cryptosystem. The application has been developed in Python version 2.7.5 using TKInter GUI.

In terms of the graphical user interface, the application is made out of five sections (GUI tabs), each section expressing a part of the fundamental functionality of a public-key cryptosystem. The sections (tabs) and their functionalities are:

- Plain– the section tab in which the user introduces the input that is to be encrypted. .
- Pub Key – the public key generation component, activated by the “Choose Key” button using the primes obtained in the Primes Generation method.
- Pri Key - the private key generation component, activated by the “Choose Key” button using the primes obtained in the Primes Generation method.
- Cipher – the content encryption section, which encrypts the data (or any file) chosen by the user, according to a public key obtained in the Pub Key section, corresponding to a public-key cryptosystem (RSA).
- Digest – the section for displaying the output of the corresponding encrypted hash-value (represented as a hexadecimal output).

Screenshots of the Python RSA GUI:

A public and private key are randomly generated the instruction "Enter plain text here." You may use that very sentence as the plain text to be encoded, or you may type your own in. Pressing the little screw head button to the left deletes the current plain text fig(1).

Plain text and cipher text may be entered by typing into the corresponding entry windows or they may be loaded in from files by using the pulldown File menu fig(2).

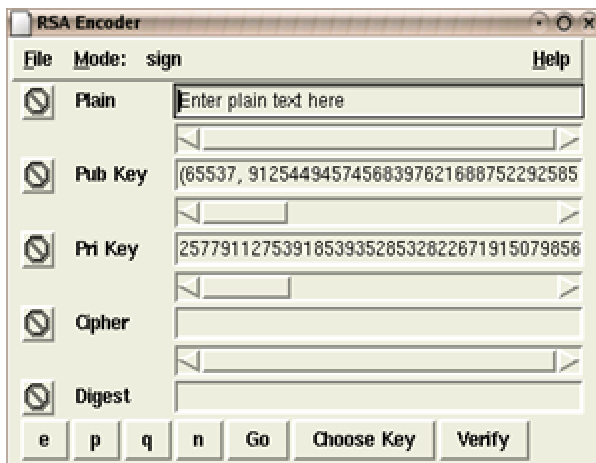

Fig. 1 Screenshot of startup state

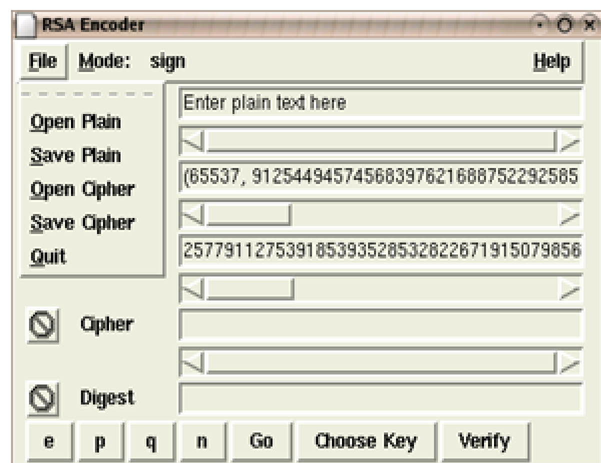

Fig. 2 Screenshot of File menu

There are 5 numbers that are important in the RSA protocol, *e, d, p, q*, and *n*. *d* is the private key. The others may be viewed by pressing the corresponding buttons at the bottom left. This shot shows the result of pressing the e key fig(3).

Choosing a mode fires up the action appropriate to the mode. Pressing the Go key while already in the mode does the same. This shot shows the result of pressing the Go key while in encode mode. The cipher window fills up with cipher text, the appropriate encoding for the current plain text using the current public key fig(4).

Fig. 3 Important RSA components

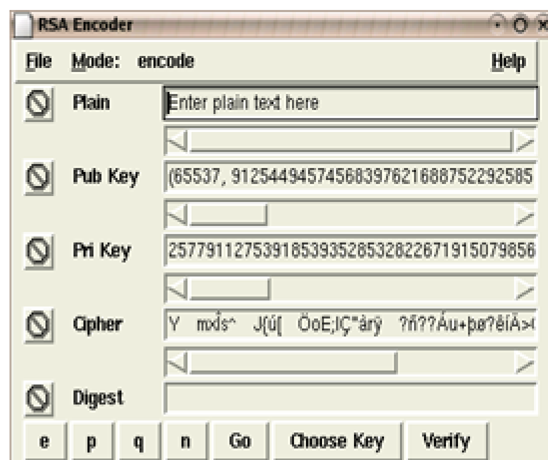

Fig. 4 Ciphertext generation

3. Tools and code:

As a tool for the application Python GUI TkInter was used to build the interface. Regarding the implementation for the encryption function, the snippet of code is presented below [5]:

```
def encrypt(self, plaintext):
        //encrypt(plaintext:string|long) : tuple
        // Encrypt the string or integer plaintext.
        wasString=0
        if isinstance(plaintext, types.StringType):
            plaintext = number.bytes_to_long(plaintext) ; wasString=1
        ciphertext = self.do_encrypt(plaintext)
        if wasString: return number.long_to_bytes(ciphertext)
        else: return ciphertext
def do_encrypt(self, plaintext):
        if self.n<=plaintext:
            raise error, 'Plaintext too large'
        return pow(plaintext, self.e, self.n)
```

The contents of the function are self-explanatory. It is necessary to emphasize the simplicity of the algorithm of the do_encrypt function. The background of these illustrated functions rely in chapter 2. The core here is that for computing the cipher text the plaintext (converted into a long format representation) must be raised at the power of the public key e modulo n.

The next section covering the implementation explains how the RSAkey object is created together with all the key components: *p,q,n,e,d* and max_len. Also, the process of working with tuples displays effectively how to construct the following key-rings:

```
def calculate_key_components(p,q, progress_func=None):
    //Given p and q calculate the other key components.
    //p shall be smaller than q
    if p > q:
        (p, q)=(q, p)
    p = p
    q = q
```

```
        n = p * q
        if progress_func:
            progress_func('e,d\n')
        (e,d,max_str_len) = calculate_e_d_and_max_len(p,q,n)
        return (p,q,n,e,d,max_str_len)
    def calculate_e_d_and_max_len(p,q,n):
        e = 65537L
        d = number.inverse(e, (p-1)*(q-1))
        max_str_len = calculate_max_str_len(n)
        return e, d, max_str_len
    def calculate_q_d_key_components(e,n,p):
        //Produce an approriate key obj based on pub/pri string info from display
        if p:
            assert (n % p) == 0
            q = n/p
            obj = build_key_obj('','private',p,q)
        else:
            obj = build_key_obj('','public',e,n)
        return obj [7],[8]
```

Conclusion

The computer application (RSA Cryptosystem Application) presents a successful approach to securing limited amounts of data, as well as providing a safe container for symmetric-key cryptosystem keys, that are about to be distributed over unsecured communication channels. The power of the application resides in the strength of its algorithms, the efficiency of its implementation and its large key size of 1536 bits, while its accessibility is enhanced by the use of intuitive graphical user interface, suitable for users having different levels of knowledge of cryptography and its inherent protocols.

Bibliography

4. BELLARE, Mihir, ROGAWAY, Phillip. *Introduction to Modern Cryptography*, 2005.
5. RIVEST, Ronald L.,VAN LEEUWEN, J. *Cryptology, Handbook of Theoretical Computer Science,* 1990.
6. MENEZES, A.J., VAN OORSCHOT, P.C., VANSTONE, S.A. *Handbook of Applied Cryptography,* ISBN 0-8493-8523-7.
7. GOLDREICH, Oded. *Foundations of Cryptography,* Cambridge University Press, 2001, ISBN 0-521-79172-3.
8. *Python documentation/* http://www.python.org (accessed on 17.05.2013).
9. *The Story of Alice and Bob/* http://www.conceptlabs.co.uk/alicebob.html (accessed on 25.04.2013).
10. *The Primes Page/* http://primes.utm.edu (accessed on 18.03.2013).
11. *True Random Number Service/* http://www.random.org (accessed on 17.06.2013).