HW3 – RSA Encryption/Decryption Design

Students are Maxwell Petersen, mpeter39, and Danny Chen, schen204. The assignment is to implement the RSA Encryption/Decryption algorithm in Java.

The purpose of this project is to create a GUI that performs the RSA algorithm to encrypt and decrypt messages. The process works with two different keys, the public and the private. The public key is a pair of values (e, n), and the private key is the pair (d, n). They are generated in the following steps:

1. Choose two distinct, large prime numbers p and q.
2. Calculate $n = p * q$
3. Calculate $\varphi = (p - 1) * (q - 1)$
4. Choose an integer for the value of e such that $1 < e < \varphi$ and $\gcd(e, \varphi) = 1$
5. Calculate d such that $(d * e) \bmod \varphi = 1$

The encryption of a value *M* into *C* is performed by the operation:

$$C = M^e \bmod n$$

The decryption of a value *C* back to *M* is performed by the operation:

$$M = C^d \bmod n$$

In order to make the encrypted message more manageable, we will be diving the message into smaller blocks of 10 or less characters and use only a subset of the ASCII values. The characters we're using include the 95 printable ASCII characters (decimal values 32 to 126) and 5 special characters. With 100 characters in our alphabet, every 2 digits in the blocked message will correspond to a character in the original message. The characters and their corresponding blocked values are in the table below.

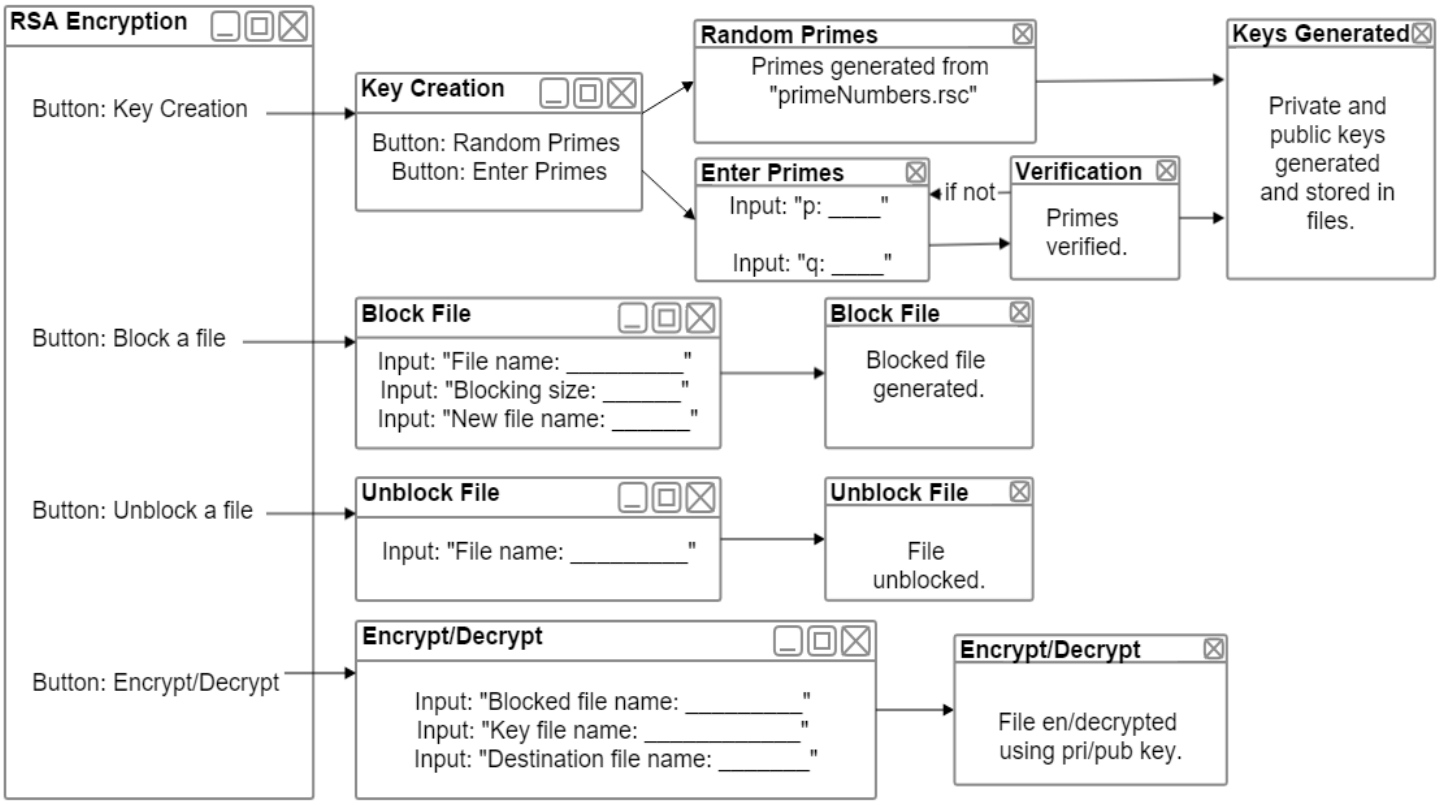
	0	1	2	3	4	5	6	7	8	9
0	\0	\v	\t	\n	\r	SP	!	"	#	$
10	%	&	'	(	)	*	+	,	-	.
20	/	0	1	2	3	4	5	6	7	8
30	9	:	;	<	=	>	?	@	A	B
40	C	D	E	F	G	H	I	J	K	L
50	M	N	O	P	Q	R	S	T	U	V
60	W	X	Y	Z	[	\	]	^	_	`
70	a	b	c	d	e	f	g	h	i	j
80	k	l	m	n	o	p	q	r	s	t
90	u	v	w	x	y	z	{	|	}	~

The idea is that the public key is used to encrypt the message to be sent, and the recipient has the private key that can decrypt the message. The security of the encryption comes down to the problem of obtaining the private key given only the public key. The difficulty in the calculation of the private key comes from factorizing the composite number *n* into *p* and *q*, because there are currently no efficient methods to factorize very large numbers.

Our implementation will consist of 5 main parts: a class that holds very large unsigned integers, a class to generate keys, a class to divide a message into blocks, a class to perform the encryption and decryption, and a GUI to tie everything together for the user.

The GUI allows the user to navigate and use the RSA algorithm to encrypt and decrypt files with ease. The "Key Creation" menu can generate random primes to be used for the keys, or the user can specify the primes and the program will validate them. In either case, the generated private and public keys will be stored in separate files in XML format. The "Block File" menu asks the user for a file containing the original message, a blocking size, and a name for the blocked file. The file produced contains the original message's corresponding values, according to the table above, divided into chunks of the blocking size per line. It is to be used with the encrypt operation. The "Unblock File" menu takes a blocked file and reverses the blocking process so that the original message can be recovered. The "Encrypt/Decrypt" menu applies the encryption/decryption algorithms above to a blocked file depending on what key the user provides. The output file is another blocked file with the message values altered and the user can specify a name for it. The interactions are shown in Figure 1 below.

Figure 1: front-end GUI design



The low level classes that perform operations under the GUI are shown in the figure on the next page. The HugeInt class is used to replace the primitive integer types because the numbers we'll be working with are much larger than 64 bits. The value is stored as a string. The class also contains binary operations for addition, subtraction, multiplication, division, modulus, and exponentiation. The current instance of the class is the first operand in these methods and the HugeInt parameter is the second operand. The compare() method takes a HugeInt as parameter and compares it to the current instance. It returns -1 if the parameter is less than the instance, 0 if they're equal, and 1 if the parameter is greater.

The RSAKeyGen class is used in the "Key Creation" menu. The constructor with no parameters will generate primes randomly from a file. The constructor taking two HugeInts as parameters will use the primes given by the user. The checkPrime() method is used to verify that the given primes are legitimate. Both constructors then use the primes to compute the private and public key pairs according to the algorithm described above. The keyToFile() method takes a key pair and stores the values into a file in XML format.

The Blocking class serves to format the message files. The block() method reads the input file according to the specified block size per line and stores the blocked message into a new file. If the number of characters in the original file is not a multiple of the block size, the last block will be padded with NULL characters. The unblock() method takes a blocked file, reads a value line by line, parses it into *B* ASCII characters, where *B* is the blocking size, and puts the message back together. NULL characters used for padding are ignored.

The Transform class is used for encryption and decryption. The encryptToFile() and decryptToFile() methods take a blocked file and apply the respective algorithm according to the provided key pair then produces a new blocked file containing the encrypted or decrypted message. The encrypt() and decrypt() methods are helpers used to apply the algorithm to each value.

Figure 2: lower level classes that perform operations of the RSA algorithm