```
<?php

/*************************************************************************
 Code to be Executed
 You should include your code below. The subsequent sections contain the
 set of functions. Some of these functions are already written for you,
 others you need to complete.
 *************************************************************************/

/* Task 1 */


/* Task 2 */


/* Task 3 */



/*************************************************************************
 LEARNING PHP
 The following code illustrates some of the necessary features of PHP
 By viewing the output, you should be able to understand how PHP works
 You should delete or comment out this code when you begin your
 development. THIS CODE MUST NOT BE INCLUDED WHEN YOU SUBMIT YOUR ASSIGNMENT
 *************************************************************************/

echo "\nLets try some basic PHP ... \n\n";

/* Variables */
/* Note that you don't have to define data types */
$i = 5;
$j = "five";
echo $i . " " . $j . "\n";

/* Create an array of integers and strings */
/* Note there are different ways to create arrays - this is just 2 examples */
$arr_of_int = array(1,2,3);
$arr_of_str = array();
$arr_of_str[0] = "hello";
$arr_of_str[1] = "world";

/* Print an array */
print_r($arr_of_int);
print_r($arr_of_str);

/* Output some text */
echo "This is a test\n";
echo $arr_of_str[0] . " " . $arr_of_str[1] . "\n";

/* Convert a data type into a string and vice versa */
/* The serial form is useful for sending a data structure over the network */
$serial_arr_of_str = serialize($arr_of_str);
echo $serial_arr_of_str;
```

```php
echo "\n";
print_r(unserialize($serial_arr_of_str));
echo "\n";

/* BC Math */
/* You must use special BC Math functions in the RSA Encrypt and Decrypt,
 * so that large calculates (exponentials) can be performed. You do so using
 * the functions with integers input as strings */
echo "5 + 3 = " . bcadd('5','3') . "\n";
echo "6 x 5 = " . bcmul('6','5') . "\n";
echo "2 to the power of 3 = " . bcpow('2','3') . "\n";
/* Note there is also a function bcpowmod */

/* You can explicitly convert between data types using settype */
$i = 5;  /* $i is an integer */
settype($i,"string");  /* $i is now a string */


/* Try some of the included functions */
echo "\nNow lets try some security related functions ...\n\n";

$n = 10;
echo "The prime numbers up to " . $n . " are:\n";
print_r(Sec_ListPrimes($n));
echo "\n";

$n = 134569;
if (Sec_IsPrime($n))
    echo $n . " is prime\n";
else
    echo $n . " is not prime\n";

$n = 187;
$factors = Sec_GetPrimeFactors($n);
echo "The prime factors of " . $n . " are:\n";
print_r($factors);

$i1 = 17;
$i2 = 3;
$a = Sec_Gcd($i1,$i2);
echo "The GCD of " . $i1 . " and " . $i2 . " is: " . $a[0] . "\n";
echo "The multiplicative inverse of " . $i2 . " mod " . $i1 . " is: " . $a[1] . "\n";

$n = 8;
$a = Sec_RelativelyPrime($n);
echo "The numbers relatively prime to " . $n . " are:\n";
print_r($a);
echo "\n";

echo "The totient of " . $n . " is:\n";
echo Sec_Totient(8) . "\n";
```

```
/***************************************************************************
 RSA
 **************************************************************************/
function RSA_Encrypt($plaintext,$key) {
/* Performs RSA encryption
 * Inputs:
 *   $plaintext: the message to be encrypted. This should be an integer
 *              represented as string (so BC math can be used)
 *   $key: the RSA key to perform the encryption. This should be an array
 *         indicating appropriate RSA key
 * Output:
 *   $ciphertext: the encrypted message. This should be an integer
 *              represented as a string (so BC math can be used)
 */
        /* ====== START YOUR CODE HERE ====== */



        /* ====== END YOUR CODE HERE ====== */
     return $ciphertext;
}

function RSA_Decrypt($ciphertext,$key) {
/* Performs RSA decryption
 * Inputs:
 *   $ciphertext: the message to be decrypted. This should be an integer
 *               represented as string (so BC math can be used)
 *   $key: the RSA key to perform the decryption. This should be an array
 *         indicating appropriate RSA key
 * Output:
 *   $plaintext: the original message. This should be an integer
 *              represented as a string (so BC math can be used)
 */
        /* ====== START YOUR CODE HERE ====== */



        /* ====== END YOUR CODE HERE ====== */
     return $plaintext;
}

function RSA_GenerateKeys($p=-1, $q=-1) {
/* Generates an RSA Public/Private key pair
 * Inputs:
 *   - none
 * Outputs:
 *   - rsakeys (array). See the end of this function for the format.
 *
 * Algorithm notes and constraint:
 *   - You must randomly generate p and q by:
 *       choosing a random prime number between the 31 (the 11th prime)
 *       and 293 (the 62nd prime) inclusive.
 *  - use BC Math in your calculation of keys
 *  - you must randomly select a value of e within the range of the 1st
*   valid number up to the 100th valid number inclusive. Note that may
*   not always be 100 valid values of e.
*/
    /* Constants. Use these if necessary. */
    $lowprime = 10;  /* The 11th prime (31) is indexed number 10 */
    $highprime = 294;  /* The 62th prime is 293. Therefore you could
            list primes up to 294 */
    $limite = 100;  /* choose an e up to the 100th valid value */

    /* ====== START YOUR CODE HERE ====== */



    /* ====== END YOUR CODE HERE ====== */
    /* Set all key varaibles to be strings (for future use in BC Math) */
    settype($n,"string");
    settype($e,"string");
    settype($d,"string");
    settype($p,"string");
    settype($q,"string");

    /* Put the keys into the appropriate data structures */
    $publickey[0] = $e;  /* e is a string */
    $privatekey[0] = $d;  /* d is a string */
    $publickey[1] = $n;  /* n is a string */
    $privatekey[1] = $n;

    $rsakeys['pu'] = $publickey;
    $rsakeys['pr'] = $privatekey;
    $rsakeys['p'] = $p;  /* p is a string */
    $rsakeys['q'] = $q;  /* q is a string */

    return $rsakeys;
}


function RSA_Sign($signerskey,$message) {
/* Creates a certificate in the format for the assignment
 * Inputs:
 *   signerskey: the key used to sign the certificate
 *   message: to be signed
 * Outputs:
 *   certificate: "Message Encrypt(Hash(Message))" as a string
 */
    /* ====== START YOUR CODE HERE ====== */



    /* ====== END YOUR CODE HERE ====== */
    return $certificate;
}

function RSA_Validate($signerskey,$receivedmessage,$receivedsignature) {
/* Validate a RSA signed message
```

```
 * Inputs:
 *   signerskey: the appropriate key of the person who signed the message
 *   receivedmessage: the unsigned part of the message received
 *   receivedsignature: the signed part of the message received
 * Outputs:
 *   result: true if valid, false if invalid
 */

    /* ====== START YOUR CODE HERE ====== */



    /* ====== END YOUR CODE HERE ====== */
    return $result;
}

function RSA_Attack($publickey,$ciphertext) {
/* Break RSA
 * Inputs:
 *    publickey: the users public key
 *    ciphertext: the capture ciphertext
 * Outputs:
 *    cracked: an array containing d and plaintext
 *           If you can only calculate d or plaintext (not both)
 *           then set the other one to 'notfound'
 */
    $e = $publickey[0];
    $n = $publickey[1];
    settype($n,"integer"); /* Be sure to set back to string when Decrypt */
    /* ====== START YOUR CODE HERE ====== */



    /* ====== END YOUR CODE HERE ====== */
    $cracked['d'] = $d;
    $cracked['plaintext'] = $plaintext;
    return $cracked;
}

function RSA_ValidateCertificate($signerskey,$receivedmessage) {
/* Validate a RSA signed message, specifically for the certificate
 * Inputs:
 *   signerskey: the appropriate key of the person who signed the message
 *   receivedmessage: the entire message received
 * Outputs:
 *   result: true if valid, false if invalid
 */
    /* Break certificate in to three parts: PU | ID | Signature */
    $n = sscanf($receivedmessage,"%s %s %s",$publickey,$id,$signature);

    $result = RSA_Validate($signerskey,$publickey . " " . $id,$signature);

    return $result;
}
```

```php
function RSA_ValidateCiphertext($signerskey,$receivedmessage) {
/* Validate a RSA signed message, specifically for the ciphertext
 * Inputs:
 *   signerskey: the appropriate key of the person who signed the message
 *   receivedmessage: the entire message received
 * Outputs:
 *   result: true if valid, false if invalid
 */
    /* Break received message in to two parts: C | Signature */
    $n = sscanf($receivedmessage,"%s %s",$ciphertext,$signature);

    $result = RSA_Validate($signerskey,$ciphertext,$signature);

    return $result;
}

function RSA_PrintKeys($rsakeys) {
/* Prints the RSA key in a user friendly manner */
    echo "RSA key information\n";
    echo "   Primes: (p=" . $rsakeys['p'] . ",q=" . $rsakeys['q'] . ")\n";
    echo "   Public key: (e=" . $rsakeys['pu'][0] . ",n=" . $rsakeys['pu'][1] . ")\n";
    echo "   Private key: (d=" . $rsakeys['pr'][0] . ",n=" . $rsakeys['pr'][1] . ")\n";
}


/**************************************************************************
 Hash
 **************************************************************************/

function Hash_Simple($message) {
/* Calculate the hash of a message
 * Input: $message - the data to be hashed. This should be a string.
 * Output: $hash - the hash value. This should be an integer.
 */
    $i = 0;
    $n = 0;
    while ($i < strlen($message)) {
        $n = $n + ord($message[$i]);
        $i++;
    }
    $hash = ($n % 128);
    return $hash;
}


/**************************************************************************
 General functions
 These function are provided for you to use. They may not be the best
 or most efficient implementations, but that are suitable for this
 assignment. You should not need to modify this code.
 **************************************************************************/
function Sec_IsPrime($n) {
/* Returns true of $n is a prime number
```

```
 * Code from http://www.phpmath.com/
 */
    $sqrtn = sqrt($n);
    for($i=2; $i <= $sqrtn; $i++ )
        if ($n % $i == 0) return false;
    return true ;
}

function Sec_ListPrimes($upTo) {
/* Return the prime numbers up to the parameter $upTo
 * Code from http://www.phpmath.com/
 */
    if ($upTo > 0) {
        $size   = $upTo + 1;
        $flags  = array();
        $primes = array();
        $limit  = sqrt($size);

        // Set flags
        for ($i=2; $i < $size; $i++)
            $flags[$i] = true;

        // Cross out multiples of 2
        $j = 2;
        for ($i=$j+$j; $i < $size; $i=$i+$j)
            $flags[$i] = false;

        // Cross out multiples of odd numbers
        for ($j=3; $j <= $limit; $j=$j+2) {
            if ($flags[$j]) {
                for ($i=$j+$j; $i < $size; $i=$i+$j)
                    $flags[$i] = false;
            }
        }

        // Build list of primes from what is left
        for ($i=2; $i < $size; $i++) {
            if ($flags[$i])
                $primes[] = $i;
        }

        return $primes;
    } else
        die("n must be greater than 0.");
}

function Sec_GetPrimeFactors($n,$max=500000) {
/* Return the prime factors of $n
 * $max is an optional parameter, limiting the search
 *  If $max is not given as an input, then the default is used
 * Code from http://www.phpmath.com/
 */

    $sqrtn = (int)ceil(sqrt($max));
    $trialDivisors = Sec_ListPrimes($sqrtn);
    $trialDivisors[] = $sqrtn;

    $factors = array();
    if ($n > 0) {
        if ($n > $max)
            die("Illegal arguments: ".$n." > ".$max);

        $k = 0;
        while ($n > 1) {
            $divisor   = $trialDivisors[$k];
            $quotient  = $n / $divisor;
            $remainder = $n % $divisor;
            if ($remainder == 0) {
                $factors[] = $divisor;
                $n         = $quotient;
            } else {
                if ($quotient > $divisor)
                    $k++;
                else {
                    $factors[]=$n;
                    break;
                }
            }
        }
        return $factors;
    } else
        die("n must be greater than 0.");
}

function Sec_Gcd($a,$b) {
/* Returns an array with the first entry $x[0] containing the GCD of $a and $b
 * and the second entry $x[1] containing the multiplicative inverse of $b in mod $a
 * Uses the function Sec_EuclidGcd to peform the main calculation
 */
    $x = Sec_EuclidGcd($a,$b);
    if ($x[1] < 0) {
        if ($a > $b)
            $x[1] = $a + $x[1];
        else
            $x[1] = $b + $x[1];
    }
    return $x;
}

function Sec_EuclidGcd($a,$b,$x=0,$y=1) {
/*
 * Calculates the GCD and Multiplicative Inverse (if exists)
 * Code from http://www.ultimatespin.com/e_art_example.php
 * Modified to simplify the input parameters
 */

    // ensure that a > b, we can do this because the same two numbers always
    // have the same gcd.
    if($a < $b) {
        Sec_Swap($x,$y);
        Sec_Swap($a,$b);
    }

    //get the m of a/b = mb+r
    if($a != 0 && $b != 0) {
        $q = (int)($a/$b);
    } else {
        $q = 0;
    }

    //if there is no multiplicative inverse
    if(0 == $b)
        return array($a,'no inverse');
    //if there is a multiplicative invers
    elseif(1 == $b)
        return array($b,$y);
    //otherwise we are not at the gcd so we keep going
    else
        return Sec_EuclidGcd($b,($a - ($q * $b)),$y,($x - ($q * $y)));

}

function Sec_Swap(&$a,&$b) {
/* Swaps the two values $a and $b */
    $temp = $a;
    $a = $b;
    $b = $temp;
}

function Sec_RelativelyPrime($a,$limit=100000) {
/* Returns a list of numbers relatively prime to $a
 * The list is limited to $limit numbers
 */
    $relprime = array(1);
    $i = 2;
    $n = 1;
    while ($n < $limit and $i < $a) {
        $x = Sec_Gcd($a,$i);
        if ($x[0] == 1) {
            $relprime[] = $i;
            $n++;
        }
        $i++;
    }
    return $relprime;
}


function Sec_Totient($a) {
/* Returns Eulers Totient of $a (if it is known) */
    $limit = 100000;
    if (Sec_IsPrime($a))
```

```
        return $a-1;
    else {
        $relprime = Sec_RelativelyPrime($a,$limit);
        if (count($relprime) < $limit)
            return count($relprime);
        else
            return 'unknown';
    }
}


?>
```