Institut Universitaire de Technologie
Aix Marseille Université

IUT Aix-en-Provence
413 Avenue Gaston Berger
13625 Aix-en-Provence CEDEX 1
Tél : 04.42.93.90.00

Projet Mathématiques : Chat crypté en RSA

Anthony Loroscio
Loïck Mahieux
Thomas Munoz
Loïc Pauletto

23 octobre 2014

Document rédigé en LaTeX

Table des matières

Chapitre 1

Introduction

L'Internet est devenu omniprésent dans notre vie quotidienne. La démultiplication des dispositifs portables met en exergue notre vie personnelle. De plus, les récents déboires des principaux acteurs du web ne font que ternir le portrait d'une vie privée déjà affaiblit par les intrusions de plus en plus oppressantes menées par les agences gouvernementales.

A tel point que les systèmes de chiffrement de données sont de plus en plus en vogue. Nombreux sont les systèmes qui proposent désormais un chiffrement des données par défaut, tel que IOS 8 et Android 5. Mais comment cela fonctionne-t-il ?

Pour illustrer le fonctionnement de tel systèmes, nous avons choisit de créer un "chat" crypté en ligne. Nous avons choisit le système de chiffrement RSA car il répondait le mieux à nos attentes : simple à mettre en oeuvre (les chiffrements DES et AES étant trop gourmands en ressources pour pouvoir fonctionner sur nos machines), mais aussi celle qui s'adaptait le mieux à notre utilisation (chaque contact possède sa propre clé de déchiffrement donc aucune information sensible n'est envoyée).

Première partie

Le Fonctionnement

Chapitre 2

La théorie

2.1 RSA

2.1.1 Historique

Le chiffrement RSA (nommé par les initiales de ses trois inventeurs qui sont Ronald Rivest, Adi Shamir et Leonard Adleman) est un algorithme de cryptographie asymétrique c'est-à-dire qu'il se décompose en deux parties : clé privée et clé publique, il a été développé en 1977. RSA a été breveté par le Massachusetts Institute of Technology (MIT) en 1983 aux États-Unis. Le brevet a expiré le 21 septembre 2000. Les premières attaques visant à casser le chiffrement ont débutées en 1985 avec l'attaque de Håstad qui utilise le théorème du reste chinois.

2.1.2 Le fonctionnement général

Dans ces paragraphes nous expliquerons le fonctionnement théorique du RSA dans l'ordre chronologique.L'implémentation du chiffrement dans notre application sera expliquée dans une autre partie.

La création des clés

Cette création n'intervient pas à chaque chiffrement de nouveaux messages, car les clés peuvent êtres réutilisées. Pour chaque utilisateur du chiffrement, il y a une paire de clés (une publique, une privée). .La taille de clé dites "sûre" est d'au moins 1024bits mais peuvent aller jusqu'à 4192bits. En dessous 1024bits, le RSA n'est pas sûr il serait préférable d'utiliser des algorithmes plus puissant tels qu'AES. La génération de ces dernières se fait de la manière suivante :

1. Il faut choisir deux nombres premiers distincts P et Q.
2. Calculer le produit des deux N = P $\times$ Q. Qui est appelé module de chiffrement.
3. Calculer ϕ(N) = (P-1)(Q-1) qui est l'indicatrice d'Euler en N.
4. Trouver E tel que ϕ(N) et E soient premiers entre eux et strictement inférieurs à ϕ(N), E est appelé exposant de chiffrement
5. Calculer l'entier naturel D, inverse de $E \pmod{\phi(N)}$, et strictement inférieur à ϕ(N), appelé exposant de déchiffrement

Le couple (N,E) donne la clé publique de l'utilisateur et le couple (N,D) donne sa clé privée.

Chiffrement du message

Le chiffrement s'effectue selon la formule suivante :

$$\mathrm{C} \equiv \mathrm{M}^{E} \pmod{N}$$

Où M étant le message que l'on souhaite crypter et C le message étant crypté.

Déchiffrement du message

Cela s'effectue de la même manière que pour le chiffrement sauf que cette fois ci la formule devient :

$$\mathrm{M} \equiv \mathrm{C}^{D} \pmod{N}$$

2.1.3 Fonctionnement général

Afin de rendre le système de chat moins trivial et de proposer une solution claire et utilisable par tous, rendant ainsi le processus de cryptage transparent pour l'utilisateur final, nous avons décidé de n'afficher aucun calcul à l'utilisateur.

Lors de son inscription, un utilisateur se voit attribuer un identifiant unique et un couple clé publique/clé privée générée grâce à son adresse mail (supposée unique à chaque utilisateur), des informations personnelles (nom, prénom) sont elles aussi stockées afin d'identifier plus simplement un utilisateur.

Chaque message transmis entre deux utilisateurs (les conversations de groupe ne sont pas possible dans ce système) fait l'objet d'un nouvel enregistrement dans la base de données et est identifié par l'identifiant de l'expéditeur et celui du récepteur, le contenu du message est crypté et la date (en nombre de secondes écoulées depuis le 1^{er} janvier 1970) permet de distinguer et d'ordonner les messages envoyés par le même couple expéditeur/récepteur.

Pour gérer tout cela nous avons utilisé des langages tel que le **PHP** ou le **JavaScript**. Qui nous permettait de faire une application légère et rapide. Pour la mise en forme nous avons simplement utilisé le **HTML5** et le **CSS3**.Toutes les données spécifiques aux utilisateurs (données personnelles, conversations, ...) sont stockées dans une base de données **MySQL**, nous avons choisi ce système de gestion de base de données car il est léger et nous n'avions pas besoin d'un système disposant de trop de fonctionnalités, car cela nous aurait fait perdre en performances et en temps.

2.1.4 Limites

Le cryptage RSA repose sur la difficulté à factoriser un nombre en facteurs premiers, il est donc important de souligner qu'avec la puissance de calcul actuelle des ordinateurs, l'utilisation d'une clef de petite taille rend la factorisation facile, il est donc possible de retrouver la clef privée d'une tierce personne.

Une autre limite du cryptage RSA est que le fait de crypter un message caractère par caractère peut nuire à la fiabilité du cryptage. En effet, lorsque l'on crypte un message de taille conséquente, il est facile en calculant la fréquence d'apparition des caractères de trouver de qu'elle lettre il s'agit en se basant sur les fréquences habituelles d'apparition de lettres dans une langue donnée.

Pour remédier à cela, nous devons utiliser des clefs de grande taille, cela a pour conséquence d'augmenter de façon exponentielle le temps de calcul nécessaire à la factorisation de ce nombre en se basant sur la puissance de calcul actuelle des ordinateurs. Cependant, l'arrivée des ordinateurs quantiques nécessitera donc d'utiliser des cléfs encore plus grande ou alors d'abandonner RSA pour passer à un autre cryptage.

Pour éviter l'identifiation des lettres dans le message, il est indispensable de faire en sorte qu'un même caractère n'ai pas pour le même code une fois crypté dans tout le message. Pour remédier à cela, il ne faut plus crypter le message caractère par caractère mais par chunks.

Deuxième partie

Mise en application

Chapitre 3

La pratique

3.1 Fonctionnement général

Nous avons décidé de mettre en application le système de chat en utilisant les langages web **HTML** et **CSS** pour la mise en page et la structure graphique générale du site, **PHP** pour les opérations et calculs nécessaires au fonctionnement du chat et **JavaScript** (utilisation du framework **jQuery**) pour rendre les différentes opérations plus dynamiques et fluides pour l'utilisateur (utilisation d'**AJAX** pour effectuer les différentes opérations sans avoir à rafraîchir la page).

3.1.1 Modèle de conception et de programmation

Afin de rendre la phase de développement plus simple à partager entre les membres de l'équipe et permettre une meilleure maintenabilité du code nous avons opté pour la mise en place de la structure **MVC** (Modèle-Vue-Contrôleur) qui se découpe en trois fonctions principales :

— **Modèle** : Le modèle contient toutes les différentes opérations possibles entre la base de données et le site. Chaque classe correspond à une table de la base données.

Exemple (*/app/models/messages.php*) :

```
1   /*
2    * Insert a message in the database
3    */
4       public function sendMessage($post = array()){
5           $sql = 'INSERT INTO ' . self::TABLE . ' '
6                   . 'VALUES (?, ?, ?, ?)';
7
8           // Execute the query
9           Database::execute($sql, array($post['idE'], $post['idR'
                ], $post['message'], time()));
10      }
```

— **Vue** : La vue correspond à ce qui est réellement vu par l'utilisateur (liste des messages, des conversations, formulaire d'inscription ou de connexion, etc.).

Exemple (*/app/view/chat/search.php*) :

```
1     <p class="result">
2     <?php
3
4         // If we do not make a search for nothing we get the list of
              users
5         // who have their username which begin by $_POST['username']
6         if(!empty($_POST['username'])){
7             $result = User::getUserByName($_POST['username']);
8
9             // We display the result
10        foreach($result as $value){
11    ?>
12
13    <a href="#" id="<?php echo $value['id'];?>" class="userStart">
14        <span class="user">
15            <?php echo $value['firstname'] . ' ' . $value['lastname'
                  ]; ?>
16        </span>
17        <span class="plus"> + </span>
18    </a>
19    <br/>
20        <?php }
21        }?>
22 </p>
```

— **Contrôleur** : Le contrôleur correspond à la liaison entre le modèle et la vue, cela inclut donc les différents calculs effectués et la vérification de la validité de ce qui est transmis à la base de données.

Exemple (*/app/controllers/chat.php*)

```
1       public function sendMessage(){
2
3
4         /* We check if the user is connected
5          * otherwise it is impossible to send
6          * a message
7          */
8         if(Auth::isConnected()){
9             $input = new Input();
10            $post['message'] = $input->post('message');
11
12            // We assume that we have the receiver id via $_POST
13            $post['idR'] = $input->post('idR');
14            $post['idE'] = $_SESSION['user']['id'];
15
16            $user = Message::getArrayUser($post['idR']);
17
18            $rsa = new RSA();
```

```
19              $rsa->setModulus($user['modulus']);
20              $rsa->setPublicKey($user['public_key']);
21
22              $post['message'] = $rsa->encrypt(str_split($post['
                    message']));
23
24              Message::send($post);
25
26              /*
27              *  The user will not leave the chat
28              *  but he will see his message thanks to AJAX)
29              */
30              $this->loadView('chat');
31              $this->view->render();
32
33          } else {
34              // We redirect the user in the homepage
35              Popup::set('ERREUR', 'Vous n\'etes pas connect ', '
                    error');
36              $this->loadView();
37              $this->view->render();
38          }
39      }
```

3.2 Gestion des utilisateurs

3.2.1 Inscription

Lorsqu'un utilisateur a fini de saisir ses informations, celles-ci sont envoyées au contrôleur *home.php* qui va vérifier la validité des informations saisies, générer les clés et envoyer le tout à la base de données (grâce à la fonction *signin*)

```
1   public function signin() {
2
3           // If the user is already connected (so he's registered), we
                redirect him to the chat page
4           if(Auth::isConnected())
5               $this->redirect ('chat');
6
7           // If all the informations aren't empty
8           if (Input::post('nom') != null && Input::post('prenom') != null
                && Input::post('mail') != null && Input::post('pseudo') !=
                null && Input::post('pass') != null && Input::post('pass2') !=
                 null)
9           {
10              $passe = Input::post('pass');
11              $passe2 = Input::post('pass2');
12
13              if ($passe == $passe2)
14              {
15                  $pseudo = Input::post('pseudo');
16                  $email = Input::post('mail');
17
```

```
18                  // We generate the Public Key and the Private Key
19                  $passe = sha1($passe);
20                  $a = new Keys;
21                  $a->generate();
22                  $pubkey = $a->getPublicKey();
23                  $prikey = $a->getPrivateKey();
24                  $mod = $a->getModulus();
25
26                  // We put these informations in the database
27                  $sql = 'INSERT INTO users (`username`, `firstname`, `
                        lastname`, `password`, `public_key`, `private_key`, `
                        modulus`) VALUES (?,?,?,?,?,?,?)';
28                  Database::execute($sql,array($pseudo,Input::post('prenom'
                        ),Input::post('nom'),$passe,$pubkey,$prikey,$mod));
29
30                  // We confirm to the user that his registration is
                        completed
31                  Popup::set("Inscription termin e",'success');
32                  $this->loadView();
33                  $this->view->render();
34              }
35              else
36              {
37                  Popup::set("Les mots de passe ne corresponde pas.",'error
                        ');
38                  $this->loadView();
39                  $this->view->render('signin');
40              }
41          }
42          elseif (Input::post('submit') == 'Inscription' && (Input::post('
                nom') == null || Input::post('prenom') == null || Input::post(
                'mail') == null || Input::post('pseudo') == null || Input::
                post('pass') == null || Input::post('pass2') == null))
43          {
44              Popup::set("Tout les champs doivent  tre  complet s.",'error
                    ');
45              $this->loadView();
46              $this->view->render('signin');
47          }
48          else{
49              $this->loadView();
50              $this->view->render('signin');
51          }
52
53      }
```

3.2.2 Connexion

La connexion se fait en vérifiant si l'utilisateur saisi possède bien le mot de passe saisi, si ce n'est pas le cas, le système affiche une erreur et invite la personne à réessayer.

Extrait du fichier *Auth.php* qui vérifie la validité des identifiants en appellant le modèle concerné (*users.php*) :

```php
public static function connect($login, $password) {
        // We load the Model users
        self::loadModel('users');

        // We get the user who have the same login and password
        $user = self::$model->getUserConnect($login, $password);

       // If it's correct we store all the informations about the user in
           the SESSION variable
        if(!empty($user)){
            $_SESSION['user'] = $user;
            return TRUE;
        }

        return FALSE;
    }
```

3.3 Gestion des messages

3.3.1 Envoi des messages

Une fois que l'utilisateur a saisi et envoyé son message, le contenu du message est transmis au contrôleur *chat.php* qui va crypter le message, le message ainsi crypté va être ensuite transmis au modèle qui va effectuer l'enregistrement dans la base de données.

L'envoi de message est fait de manière dynamique grâce à **jQuery** et **AJAX** qui permet de transmettre le message au PHP sans avoir à raffraichir la page.

Extrait du fichier *chat.php* à la fonction *sendMessage* :

```php
public function sendMessage(){


        /* We check if the user is connected
         * otherwise it is impossible to send
         * a message
         */
        if(Auth::isConnected()){
            $input = new Input();
            $post['message'] = $input->post('message');

            // We assume that we have the receiver id via $_POST
            $post['idR'] = $input->post('idR');
            $post['idE'] = $_SESSION['user']['id'];
```

```
15
16              $user = Message::getArrayUser($post['idR']);
17              $rsa = new RSA();
18              $rsa->setModulus($user['modulus']);
19              $rsa->setPublicKey($user['public_key']);
20
21              $post['message'] = $rsa->encrypt(str_split($post['message']))
                    ;
22
23              Message::send($post);
24
25              /*
26              *  The user will not leave the chat
27              *  but he will see his message thanks to AJAX)
28              */
29              $this->loadView('chat');
30              $this->view->render();
31          } else {
32              // We redirect the user in the homepage
33              Popup::set('ERREUR', 'Vous n\'etes pas connect ', 'error');
34              $this->loadView();
35              $this->view->render();
36          }
37      }
```

Extrait du fichier */public/js/ajax.js*

```
1       // When the user press the enter key on the message's form
2       $('.form_message').on('keyup', function(e) {
3               if (e.which == 13 && ! e.shiftKey) {
4
5               // Stop the form from redirecting to sendMessage page
6               e.preventDefault();
7
8               // Store form values into variables
9               var idR = $('.idR').val();
10              var message = $('.form_message').val();
11
12              if(message !== ''){
13
14                  // Send the POST request
15                  $.post('/chat/sendMessage', {
16                      idR: idR,
17                      message: message
18                  });
19
20                 // Load the messages in the chat
21                 $('.messages').load('/chat p').html();
22
23                 //Auto-scroll down
24                 var oldscroll = $('.messages').scrollTop();
25                 $('.messages').animate({ scrollTop: oldscroll + $('.
                       messages').height() },500);
```

```
26
27                  // Clear the value of the <textarea> input
28                  $('.form_message').val('');
29
30              } else {
31                  alert('Il n\'y a rien dans votre message ...');
32              }
33          }
34      });
```

3.3.2 Réception des messages

La réception des messages d'une conversation donnée demande de déchiffrer les messages reçus (les messages envoyés ne sont pas déchiffrés dans notre système afin de montrer la transformation du message) et de les afficher (sous forme de bulle), le tout de manière dynamique (grâce à **jQuery** et **AJAX** on vérifie toutes les secondes la présence ou non d'un nouveau message afin de l'afficher si c'est le cas, le tout sans que l'utilisateur ait besoin de rafraîchir la page).

Extrait du fichier */public/ajax/ajax.js*

```
1   setInterval(function() {
2
3           // We store the last message of the conversation
4           var oldmess = $('.last').html();
5
6           // We load the messages
7           $('.messages').load('/chat .conversation').html();
8
9           //We load the conversation list
10          $('.list-conversation').load('/chat .list-conversation').html();
11
12          // We store the last message of the conversation
13          var newmes = $('.last').html();
14
15          // If there is a new message we scroll to the last message of the
                conversation
16          if (newmes !== oldmess) {
17            $('.messages').scrollBottom();
18          };
19  }, 1000);
```

3.4 Le chiffrement

3.4.1 Les clés

La fonction *generate* permet de générer la paire clé publique/privée de chaque nouvel utilisateur. Cette fonction et toutes les autres fonctions relatives au chiffrement on été développées en **PHP**.

Extrait du fichier *app/RSA/Keys.php*

```
1   public function generate($length = 16) {
2       //Definit la taille de la cle
3       $pLength = (int) ($length + 1) / 2; // La taille pour P
4       $qLength = $length - $pLength; //La taille pour Q
5
6       //Genere les deux entiers premiers distincts P et Q. L'op ration
            et r p te tant que P vaut Q, pour obtenir deux valeurs
           distinctes.
7       while($this->p == $this->q) {
8           //On affecte leurs valeurs    P et Q en utilisant la fonction
                "generatePrime" montr e apres
9           $this->p = $this->mathsTools->generatePrime($pLength);
10          $this->q = $this->mathsTools->generatePrime($qLength);
11      }
12
13      //Calcule le modulo grace    la fonction "mul" montr e apres
14      $this->n = $this->mathsTools->mul($this->p, $this->q);
15
16      //Calcul de Phi grace    "mul"
17      $this->phi =  $this->mathsTools->mul($this->p - 1, $this->q - 1);
18
19      //Calcul de E/D
20      do{
21
22           do{//Calcul de E
23
24              //Affecte    $min la valeur de Q si P<Q sinon affecte la
                    valeur de P
25              $min = ($this->p < $this->q) ? $this->q : $this->p;
26
27              //Affecte    E un nombre al atoire entre min+1 et Phi-1
28              $this->e = rand($min + 1,  ($this->phi)-1);
29
30              //Refait les operations precedentes tant que E et N (
                    modulo) ne sont pas premiers entre eux. V rification
                    effectu e par "isComprime" montr e plus tard.
31           }while($this->mathsTools->isCoprime($this->e,  $this->n));
32
33      //Une fois que E est trouv  on calcule D qui est son inverse
           modulo N cette op ration est effectue par "invert"
34      $this->d = $this->mathsTools->invert($this->e, $this->phi);
35
36      //Ces operations sont r  p  t  es  tant que d vaut 0
37      }while($this->d == 0);
```

```
38      }
```

Voici les diverses fonctions utilisées par *generate* pour effectuer la génération de clé. À divers endroits nous avons dû utiliser des fonctions systèmes ou provenant de bibliothèques externe car il était trop coûteux pour nous de refaire ces fonctions (et ainsi se concentrer au mieux sur l'essentiel)

— La fonction *generatePrime* cette fonction permet simplement de générer un entier premier d'une taille donnée, elle est extraite du fichier */app/RSA/RSAMathTools.php* :

```
1   public function generatePrime($length) {
2           //On affecte    une variable temporaire la valeur 1
3           $bin = "1";
4
5           //On affecte    une autre variable temporaire la valeur 0
6           $i = 0;
7
8           //Tant que i est inf rieur    la taille - 2 on ajoute  la
               variable bin de type string soit 1 ou 0 de mani re
               al atoire.
9           while($i++ < $length - 2)
10              $bin .= rand(0, 1);
11
12          //On ajoute 1    la fin
13          $bin .= "1";
14
15          //On transforme la variable bin qui contient un code binaire
                en d cimal et on affecte ce nombre a nb
16          $nb = bindec($bin);
17
18          //On retourne l'entier le plus proche du nombre obtenu
               gr ce a la fonction "gmp_nextprime" qui est une fonction
                fourni par le langage. "gmp_intval" permet de retouner
               un entier.
19          return gmp_intval(gmp_nextprime($nb));
20      }
```

— La fonction *mul* , qui multiplie les deux entiers qui lui sont passés en paramètres et retourne le résultat.

```
1   public function mul($a, $b) {
2           //Les fonctions utilis es sont des fonctions du systeme qui
                multiplient a par b.
3           return gmp_intval(gmp_mul($a, $b));
4       }
```

— La fonction *isComprime* retourne vrai ou faux en fonction du chiffre qui lui est passé en paramètre si il est premier ou non.

```
1   public function isPrime($a) {
2           //La fonction "gmp_prob_prime" retourne 2 si le nombre qui
               lui est pass  en param tre est premier, la fonction
               utilise le test de probabilit  Miller-Rabin.
```

```
3           return gmp_intval(gmp_prob_prime($a)) == 2;
4       }
```

— La fonction *invert* inverse le nombre à modulo n

```
1       public function invert($a, $n) {
2           //La fonction "gmp_invert" inverse    modulo n
3           return gmp_intval(gmp_invert($a, $n));
4       }
```

3.4.2 Le chiffrement des messages

Les fonctions suivantes sont appelées à chaque nouveau message envoyé par un utilisateur. À la fin de l'explication nous expliquerons les problèmes que nous avons rencontrés et les solutions que nous avons déployées pour faire face à ces derniers.

Le cryptage

Le cryptage est fait par une seule fonction qui fait appel à diverses fonctions que nous expliquerons dans l'ordre chronologique.
La fonction s'appelle *encrypt*, elle se trouve dans */app/RSA/RSA.php* :

```
1  public function encrypt($message) {
2          //Cette fonction permet de transformer un tableau en string
3          $message = implode($message);
4
5          //Ceci est une solution que nous avons trouv  au probl me que
               nous exposerons plus tard.
6          $message .= '~';
7
8          //Cette fonction change la string en tableau avec l'  quivalant
               ASCII de chaque lettre qui se trouvait dans la string
9          $message = $this->stringTools->str2Int($message);
10
11         //Cette fonction d coupe les blocs pr cedents en blocs de
               taille differente ce qui permet d'avoir un code diff rent
               pour chaque lettre.
12         $message = $this->stringTools->createChunk($message);
13
14
15         //c'est la variable qui va acceuillir le code crypt
16         $crypted = array();
17         foreach($message as $chunk) {
18             $letter = gmp_intval($this->mathsTools->modExp((int)$chunk,
                   $this->public_key, $this->modulus));
19             array_push($crypted, (string)$letter);
20         }
21
22         //$crypted = $this->stringTools->int2Str($crypted);
23         $crypted = implode(' ',$crypted);
24         return $crypted;
25     }
```

str2Int

Cette fonction rend un tableau d'équivalent ASCII d'une string qui lui est passée en paramètre.

```
 public function str2Int($tab) {
        $ASCII = array();
        if(is_string($tab))
            foreach(str_split($tab) as $value)
                array_push($ASCII, ord($value));
        else
            foreach($tab as $value)
                array_push($ASCII, ord($value));

        return $ASCII;
    }
```

createChunk

Cette fonction coupe des blocs en blocs d'une nouvelle taille.

```
public function createChunk($tocut, $size = 4) {
        $temp = strlen((string)$this->phi);
        if($temp != 0){
            if($size ==  $temp)
                $size-=1;
            elseif ($size > $temp)
                $size -= $temp -1;
            }
        $cut = '';
        foreach ($tocut as $val) {
        $str = '';
            $cout = (($size - 1) - strlen((string)$val));
            if($cout > 0)
            {
                for($i=0;$i<$cout;++$i)
                {
                    $str .= '0';
                }

            }
            $cut .= $str . $val;
        }
        $zero = array();
        for ($i = 0;$i < strlen($cut);++$i)
        {
            if($cut[$i] == '0')
            {
                array_push($zero,$i);
            }
        }
        $cut = str_split($cut,$size);
        for ($i=0; $i < count($zero) ; $i++) {
```

```
33              if ($i == 0)
34                  array_push($cut,ord('^').$zero[$i]);
35
36              array_push($cut,$zero[$i].ord('^'));
37          }
38          return $cut;
39      }
```

modExp

Cette fonction met a à la puissance b modulo n

```
1  public function modExp($a, $b, $n) {
2          return gmp_intval(gmp_powm($a, $b, $n));
3      }
```

Le decryptage

C'est le même fonctionnement que *encrypt*, sauf que cette fois la fonction se nomme *decrypt*.

```
1  public function decrypt($message) {
2          $message = explode(' ', $message);
3          $crypted = array();
4          foreach($message as $chunk) {
5              $letter = gmp_intval($this->mathsTools->modExp((int)$chunk,
                   $this->private_key, $this->modulus));
6              array_push($crypted, $letter);
7          }
8
9          $crypted = $this->stringTools->reformChunk($crypted);
10         $crypted = $this->stringTools->int2Str($crypted);
11         return $crypted;
12     }
```

reformChunk

Cette fonction, à l'inverse de *createChunk*, remet les blocs à leur taille originale pour ensuite être remit en caractère.

```
1  public function  reformChunk($toreform, $size = 4)
2      {
3          $reform = "";
4          $rech = implode("",$toreform);
5          $pos = strpos($rech, '126');
6          for ($i = 0 ;$i<$pos;$i++)
7          {
8            $reform .= $rech[$i];
9
10         }
11         $a = "";
12         for ($i = $pos+3; $i < strlen($rech) ;$i++ ) {
13             $a .= $rech[$i];
```

```
        }
        $a = str_replace(ord('^'), ' ', $a);
        $zero = explode(" ", $a);
        for ($i=0; $i < count($zero) ; $i++) {
            if ($zero[$i] == '') {
                        unset($zero[$i]);
                    }
            }
        unset($zero[count($zero)+1]);
        foreach ($zero as $temp) {
            if(!empty($reform[$temp]) && $reform[$temp]  != '0')
                $reform = substr_replace($reform, '0', $temp, 0);
            else
                continue;


        }
        $reform = str_split($reform,$size-1);
        return $reform;
    }
```

int2Str

Cette fontion repasse du code ASCII d'un caractère au caractère en lui même.

```
  public function int2Str($tab) {
        $ASCII = '';
        foreach($tab as $value)
            $ASCII .= chr($value);

        return $ASCII;
    }
```

3.4.3 Exemple de cryptage RSA

Soit la clé :
$P = 41$
$Q = 23$
$N = P \times Q = 41 \times 23 = 943$
$\phi(N) = (P-1)(Q-1) = (41-1)(23-1) = 880$
$E = 723$
$D = E^{-1} \pmod{\phi(N)}$
$723^{-1} \pmod{880}$
$723 \times y = 1 \pmod{880}$
$723 \times u + 880 \times v = 1$
$723 \times 667 + 880 \times 548 = 1$
On a donc $D = 667$

Nous avons donc pour clé privée (943, 723) et pour clé publique (943, 667)

On va donc crypter le message "BONJOUR"
On convertit le message "BONJOUR" en ASCII :

"Bonjour" => {66, 111, 110, 106, 111, 117, 114}

$66^{723} \pmod{943} = 86$
$111^{723} \pmod{943} = 773$
$110^{723} \pmod{943} = 591$
$106^{723} \pmod{943} = 171$
$111^{723} \pmod{943} = 773$
$117^{723} \pmod{943} = 440$
$114^{723} \pmod{943} = 91$

Le message crypté est donc : {86, 773, 591, 171, 773, 440, 91}

$86^{667} \pmod{943} = 66$
$773^{667} \pmod{943} = 111$
$591^{667} \pmod{943} = 110$
$171^{667} \pmod{943} = 106$
$773^{667} \pmod{943} = 111$
$440^{667} \pmod{943} = 117$
$91^{667} \pmod{943} = 114$

Le message décrypté est donc : {66, 111, 110, 106, 111, 117, 114}

On retrouve donc bien le message "BONJOUR"

3.4.4 Les problèmes et leurs solutions

Nous pensions utiliser un système de création de blocs. Cependant cette fonctionnalité nous a posé beaucoup de problèmes comme par exemple la disparition du 0 lors de la phase de chiffrement/déchiffrement et la consommation excéssive de ressources dans notre système (lors d'une gestion plus importante de messages et de contenu). Donc, pour palier au problème des 0, nous avons décidé de passer les positions des 0 enrobées d'une numéro spécifique qui nous permettait de les récuperer au décryptage des messages. Concernant la consommation excéssive de ressources, nous avons décidé de n'inclure la gestion des chunks uniquement dans une page de test.

Chapitre 4

Conclusion

La mise en place d'un "chat" montre bien le fonctionnement de systèmes chiffrés : au plus la valeur de la clé est importante, au plus la sécurité est garantie. Cependant, une clé de grande taille entraine de facto de fortes lenteurs, la sécurité prenant alors le dessus sur l'ergonomie.

Ce projet nous a permis d'appliquer nos connaissances et nos compétences en informatique dans un domaine dans lequel nous n'avions jusqu'à présent qu'une approche théorique.