PARTIE 1

exponentiation.h

```
int is_prime_naive(long p);
void print_is_prime(long p);
long modpow_naive(long a, long m, long n);
long modpow(long a, long m, long n);

Fonctions de tests de primalite naive et
d'exponentiation modulaire
```

miller_rabin.h

```
int witness(long a, long b, long d, long p);
long rand_long(long low, long up);
int is_prime_miller(long p, int k);
long random_prime_number(int low_size, int
up_size, int k);

Fonctions de test de primalite probabiliste
(Miller Rabin) et de generation aleatoire de
nombre premier
```

keys.h

```
long extended_gcd(long s, long t, long *u, long *v);
void generate_key_values(long p, long q, long* n, long *s, long *u);
char* decrypt(long* crypted, int size, long u, long n);
long* encrypt(char* chaine, long s, long n);

Fonctions de generation de pairs de cles pour le chiffrement RSA
ainsi que des fonctions de chiffrement/dechiffrement a l'aide de ces
cles.
```

PARTIE 2

Key

```
long val;
long n;

n: produit des deux nombres
premier
val: u ou s
```

Signature

```
long *content;
int size;
char *message;

content: message que la personne qui
signe a chiffre avec sa cle secrete
size: taille du tableau content
message: message chiffre avec la cle
publique du candidat
```

Protected

```
Key *pKey;
char *declaration_vote
Signature *sgn;

pKey: cle publique de celui
qui vote qui permet de valider
la signature
declaration_vote: cle publique
de pour qui il vote
sgn: signature du votant
```

keys_struct.h

```
void free_signature(Signature *s);
void free_protected(Protected *p);

void init_key(Key* key, long val, long n);
void init_pair_keys(Key* pKey, Key* sKey, long low_size, long up_size);

char* key_to_str(Key* key);
Key* str_to_key(char* str);

Signature* init_signature(long* content, int size);
Signature* sign(char* mess, Key* sKey);
char *signature_to_str(Signature *sgn);
Signature *str_to_signature(char *str);

Protected* init_protected(Key* pKey, char* mess, Signature* sgn);
int verify(Protected* pr);
char* protected_to_str(Protected *pr);
Protected* str_to_protected(char* str);

void print_long_vector(long *result, int size);

Fonctions permettant la creation de structure necessaire au processus electoral:
• Key -> cle publique ou secrete necessaire au chiffrement
•Signature -> Permet de signer une declaration a l'aide de sa cle secrete
• Protected -> contient la declaration de vote, la signature ainsi que la cle publique du
votant qui permet de verifier sa signature
Les fonctions to_str et str_to permettrons la serialisation/deserialisation dans des
fichiers texte
Les fonctions sign et verify permettent de signer une declaration puis de la valider à
l'aide de la cle publique du votant
```

PARTIE 3

cellKey

```
Key *data;
struct cellKey* next;

data: pointeur vers une cle
next: pointeur vers l'element suivant de
la liste chainee
```

cellProtected

```
Protected* data;
struct cellProtected* next;

data: pointeur vers une protected
next: pointeur vers l'element suivant
de la liste chainee
```

linked_keys.h

```
CellKey* create_cell_key(Key* key);
CellKey* inserer_cell_tete(Key* key, CellKey *next);

CellKey* read_public_keys(char* nomfic);
void print_list_keys(CellKey* c);
void delete_cell_key(CellKey* c);
void free_list_keys(CellKey *c);

CellProtected* inserer_list_protected(Protected *pr,
CellProtected *next);
CellProtected* create_cell_protected(Protected *pr);
CellProtected* read_declarations(char* nomfic);
void print_list_protected(CellProtected *pr);
void delete_cell_protected(CellProtected *pr);
void free_cell_protected(CellProtected *pr);

Fonctions permettant de deserialiser les
cles/declarations contenu dans les .txt et de mettre dans
une liste chainee
```

check_declaration.h

```
void delete_fake_signature(CellProtected *c);

Fonction permetant de supprimer les declarations contenant
une signature invalide à partie d'une liste chainee
```

PARTIE 4

PARTIE 5