1 Documentation Technique de l'UCL Benchmark

Ce document est provisoire, il sera sûrement modifié durant l'élaboration du benchmark. Il est uniquement réservé aux utilisateurs avancés désirant développer leurs scénarios manuellement, aux administrateurs désirant configurer correctement le benchmark et bien sur aux développeurs soucieux de faire évoluer le benchmark en y intégrant de nouvelles librairies.

Merci à Frédéric Lefèbvre pour son assistance technique sur l'architecture matlab et pour ses codes sources.

1.1 Création d'un scénario

1.1.1 Commandes du fichier de scénario

```
load txt_filename pvn_out
```

La commande *load* permet de charger le fichier image *txt_filename*, et de stocker sont contenu dans la variable *pvn_out*.

```
watermark WATERMARK_ID pvn_in pvn_out [[parametre valeur]...]
```

La commande *watermark* lance le watermark dont l'identifiant est *WATERMARK_ID*. Le watermark s'éffectue sur l'image *pvn_in*, et le résultat est stocké dans la variable *pvn_out*. Cette commande peut prendre des arguments optionnels.

```
attack ATTACK_ID pvn_in pvn_out [[parametre valeur]...]
```

La commande *attack* lance l'attaque *ATTACK_ID* sur l'image *pvn_in* et stock le résultat dans *pvn_out*. Cette commande peut prendre des arguments optionnels.

```
readmark WATERMARK_ID pvn_in txt_markres [[parametre valeur]...]
```

La commande *readmark* teste si l'image watermarkée par l'algorithme *WATERMARK_ID* contient bien le texte *txt_markres*. Cette commande peut prendre des arguments optionnels.

```
test TEST_ID pvn_before pvn_after [[parametre valeur]...]
```

La commande *test* lance le test de qualité *TEST_ID* entre l'image *pvn_before* et *pvn_after*. Cette commande peut prendre des arguments optionnels.

Le benchmark n'interprète pas les lignes commençant par le caractère #.

Chaque commande tient sur une ligne, le premier mot de la ligne correspond au nom de la commande interne permettant au benchmark d'identifier l'action à effectuer. Elle est suivit de 1 à 2 arguments obligatoires permettant de distinguer les données d'entrée et de sortie des différentes actions.

Tous les paramètres initialisés derrière la fonction seront passés aux librairies sans aucune interprétation du benchmark. Ce système permet à l'utilisateur de créer des nouveaux paramètres d'entrée pour sa librairie, dont il pourra librement affecter les valeurs lors de la génération du scénario. (CF char** param **§1.3.5**)

1.1.2 exemple de scénario

```
load lena.jpg my_lena
watermark digimark my_lena wm_lena text helloworld
#check if the watermark work
readmark digimark wm_lena helloworld

#check if the watermark is robust
attack blur wm_lena atk_lena blur_x 4 blur_y 4
readmark digimark atk_lena helloworld

#check the quality of image
test psnr my_lena wm_lena
test psnr my_lena atk_lena
```

1.1.3 scénario du comparatif

Le scénario du comparatif est un scénario commun à tout les watermark, il doit être saisit dans le fichier *conpare.cfg* contenu par le répertoire du serveur de benchmark.

La syntaxe de ce fichier de scénario est la même que les autres scénarios exception faite du nom de l'algorithme de warermarking qui devra être remplacer par cette macro-définition: **WATERMARK**

Le benchmark substituera le nom de l'algorithme utilisé à cette macro lors de l'exécution.

1.2 Administration du benchmark

Le système de benchmarking est composé de deux parties :

- Le serveur de benchmark. C'est la partie communiquant avec l'utilisateur, exécutant les fichiers de scénario et traitant la base de donnée d'image.
- Les serveurs de librairie. C'est une application qui communique au serveur de benchmark par l'intermédiaire du réseau et qui se charge de l'exécution des différents algorithmes de watermarking, d'attaque et de test.

Cette séparation était nécessaire pour permettre une abstraction de la plate-forme et du format utilisé pour les librairies.

Le paramétrage du benchmark est simplifié grâce à l'utilisation de fichier de configuration :

- *benchmark.cfg* est le fichier de configuration du serveur de benchmark, il devra se trouver dans le même répertoire que l'exécutable.
- *libserver.cfg* est le fichier de configuration d'un serveur de librairie. Il devra en exister un par serveur de librairie et devra se trouver dans le même répertoire que l'exécutable correspondant.

1.2.1 Environnement et dépendances

Le serveur de benchmark et le serveur de librairie Unix ont été développés sous Linux, ils peuvent être compilés sous Unix sans modification du code. Ces deux parties du benchmark sont complètement autonomes.

La librairie de chargement d'image jpeg nécessite que le fichier *libjpeg.so* soit disponible sur le système.

Le serveur de librairie Matlab nécessite la présence de Matlab sur le système ainsi que celle du fichier *libz.so* correspondant à la librairie gzip.

1.2.2 Syntaxe des fichiers de configuration

Les fichiers de configuration sont des fichiers ASCII. Ils pourront être indifféremment créés sous un éditeur Unix ou Windows. Ces fichiers sont découpés en section :

Le début de chaque section est représenté par une ligne contenant une chaîne du type [NomDeSection]

Dans chaque section se trouve une liste d'affectation de paramètre de la forme :
paramètre = valeur

On peut déclarer des paramètres de même noms dans des sections différentes.
Si plusieurs affectations d'un paramètre sont définies dans une même section, c'est la dernière affectation qui l'emporte sur les autres.

Le fichier de configuration n'est pas sensible aux lignes blanches et aux espaces en début et fin de ligne, par contre, il est très important de respecter la case.

1.2.3 Configuration du serveur de benchmark

Le fichier de configuration du serveur de benchmark se nomme *benchmark.cfg*.
Il doit contenir une section [*Général*] ou sera affecté le paramètre *port*.

La valeur du paramètre *port* correspondra au port d'écoute du benchmark.
C'est par ce port qu'il recevra les directives de l'utilisateur.

Dans la section [*Directory*] on doit affecter le paramètre *picdatabase*, qui contiendra le chemin relatif ou absolu des images à lire.

Dans la section [*PictureLoader*] une liste de paramètre portant le nom des extentions des images. Ces derniers devront contenir le nom de la librairie susceptible de lire le format correspondant.

Dans la section [*Libserver*] une liste de paramètres correspondant à la liste des serveurs de librairie devra être affectée de la manière suivante : mon_serveur = addresse_ip:port.

NB : l'adresse IP peut être un nom de machine.

La section [*Uploadhost*] devra comporter une liste de paramètres portant les noms des extensions des librairies à uploader. Sa valeur sera le nom du serveur de librairie pouvant exécuter ce type de librairie.

La section [*MySql*] doit contenir toutes les informations concernant la connexion à la base de donnée recevant les résultats.

Le paramètre *host* devra recevoir le nom ou l'adresse IP de la machine où se trouve le serveur MySql. Le paramètre *port* devra recevoir le port du serveur MySql. Les paramètres *user* et *password* devront contenir le nom et le mot de passe de l'utilisateur de connexion. Le paramètre *database* devra contenir le nom de la base contenant les tables du benchmark.

1.2.4 Exemple de benchmark.cfg

```
[General]
port = 54321

[MySql]
host = 130.104.224.230
port = 3306
user = wb
password = water
database = wb

[Directory]
picdatabase = /var/pics

[PictureLoader]
jpg = ldr_jpg.so
jpeg = ldr_jpg.so
gif = ldr_gif.so
ppm = ldr_ppm.so

[Libserver]
win32 = 130.210.21.42:12222
linux = caph.tele.ucl.ac.be:12345
linux2 = 12.153.55.123:12345
matlab = localhost:12345

[Uploadhost]
so = linux2
tgz = matlab
tar.gz = matlab
dll = win32
```

1.2.5 Configuration d'un serveur de librairie

Le fichier de configuration d'un serveur de librairie se nomme *libserver.cfg* et doit contenir une section [*Général*] ou sera affecté le paramètre *port*.

La valeur du paramètre *port* correspondra au port de communication entre le serveur de librairie et le serveur de benchmark. Il doit correspondre au port spécifié dans la section [*Libserver*] du fichier *benchmark.cfg*.

Le fichier *libserver.cfg* devra également contenir une section [*Uploadlib*] où trois paramètres seront définis :

- *watermarklib* : devra contenir le chemin relatif ou absolu du répertoire où seront uploadées les librairies de watermarking.
- *attacklib* : devra contenir le chemin relatif ou absolu du répertoire où seront uploadées les librairies d'attaque.
- *benchlib* : devra contenir le chemin relatif ou absolu du répertoire où seront uploadées les librairies de test de qualité.

NB : Il faudra s'assurer que le système d'exploitation est configuré pour aller chercher les librairies dans les répertoires spécifiés. Par exemple sous Linux, il faudra positionner la variable d'environnement *LD_LIBRARY_PATH* de manière à ce que le chemin absolu des répertoires y apparaisse.

NB : Les chemins relatifs sont autorisés mais l'utilisation de l'alias '~' est interdit.

1.2.6 Exemple de libserver.cfg

```
[General]
port = 12222

[Uploadlib]
watermarklib = c:/temp/watermark
attacklib = c:/lib/attack
benchlib = ./benchlib
```

1.3 Création d'une librairie C pour le benchmark

Toutes les fonctions des librairies (sauf *testfunc*) devront retourner 0 en cas de succès ou un code d'erreur dans le cas contraire.

Les entêtes des fonctions et les types utilisés sont identiques sur toutes plates-formes confondues et sur tous les types de système d'exploitation. Ceci pour permettre un portage aisé et rendre indépendant votre algorithme face au système utilisé. C'est pourquoi nous vous déconseillons d'utiliser les appels systèmes et autres descripteurs de fichier dans vos librairies, ce qui rendrait votre code C spécifique à une architecture.

1.3.1 Librairie de watermark

Une librairie de watermark devra contenir au moins 2 fonctions :

- La fonction *markfunc*, de type *markfunc_t*, permettra de modifier l'image source et créer l'image de destination en fonction des paramètres passés par l'utilisateur.
- La fonction *decodefunc*, de type *decodefunc_t*, permettra de lire le texte contenu dans une image watermarkée toujours avec des paramètres passés par l'utilisateur.

1.3.2 Librairie d'attaque

Une librairie d'attaque devra contenir au moins la fonction *attackfunc*, de type *attackfunc_t*. Cette fonction créera une nouvelle image en fonction de l'image source et des paramètres utilisateur.

1.3.3 Librairie de test de qualité

Une librairie de test de qualité devra permettre à l'utilisateur de recevoir un indice de qualité d'une image par rapport à une autre (PSNR, MSE…) L'utilisateur devra développer une fonction *testfunc* de type *testfunc_t* qui prendra les deux images en argument ainsi qu'une liste exhaustive de paramètres, et devra retourner un double contenant le résultat du test.

Si le test n'est pas faisable, la fonction devra retourner la valeur *QUA_ERR_VAL*.

1.3.4 Librairie de lecture d'image

Une librairie de lecture d'image devra permettre au benchmark de transformer un fichier image en une structure interprétable par ce dernier. Elle le fera au travers de la fonction *picldrfunc* de type *picldrfunc_t*.

Ce type de librairie est le seul à être stocké du coté 'serveur de benchmark', et devra donc être compilé pour l'architecture de ce dernier.

1.3.5 Types et arguments des fonctions

```
/* stucture definition of a bitmap in the benchmark scheme */
typedef struct
{
  int size_x;
  int size_y;
  unsigned char* bitmap;
} bitmap_t;

/* header of the watermarking function */
typedef int (markfunc_t)
 (bitmap_t* dest, bitmap_t* src, char** param);

/* header of the decoding function */
typedef int (decodefunc_t)
     (char** text, bitmap_t* src, char** param);

/* header of the attack function */
typedef int (attackfunc_t)
   (bitmap_t* dest, bitmap_t* src, char** param);

/* header of the test function */
typedef double (testfunc_t)
   (bitmap_t* before, bitmap_t* after, char** param);

/* header of the picture loader */
typedef int (picldrfunc_t)(bitmap_t* dest, char* filename);
```

Le paramètre *param* est un tableau de chaîne de caractères (terminées par '\0')
Il est architecturé de la manière suivante :

```
char* param[] =
{
  "param1", "value1",
  "param2", "value2",
  "param3", "value3",
  ...
  "paramN", "valueN",
  NULL, NULL
} ;
```

Cette structure contient la liste des paramètres utilisateur : les noms suivis de leurs valeurs. Ces paramètres auront précédemment été affectés dans le fichier de scénario. L'ordre des paramètres dans le tableau est celui de saisie dans le fichier de scénario.

text est le texte éventuellement watermarké dans l'image. *src* et *dest* représentent les images sources et destinations. La structure *bitmap_t* définie une image. Les variables membre *size_x*, *size_y* correspondent respectivement à la longueur et la hauteur de l'image en pixels, *bitmap* est un tableau de pixel : c'est une suite des valeurs des composantes RVB de chaque point de l'image.

1.4 Création d'une librairie Matlab pour le benchmark

Une librairie Matlab est composée généralement de plusieurs fichiers avec l'extension '.m'.

Pour passer la librairie 'malib' au benchmark il faut que le point d'entrée de la librairie se trouve dans la fonction malib du fichier malib.m. D'autres fichiers matlab pourront être joints à ce fichier. Le tout devra être compressé dans l'archive malib.tgz. Sous cette forme, la librairie est maintenant prête à être envoyée au benchmark.

Voici la commande Unix permettant de stoker tout les fichiers du répertoire en cours dans l'archive malib.tgz :

```
tar –czf malib.tgz *
```

Si la librairie malib est un algorithme de watermark, alors l'archive malib.tgz devra également comporter un fichier malibdec.m contenant la fonction malibdec servant au décodage du watermark.

1.4.1 Types des arguments

Les librairies prendront en argument une ou plusieurs matrices contenant une image. Le format des images est le même que celui pris par la fonction '*imwrite*' : l'image est stockée dans une matrice à trois dimensions : la hauteur, la largeur, la couleur. Trois choix sont possible pour la dernière dimension : 1 => rouge, 2 => vert, 3 => bleu. Chaque valeur contenue dans la matrice est un double compris entre 0.0 et 255.0 représentant le niveau d'intensité dans la couleur actuelle.

NB : une fois le traitement réalisé, le benchmark convertie les composantes double de l'image en entier, il faut donc faire attention aux arrondis lors du développement.

Les paramètres saisis dans le fichier de scénario seront également passé à la fonction par un mécanisme de tableau de chaînes de caractères.

NB : en Matlab il faut être prudent sur la manipulation des tableaux de chaînes, car dans un tableau, chaque chaîne à un nombre de caractère égal. Ce qui implique que les chaînes n'ayant pas assez de caractères, sont complétées par des blancs. Ce mécanisme peut générer des disfonctionnement lors de la comparaison des chaînes de caractères. Pour palier à ce problème, il faut penser à éliminer les caractères non significatifs rajoutés par Matlab. La ligne suivante suffit à lire correctement le nom du premier paramètre d'un tableau de chaînes :

```
trimname = deblank(param(1,:))
```

NB pour les programmeurs C : le *strcmp* de Matlab retourne vraie lorsque les chaînes sont identiques, contrairement à celui de la libc.

1.4.2 Librairie de watermark

L'entête d'une librairie de watermark devra se conformer au prototype suivant :

```
function output = my_watermark(input, param)
output = [];
...
function output = my_watermarkdec(input, param)
output = char;
...
```

input correspond à la matrice de l'image à watermarker, *param* contient le tableau de paramètres optionnels sous la forme : nom_param1, val_param1, nom_param2, val_param2…

La fonction doit retourner une matrice au format spécifié au chapitre précédent contenant l'image watermarkée.

La fonction de décodage doit retourner la chaîne de caractère contenu dans l'image input.

1.4.3 Librairie d'attaque

L'entête d'une librairie d'attaque devra se conformer au prototype suivant :

```
function output = my_attack(input, param)
output = [] ;
...
```

input correspond à la matrice de l'image à attaquer, *param* contient le tableau de paramètres optionnels.

La fonction doit retourner une matrice contenant l'image attaquée.

1.4.4 Librairie de test de qualité

L'entête d'une librairie de test de qualité devra se conformer au prototype suivant :

```
function output = my_psnr(img1, img2, param)
...
```

img1 et *img2* correspondent aux matrices des images à comparer, *param* contient le tableau de paramètres optionnels.

La fonction doit retourner un double correspondant à l'indice de qualité.

1.5 Développement d'un serveur de librairie

Le but du développement d'un nouveau serveur de librairie est d'étendre la palette de format de librairie gérée par le Benchmark. A l'heure actuelle il existe trois serveurs de librairie pour les trois formats de librairie gérés :

-.so : serveur de librairie unix.
-.dll : serveur de librairie Windows.
-.tgz : serveur de script matlab compressé.

Rien ne vous empêche, en respectant le protocole réseau décrit ci dessous, de créer un nouveau serveur de librairie pour une autre architecture que celles proposées en standard, voir même de gérer un script pour algorithme de watermarking de votre invention. Ce système permettra éventuellement d'exécuter des librairies développées pour un autre benchmark existant.

1.5.1 Structures réseau

- Les nombres entiers qui circuleront sur le réseau seront au format Big Endian.
- Les chaînes de caractères seront précédées par leur taille (*int*) en caractère, et ne se termineront pas par '/0'.
- Les mesures de temps seront envoyées grâce à deux entiers : d'abord le nombre de secondes suivi du nombre de millisecondes
- Les bitmaps circuleront de la manière suivante : d'abord la longueur (*int*), la hauteur (*int*), suivies d'une suite d'octets (*unsigned char[L x H x 3]*) représentant respectivement les composantes (rouge, vert, bleu) de chaque pixel.
- Les listes de paramètres seront passées de la manière suivante : d'abord le nombre de paramètres, puis une suite de 2 chaînes de caractères correspondant respectivement au nom et à la valeur de chaque paramètre.

1.5.2 Protocole de communication

Le serveur de librairie devra être capable de répondre à toutes ces commandes :

- *CMD_ACK* : test si le serveur de librairie est bien en écoute.
- *CMD_WM_EXIST* : demande si le watermark spécifié est disponible.
- *CMD_AK_EXIST* : demande si l'attaque spécifiée est disponible.
- *CMD_QT_EXIST* : demande si le test de qualité est disponible.
- *CMD_WM_UPLOAD* : upload un algorithme de watermark sur le serveur.
- *CMD_AK_UPLOAD* : upload un algorithme d'attaque sur le serveur.
- *CMD_QT_UPLOAD* : upload un algorithme de test de qualité sur le serveur.
- *CMD_WM_DEL* : détruit un algorithme de watermark sur le serveur.
- *CMD_AK_ DEL* : détruit un algorithme d'attaque sur le serveur.
- *CMD_QT_ DEL* : détruit un algorithme de test de qualité sur le serveur.
- *CMD_WE_LAUNCH* : lance un algorithme de watermark.
- *CMD_WD_LAUNCH* : lance un algorithme de décryptage.
- *CMD_AK_LAUNCH* : lance un algorithme d'attaque.
- *CMD_QT_LAUNCH* : lance un algorithme de test de qualité.

1.5.3 Traitement des commandes

CMD_ACK :

Lors de la réception de cette commande, le serveur de librairie doit se contenter d'envoyer un *CMD_ACK* au client.

CMD_xx_EXIST :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère (*int*) et le nom de la librairie recherchée. Une fois l'extension correcte ajoutée au nom, on retourne *CMD_ACK* si la librairie existe et si elle est valide, sinon on retourne *CMD_ERR* ;

CMD_xx_UPLOAD :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère (*int*) et le nom de la librairie à uploader. Une fois l'extension et le chemin de la librairie ajoutés, on créé le fichier, puis on lit sur le réseau la taille en octet (*int*) et les données binaires à stocker dans le fichier. Puis on renvoie la commande *CMD_ACK* ou *CMD_ERR* si l'upload n'a pas pu avoir lieu.

CMD_xx_DEL :

Lors de la réception de ce message, le serveur doit lire le nom de la librairie sur le réseau et doit détruire le fichier correspondant.

CMD_WE_LAUNCH :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère et le nom de la librairie à lancer. Le serveur devra en suite lire la bitmap d'entrée, puis une liste exhaustive de paramètres qui serviront à spécifier les données à insérer dans l'image.

Le serveur pourra ainsi lancer la fonction *markfunc* de la librairie avec les bons arguments et récupérer la bitmap de sortie ainsi que son temps d'exécution. Il se contentera de transmettre au benchmark ces informations précédées de la commande *CMD_ACK* ou *CMD_ERR* si le traitement n'a pu avoir lieu.

CMD_WD_LAUNCH :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère et le nom de la librairie à lancer. Le serveur devra en suite lire la bitmap d'entrée, puis une liste exhaustive de paramètres.

Le serveur pourra ainsi lancer la fonction *decodefunc* de la librairie avec les bons arguments et récupérer le texte de sortie ainsi que son temps d'exécution. Il se contentera de transmettre au benchmark ces informations précédées de la commande *CMD_ACK* ou *CMD_ERR* si le traitement n'a pu avoir lieu.

CMD_AK_LAUNCH :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère et le non de la librairie à lancer. Le serveur devra en suite lire la bitmap d'entrée puis une liste exhaustive de paramètres.

Le serveur pourra ainsi lancer la fonction *attackfunc* de la librairie avec les bons arguments et récupérer l'image modifiée ainsi que son temps d'exécution. Il se contentera de transmettre au benchmark ces informations précédées de la commande *CMD_ACK* ou *CMD_ERR* si le traitement n'a pu avoir lieu.

CMD_QT_LAUNCH :

Lors de la réception de ce message, le serveur doit lire sur le réseau la taille en caractère et le nom de la librairie à lancer. Le serveur devra en suite lire les deux bitmas d'entrée puis une liste exhaustive de paramètres.

Le serveur pourra ainsi lancer la fonction *testfunc* de la librairie avec les bons arguments et récupérer le résultat du test ainsi que son temps. Il se contentera de transmettre au benchmark ces informations précédées de la commande *CMD_ACK* ou *CMD_ERR* si le traitement n'a pu avoir lieu.

1.6 Annexe

1.6.1 Exemple de code C pour une librairie de watermark

```
#include "../common.h"  /* for bitmap_t */
#include <stdlib.h>  /* for malloc */
#include <netinet/in.h> /* for htonl and ntohl */

/* read the value of the bit number Bitnum in Byte
return 0 if false, a not null value else.
Bitnum is zero based */
#define GETBIT(Byte, Bitnum) \
   ((Byte) & (1 << (Bitnum)))


/* Set the first bit value of Byte at Bitval */
#define SETBIT(Byte, Bitnum, Bitval)  \
     if (Bitval)                      \
       (Byte) |= (1 << (Bitnum));     \
     else                             \
       (Byte) &= ~(1 << (Bitnum))



/* store each bit of indata in the first bit of each byte of outdatat */
static void stegano(unsigned char* outdata, void* indata, int size)
{
  int i;
  int bitnum;
  int bitval;

  for(i = 0; i < size; i++)
    for(bitnum = 0; bitnum < 8; bitnum++)
    {
      bitval = GETBIT(((unsigned char*)indata)[i], bitnum);
      SETBIT(outdata[i * 8 + bitnum], 0, bitval);
    }
}


/* invert function of stegano */
static void readstegano(void* outdata, unsigned char* indata, int size)
{
  int i;
  int bitnum;
  int bitval;

  for(i = 0; i < size; i++)
    for(bitnum = 0; bitnum < 8; bitnum++)
    {
      bitval = GETBIT(indata[i*8 + bitnum], 0);
      SETBIT(((unsigned char*)outdata)[i], bitnum, bitval);
    }
}
```

```
/* watermark function  : MANDATORY ENTRY POINT FOR BENCHMARK */
int      markfunc(bitmap_t* dest, bitmap_t* src, char** param)
{
  int    i;
  char*  text;
  int    textlen;
  int   bigendianlen;
  int    bitmapsize;

  i = 0;             /* Find and read the text parameter value */
  while(param[i] && strcmp(param[i], "text"))
    i += 2;
  if(!(text = param[i + 1]))
    return 1;                                /* Text parameter not found */
  textlen = strlen(param[i + 1]);
  bitmapsize = src->size_x * src->size_y * RGB_SIZE;
  if(bitmapsize < ((textlen + sizeof(int)) * 8))
    return 2;                                 /* bitmap is too little */
  BMP_CREATE(dest, src->size_x, src->size_y);
  memcpy(dest->bitmap, src->bitmap, bitmapsize);
  /* convert the text size in an independant platforme BigEndian format
*/
  bigendianlen = htonl(textlen);
  stegano(dest->bitmap, &bigendianlen, sizeof(int));       /* mark the
len */
  stegano(dest->bitmap + sizeof(int) * 8, text, textlen); /* mark the
text */
  return 0; /* OK */
}


/* decode function : MANDATORY ENTRYPOINT FOR BENCHMARK */
int decodefunc(char** text, bitmap_t* src, char** param)
{
  int    bigendianlen;
  int    textlen;
  int   bitmapsize;

  bitmapsize = src->size_x * src->size_y * RGB_SIZE;
  if(bitmapsize < sizeof(int) * 8)
    return 1; /* bitmap to little */
  readstegano(&bigendianlen, src->bitmap, sizeof(int));
  textlen = ntohl(bigendianlen);
  if(bitmapsize < (textlen + sizeof(int)) * 8)
    return 1; /* bitmap to little */
  *text = malloc(textlen + 1);
  readstegano(*text, src->bitmap + sizeof(int) * 8, textlen);
  (*text)[textlen] = '\0';
  return 0;
}
```

1.6.2 Exemple de code C pour une librairie d'attaque

```
#include "../common.h" /* for bitmap_t*/
#include <stdlib.h> /* for malloc */

/* blur algorithm */
static void blur(bitmap_t* dest, bitmap_t* src, int blur_x, int blur_y)
{
  int x;       /* navigation variable in the picture */
  int y;
  int wx;            /* navigation variable in the blur quad */
  int wy;
  int colr;          /* new color value for destination */
  int colg;
  int colb;
  int quadxtl; /* quad position in picture(top left bottom right)*/
  int quadytl;
  int quadxbr;
  int quadybr;
  int quadsize;            /* number of pixel in the quad */

  for(y = 0; y < src->size_y; y++)
    for(x = 0; x < src->size_x; x++)
    {
      /* compute the quad around the actual point */
      quadxtl = x - blur_x;
      if(quadxtl < 0)
        quadxtl = 0;
      quadytl = y - blur_y;
      if(quadytl < 0)
        quadytl = 0;
      quadxbr = x + blur_x;
      if(quadxbr > (src->size_x - 1))
        quadxbr = src->size_x - 1;
      quadybr = y + blur_y;
      if(quadybr > (src->size_y - 1))
        quadybr = src->size_y -1;

      /* compute the average color of the quad */
      colr = 0;
      colg = 0;
      colb = 0;
      for(wy = quadytl; wy <= quadybr; wy ++)
         for(wx = quadxtl; wx <= quadxbr; wx ++)
         {
          colr += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE];
          colg += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE + 1];
          colb += src->bitmap[(wy * src->size_y + wx) * RGB_SIZE + 2];
        }
      quadsize = (quadxbr - quadxtl + 1) * (quadybr - quadytl + 1);
      colr /= quadsize;
      colg /= quadsize;
      colb /= quadsize;
      /* store the average color in the result */
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE] = colr;
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE + 1] = colg;
      dest->bitmap[(y * dest->size_y + x) * RGB_SIZE + 2] = colb;
    }
}
```

```
/* attack mandatory entry point */
int attackfunc(bitmap_t* dest, bitmap_t* src, char** param)
{
  int    i;
  char  *text;
  int   blur_x;
  int   blur_y;

  /* create de new bitmap */
  CREATE_BMP(dest, src->size_x, src->size_y);
  /* find the optionnal parameter value blur_x, blur_y */
  i = 0;
  while(param[i] && strcmp(param[i], "blur_x"))
    i += 2;
  if(!(text = param[i + 1]))
    blur_x = 1;
  else
    blur_x = atoi(text);
  i = 0;
  while(param[i] && strcmp(param[i], "blur_y"))
    i += 2;
  if(!(text = param[i + 1]))
    blur_y = 1;
  else
    blur_y = atoi(text);
  /* launch the attack */
  blur(dest, src, blur_x, blur_y);
  return 0;
}
```

1.6.3 Exemple de code C pour une librairie de test de qualité

```
#include "../common.h" /* for bitmap_t */
#include <math.h> /* for log */


/* return the mean square error value */
static double mse(bitmap_t* before, bitmap_t* after)
{
  double result;
  int i;
  int picturesize;
  int temp;

  result = 0;
  picturesize = after->size_x * after->size_y * RGB_SIZE;
  for(i = 0; i < picturesize; i ++)
  {
    temp = after->bitmap[i] - before->bitmap[i];
    temp *= temp;
    result += temp;
  }
  return result / picturesize;
}


/* quality test mandatory entry point for benchmark. Return the PSNR */
double testfunc(bitmap_t* before, bitmap_t* after, char** param)
{
  /* check if pictures have got the same size */
  if((before->size_x != after->size_x) ||
   (before->size_y != after->size_y))
    return QUA_ERR_VAL;

  return 10 * log(65536 / mse(before, after));
}
```

Pour plus d'information sur les exemples C, veuillez lire les commentaires des fichiers sources originaux fournis avec le Benchmark.

1.6.4 Exemple de librairie Matlab

Fichier *getparamvalue.m* :

```
function output = getparamvalue(param, paramname)
output = '';
[nbline, nbcar] = size(param);
for i = 1:nbline
  if mod(i, 2) & strcmp(deblank(param(i,:)),paramname )
    output = deblank(param(i + 1,:));
  end
end
```

Fichier *attack_filter.m* :

```
function output=attack_filter(input,parameter)
%input is an image matrix
%parameter sequence choose the filter with its personal parameters:
%  -> 1: 'gaussian' for a Gaussian lowpass filter (n,sigma)
%  -> 2: 'sobel' for a Sobel horizontal edge-emphasizing filter
%  -> 3: 'prewitt' for a Prewitt horizontal edge-emphasizing filter
%  -> 4: 'laplacian' for a filter approximating the two-dimensional
Laplacian operator (alpha)
%  -> 5: 'log' for a Laplacian of Gaussian filter (n,sigma)
%  -> 6: 'average' for an averaging filter (n)
%  -> 7: 'unsharp' for an unsharp contrast enhancement filter (alpha)

[height,width,depth]=size(input);
filtre = str2num(getparamvalue(parameter, 'filter'));

output=[];
h=[];

switch filtre
    case 1
       disp('Gaussian lowpass filter')
       n = str2num(getparamvalue(parameter, 'n'));
       sigma = str2num(getparamvalue(parameter, 'sigma'));
       if length(n)==0
          n=[3 3];
       end
       if length(sigma)==0
          sigma=0.5;
       end
       h = fspecial('gaussian',n,sigma);

    case 2
       disp('Sobel horizontal edge-emphasizing filter')
       h = fspecial('sobel');

    case 3
       disp('Prewitt horizontal edge-emphasizing filter')
       h = fspecial('prewitt');

    case 4
       disp('Laplacian operator 2D filter')
       alpha = str2num(getparamvalue(parameter, 'alpha'));
```

```
        if length(alpha)==0
           alpha=0.2;
        end
        h = fspecial('laplacian',alpha);

    case 5
        disp('Laplacian of Gaussian filter')
        n = str2num(getparamvalue(parameter, 'n'));
        sigma = str2num(getparamvalue(parameter, 'sigma'));
        if length(n)==0
           n=[3 3];
        end
        if length(sigma)==0
           sigma=0.5;
        end
        h = fspecial('log',n,sigma);

    case 6
        disp('Averaging filter')
        n = str2num(getparamvalue(parameter, 'n'));
        if length(n)==0
           n=[3 3];
        end
        h = fspecial('average',n);

    case 7
        disp('Unsharp contrast enhancement filter')
        alpha = str2num(getparamvalue(parameter, 'alpha'));
        if length(alpha)==0
          alpha=0.2;
        end
        h = fspecial('unsharp',alpha);

    otherwise
        disp('Unknown method.')
        output = input;
        break;
end

if depth==3
   output(:,:,1) = filter2(h,input(:,:,1));
   output(:,:,2) = filter2(h,input(:,:,2));
   output(:,:,3) = filter2(h,input(:,:,3));
else
   output = filter2(h,input);
end
```