



# Verslag Stirmark



Frederic Wegge

Tim Langens

Dit document bevat het volledige verslag van het project  
Stirmark Benchmark, ThemawEEK Multimedia 1

# Finaal Verslag project Stirmark

## Stirmark workshop

De workshop van de Stirmark Benchmark is zonder al te grote problemen volbracht, hierna hebben we gekozen om opdracht 5 te doen.

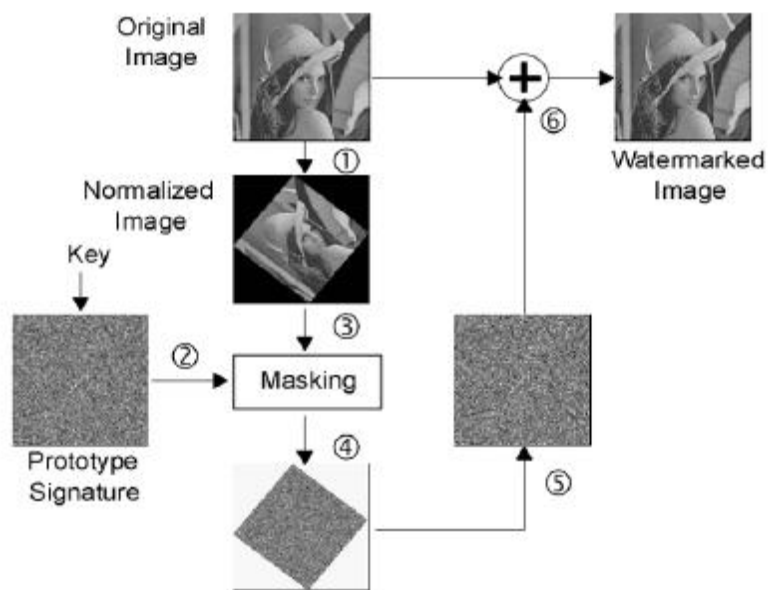
## Keuze opdracht 5:

### OPDRACHT 5: DS-CDMA

Een variant op standaard Spreadpectrum embedding is DS-CDMA. Hierbij kunnen meerdere bits in de image verstopt worden, zonder deze op te splitsen in subimages. Hoe dit werkt wordt uitgelegd in de paper "Watermarking digital image and video data-A state-of-the-art overview", aan het begin van pagina 27 (paper telt vanaf 20).

Literatuuronderzoek:

DS-CDMA (direct-sequence code division multiple acces) is een multiple acces schema gebaseerd op direct-sequence spread spectrum, door de signaal van/naar verschillende gebruikers met verschillende codes. Het is de meest gebruikte type van CDMA.



Dit schema illustreert het watermerk embedding proces.

## Werkwijze :

### Aan de embedder:

We zijn begonnen met het genereren van de RP's, dit in een 2Dimensionale array. De kolomen optellen en de resultaten hiervan optellen bij de pixel waarden van de originele image. In de code hier gebruiken we een vaste message van 7 lang (0011010) en een watermerk lengte van 11. Met de test hebben we deze verander naar een message van 30 lang en 200 voor het watermerk. Het watermerk wordt over heel de afbeelding herhaalt.

$$\begin{array}{llll}
 RP_0 : -1 & 1 & 1-1-1 & 1-1-1 & 1 & 1-1 & b_0 : 0 & \rightarrow & +RP_0 : -1 & 1 & 1-1-1 & 1-1-1 & 1 & 1-1 \\
 RP_1 : 1 & 1-1-1 & 1-1-1 & 1 & 1-1 & 1 & b_1 : 0 & \rightarrow & +RP_1 : 1 & 1-1-1 & 1-1-1 & 1 & 1-1 & 1 \\
 RP_2 : 1-1-1 & 1-1-1 & 1 & 1-1 & 1-1 & & b_2 : 1 & \rightarrow & -RP_2 : -1 & 1 & 1-1 & 1 & 1-1-1 & 1-1 & 1 \\
 RP_3 : -1-1 & 1-1-1 & 1 & 1-1 & 1-1-1 & & b_3 : 1 & \rightarrow & -RP_3 : 1 & 1-1 & 1 & 1-1-1 & 1-1 & 1 & 1 \\
 RP_4 : -1 & 1-1-1 & 1 & 1-1 & 1-1-1 & 1 & b_4 : 0 & \rightarrow & +RP_4 : -1 & 1-1-1 & 1 & 1-1 & 1-1-1 & 1 \\
 RP_5 : 1-1-1 & 1 & 1-1 & 1-1-1 & 1 & 1 & b_5 : 1 & \rightarrow & -RP_5 : -1 & 1 & 1-1-1 & 1-1 & 1 & 1-1-1 \\
 RP_6 : -1-1 & 1 & 1-1 & 1-1-1 & 1 & 1 & b_6 : 0 & \rightarrow & +RP_6 : -1-1 & 1 & 1-1 & 1-1-1 & 1 & 1 & 1 & + \\
 & & & & & & & & W & : -3 & 5 & 1-3 & 1 & 3-7 & 1 & 3-1 & 3
 \end{array}$$

▲ 10. Example of a CDMA watermark generation for 7 bits  $b_0b_1 \dots b_7$ .

### De decoder:

Eerst hebben we geprobeerd het originele watermerk er terug uit te halen. Dus we genereren dezelfde waarden voor de RP's door dezelfde nSeed te gebruiken. We nemen de gewatermerkte afbeelding en we gebruiken de formule die in het onderstaand voorbeeld gebruikt wordt om het originele bericht eruit te halen.

$$\begin{array}{ll}
 W & : -3 \quad 5 \quad 1-3 \quad 1 \quad 3 \quad -7 \quad 1 \quad 3 \quad -1 \quad 3 \\
 I & : \underline{98 \quad 98 \quad 97 \quad 98 \quad 97 \quad 96 \quad 97 \quad 96 \quad 95 \quad 94 \quad 94} + \\
 I_W & : 95 \quad 103 \quad 98 \quad 95 \quad 98 \quad 99 \quad 90 \quad 97 \quad 98 \quad 93 \quad 97
 \end{array}$$

$$\begin{array}{ll}
 E[(RP_0 - E[RP_0]) \cdot (I_W - E[I_W])] & = +15.6 \rightarrow b_0 = 0 \\
 E[(RP_1 - E[RP_1]) \cdot (I_W - E[I_W])] & = +16.4 \rightarrow b_1 = 0 \\
 E[(RP_2 - E[RP_2]) \cdot (I_W - E[I_W])] & = -26.4 \rightarrow b_2 = 1 \\
 E[(RP_3 - E[RP_3]) \cdot (I_W - E[I_W])] & = -3.1 \rightarrow b_3 = 1 \\
 E[(RP_4 - E[RP_4]) \cdot (I_W - E[I_W])] & = +21.6 \rightarrow b_4 = 0 \\
 E[(RP_5 - E[RP_5]) \cdot (I_W - E[I_W])] & = -23.6 \rightarrow b_5 = 1 \\
 E[(RP_6 - E[RP_6]) \cdot (I_W - E[I_W])] & = +0.4 \rightarrow b_6 = 0
 \end{array}$$

▲ 11. Example of CDMA watermark extraction, compare to Fig. 10.

Maar omdat we de certainty moesten bereken met de benchmark was dit eigenlijk niet nodig. Om de certainty te berekenen moesten we de code van Stirmark aanpassen omdat deze fout was, we gebruiken wel de code uit de paper.

$$R_{I'_w(x,y)W(x,y)} = \frac{1}{N} \sum_{i=1}^N I'_w(x,y)W_i(x,y)$$

## De Code:

### Embedder:

```

////////////////////////////////////
// Embed a watermark by adding a random pattern to the image
//
// See Base/StirMarkBenchmark.h for function behaviour

ErrorNum Embed_Image_Lib(const SMBImage      in_imgOriginal, // original
                        SMBImage *         out_pimgTarget, // image to
                        const SMBSchemePars * in_pPars)      //
parameters for embedding
{
    assert((out_pimgTarget != NULL) &&
           (out_pimgTarget->pgData != NULL) &&
           (in_imgOriginal.pgData != NULL) &&
           (in_pPars != NULL) &&
           (in_pPars->in_nKeyLength <= SCHEME_MAX_KEY_SIZE) &&
           (in_pPars->in_pszKey != NULL) &&
           (in_pPars->inout_nMsgLength <= SCHEME_MAX_PAYLOAD_SIZE) &&
           (in_pPars->in_dpStrength >= 0.0) &&
           (strlen(in_pPars->in_pszKey) == in_pPars->in_nKeyLength));

    uint nWidth   = in_imgOriginal.nCols;
    uint nHeight  = in_imgOriginal.nRows;
    uint nChannel  = in_imgOriginal.nChannels;

    uint nPixels  = nWidth * nHeight * nChannel;

    // The random number generator is seeded with the
    // watermarking key and used to generate a pseudo
    // random watermark reference pattern which is added
    // to the signal without taking into account the
    // different colour components
    uint nSeed = 1;
    srand(nSeed);
    double dAmp = in_pPars->in_dpStrength / 4;

    //message b
    bool bit[30] =
{0,1,0,0,1,1,0,0,0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1};

```

```

//Hier komt de nieuwe code voor DS-CDMAB
//Bereken per bit een random key van 200bits en invertteer indien nodig.
//double rp[30][(int)nPixels];

//2D array example
/*double** dNoise0 = new double*[nWidth];
   for(int x=0; x<nWidth; x++)
       dNoise0[x] = new double[nHeight];*/

double rp[30][200];

for(int i = 0; i < 30; i++)
{
    //for(int j = 0; j < nPixels; j++)
    for(int j = 0; j < 200; j++)
    {
        rp[i][j] = (double)(rand() > RAND_MAX/2 ? 1.0 : -1.0);
        if(bit[i] == 1)
            rp[i][j] *= -1.0;
    }
}
//Bereken het watermerk
//double W[(int)nPixels]; // = {1,1,1,1,1,1,1};
double W[200];
for(int i = 0; i < 200; i++)
{
    W[i] = 0;
}
//for(int i = 0; i < nPixels; i++)
for(int i = 0; i < 200; i++)
{
    //Bereken de nieuwe pixel
    //schrijf pixel weg
    for(int j = 0; j < 30; j++)
    {
        W[i] += (int)rp[j][i];
    }
}

for (int i = 0; i < (int)nPixels; i++)
{
    //Bereken de weg te schrijven boodschap

    //schrijf het watermerk weg.
    out_pimgTarget->pgData[i] = Clip(in_imgOriginal.pgData[i] + (W[i%200]
* dAmp));
}
cout <<endl<<"Eddy en Freddy edited this dll" << endl;

return SMBSUCCESS;

```

### Decoder:

```

////////////////////////////////////
// Detect a watermark using linear correlation
//
// See Base/StirMarkBenchmark.h for function behaviour

```

```

ErrorNum Extract_Image_Lib(const SMBImage    in_imgTest,    //
Marked/attacked image
                        const SMBImage    in_imgOriginal, //
Marked/unmarked image | 0
                        SMBSchemePars *inout_pPars)        // Pars for
extraction & output
{
    // in_imgOriginal is not used as this is a blind scheme
    assert((in_imgTest.pgData != NULL) &&
           (inout_pPars != NULL) &&
           (inout_pPars->in_nKeyLength <= SCHEME_MAX_KEY_SIZE) &&
           (inout_pPars->in_pszKey != NULL) &&
           (strlen(inout_pPars->in_pszKey) == inout_pPars->in_nKeyLength));

    uint nWidth    = in_imgTest.nCols;
    uint nHeight   = in_imgTest.nRows;
    uint nChannel   = in_imgTest.nChannels;

    uint nPixels   = nWidth * nHeight * nChannel;

    // The random number generator is seeded with the
    // watermarking key and used to generate a pseudo
    // random reference pattern. The test image is then
    // correlated with this pattern.
    uint nSeed = 1;
    srand(nSeed);

    //RP waarden berekenen
    double rp[30][200];
    bool bit[30] =
{0,1,0,0,1,1,0,0,0,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,1,1,1,1};

    for(int i = 0; i < 30; i++)
    {
        //for(int j = 0; j < nPixels; j++)
        for(int j = 0; j < 200; j++)
        {
            rp[i][j] = (double)(rand() > RAND_MAX/2 ? 1.0 : -1.0);
            if(bit[i] == 1)
                rp[i][j] *= -1.0;
        }
    }

    //Bereken het watermerk
    double W[200];
    for(int i = 0; i < 200; i++)
    {
        W[i] = 0;
    }
    //for(int i = 0; i < nPixels; i++)
    for(int i = 0; i < 200; i++)
    {
        //Bereken de nieuwe pixel
        //schrijf pixel weg
        for(int j = 0; j < 30; j++)
        {
            W[i] += (int)rp[j][i];
        }
    }
}

```

```

// dCor formule uit paper

double dCor = 0.0;

for (int i = 0; i < (int)nPixels; i++)
{
    dCor += in_imgTest.pgData[i] * W[i%200];
}

    dCor /= nPixels;
// Return the linear correlation value as a measure of certainty
// TODO: this is wrong
inout_pPars->out_dpCertainty = dCor;

return SMBSUCCESS;
}

```

### Problemen die we zijn tegengekomen:

1. Bij het random generen hebben we die 0,5 verandert door RAND\_MAX/2  
 Dus: `rp[i][j] = (double)rand() > RAND_MAX/2 ? 1.0 : -1.0;`  
 Hiervoor kregen we steeds allemaal 1'nen als onze RP's
2. De resulterende images waren volledig zwart buiten de eerst pixels waar het watermerk in zat. Dit was een foutje in de lus, de lus stopt na de lengt van het watermerk, dit hebben we dan verandert door de lus nPixels keer uit te voeren.
3. Eerst hebben we enkel geprobeerd om een vaste lengte voor het watermerk te gebruiken, maar om de afbeelding te watermerken laten we het watermerk steeds herhalen. Dit omdat je de lengte van een array op voorhand moet definiëren en niet van een wisselende variabele kunt laten afhangen.

```

for (int i = 0; i < nPixels; i++)
{
    //Bereken de weg te schrijven boodschap
    //schrijf het watermerk weg.
    out_pimgTarget->pgData[i] = Clip(in_imgOriginal.pgData[i] + W[i%11]);
}

```

4. Ook de originele dCor formule hebben we verandert door de formule uit de paper.

$$R_{I'_w(x,y)W(x,y)} = \frac{1}{N} \sum_{i=1}^N I'_{w_i}(x,y)W_i(x,y)$$

5. In onze resultaten komen ook negatieve waarden voor, dit komt door dat er meer negatieve waarden voorkomen dan positieve. Dit vonden we eerst raar, maar bij nader onderzoek was dit toch mogelijk.
6. Als je de message vergoot naar 30 en de lengte van het watermerk vergroot naar 200 krijg je iets logischere waarden voor de correlatie, stijgend positief en dalend negatief, maar dit klopte niet omdat de strength niet gebruikt werd. Dit hebben we opgelost door bij het declareren van een veranderlijk alles op nul te zetten.
7. De for-lus zorgt dat nSeed altijd hetzelfde zou zijn , maar nSeed verandert toch nog in het programma waardoor de correlatie toch nog verandert.

```

uint nSeed = 1;
//lus is fout, de nSeed verandert tijdens het programma
for(uint i = 0; i < in_pPars->in_nKeyLength; i++)
if (in_pPars->in_pszKey[i] == '1')
nSeed += 1 << i;
srand(nSeed);

```

Maar nu hebben we de for-lus verwijderd en nu blijkt de correlatie waarde gedurende de test hetzelfde te zijn, wat juist is want de test is 10 maal dezelfde.

## Resultaten:

### SMBReports:

De test is 10 maal dezelfde omdat we in onze code de strength (dAmp) niet meer gebruiken. Maar in volgend resultaat is er plots een verandering van de bekomen waarde, dit ligt volgens ons aan de lus waar nSeed wordt gedefinieerd.

### *Resultaat met lus:*

Test_PSNR	0	Images/Set1/Lenagray	Certainty:	81,5843	33,4696
Test_PSNR	10	Images/Set1/Lenagray	Certainty:	81,5843	33,4696
Test_PSNR	20	Images/Set1/Lenagray	Certainty:	81,5843	33,4696
Test_PSNR	30	Images/Set1/Lenagray	Certainty:	81,5843	33,4696
Test_PSNR	40	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	50	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	60	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	70	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	80	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	90	Images/Set1/Lenagray	Certainty:	97,28	32,8354
Test_PSNR	100	Images/Set1/Lenagray	Certainty:	97,28	32,8354

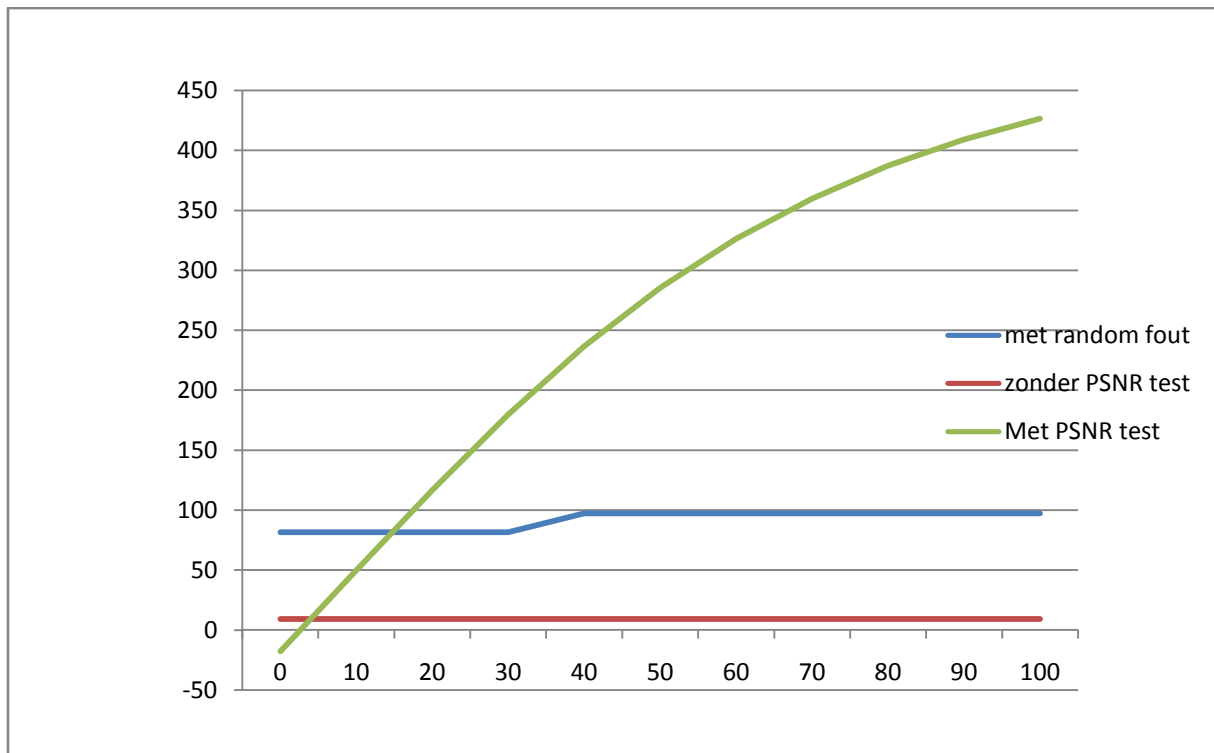
Resultaat zonder lus:

Test_PSNR	0	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	10	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	20	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	30	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	40	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	50	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	60	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	70	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	80	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	90	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB
Test_PSNR	100	Images/Set1/Lenagray	Certainty:	9,24964	33.389 dB

Resultaat met PSNR:

Test_PSNR	0	Images/Set1/Lenagray	Certainty:	-17,8095	1.#INF dB
Test_PSNR	10	Images/Set1/Lenagray	Certainty:	49,8283	25.8501 dB
Test_PSNR	20	Images/Set1/Lenagray	Certainty:	116,642	19.8735 dB
Test_PSNR	30	Images/Set1/Lenagray	Certainty:	179,854	16.4951 dB
Test_PSNR	40	Images/Set1/Lenagray	Certainty:	236,665	14.2418 dB
Test_PSNR	50	Images/Set1/Lenagray	Certainty:	285,446	12.6377 dB
Test_PSNR	60	Images/Set1/Lenagray	Certainty:	326,239	11.4459 dB
Test_PSNR	70	Images/Set1/Lenagray	Certainty:	359,794	10.5348 dB
Test_PSNR	80	Images/Set1/Lenagray	Certainty:	387,202	9.82203 dB
Test_PSNR	90	Images/Set1/Lenagray	Certainty:	409,153	9.2629 dB
Test_PSNR	100	Images/Set1/Lenagray	Certainty:	426,446	8.8238 dB

Grafisch:



De blauwe grafiek geeft de resultaten met de lus, die een random fout veroorzaakt.

De rode grafiek geeft een mooi constante waarde weer omdat hier de lus verwijderd is en door het niet gebruiken van de dAmp in de code is de test 10 maal dezelfde.

De groene grafiek geeft het resultaat met de PSNR test, de certainty stijgt naargelang de strength toeneemt. Dit is correct omdat de DS-CDMA methode vrij robuust is en het watermerk blijft zitten bij stijging van de dAmp.

## Conclusie:

De workshop ging goed. Na het kiezen van een opdracht hebben we eerst wat opgezocht en bleek de embedder code vrij simpel, maar omdat we dan dachten dat we met de formule het originele message terug moesten berekenen, hebben we veel tijd verloren met het zoeken naar fouten hiervoor. Toen we wisten dat we de certainty moesten meten met de benchmark en dat we de formule uit de paper moesten gebruiken, konden we de code hiervoor aanpassen. Hier kregen we dan onlogische resultaten in de log, we gebruikte de dAmp niet en toch steeg de waarde van de certainty. Dan hebben we in de code de nSeed veranderd en toen werd het resultaat logischer. Ook kregen we negatieve waarden voor de certainty. We dachten eerst dat dit niet kon, maar na nader toezicht zagen we dat dit goed mogelijk was doordat er meer random negatieve waarden konden worden gegenereerd.

Het was vrij moeilijk om de resultaten te begrijpen omdat we hier tamelijk nieuw in zijn en we niet wisten wat we konden verwachten. Ook leek het dat we veel tijd nodig hadden om de opdracht te

voltooiën, maar al bij al is deze toch goed gelukt en hebben we meer inzicht gekregen in deze methode en hoe men watermerken aanbrengt in afbeeldingen.

We kunnen dus besluiten dat het een leerrijke opdracht was, men zegt altijd dat men al doende het meeste leert en dit bleek hieruit ook.