IST-2002-507932

ECRYPT

European Network of Excellence in Cryptology

Network of Excellence

Information Society Technologies

D.WVL.10
Audio Benchmarking Tools and Steganalysis

Due date of deliverable: 31 January 2006
Actual submission date: 22 February 2006

Start date of project: 1 February 2004 Duration: 4 years

Lead contractor: Katholieke Universiteit Leuven (KUL)

Revision 1.1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Audio Benchmarking Tools and Steganalysis

Editor
Jana Dittmann (GAUSS)
Christian Kraetzer (GAUSS)

Contributors
Andreas Lang (GAUSS), Christian Kraetzer (GAUSS),
Natalia Trofimova (GAUSS), Christian Ullerich (GAUSS), Andreas Westfeld (GAUSS), Patrick Bas (CNRS), Jordi Herrera Joacomartí (UVIGO-UOC), David Megías Jiménez (UVIGO-UOC)

22 February 2006
Revision 1.1

The work described in this report has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932. The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Contents

1 Abstract – Executive Summary

The focus of WAVILA WVL3s research activities is on the development of benchmarking tools and schemes for digital watermarking and steganography as well as the evaluation of selected algorithms.
Digital watermarking and steganography are two of the most important aspects of information hiding in digital media. While the first is most commonly used for authentification, proof of ownership, proof of integrity and non-repudiation mechanisms it is part of many Digital Rights Management schemes and has therefore a huge commercial interest. Steganography, as the second information hiding technique considered by WVL3, provides hidden communication channels in seemingly "harmless" media like images, audio material or VoIP telephony sessions and is therefore of huge interest for security considerations and for the development of steganalysis techniques to detect such hidden communication channels in their cover mediums.
Benchmarking itself has not only the possibilities to identify possible weaknesses of tested algorithms. It can also provide a fair comparison of different algorithms under different evaluation aspects, making it possible to identify from a list of given solutions the algorithm most fitting for a concrete application scenario.
In this report we introduce the results of WVL3s activities in audio watermarking benchmarking where a lot of research results have been reached by evaluations with SMBA – a audio watermarking benchmarking tool developed by ECRYPT partner GAUSS and provided as a commonly available tool to the ECRYPT consortium. The results introduced here range from a comparison of different available watermarking algorithms for their possible application in different scenarios to recommendations for improvement regarding the performance of selected algorithms.
The second large part of WVL3s research activities described here is concerned with the results in steganography and steganalysis. In these fields WVL3 is on its way to identify the characteristics necessary for a fair benchmarking and to propose benchmarking procedures relevant consistent for various applications of steganographic methods. The results presented here show a first classification of available steganographic algorithms, followed by a more detailed evaluation of transparency for steganographic algorithms (based on the fact that it is one of the most important general characteristics of data hiding methods). The presentation of the research results in the field of steganography is ended by considerations about an specific steganalytical approach to illustrate the fact that the ECRYPT partner are not only doing basic research in the fields of steganography and steganalysis.

2 Introduction

In this report we summarise the WVL3 activities in the fields of a) benchmarking methods and tools for digital watermarks and b) steganography and steganalysis for the second ECRYPT year.
The evaluation focuses in general on the most important properties robustness, security, imperceptibility/ transparency, complexity, capacity and possibility of verification as well as invertibility of digital watermarking techniques. As a tool of choice for watermarking benchmarking SMBA (StirMark Benchmark for Audio) was chosen for this report, since it is not only developed by one of the ECRYPT partners

(GAUSS) but is also used for the benchmarking of audio watermarking algorithms by other ECRYPT partners, too, to provide a base for comparability.
The steganography and steganalysis part of the work done in WVL3 focuses on the identification of available steganographic tools and the proposal of a classification scheme for a huge number of hidden communication applications. Furthermore the possibilities for steganalytical detection for selected algorithms are discussed.

This report is structured as follows: Section 3 addresses investigated watermarking benchmarking methods that allow assessing watermarking algorithm properties.
Especially we summarise the functions of the developed benchmarking tool StirMark Benchmark for Audio as well as its approach for audio streams and indicate how SMBA was used for evaluation of different audio watermarking algorithms, exchanged and developed in WVL2. Since the results of these evaluations were already published in other documents they are only referred to here. Furthermore the approaches of attack tuning with attack transparency models and their impact to geometric attacks are reported, which were presented and discussed during the Ecrypt workshop WaCha 2005, Barcelona. Section 4 summarises the activities of WVL3 in the fields of steganography and steganalysis. Here first an overview and classification of a large number of available steganographic tools is given. Then a selected steganographic approach is evaluated using a steganalytical approach.

3 Audio Benchmarking Tool SMBA

This section describes the evaluation tool StirMark Benchmark for Audio (SMBA). This tool includes a collection of single attacks against the robustness of digital audio watermarks. These attacks work in time or frequency domain and modify the audio signal with the goal to weak or destroy an embedded watermark for benchmarking and evaluation purposes.

3.1 SMBA architecture

Note, that in this paper we use the following notation: SMBA stands for the overall benchmarking system and SMFA denotes the single attack module. The architecture of SMBA consists of four different types of modules. First, the attack module SMFA itself, second the read write stream module to convert audio files into streams and back into files, which is needed for input and output of audio signals. The third module SM-Bell is a wrapper for SMFA and read write to make it easier to use. The fourth module SM-Bell_GUI is a graphical user interface for SM-Bell. Figure 3-1 shows the modules and the module dependencies. The line between the read write process and SMFA and the other SMFA processes is a symbol for a pipe. The audio file is read by the read write module. The audio data are given to the first SMFA process which runs the first attack. The resulting audio signal of this process is the new input for the second SMFA process and so on. At the end of the pipes can be the read write module to save the audio signal in an audio file. If the user does not want to store the audio signal in a file, then the audio stream can be sent to the sound device to play it.

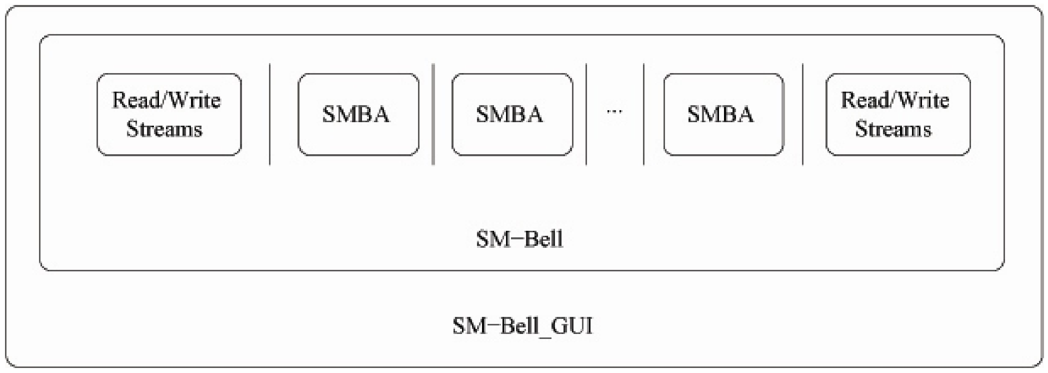

Figure 3-1: Modules of SMBA

The main benchmarking process is SMFA, the attack process, where a selected single attack is running (or concatenations of single attacks) to change the audio signal. In this paper, we set our main focus on this module (SMFA). Due to the design of our system, by running one single process, it is only possible to run one single attack on the audio signal. If it is feasible to run more than one attack on the audio signal, it is simple to use multiple instances of SMFA. This is possible by connecting the stdout of one SMFA process with the stdin of the following SMFA process by using a pipe. Another advantage of using SMFA for multiple attacks is, that each attack runs in its own SMFA process and the operating system can independently allocate each process to a processing unit. This is especially advantageous if there are multiple processing units available.

3.2 Current Single Attacks

Currently 40 attacks[1] are provided by StirMark Benchmark for Audio (SMBA) [SMBA] in version 1.2, with their default attack parameters [LANG2005].

From the overall point of view, a digital audio signal S depends on different parameters based on the capturing and sampling processes (with the following default values for SMBA):

- Sampling Frequency: $f_{SR} = 44.1$ kHz
- Sampling Quantization: 16 bits (MaxQantization = 2^{16})
- Number of channels: 2 (stereo)

Based on the digital audio representation, we differ between time and frequency domain. The frequency domain representation can be provided by transforming the time domain audio signal into the frequency domain for example by using a Fourier transformation [IFEACHOR2002]. As notation for the attacks of SMFA working in time domain, we use Si(x) as input signal for SMFA (marked audio signal) and So(x) as output signal from SMFA which is the attacked, modified, marked audio signal. The value x is the sample value at a discrete point of time in the input and output stream, we use $x = x(t_i)$. The value f denotes one frame that contains n sample values

[1] AddBrumm, AddDynNoise, AddFFTNoise, AddNoise, AddSinus, Amplify, BassBoost, Compressor, CopySample, CutSamples, DynamicPitchScale, DynamicTimeStretch, Echo, Exchange, ExtraStereo, FFT_HLPassQuick, FFT_Invert, FFT_RealReverse, FFT_Stat1, FlippSample, Invert, LSBZero, Noise_Max, Normalizer1, Normalizer2, Nothing, Pitchscale, RC_HighPass, RC_LowPass, Resampling, Smooth, Smooth2, Stat1, Stat2, TimeStretch, VoiceRemove, ZeroCross, ZeroLength1, ZeroLength2, ZeroRemove

x, f = $f_x(t_0)$; $x(t_1)$; ... ; $x(t_{n-1})$g, t_i; i = 0 ... n - 1, n in N, n=framelength. Avg(f) is the average of all sample values for all channels within the frame f. As notation for the attacks of SMFA working in frequency domain, we use $F_i(x)$ to identify the frequency input signal and $P_i(x)$ to identify the phase of the signal represented in the frequency domain. Furthermore, we use $F_o(x)$ and $P_o(x)$ as the corresponding output signal in frequency domain. The attacks in the frequency domain are parameterized with a window size (FFTSize) equal to frame size f for the Fast Fourier Transformation used for processing. As a default value we use a size of 1024 samples. To scale the attacks, we introduce an attack strength scalar value AS which specifies the strength of the attack performed on the audio signal.

The motivation for all attacks in SMFA is to destroy or weaken the embedded watermark signal, as Kutter et. all [Kutter2000] described for geometric attacks. From the signal processing point of view, we can classify the SMFA attacks into three attack classes. The first class adds or removes a signal k to or from $S_i(x)$: $S_o(x) = a * S_i(x) + b * k(t)$. The value a scales the input audio signal and the value b scales k(t) to a specific limit. The second class can be described as filter attacks: $S_o(x) = F_{Attack}(S_i(x))$, where F_{Attack} is the corresponding attack from this attack class. The third attack class can be seen as modification attacks, by modifying the overall structure of the signal representation for example the overall length of the audio signal or shifting audio samples: $S_o(x) = M_{Attack}(S_i(x))$. Table 3-1 summarises all current single attacks of SMFA into these three classes by indicating time and frequency domain.

Add/Remove Attacks	Domain	Filter Attacks	Domain	Modification Attacks	Domain
$S_o(x) = a * S_i(x) + b * k(t)$		$S_o(x) = F_{Attack}(S_i(x))$		$S_o(x) = M_{Attack}(S_i(x))$	
AddBrumm	Time	Amplify	Time	Invert	Time
AddSinus	Time	Normalizer1	Time	FFT_Invert	Frequency
AddNoise	Time	Normalizer2	Frequency	CopySample	Time
AddDynNoise	Time	Compressor	Time	FlippSample	Time
AddFFTNoise	Frequency	BassBoost	Time	CutSample	Time
NoiseMax	Time	RC-HighPass	Time	ZeroCross	Time
Denoise	Time	RC-LowPass	Time	ZeroLength1	Time
LSBZero	Time	FFT_HLPassQuick	Frequency	ZeroLength2	Time
Echo	Time	Stat1	Time	ZeroRemove	Time
		Stat2	Time	PitchScale	Frequency
		FFT_Stat1	Frequency	DynamicPitchScale	Frequency
		Smooth1	Time	TimeStretch	Frequency
		Smooth2	Time	DynamicTimeStretch	Frequency
				Exchange	Time
				Resampling	Time
				ExtraStereo	Time
				VoiceRemove	Time

Table 3-1: Classification of SMFA attacks

In the following sub sections, all 40 attacks are described and discussed how they work and what parameters they need. At the beginning, we introduce the attacks in Time Domain and then the attacks in Frequency Domain. At the end of each attack, we present default parameters.

3.2.1 Time domain attacks

As notation for the time-domain attacks we use $S_i(x)$ as input signal for SMBA (marked audio signal) and $S_o(x)$ as output signal from SMBA which is the attacked, modified audio signal. The value *x* is the sample number in the output stream. The value *f* is the frame that contains the sample *x*. *Avg(f)* is the average of all sample values for all channels within the frame that contains *x*.

- **AddBrumm:** Adds buzz as sinus tone to the audio signal. This attack is useful to simulate the buzz of a power supply unit of analogue devices. This attack has 2 parameters. *Frequency* indicates the buzz frequency and *Strength* for the attack strength (amplitude).

$$S_o(x) = S_i(x) + A_{AddBrumm}(x)$$

$$A_{AddBrumm}(x) = \sin(\omega * Frequency) * Strength$$

  The default parameter values are: *Frequency = 55 and Strength = 2500*

- **AddDynNoise:** Adds a dynamic noise that depends on the signal amplitude to the audio signal. This attack is useful for simulating certain noise effects found in analogue audio equipment. This attack has one parameter. *Strength* is the relative strength of the attack.

$$S_o(x) = S_i(x) + Pseudorandom(t) \bmod \left(\left| S_i(x) * \frac{Strength}{100} \right| + 1 \right)$$

  The default parameter value is: *Strength = 20*

- **AddNoise:** Adds a white noise to the audio signal. This attack is useful for analogue noise produced in many analogue devices. This attack has one parameter. *Strength* is the relative strength of the attack.

$$S_o(x) = S_i(x) * \left(1 - \frac{Strength}{MaxQuantitization()} \right) + Pseudorandom(t) \bmod (Strength)$$

  The default parameter value is: *Strength = 1000*

- **AddSinus:** Adds a sinus tone to the audio signal. This attack is useful to simulate various sinusoidal audio phenomena or to add a distortion signal in exactly the same frequency range where the watermark embedder would embed the watermark information. This attack is similar to Addbrumm and has two parameters. *Frequency* is the buzz frequency and *Strength* the attack strength (amplitude).

$$S_o(x) = S_i(x) + A_{AddSinus}(x)$$

$$A_{AddSinus}(x) = \sin(\omega * Frequency) * Strength$$

  The default parameter values are: *Frequency = 3000* and *Strength = 120*

- **Amplify:** Changes the amplitude of the audio signal to simulate amplification. This attack has one parameter. *Factor* is the percent increase or decrease percentage with 100 being neutral.

$$S_o(x) = S_i(x) * \frac{Factor}{100}$$

  The default parameter value is: *Factor = 50*

- **BassBoost:** Boosts the bass range of the audio signal, similar to analogue bass boosting circuits that are prevalent in consumer audio devices. This attack uses code from the Open-Source Media Editor Audacity [AUDACITY]. This attack has two parameters. *ThresholdFrequency* is the threshold under which all frequencies are boosted by *BoostDB* (in decibels).

  The default parameter values are: *ThresholdFrequency = 150* and *BoostDB = 6.123*

- **Compressor:** Scales all samples under a given threshold by a given amount. This attack is similar to various analogue compressor devices that are available on the market. This attack has 2 parameters. *ThresholdDB* is an amplitude threshold (in dB) under which all samples are compressed using the factor *CompressorValue*.

$$S_o(x) = \begin{cases} S_i(x) & S_o(x) \leq dbToAmplitude(-ThresholdDB) \\ S_i(x) * CompressorValue & S_o(x) > dbToAmplitude(-ThresholdDB) \end{cases}$$

$$dbToAmplitude(x) = 2 * \log\left(\frac{x}{MaxQuantitization()}\right)$$

The def. parameter values are: *ThresholdDB = 6.123* and *CompressorValue = 2.1*

- **CopySample:** Inserts copies of a group of samples at a later point in the stream. This is meant to test watermarking algorithms for dependence on quantity and order of samples. This attack uses a modified buffer-wise processing where input and output are asymmetrical and successive iterations do not necessarily use the same length buffer.
  This attack has three parameters. Within every *Period* samples, the first *Count* samples are copied *Distance* away from their original position. *Period* must be greater than *Distance* which in turn must be greater than *Count*.

$$S_o(x) = \begin{cases} S_i(z) & 0 \leq z < Distance \\ S_i(z - distance) & Distance \leq z < Distance + Count \\ S_i(z - count) & Distance + Count \leq z < Period + Count \end{cases}$$

$$z = x \bmod (Period + Count)$$

The default parameter values are: *Period*=10000, *Distance*=6000 and *Count*=2000

- **CutSample:** Drops a certain number of samples periodically. This is meant to test watermarking algorithms for dependence on quantity and order of samples. This algorithm also uses a modified buffer-wise algorithm, input and output are again asymmetrical, but in this case every other buffer is simply thrown away without being outputted.
  This attack has two parameters. Drops *RemoveNumber* samples at the beginning of every *Remove* input samples.

  $$S_o(x) = S_i(z + RemoveNumber)$$

  $$z = x \bmod (Remove - Removenumber)$$

  The default parameter values are: *Remove=1000* and *RemoveNumber=7*

- **Echo:** Adds a simple echo to the audio signal. This is meant to test watermarking algorithms that hide information in echos in the signal or test the watermark against echo attacks. This algorithm uses a "marching blocks" type algorithm to support streaming. Two buffers are always in memory, one front and one back. Front is the most recently read in samples. It is used to calculate the echo for back and then back is outputted, a front becomes the new back and a new front buffer of samples is read in.
  This attack has one parameter. The echo is produced with an echo delay of *Distance* samples.

  $$S_o(x) = \frac{S_i(x) + S_i(x + Distance)}{2}$$

  The default parameter value is: *Distance=2000*

- **Exchange:** Swaps consecutive samples in the audio sample. This is meant to test watermarking algorithms for their sensitivity to the exact order of the samples. This attack has no parameters.

  $$S_o(x) = \begin{cases} S_i(x+1) & z = 0 \\ S_i(x-1) & z = 1 \end{cases}$$

  $$z = x \bmod 2$$

- **ExtraStereo:** Adds the average between the channels in a frame to all samples in the frame. This is meant to test watermarking algorithms that use channel difference in multi-channel systems. This attack is based on the extra stereo function of XMMS [XMMS]. This attack has one parameter. *Factor* scales the strength of the effect.

$$S_o(x) = S_i(x) + (S_i(x) - Avg(f)) * Factor$$

The default parameter value is: *Factor=20*

- **FlippSample:** Flips a group of samples at a later point in the stream. This is meant to test watermarking algorithms for dependence on the order of samples. This attack uses a modified buffer-wise processing where sections of input and output are asymmetrical and successive iterations do not necessarily use the same length buffer. This attack has three parameters. Within every *Period* samples, the first *Count* samples are copied *Distance* away from their original position. *Period* must be greater than *Distance* which in turn must be greater than *Count*.

$$S_o(x) = \begin{cases} S_i(z + Distance) & 0 \leq z < Count \\ S_i(z) & Count \leq z < Distance \\ S_i(z - Distance) & Distance \leq z < Distance + Count \\ S_i(z) & Distance + Count \leq z < Period \end{cases}$$
$$z = x \bmod (Period + Count)$$

The default parameter values are: *Period=10000, Count=2000* and *Distance=6000*

- **Invert:** Inverts a sample: each sample value is replaced by its opposite (phase shift of 180°). This is meant to test watermarking algorithms for their sensitivity to the phase of the signal. This attack has no parameters.

$$S_o(x) = -S_i(x)$$

- **LSBZero:** Set the values of Least Significant Bits (LSBs) to zero. This is meant to challenge algorithms that may embed in the LSBs of the samples. This attack has no parameters.

$$S_o(x) = S_i(x) \wedge (MaxQuantitization() - 1)$$

- **NoiseMax:** Introduces a noise into the signal based on the Maximum Length Sequence (RIFE1989) [RIFE1989]. This attack has three parameters. *Mask* and *Length* are parameters to the RIFE1989 function, while *Strength* controls the strength of the noise that is introduced.

$$S_o(x) = S_i(x) + Strength * MLS(Mask, Length)$$

The default parameter values are: *Mask=300, Length=23* and *Strength=1365*

- **Normalizer1:** Normalizes the signal in a streaming manner, based on local peak information or based on cumulative peak information. This attack simulates various real-time normaliser devices. This attack has three parameters. *BufferSize* determines the length of each buffer for each processing cycle, *Level* is quantisation step that the samples are to be normalized to and *Zeroing* is a boolean switch to switch between local and cumulative peak finding (0 = local, else cumulative). The function *Peak(x)* finds the maximum peak value in the buffer.

$$S_o(x) = S_i(x) * \frac{Level}{Peak(S_i(x_j), Zeroing, PreviousPeak)}$$

$$j = 0,1,\ldots, BufferSize$$

The default parameter values are: *BufferSize=2048, Level=28000* and *Zeroing=0*

- **Nothing:** A simple pass-through, nothing is altered. This attack has no parameters.

$$S_o(x) = S_i(x)$$

- **RC-HighPass:** Simulates an analogue Resistor/Capacitor (RC) High-Pass Filter. This algorithm comes from [SMITH1997]. This attack has one parameter. *Threshold* determines the cut-off frequency (in Hz) for the filter.

$$S_o(x) = A_0 * S_i(x) + B * S_o(x-1) + A_1 * S_i(x-1)$$

$$B = e^{\frac{-2\pi * Threshold}{Samplerate}}$$

$$A_0 = \frac{1+B}{2},\ A_1 = -\frac{1+B}{2}$$

The default parameter value is: *Threshold=150*

- **RC-LowPass:** Simulates an analogue Resistor/Capacitor (RC) Low-Pass Filter similar to **RC-HighPass**. This attack has one parameter. *Threshold* determines the cut-off frequency (in Hz) for the filter

$$S_o(x) = A * S_i(x) + B * S_o(x-1)$$

$$B = e^{\frac{-2\pi * Threshold}{Samplerate}}$$

$$A = 1 - B$$

The default parameter value is: *Threshold=15000*

- **Resampling:** Adjusts the sample rate to a new rate. This attack uses the libsamplerate library [libsamplerate]. The library libsamplerate is written to support streaming, so the library handles streaming. This attack has one parameter *SampleRate* is the new rate which the signal has to be resampled to. The default parameter value is: *SampleRate=22050*

- **Smooth1:** Applies a smoothing algorithm to the signal. This attack has no parameters.

$$S_o(x) = \begin{cases} \frac{S_i(x-1)+S_i(x+1)}{2} & S_i(x-1), S_i(x+1) > S_i(x) \\ \frac{S_i(x-1)+S_i(x+1)}{2} & S_i(x-1), S_i(x+1) < S_i(x) \\ S_i(x) & \text{otherwise} \end{cases}$$

- **Smooth2:** Applies a smoothing algorithm to the signal. This attack has no parameters.

$$S_o(x) = \begin{cases} \frac{S_i(x-1)+S_i(x+1)}{2} & S_i(x-1) < S_i(x) < S_i(x+1) \\ \frac{S_i(x-1)+S_i(x+1)}{2} & S_i(x-1) > S_i(x) > S_i(x+1) \\ S_i(x) & \text{otherwise} \end{cases}$$

- **Stat1:** Applies a distortion algorithm to the signal. This attack has no parameters.

$$S_o(x) = \frac{S_i(x) + S_i(x+1)}{2}$$

- **Stat2:** Applies a distortion algorithm to the signal. This attack has no parameters.

$$S_o(x) = \frac{S_i(x-1) + 3 * S_i(x) + S_i(x+1)}{5}$$

- **VoiceRemove:** Subtracts the average between the channels in a frame from all samples in the frame. This is meant to test watermarking algorithms that use channel difference in multi-channel systems. This attack is based on the voice remove function of XMMS [XMMS]. This attack has no parameters.

$$S_o(x) = S_i(x) - Avg(f)$$

- **ZeroCross:** Sets all samples with an absolute value less than a given threshold to zero. This attack has one parameter. *ZeroCross* is the threshold below which all values are set to zero.

$$S_o(x) = \begin{cases} 0 & |S_i(x)| < ZeroCross \\ S_i(x) & \text{otherwise} \end{cases}$$

  The default parameter value is: *ZeroCross=1000*

- **ZeroLength1:** Sends a given number of zero samples as output after the detection of a zero value in the input, treating each channel independently. The streaming implementation of this attack uses queues to store incoming samples while it is outputting a set of zero-samples. This attack has one parameter. *ZeroLength* is the number of zero samples sent to output after the detection.

$$S_o(x) = \begin{cases} 0 = S_o(x+1) = \ldots = S_o(x + ZeroLength) & S_i(x) = 0 \\ S_i(x) & \text{otherwise} \end{cases}$$

  The default parameter value is: *ZeroLength=10*

- **ZeroLength2:** Sends a given number of zero samples to all channels after the detection of a zero value in the any channel of the input. The streaming implementation of this attack is buffer-wise but input and output are not always symmetrical. (it depends whether we are in a run of zeros or not) This attack has one parameter. *ZeroLength* is the number of zero samples sent to output after the detection.

$$S_o(x) = \begin{cases} 0 = S_o(x+1) = \ldots = S_o(x + ZeroLength) & S_i(x) = 0 \\ S_i(x) & \text{otherwise} \end{cases}$$

  The default parameter value is: *ZeroLength=10*

- **ZeroRemoves:** Removes all zero samples from the signal. This attack has no parameters.

$$S_o(x) = \begin{cases} S_i(x) & S_i(x) \neq 0 \\ S_i(x+1) & S_i(x) = 0 \end{cases}$$

3.2.2 Frequency Domain Attacks

In the following, we introduce all attack in frequency domain. As notation for the frequency-domain attacks we use $F_i(x)$ to signify the frequency input signal and $P_i(x)$ to specify the phase of the signal represented in the frequency domain using a Fourier transform [IFEACHOR2002] with $F_o(x)$ and $P_o(x)$ as the corresponding output signal in frequency domain. The value x is the bucket number in the Fourier representation. All attacks in frequency domain convert to the Fourier representation in frames. Some of them need as attack parameter the size of a frame (*FFTSize*) of the Fast Fourier Transformation used in processing. As a default value we assume a size of 1024 samples for the following attacks. Unless otherwise noted, a simple buffer-wise implementation was used to introduce streaming: each FFT buffer is read in, transformed, processed, transformed back and then outputted.

- **AddFFTNoise:** Adds a white noise to the signal in the frequency domain. This attack has one parameter. *Strength* is the relative strength of the attack.

$$F_o(x) = F_i(x) * \left(1 - \frac{Strength}{MaxValue()}\right) + Pseudorandom(t) \bmod (Strength)$$

  The default parameter value is: *Strength=3000*

- **DynamicPitchScale:** This attack performs a nonlinear pitch scaling on the audio signal and provides five different working modes. These modes and the default attack parameters are similar to the **DynamicTimeStretch** attack with the difference, that the pitch is scaled instead the time is stretched.

- **DynamicTimeStretch:** This attack performs a nonlinear time stretch on the audio signal. This can be categorized into five different working modes, which are described in the following. Furthermore, all modes have five parameters (scale factor (s), mode of attack (m), frame size - lower bound (FS_{LB}), frame size - upper bound (FS_{UB}) and buffer size (bs)).

  - [**Mode 0:**] This mode stretches the audio signal in fixed direction either left or right but scaling factor varies over time randomly between scale factor and 1. Direction of scaling is determined by scale factor. If it is greater than 1 scaling will be right direction (stretch), otherwise in left direction (shrink). The following figure 3-2 illustrates the attack.

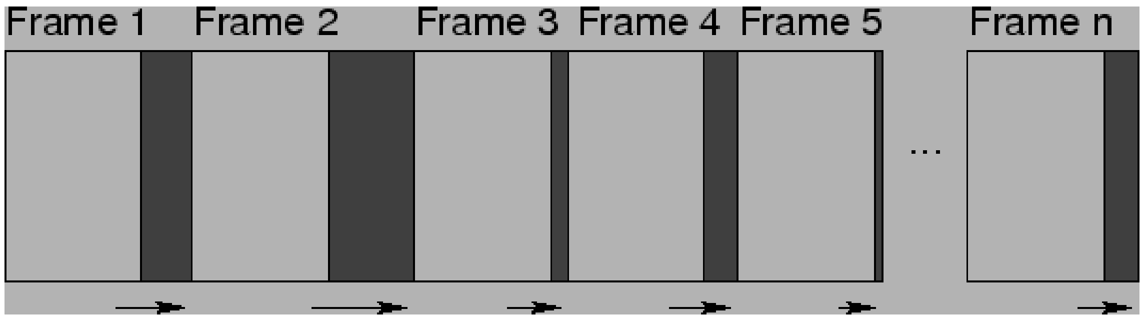

Figure 3-2: DynamicTimeStretch with fixed direction and randomly stretched

- [**Mode 1:**] This mode stretches the audio signal with a fixed scale factor but the direction of scaling changes randomly while processes sequence of all the frames. The following figure 3-3 shows it. The direction which frame is going to be stretched is selected randomly from a pool of random numbers.

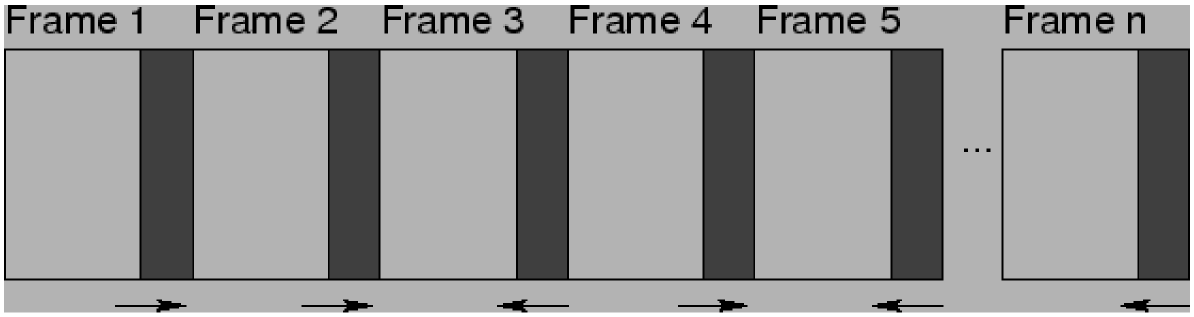

Figure 3-3: DynamicTimeStretch with random direction and fixed stretch

- [**Mode 2:**] This mode stretches the audio signal with a fixed scaling factor similar to mode 1 but the difference is that direction of scaling changes to right and left alternatively for each frame. The following figure 3-4 shows it.

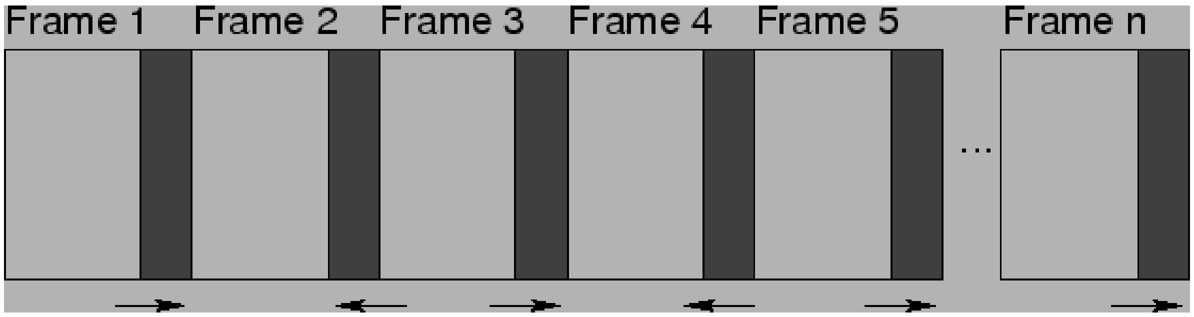

Figure 3-4: DynamicTimeStretch with an alternating direction and fixed stretch

- [**Mode 3:**] This mode stretches the audio signal in random direction with a randomly scaling factor varies over time. The figure 3-5 shows it.

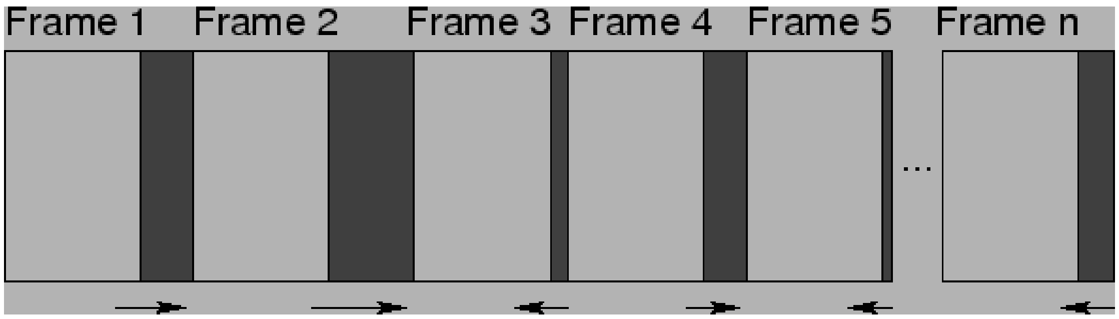

Figure 3-5: DynamicTimeStretch with random direction and random stretch

- o [**Mode 4:**] This mode stretches the audio signal in alternating directions with a randomly scaling factor varies over time. The figure 3-6 shows it.

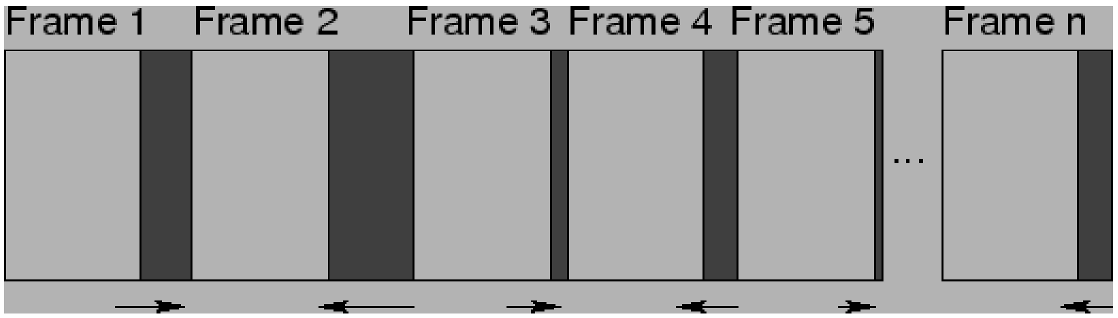

Figure 3-6: DynamicTimeStretch with alternating direction and random stretch

The default parameters are: *scalefactor=1.4, mode=3, framesize-lowerbound=32000, framesize-upperbound=64000* and *buffersize=32384*

- **FFT_HLPassQuick:** Filters selected frequencies out of the audio signal or let pass through selected frequencies. Similar to **RC-High-** and **RC-LowPass** but works in the frequency domain. This attack has two parameters. *HighPassFrequency* is the cut-off frequency for the high-pass filtering. *LowPassFrequency* is the cut-off for the low-pass filtering.

$$F_o(x) = \begin{cases} F_i(x) & F_i(x) < LowPassFrequency \\ F_i(x) & F_i(x) > HighPassFrequency \\ 0 & \text{otherwise} \end{cases}$$

The default attack parameters are: *HighPassFrequency=15000* and *LowPassFrequency=150*

- **FFT_Invert:** Inverts (phase shift 180°) the frequency and phase in this domain and needs no parameters. The effect of this attack depends on the FFT implementation that is used.

$$F_o(x) = -F_i(x)$$

$$P_o(x) = -P_o(x)$$

- **FFT_Stat1:** Computes a mathematical signal modification, similar to **Stat1** but in the frequency domain. This attack has no parameters.

$$F_o(x) = \frac{F_i(x-1) + F_i(x+1)}{2}$$

- **Normalizer2:** This attack is similar to **Normalizer1**, but works in the frequency domain. Additionally, the audio signal is split into two frequency bands, which are normalized independently of one another. This attack uses the FFTW library for Fourier Transformation [IFEACHOR2002]. This attack has three parameters. *Level* is quantisation step that the samples are to be normalized to and *Zeroing* is a boolean switch to switch between local and cumulative peak finding (0 = local, else cumulative). The function *Peak(x)* finds the maximum peak value in the buffer. *Threshold* is the boundary between the two separately-analysed frequency bands. The default attack parameters are: *Level=28000, Zeroing=0* and *Threshold=5000*

- **PitchScale:** This attack scales the frequency up or down without changing the tempo of the signal. This attack uses the library SoundTouch [soundtouch] to alter the signal. This attack uses the streaming facility built into the libsoundtouch library. All data is processed in an asynchronous FIFO. This attack has one parameter. *ScaleFactor* is the pitch scaling factor (in octaves). The default attack parameter is: *ScaleFactor=0.95* or *1.05.*

- **TimeStretch:** This attack stretches or shrinks the signal playing time without changing the pitch. This attack uses the library SoundTouch [soundtouch] to alter the signal. This attack uses the streaming facility built into the libsoundtouch library. All data is processed in an asynchronous FIFO. This attack has one parameter. *Factor* is the time scaling factor. The default attack parameter is: *Factor=0.95* or *1.05*

3.3 Application oriented attacks

The future of attack based benchmarking lies in application oriented attacks (profile attacks) as they are described in [LANG2004] and [LANG2005]. Profiles serve the purpose of simulation real world application scenarios or special parameter settings for the watermarking algorithm. The simplest way to attack a digital watermark is a brute force attack by using all possible attacks against the watermark. For each attack, SMBA has default attack parameters, which can be used to evaluate the digital audio watermark very quickly. It is also possible to change and optimise the attack parameters to improve the attack strength or attack transparency. Each attack is part of a single evaluation process to determine the watermarking algorithm weakness e.g. with which attack a watermark can be broken. These single attacks are atomic signal modification processes. This scenario is also called single attack process [DITTMANN2004]. In this mode, the watermarked audio file undergoes many separate instances of attacks and for each attack a separate output audio file is produced. Each of these audio files is only modified by a single attack (e.g. AddNoise, PitchScale, Amplify or CutSample). This is useful to find a single specific weakness of a watermark algorithm. By using this attack method, the problem is twofold: Firstly, for each attack we produce and evaluate an attacked audio file to test the watermarking detection/verification computing a huge amount of data. Secondly, weaknesses caused by combinations of audio effects or artefacts are not tested.

Therefore, another attack mode, called profile attack is introduced by [LANG2004] to run more than one attack in serial order. An evaluation profile is an ordered sequence of processes that may be applied to a signal. Each of the individual processes in the profile is defined by its own set of parameters. While a profile may seem to be merely an attack or process macro, profiles serve a very useful purpose in benchmarking. Profiles allow the evaluation system to model or simulate scenarios of interest to particular applications, like internet radio, audio production or audio archives. Based of our classification of profiles [LANG2005C], an evaluation profile may be defined in terms of other (existing) profiles, which allow a complex process or attack to be modelled as a sequence of previously-defined (or elementary) processes (for example the DA/AD conversion). The approach used by SMBA composes profile attacks from a concatenation of single attacks (basic profiles). Therefore is has to be assumed that an improvement of the single attacks (their number and functionality as well as their parameterisation) will lead to improved (closer to the desired application) profile attacks.

3.4 Attack tuning - Attack Transparency Models and their Impact to Geometric Attacks

By tuning geometric attacks using attack transparency models and non-linear methods found in psychoacoustic models like described in [LANG2003] and [DITTMANN2005] the modification itself becomes context aware, resulting in attacked sequences which are the output of a transparent modification. As a second benefit of the use of psychoacoustic models, attacks could be maximised by refraining from the use of single attack parameters and instead using functions like the masking threshold to parameterise the attack.

In [DITTMANN2005] it was shown that using psychoacoustic modelling in geometric attacks (used as single attacks in an audio watermark benchmarking environment like SMBA) is capable of improving the results of the attacks. The exhaustive search needed by the detector to compute the correlation between the attacked sequence and all cyclically shifted versions of the watermark signal becomes far more complicated. A much wider scope of attacks (resulting from the perceptually scaled cyclical shifts) has to be considered for the exhaustive search.

When observing single attacks from the SMBA suite it can be stated from the results presented in [KRAETZER2006A] that the trade-off between the two characteristics impact on robustness and transparency seems to be the same like in data hiding. When the psychoacousticaly modified SMBA attacks from [DITTMANN2005] where evaluated in a larger scope further proof for the benefit of using transparency enhancing measures in attack based evaluation was given.

3.5 First Benchmarking Tests

First benchmarks on watermarking algorithms were performed in [KRAETZER2006A], [LANG2006] and [Megías2006] using SMBA to evaluate the transparency, complexity, robustness and capacity of selected watermarking and steganography algorithms. In the following section 3.5.1 we describe the evaluated watermarking and steganographic algorithms in detail, in section 3.5.2 we describe the used parameterisation. The first test results can be found in the publications mentioned above.

3.5.1 Algorithms used for testing

The following subsections introduce the used algorithms in detail.

3.5.1.1 AMSL LSB Watermarking

This watermarking algorithm works in time domain and embeds the watermark in the least significant bits of the audio sample values [Klimant2003, Matev2005] by overwriting the original bits. The watermarking algorithm has the following three parameters:

- The parameter *k* is a secret key. If *k* is used, then it initialized a pseudo random noise generator (PRNG) which selects the LSB's which are used for embedding the digital watermark. It means that not all LBS's are used for embedding. If no *k* is used (the parameter is not set), then all sample values of the audio signal are used for embedding. It directly inferences the embedding capacity.
- The parameter *c* is for the usage of error correction codes (ECC) [Klimant2003] and turns error correction on or off. If an ECC is used, then the length of the embedding message is doubled and errors during the retrieval process can be detected and corrected up to a threshold.
- The parameter *m* is for embedding a message and specifies the secret message which is embedded into the audio signal.

3.5.1.2 AMSL Spread Spectrum Watermarking

This watermarking algorithm works in frequency domain and embeds the watermark in the frequency coefficients [Kim2003]. The audio signal is transformed into the frequency domain by using a Fourier Transformation [IFEACHOR2002]. The following itemization introduces the parameters for this watermarking algorithm.

- The secret message *m* is used for embedding into the audio signal.
- A secret key *k* initialized a PRNG and the random numbers *r* are computed.
- Other parameters *l* and *h* determine the bandwidth by selecting the lower (*l*) and upper (*h*) frequency bound for embedding. Both parameters represent the frequency range which is used for embedding.
- A scalar value α determines the embed strength of the watermark.

- An error correction code [Klimant2003] selected with the parameter *c*, corrects the embedded information up to a specific threshold if errors occur on the embedded message during the retrieval process.

The secret message *m* is a binary message $m = \{0,1\}$ or an equivalent bipolar variable $m=\{-1,1\}$, which is modulated by a pseudo-random sequence $r(n_i)$. This sequence is generated by *k*. The value α specifies the embed strength. The index *i* runs from 1 to N and N is the length of the audio signal. The following equation shows the watermarking process: $S_E(n_i) = S(n_i) + \alpha\ w(n_i)$. The scaling factor α controls the adjustment between robustness and inaudibility. The modulated watermark $w(n_i)$ is equals to $r(n_i)$ or $–r(n_i)$ depending on m = 0 or m = 1.

3.5.1.3 2W2A – AMSL Audio Water Wavelet

This watermarking algorithm works in wavelet domain and embeds the watermark on selected zero tree nodes. It does not use a secret key. An additional file is created, where the marking positions are stored to retrieve the watermark information in detection process (non blind).

This watermark scheme embeds the watermark information *m* in the wavelet based frequency domain and uses a digital watermarking technique called zerotree (ZT) [Shapiro1993]. It is a non blind method, because additional information (specific file, where the wavelet coefficient used for embedding are stored) for watermark detection are needed. A classification of the wavelet coefficients which are significant by using zerotrees is performed in [INOUE1999]. In [INOUE1999] are two methods of digital watermark described. The first method uses the insignificant coefficients and embeds the watermark information redundantly into these. Zerotrees are constructed for three pairs of sub-bands. For detecting the watermark the zerotree root is used after the wavelet decomposition. The second method uses the significant coefficients by thresholding and modifying these coefficients at the coarsest scale. Two thresholds T1 and T2, where T1 < T2, and one the sub-bands must be selected. The coefficients are calculated and must lie between T1 and T2. Then the watermark information is embedded by modifying the calculated coefficients. For detection the embedded position and the threshold value are needed after the wavelet decomposition. This is the reason, why 2A2W is a non blind watermarking algorithm. In the following, the parameters are introduced.

- The parameter *m* is the watermarking message, which is embedded.
- The Parameter *w* specifies the watermarking method and at this time, only ZT (zerotree) is possible.
- The parameter *c* specifies the coding method and at this time, only binary (BIN) is possible.

3.5.1.4 Publimark

This watermarking algorithm is an open source tool, developed from Gaëtan Le Guelvouit [Publimark] and it is a command line tool, which embeds a secret message into an audio file. Therefore it uses a pair of keys, a public and a private key. The public key k_{pub} can be shared so that everybody can send a secrete message. The private key k_{priv} must be kept secret that only the owner can detect and retrieve the hidden information. The embedding process consists of two phases. First the sender chooses a random key, denoted seed, which is encoded with the shared public key to embed the secrete message. Second the sender transmits the audio file to the recipient, using an efficient private key steganographic algorithm [GUILLON2002]. For this the Scalar Costa scheme [EGGERS2003] could be used. Therefore the main problem lies on how to transmit the key. In [GUELVOIT2005] are further details described. The algorithm has the following three parameters:

- The parameter *m* specifies the watermark message, which is embedded.
- The private (secret) key is k_{priv}.
- The public key is k_{pub}.

3.5.1.5 WAUC and WAUC-sec

These watermarking algorithms work in the frequency domain and rely on MPEG 1 Layer 3 compression to determine where and how the embedded mark must be introduced [Megías2003, Megías2005]. The mark is embedded by modifying the magnitude of the spectrum at several frequencies which are chosen according to the difference computed between the original and the compressed audio content. The main advantage of this scheme is that the perceptual masking of the compressor is implicitly used and, thus, the scheme can be directly tested with different masking models by replacing the compressor. Since repeat coding of the mark is used, a majority voting scheme is applied to improve robustness. The scheme also uses a dual Hamming error correcting code for the embedded mark, which makes it possible to apply it for fingerprinting, achieving robustness against collusion of two buyers.

Mark embedding

Let the signal *S* to be marked be a collection of PCM samples. The spectrum of *S*, denoted as S_F, is computed with a Fast Fourier Transform (FFT) algorithm. Then, the signal *S* is compressed using an MP3 algorithm with a rate of *R* kbps and decompressed again to PCM format. The result of this compression/decompression operation is a new signal *S*', and its spectrum S_F' is obtained. The set of marking frequencies F_{mark} is chosen as follows. Firstly, all $f_{\text{mark}} \in F_{\text{mark}}$ must belong to the relevant frequencies F_{rel} of the original signal S_F:

$$F_{\text{rel}} = \left\{ f \in [0, f_{\max}] : |S_F(f)| \geq \frac{p}{100} |S_F|_{\max} \right\},$$

where $f_{\max}$ denotes the maximum frequency of the spectrum, which depends on the sampling rate and the sampling theorem, $p \in [0, 100]$ is a percentage and $|S_F|_{\max}$ is the maximum magnitude of the spectrum S_F. Secondly, the frequencies to be marked are those for which the magnitude remains "unchanged" after lossy compression and decompression, where "unchanged" means a relative error below a given threshold ε:

$$F_{\text{mark}} = \{f_1, f_2, \cdots, f_n\} = \left\{ f \in F_{\text{rel}} : \left| \frac{S_F(f) - S_F{}'(f)}{S_F(f)} \right| < \varepsilon \right\}.$$

- The parameter R is the bit rate used by the MP3 compressor needed by the embedding algorithm. R can be chosen in the range [32, 320] kbps.
- The parameter p is a percentage in the range [0, 100].
- The parameter ε is a relative error in the range [0, 1].

Once the frequencies in F_{mark} have been chosen, the spectrum of the marked signal is computed as:

$$\hat{S}_F(f) = \begin{cases} S_F(f), & f \notin F_{\text{mark}}, \\ S_F(f) \cdot 10^{+d/20}, & f \in F_{\text{mark}}, \text{to embed '1'}, \\ S_F(f) \cdot 10^{-d/20}, & f \in F_{\text{mark}}, \text{to embed '0'}. \end{cases}$$

Since spectrum components in S_F are paired (pairs of complex-conjugate values), the same transformation (increase or decrease d dB) must be performed to $S_F(f_{\text{mark}})$ and to its conjugate.

In addition a dual Hamming error correcting code DH(31,5) is used prior to the magnitude modification step and a pseudo-random binary sequence (PRBS) is added to the sequence of embedded bits.

- The parameter d is the disturbance (measured in dB) introduced at the marking frequencies to embed a bit.
- The parameter k is a secret key used to generate the PRBS.

The modified **WAUC-sec** scheme is suggested in (Megías2005) introducing some randomness in the selection of the marking frequencies. This modification uses some additional parameters:

- The parameters p_1 and p_2 are two probabilities (in the range [0, 1]). p_1 is the probability of choosing the most perceptually significant frequencies and p_2 is the probability of choosing the other frequencies.
- The parameter k_{sec} is the initial seed of a pseudo-random number generator in the range [0, 1] used to choose the marking frequencies according to the probabilities p_1 and p_2.

Mark detection

The objective of the mark detection algorithm is to determine whether an audio test signal T is a (possibly attacked) version of the marked signal $\hat{S}$. It is assumed that T is in PCM format or can be converted to it. First of all, the spectrum T_F is obtained applying the FFT algorithm and, then, $\left|T_F\left(f_{\text{mark}}\right)\right|$, the magnitude at the marking frequencies, is computed for all $f_{\text{mark}} \in F_{\text{mark}}$ When the magnitudes $\left|T_F\left(f_{\text{mark}}\right)\right|$ are available, a scaling (Least Squares) step can be undertaken in order to minimize the distance between the sequences $\lambda\left|T_F\left(f_{\text{mark}}\right)\right|$ and $\left|\hat{S}_F\left(f_{\text{mark}}\right)\right|$. This LS step implicitly uses the embedded mark (since $S_F(f_{\text{mark}})$ is needed) but it can be omitted ($\lambda = 1$) or performed with the original signal $S_F(f_{\text{mark}})$ instead of the marked one $\left|S_F\left(f_{\text{mark}}\right)\right|$.

The ratios $\mathbf{r}_i = \lambda |T_F(f_i)| / |S_F(f_i)|$, are computed to decide whether a '0', a '1' or a '*' (not identified) might be embedded at the *i*-th position:

$$\mathbf{r}_i \in \left[10^{d/20}\left(\frac{100-q}{100}\right), 10^{d/20}\left(\frac{100+q}{100}\right)\right] \Rightarrow \hat{\mathbf{b}}_i := '1',$$
$$\frac{1}{\mathbf{r}_i} \in \left[10^{d/20}\left(\frac{100-q}{100}\right), 10^{d/20}\left(\frac{100+q}{100}\right)\right] \Rightarrow \hat{\mathbf{b}}_i := '0'.$$

If none of these two conditions are satisfied, then $\hat{\mathbf{b}}_i$:= '*'. Here, $q \in [0, 100]$ is a percentage and $\hat{\mathbf{b}}_i$ is the *i*-th component of the vector $\hat{\mathbf{b}}$ which contains a sequence of "detected bits".

- The parameter *q* is a percentage in the range [0, 100].

Once **b** has been obtained, its length *n* will be greater than the length of the extended mark. Hence, each bit of the mark appears at different positions in **b**. A *voting* scheme is applied to choose whether the *i*-th bit of the mark is '1', '0' or unidentified ('*'). The PRBS signal is then removed and the error correcting code is applied in order to recover the identified mark *W*'.

3.5.2 Parameterisation

In this subsection, the test environment, chosen parameters for the watermarking algorithms and the methods for evaluation are introduced.
For the watermarking algorithms LSB, Spread Spectrum, 2A2W and Publimark the embed message *m* is the phrase *m*="*UniversityofMagdeburg*" is used. If a key can be indicated, then *k=22* is used. If the watermarking algorithms provide error correction, then the algorithm is used two times – one with and another without error correction. The following table 3-2 shows the exact parameters for the watermarking algorithms. A X in the column *m* indicates the usage of *m*="*UniversityOfMagdeburg*". If a cell is empty, it indicates that the parameter is not available for the watermarking algorithm. The first row shows the parameters which are introduced in the watermarking algorithms section.

m	k	c	w	l	H	a
Least Significant Bit – LSB						
X	Ø	Yes	n.a.	n.a.	n.a.	n.a.
X	22	No	n.a.	n.a.	n.a.	n.a.
X	Ø	No	n.a.	n.a.	n.a.	n.a.
X	22	Yes	n.a.	n.a.	n.a.	n.a.
Publimark						
X	k_{priv}, k_{pub} 1024bit	n.a.	n.a.	n.a.	n.a.	n.a.

AMSL Audio Water Wavelet - 2A2W						
X	n.a.	n.a.	ZT	n.a.	n.a.	n.a.
Spread Spectrum						
X	22	Yes	n.a.	9000	11000	5000
X	22	No	n.a.	9000	11000	5000
X	22	Yes	n.a.	17000	19000	5000
X	22	No	n.a.	17000	19000	5000

Table 3-2: Used Parameters for the Evaluation Tests

The Spread Spectrum algorithm runs four times in two different frequency bands and with and without ECC. One frequency band is from 9-11 kHz and the other 17-19 kHz. The frequency band from 9-11 kHz was used in some test made in the past [LANG2005], and the same frequency range is used for this evaluation. The frequency range 17-19 kHz was selected, because it is close the audible frequency bound for humans and we expected a good transparency for this parameters. As embedding strength 5000 is used, because it is the default value. For all embedding parameters ECC was enabled and disabled.
All these watermarking algorithms use the same audio test set which contains 389 different audio files. The audio signal category contains four main classes and 24 subclasses. The main classes and their subclasses are music (blues, classical, country, hiphop, jazz, metal, pop, reggae, synthetic, techno), sounds (computergen, natural, noise, silence), speech (computergen, female, male, sports) and SQAM (instrumental,voice). The subclasses divide the main classes, for example speech into male, female, computer generated and sports.

Test results for the introduced algorithms using the described parameterisations can be found in [LANG2006].

The WAUC algorithm is extensively evaluated in (Megías2003, Megías2004a, Megías2004b and Megías2006) in terms of *imperceptibility* (or transparency), *capacity* and *robustness*. The Sound Quality Assessment Material (SQAM) has been used to evaluate both the WAUC and WAUC-sec schemes, besides some other files in the Watermark Evaluation Testbed (WET) for audio (Lang2005).

- Capacity: the WAUC watermarking scheme is shown to produce capacity results in the range [20, 300] bps, depending on the audio signal to be marked.

- Imperceptibility: with appropriate tuning settings, imperceptibility results can be about 30 dB (in terms of Signal to Noise Ratio) or Objective Difference Grade (ODG) results in [–1.5, 0] (perceptible but not annoying to imperceptible).

- Robustness: robustness has been assessed against the StirMark Benchmark for Audio (SMBA), the WET system and also MP3 compression attacks. The scheme is shown to be very robust against MP3 compression, since it is able to overcome these attacks with a bit rate even lower than 56 bps. Most of the attacks in the SMBA and the WET system are also overcome, but the WAUC scheme needs be enhanced against some attacks, such as “Echo”, “Stat1”, “Stat2” and some filters.

Tuning guidelines for the tuning parameters are suggested in (Megías2004b):

- The parameter R can be tuned for capacity and imperceptibility. The increase in capacity is low for rates $R > 128$ kbps, thus, $R = 128$ kbps is a good choice. If imperceptibility is the priority, lower values of R can be chosen.
- As d is concerned, a trade-off between imperceptibility and capacity should be obtained. A value of $d = 1$ yields SNR about 20 dB, though this value, of course, depends on the particular file. If better imperceptibility is required, $d = 0.5$ or lower might be used, but taking into account that this would affect robustness. In most cases, d in [0.1, 0.5] will be appropriate.
- The parameters p and ε have a similar effect on capacity, imperceptibility and robustness. In general, $1 \leq p \leq 5$ would provide good robustness and imperceptibility (larger values produce very low capacity). On the other hand, low values of ε are advisable, for example $\varepsilon = 0.01$. It must be taken into account that reducing ε and increasing p at the same time will have a double effect on imperceptibility and capacity, thus, caution must be taken when tuning these parameters
- Finally, the best choice for q is the largest possible value.

The results for the modified **WAUC-sec** scheme are presented in (Megías2005), where the *security* of these schemes is also considered. The experiments show that it is possible to tune the WAUC-sec such that the capacity of the original scheme can be preserved. In addition, the WAUC-sec scheme obtains better imperceptibility results (both in ODG and SNR measures) than the original counterpart for the same capacity. As robustness is concerned, both schemes produce similar results against the SMBA, but the original WAUC scheme is more robust against MP3 compression. Finally, concerning security, both false positive experiments and ad-hoc attacks are performed. On the one hand, it is shown that false positives are quite improbable if different secret keys are used for embedding and detection. On the other hand, the ad-hoc attacks described in the paper can be survived by the WAUC-sec scheme, whereas they successfully erase the mark when the original WAUC scheme is used. In short, the WAUC-sec scheme provides with a trade-off solution between security and robustness against MP3 compression.

Some rules are also suggested to tune the additional parameters p_1 and p_2:

- p_1 should be chosen in some interval centred at 0.5 and too small values should be avoided. For example, p_1 should be in the interval [0.3, 0.7]. This way, a good trade-off between robustness and security would be obtained. The default value for the parameter p_2 (as a function of p_1 and other variables) is also suggested in (Megías2005).

3.5.3 First benchmarking results

To examine the results for transparency benchmarking of digital watermarks presented first in [DITTMANN2005] on a larger scale, more detailed tests were performed and described in [KRAETZER2006A]. Here transparency tests in the context of steganographic methods and digital watermarking algorithms (for audio signals) as well as in the context of attack based watermarking evaluation were performed. On a large test-set (389 audio files divided into 24 categories) the embedding transparency of the algorithms (four steganographic and four watermarking algorithms (see section 3.5.1 for detailed descriptions of the watermarking algorithms used)) and the improvement of selected attacks using psychoacoustic modelling were measured and evaluated considering the context dependency. These benchmarks were performed using the average |ODG| (absolute value of the Objective Difference Grade computed using the Open Source software tool EAQUAL; see [KRAETZER2006A] for details) as a transparency measure. The watermarking algorithms (except the used LSB watermarking) return bad results (in terms of transparency) if compared to steganographic algorithms, but this was expected since watermarking algorithms are more focused on robustness then on transparency. The used LSB watermarking algorithm evaluated is with considerable distance the most transparent algorithm but was destroyed by a fair number of attacks. This is making it interesting for integrity watermarking applications. A wavelet algorithm was found to be the most robust algorithm/parameter combination tested but it lacked high transparency. This constellation would make it useful in forensic tracking applications if transparency would play an inferior role – otherwise the transparency would have to be improved before usage in such an application. Based on the benchmarking results for the Spread Spectrum algorithm considered recommendations for the improving of the implementation of the algorithm were given. Considering the performance of the algorithms on all context categories it can be concluded that all watermarking algorithms show context dependencies for the embedding transparency. The reasons for this context dependency and its possible benefits should be determined in future research.

Benchmarking tests considering a wider scope of characteristics were performed in [LANG2006]. There the transparency, complexity, robustness and capacity of selected watermarking and steganography algorithms for different algorithms were measured and compared. As a benchmarking suite for these evaluations SMBA (Stirmark Benchmark for Audio) was chosen. In [Megías2006] a selected algorithm was described and evaluated in detail.

4 Steganography and Steganalysis

In this section first an overview over a large number of Steganographic Tools available on the internet is given with the goal to propose a standardised classification scheme for the key features of steganographic tools. This scheme is applied to a number of about 100 tools that are available from the Internet. The classification criteria contain properties, such as the supported carrier media, the availability of the source code, the licence model, and the embedding function. It is written with the intention to spot on fast-developing areas, which, as a consequence, may require further attention in academic research.

This general overview of Steganographic Tools is followed by an advanced steganographic approach for a selected form of steganographic embedding. Since the LSB embedding scheme is a widely used method in steganography the considerations on estimation accuracy introduced in section 4.2 can be considered to be of large interest for the development of steganographic algorithms.

4.1 Steganographic Tools on the Internet

The scope of this overview is limited to pure steganographic tools, characterised as software, embedding secret messages into a carrier medium with the intention to be imperceptible – or even undetectable against passive attackers. This definition explicitly excludes watermarking software.

The nature of a quantitative analysis is very well suited for giving a consolidated overview, however the level of abstraction implies that particular tools cannot be reflected in detail.

We provide a standardised classification scheme for the key features of steganographic tools. This scheme is applied to 87 tools. The classification criteria contain properties such as the supported carrier media, the availability of the source code, the licence model, the embedding function (if indicated), the encryption scheme (if implemented), the message spread (if specified), and the supported operating system. Security assessments and references to suitable steganalytic methods are given for selected entities.

4.1.1 Introduction

Steganography is the art and science of writing messages in such way that the existence of the communication is hidden. It has been used in various forms for thousands of years. In the computer era data hiding techniques gain importance and serve security, primarily the authenticity and integrity of a message in the context of computer-supported communication.

A common application of modern data hiding is digital watermarking. Digital watermarks protect intellectual property and may be used to trace, identify, and locate digital media across networks. They are attributes of the carrier, as a watermark typically includes information about the carriers or the owner. In another steganographic appliance data is hidden for transmission over the internet. In addition, the concept and application of steganography is a vital argument against a prohibition of cryptography. Unfortunately it can also be used for communication among terrorists and criminals as well as hard core pornography.

In contrast to cryptography where it is allowed to detect and intercept messages without being able to violate certain security premises guaranteed by a cryptosystem the goal of steganography is to hide messages inside other "harmless" media in a way that prevents anybody even detecting it. A good steganographic system should fulfil the same requirements posed by "Kerckhoffs' law" in cryptography. This means that a cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

4.1.2 Tools

4.1.2.1 Development

In our investigation we refer to steganographic tools which appeared in the period 1993-2004 on the Internet. Chart 4-1 presents the number of utilities that were published within this time range. Thereby the figure above the bars displays the number of tools which were released in the specified year.

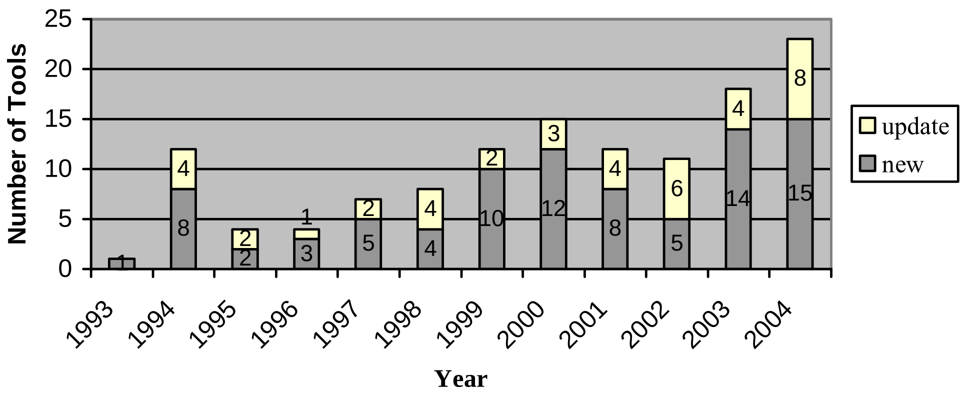

Chart 4-1: Version development overview

Another important fact is the timeline of the analysed software. It gives a global overview in respect to development processes and update frequencies. Chart 4-2 illustrates such a timeline. The x-axis defines the examined year and the y-axis displays the number of appeared tools. Dark bars within the chart mark newly released steganographic utilities, whereas bright bars represent the number of updated ones. The chart reveals an increasing appearance tendency for the last years of newly developed tools as well as update releases.

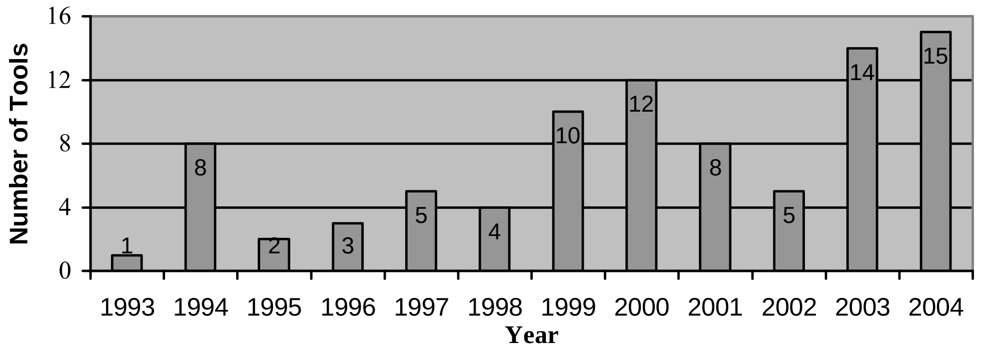

Chart 4-2: Tool development overview

4.1.2.2 Licence Model

The licence model of tools is another noteworthy attribute. Table 4-1-1 gives an overview of the models which attracted attention to us during our analysis. As it can be observed, an amount of 65 tools are freely available and distributable but there are also a significant number of utilities which were developed on a commercial basis.

Licence model	Number of tools
freely available	**65**
Freeware	43
GPL	20
BSD	1
Public Domain	1
commercial	**22**
Shareware	22

Table 4-1-1: Models of software licences

4.1.2.3 Price

The price of commercial steganographic software is a further noteworthy fact. The range of it is illustrated in Table 4-1-2. The most expensive commercially distributed software is Stealth Files 4.0 with 100 EUR but this does not ensure high quality. In fact Stealth Files simply appends data. Thus, a high price does not mean highly secure. Within the group of commercialised tools Encrypt Pic is available for the lowest price. It provides more security than Stealth Files as it embeds the data into image content. Nevertheless, the implementation of the embedding function leads to detectable steganograms.

Price class	Price
highest price	100 EUR
average price	43 EUR
lowest price	10 EUR

Table 4-1-2: Tool price range

4.1.2.4 Availability of Source Code

To judge the security of a tool the availability of its source code is helpful. One requirement of Kerckhoffs' law is that the security of an algorithm must not rely on its nondisclosure. As tools with available source code are easier to analyse it is much more likely that its security is evaluated by a larger community. The source code for more than half of the tools is freely available as shown in table 4-1-3.

Source code available	Percentage
yes	55
no	45

Table 4-1-3: Availability of source code

4.1.2.5 Supported OS Platform

Steganographic applications available on the internet run on a variety of platforms which are specified in table 4-1-4. As one single tool can support several operating systems each platform is assessed separately. Thus the number of tools in table 4-1-4 exceeds 87. Our observation in this classification is that most steganographic tools support Microsoft Windows.

Operating system	Number of tools	Percentage of tools
platform dependent	**85**	**89**
Windows	44	46
Linux / Unix	22	23
DOS	15	16
Macintosh	3	3
Psion Serie 5	1	1
platform independent	**11**	**11**
OS Independent	11	11

Table 4-1-4: Variety of OS platforms

4.1.3 Techniques

4.1.3.1 Media type

In the context of steganography it is necessary to take a closer look at the choosing of an appropriate carrier. While in ancient Greece slaves were used to play this role, in present times, various types of data files have the potential for this function. Therefore file formats can be assigned to the following domains: text, image, audio, video, and program files. In addition, within the analysed set of tools some software even uses its own file format, network protocol, or communication data (e.g. StegoGO uses game movements). Table 4-1-5 presents a selection of supported media and their distribution. Some tools can handle more than one format and hence are counted separately. On this account the utilities support two media formats on average.

Medium	Format	Number of tools	Percentage of this tool category	Percentage of all tools
Image		**107**		**56**
	BMP	34	32	18
	JPEG	22	20	11
	GIF	21	20	11
	PNG	9	8	5
	Other	21	20	11
Audio		**32**		**18**
	WAV	16	50	9
	MP3	9	28	5
	MIDI	2	6	1
	Other	5	16	3
Video		**7**		**4**
	AVI	4	57	2
	MPEG	2	29	1
	Other	1	14	1
Text		**22**		**13**
	TXT	7	31	4
	ASCII	5	23	3
	HTML	5	23	3
	Other	5	23	3
Program		**10**		**6**
	Windows Executables	7	70	4
	DLL	2	20	1
	Linux Binaries	1	10	1
Other		**5**		**3**

Table 4-1-5: Distribution of supported media

However it is worth mentioning that the majority of steganographic utilities which work with a large number of media types use the same hiding technique without adaptation for all supported file formats. As a consequence it can be stated that the support quantity does not ensure the support quality.

4.1.3.2 Encryption

The encryption scheme indicates the supported encryption features of the specified tool. Table 4-1-6 shows the disposability of encryption schemes among the analysed software. In fact, 71 % of the analysed steganographic tools feature the use of a cryptographic key for the encryption of the message before the embedding process. Encryption can increase the security level of the hiding procedure, e.g. by preventing the comprehension of the message content if the embedding is discovered and/or by creating a uniformly distributed secret message out of any original secret message. Furthermore, many encryption applications add a header in plaintext to the encrypted message. This header might yield information about the encrypted message (e.g. file type) which simplifies a cryptographic attack by enabling a more precise attack.

Cryptography available	Number of tools	Percentage of tools
implemented	62	71
not implemented	19	22
not indicated	6	7

Table 4-1-6: Availability of encryption option

If the presence of the message is discovered and successfully extracted, a header for the encryption method can be used for cryptoanalysis.
Therefore we can divide the encryption using software into two groups:

- tools that utilise cryptography based upon widely used and proven open source systems (i.e. standardised algorithms) and
- tools which don't (e.g. EncryptPic which uses a self-built cryptographic algorithm that is neither published nor verifiable).

Table 4-1-7 shows the number of tools of each category mentioned above. In addition a further subdivision within the standardised algorithms states in more detail which algorithms are used. The particular tools are assessed separately because one single tool can dispose of more than one encryption scheme. On this account the total number of tools is 108 instead of only 87. The encryption schemes which are used by just one single tool are recorded in the group "other". Our analysis shows that one of the more commonly used algorithms is Blowfish. In fact a number of 15 investigated tools use this encryption scheme. The DES algorithm is the second most frequently used cryptographic function. Finally, self-constructed algorithms without published source code infringe Kerckhoffs' law. They are provided by small developer groups only who generally cannot ensure their security whereas open source algorithms (e.g. Blowfish, DES, etc.) are (usually) more secure than self-constructed ones because open source is reviewed by much more people. In addition, the standardised encryption schemes such as AES experienced no severe attack.

Cryptographic category	Encryption scheme	Number of tools	Percentage of tools
standardised		**86**	**80**
	Blowfish	15	14
	DES	10	9
	RC4	5	4
	Twofish	5	4
	ICE	4	4
	IDEA	4	4
	AES (Rijndael)	3	3
	GOST	3	3
	PGP	3	3
	RSA	3	3
	CAST	2	2
	RC5	2	2
	RC6	2	2
	TEA	2	2
	Other	23	21
non standardised		**22**	**20**

Table 4-1-7: Variety of encryption schemes

4.1.3.3 Embedding

Steganography encompasses methods of transmitting secret messages in such a manner that the existence of the embedded message is undetectable. The analysed software tools provide a variety of information-hiding techniques. Among these, most methods are employed depending upon characteristics specific to a carrier type or format while other methods may work without relying on a specific file format. Based on the result of our investigation we could classify the hiding techniques as it is illustrated in table 4-1-8.

Domain	Method	Number of tools	Percentage of this domain methods	Percentage of all methods
Text		**16**		**15**
	manipulation by white spaces	10	63	9
	transformation into other text	4	25	4
	syntactic method	1	6	1
	Rephrasing	1	6	1
Image, Audio, Video		**81**		**76**
	LSB	56	69	53
	appending data	13	16	12
	header insertion	3	4	3
	masking	1	1	1
	other	8	10	7
Program files		**7**		**7**
	appending data	5	72	5
	substitution of equivalent instructions	1	14	1
	other	1	14	1
Protocols		**2**		**2**
	header insertion	2	100	2

Table 4-1-8: Summary of hiding techniques

4.1.3.3.1 Embedding Data into Text

The common method of hiding information in text is a manipulation of white spaces. The manipulation can be done by appending blanks or tabs at the end of lines or at the end of the text. (With HTML it is also possible to add blanks between words because consecutive white spaces are displayed as a single one.) Since blanks and tabs are invisible to most text viewers the message is effectively hidden from casual observers. However, there exist tools like FFencode which are optimised for special character sets (e.g. IBM-PC) and replace blanks with others blanks e.g. 20_{hex} with FF_{hex}. These substituted characters are displayed in text viewers that use the ISO character set and therefore it is questionable if these tools truly implement steganography. Another example of a text-based embedding is transforming data into English sentences, nonsense English phrases, or into pseudo-random natural language text. They are based on a dictionary either given explicitly or built “on the fly” from a source document. Take the following sentence:

Steganography is the art and science of writing hidden messages in such a way which hides the existence of the communication. (125 characters)

and apply some transformation to create nonsense pseudo-random natural language text:

Miasma seamers frounce bedtimes beanbag, spell kyanize ... skirr. (CryptoMX) (327 characters)

Characteristic for this technology is that it does not use a carrier for hiding information but "scrambles" a message in a way that it cannot be understood although it has some word syntax and therefore is less likely detected as an encrypted message by a machine.
The following technique is called rephrasing of text [Comp03]. It allows changing the tense of the narration or the point of view as well as substituting words whereby the meaning of the text is preserved. An example is the ensuing text that is used as carrier for the secret message "test":

Eine Lampe leuchtet in der Nacht. Das Bild an der Wand ist mit einer schmalen Rente bezahlt worden, doch hat sich der Kauf gelohnt ... (A lamp shines at night. The painting on the wall has been paid with a frugal superannuation, yet the bargain was worthwhile …translated by author)

The following steganogram is created by applying the technique of replacing single words by one of its synonyms representing the possible hiding data according to a thesaurus.

Eine Lampe leuchtet in der Nacht. Das Abbild an der Wand ist mit einer schmalen Pension bezahlt worden, doch hat sich der Kauf gelohnt ... (A lamp shines at night. The portrait on the wall has been paid with a frugal pension, yet the bargain was worthwhile ...translated by author, TextHide*)*

Another possibility of hiding information in text is known as syntactic method. This one is based on punctuation, construction, or spelling changes.

4.1.3.3.2 Embedding Data into Images, Audio, and Video

Numerous methods exist for hiding information in audio, images, and video. Some common embedding techniques range from least significant bit (LSB) manipulation over masking and filtering to applying more sophisticated image or audio processing algorithms and transformations. Each of these approaches can be developed with varying degrees of success for different file formats.
LSB methods insert the embedding data in the carrier byte stream, substituting insignificant information in a carrier file with secret data. Some tools utilise two least significant bits or even more to hide a message.
In general there are two types of LSB embedding which apply to images:

- simple LSB embedding in raw images
  - change LSB in one up to all three colour channels of the pixel or in the frequency coefficients of a discrete cosine transformation (DCT)
  - increment/decrement the pixel value instead of flipping the LSB
  - matrix encoding
- LSB embedding in palette images
  - change colour index to similar palette entry (e.g. EzStego)
  - change palette entry

The LSB manipulation concept can also be applied to audio. The least significant bit of information at each audio sampling point is replaced with a bit from the hidden message. This method introduces significant noise into the audio file.
LSB manipulation is a quick and easy way to hide information but is vulnerable to small changes resulting from file processing or lossy compression.
Masking methods such as hiding secret messages into higher-order bits with simultaneous decrease of luminance or volume are more robust than LSB insertion in respect of compressing, cropping, and some image or audio processing. These techniques allow embedding in more significant areas in order to integrate a hidden message further into the cover file.
Another technique for hiding data into image or multimedia files is called appending which means that the secret data is added after the very last byte of the carrier file. The carrier file size could increase up to the sum of the size of the original carrier file and the secret file yet the size will change with a very high probability. This method is very simple and very easy to detect because the secret message will be added in plain form. Furthermore, the probability of detecting the secret message increases if the steganographic tool uses such embedding techniques as inserting in junk or comment fields in the header of the file structure. On the one hand the hidden data congregates at the same place and on the other hand the file header is rather vulnerable for steganalysis.

4.1.3.3.3 Embedding Data into Program Files

The common technique for hiding data in program files is appending the data at the end of the carrier file as practised with image, audio, and video files. Another possibility is stashing a secret message by transforming program instructions. This technique substitutes an instruction by an equivalent which represents the bit(s) of the secret data. A simple example: “add %eax, 50” can be substituted by “sub %eax, -50”.

4.1.3.3.4 Embedding Data into Archive Files

There is only one example among the investigated software which uses archive files (gzip-files) as carrier medium. It embeds the secret data during the compression process through overwriting the least significant bits.

4.1.3.3.5 Embedding Data into Network Protocols

The embedding process in network protocols takes place via manipulation of unused spaces and other features of the packet header.

4.1.3.4 Message Spread

The embedded message is spread over the carrier by selecting more or less equally distributed positions in order to carry the bits of the secret message. This characteristic is relevant for secret message bits in cover files like images, audio, and video. In our classification we distinguish three types of message spreads: linear continuous, random continuous, and straddling.

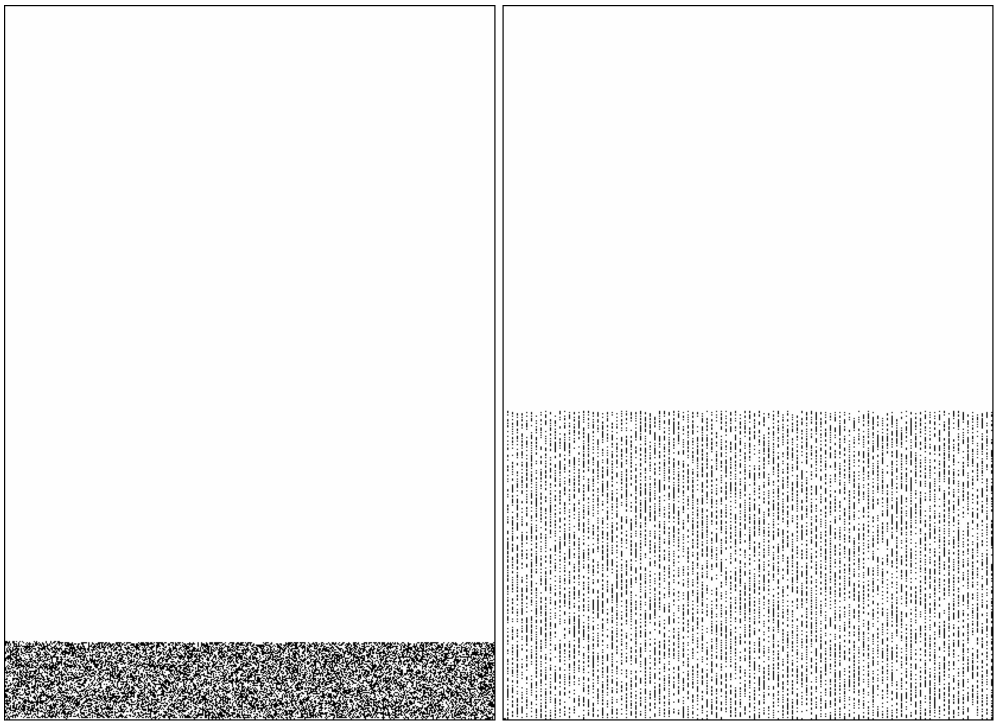

Figure 4-1: Steghide (LSB embedding); linear continuous message walk – Both pictures are true colour pictures that are spread and transformed into b/w so that each colour channel is displayed separately for better visibility of the changes caused by the embedding. Black pixels represent a change of the carrier medium. They contain a secret message of 1.35 % of the picture size.

Linear continuous message walk means that the bytes of the carrier file are sequentially selected with fixed space between the selected carrier bytes, where the first secret-hiding byte is chosen at the beginning of the carrier file. Figure 4-1 shows the difference between a linear continuous embedding with no space between the used carrier bytes2 and an embedding with the fixed space of three carrier bytes (i.e. every fourth carrier byte contains a stego bit). Additionally it is possible to fill up the unused capacity of the carrier medium with white noise.

[2] Every carrier byte is used.

Random continuous message walk uses a (pseudo) random generator to distribute the secret message across the carrier medium. Figure 4-2 shows the difference between a continuous message walk with a fixed interval of 3 bits per "stego bit" on the left side and the random continuous message walk on the right side where the space between the used carrier bytes is (pseudo) random. The initial value of the (pseudo) random generator can be calculated by hashing a pass phrase or user interactions or it can be manually given. The secret message will be hidden sequentially but the space between two data hiding carrier bytes ("stego bytes") will be determined by the (pseudo) random value. Again, a steganographic tool could fill

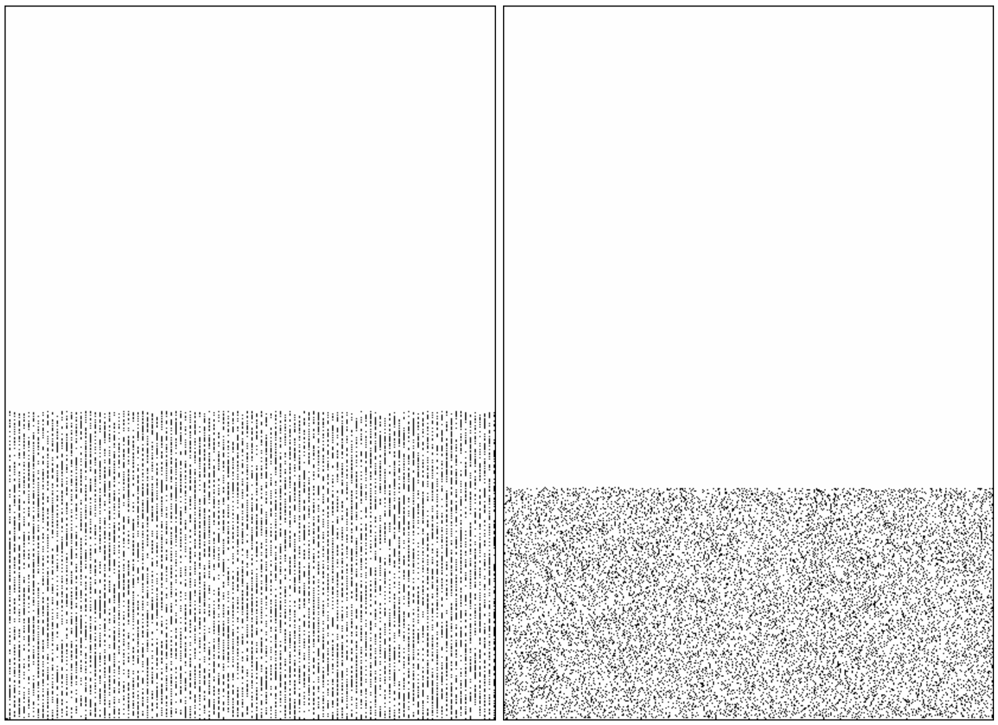
Figure 4-2: Steghide (LSB embedding): continuous message walk vs. random continuous message walk – Both pictures are true colour pictures that are spread and transformed into b/w so that each colour channel is displayed separately for better visibility of the changes caused by the embedding. Black pixels represent a change of the carrier medium. Both pictures contain a secret message of 1.35 % of the picture size – embedded with Steghide v0.3.

up the unused space in the carrier medium with white noise.

Straddling message walk distributes the secret message over the whole carrier medium irrespective of the secret message size (Figure 4-3). The advantage of this distribution method – as well as the random continuous one – is the fact that it decreases the probability to detect the content of secret data in the carrier medium because of the smaller change density compared to the sequential method. An example for this type of message spread is the so-called "permutative straddling". It basically works in three steps. At first a permutation of the bytes of the carrier file is performed. In the second step the secret message is embedded sequentially. Finally after repermutation of the carrier bytes, back to the original order, the secret message is evenly distributed over the carrier file.

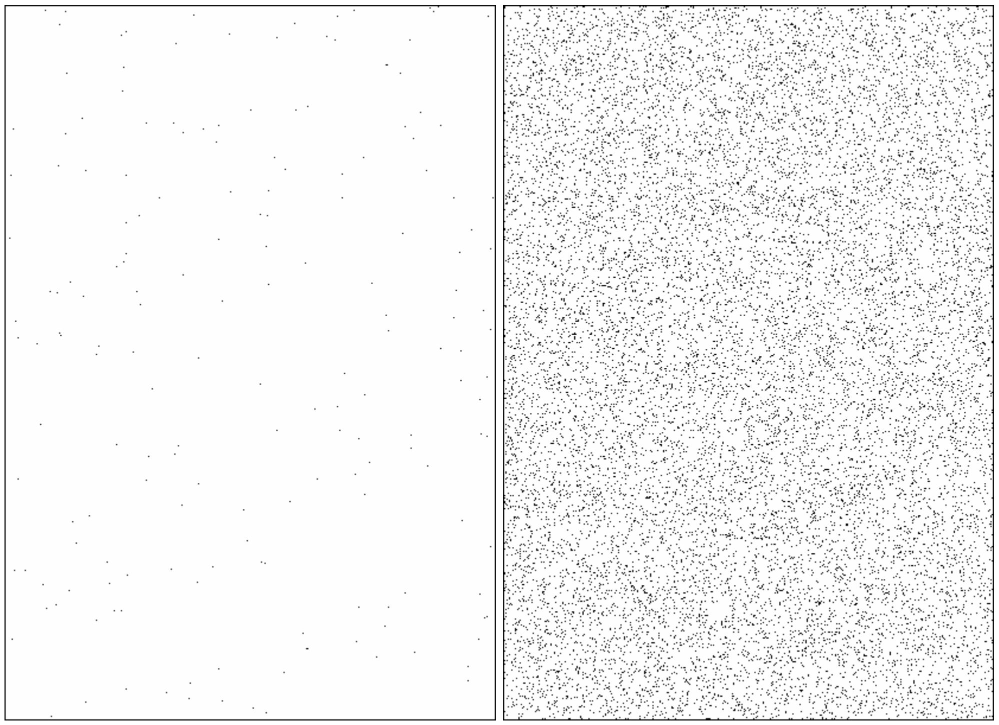

Figure 4-3: S-Tools 4 (LSB embedding); straddling message walk – Both pictures are true colour pictures that are spread and transformed into b/w so that each colour channel is displayed separately for better visibility of the changes caused by the embedding. Black pixels represent a change of the carrier medium. The left picture contains a secret message with 0.006 % and the right one with 1.65 % of the carrier size – embedded with S-Tools v4.0 and encrypted by IDEA.

4.1.3.5 Capacity

A further essential fact is the amount of secret bits that can be hidden in a carrier medium (by a specific tool). On the one hand there is the effort to maximise the payload which can be covered. On the other hand there is the attempt to minimise the detection possibility. Both objectives are interdependent because larger payloads lead to higher levels of change density and thus a higher detection possibility. The analysis of the steganographic utilities shows that the tools offer different amounts of payloads. We group five capacity categories which are displayed in table 4-1-9. Taking this into account the question arises which factors are responsible for determining how much data can be embedded. Primarily it depends on the embedding technique that the software uses. For example all tools that belong to the category "unlimited" apply the appending technique. This method usually has no data length restriction. The majority of software that is based on the header insertion can carry a large amount of data as well. By contrast, LSB, masking, substituting of program instructions, and most text-based techniques provide capacity limitation. Tools which use these embedding methods constitute the remaining capacity categories.

Looking at several carrier file formats discloses further aspects and results. The embedding capacity of images in bitmap (BMP) format is determined by the number of bits per pixel that can be changed and the number of pixels that are selected to hide the secret data. The method most used to hide data in images – which is LSB – provides a capacity up to 12 % (changing 1 bit per pixel) or up to 33 % (changing 3 bits per pixel). The capacity of JPEG, GIF and PNG files depends on the level of compression.

Capacity category	Number of algorithms	Percentage of all algorithms
< 1 %	18	15
1 % – 5 %	23	19
5 % – 15 %	47	39
> 15 %	14	11
“unlimited”	20	16

Table 4-1-9: Capacity overview of steganographic algorithms

Audio files in WAV format show a similar behaviour to BMP format image files.
In most cases the capacity of text files to carry hidden data depends significantly on the file’s content but can be estimated as follows (in bits):

- standard method: capacity = number of words
- compatible method: capacity = number of lines

The capacity of the standard method is achieved by most tools whereas the capacity of tools with further restraints, such as adaptivity, can be estimated with one bit per line as a rule of thumb.
There are no general rules for the amount of secret data that a PDF file can hold but the number decreases by numerous and large embedded objects. The quantity of hidden bits a HTML file can hold is approximately equivalent to its number of lines.

4.1.3.6 Adaptivity

The data length restriction for secret data is determined by the file composition. For example an image that contains high frequency areas (such as grass) can be manipulated imperceptibly to a much greater extend than an image containing primarily low frequency areas (such as clear blue sky or mono-colour pictures). The same principle can be applied to any other media. In the case of an audio file one possibility is to choose those bytes of the audio stream which follow immediately after high sound levels because the human hearing cannot perceive low sound levels right after high ones. Therefore the noise which is caused by the secret message will be irrecognisable for the human hearing. This approach of embedding is called “adaptive embedding”. In our analysis we observed that only 20 % of the investigated steganographic tools utilise this technique (table 4-1-10).

Adaption capability	Percentage of tools
yes	20
no	80

Table 4-1-10: Usage overview of adaptive embedding

4.1.4 Discussion and conclusion

In present times steganography is an increasingly used concept due to the need to protect privacy even in case of domestic crypto regulation. Therefore the number of supported file formats increases enormously. While in former times steganography was limited to images today's software also works with more current formats like audio. Following general trends utilities also become more user-friendly. Previous generations were mainly based on command line handling with some knowledge about shell syntax necessary. Now even inexperienced users are able to run steganographic software by means of graphic user interfaces. However the core functions – the embedding methods in file formats, which are known for a long time – have not principally changed. Only adaptations of established methods like LSB were created despite reliable steganalysis techniques known as chi-square attack, RS analysis, and SPA (sample pairs analysis). Although LSB matching, which twiddles the steganographic values by addition of (randomly chosen) +1 or -1 (also called plus minus one steganography), is little more complicated to implement than LSB replacement, it only cautiously gains currency. Merely for the boundaries a case discrimination is necessary, because -1 leads to an overflow at the lower and +1 at the upper boundary. With just this little more effort a higher steganographic security is ensured since the existing detection methods for LSB matching are far less powerful than the steganalysis of LSB replacement. Another visible development is the growing commercialisation of steganography usage. Nearly all investigated commercial tools were published within the last four years. But still today's development impulses are only given by the researching sector.

Improvements were made in the field of preventing detection by means of capacity limitations. Some of the new tools have an explicit capacity limitation which is not obeyed in general und therefore cannot be seen as standard functionality of new tools. Again non-commercial tools have a leading role since they more often implement capacity limitations.

Lastly it can be said that there exist strong deficits in the sector of documentation. Even tools which are available as freeware or something of similar character are described insufficiently. Furthermore, a uniform classification beyond simple documentation of supported file types is unavailable until now. Concluding, more transparency surely would encourage further developments.

4.2 Transparency Benchmarking for Steganographic Algorithms

Considering only the three characteristics capacity, transparency, and robustness of a data hiding method, it is obvious that there is a trade-off between these three characteristics. No algorithm can provide maximum capacity and maximum transparency at the same time. This principle is shown in figure 4-4.

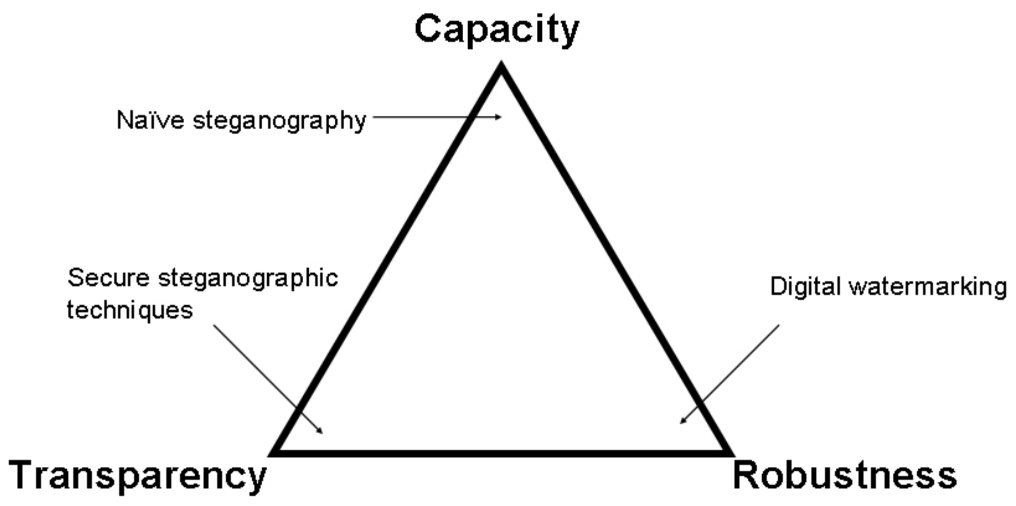

Figure 4-4: Trade-off between capacity, transparency, and robustness [FRIDRICH99].

In the corners of the triangle shown in figure 4-4 can be found the ideal positions for secure steganographic techniques, naive steganography and digital watermarking.
As indicated by the approach from Fridrich [FRIDRICH99] the transparency and capacity are the two characteristics most important to steganographic methods. In the focus of this document the secure steganographic techniques (i.e. high transparency) are more relevant than the naive steganography (high capacity). The later is neglected in the following considerations.
When dealing with secure steganographic algorithms it is assumed in the most cases that they perform very transparent but this characteristic so crucial to the performance is very seldom tested systematically on a larger scale.
As already described in section 3.5.3 in [KRAETZER2006A] watermarking and steganographic algorithms (Publimark version 0.1.2, Steghide versions 0.4.3 and 0.5.1 and a LSB steganographic algorithm implemented at the Otto-von-Guericke University of Magdeburg, Germany) where tested and compared for their embedding transparency. These benchmarks were performed on a large test-set of categorised audio signals using the average |ODG| (absolute value of the Objective Difference Grade computed using the Open Source software tool EAQUAL; see [KRAETZER2006A] for details) as a transparency measure. The results form [KRAETZER2006A] show for the selected steganographic algorithms that they perform indeed very transparent in the average for the test-set, but if single categories of audio signals are considered the algorithms become clearly distinguishable and lead in single cases to bad results (abnormal programme termination as well as bad (perceptible) embedding transparencies).
Other transparency benchmarking results for steganographic algorithms performed in [KRAETZER2006B] were used to compare two versions of a steganographic embedder for a VoIP scenario for their performance. The results there did show the strong impact of the usage of a silence detection algorithm on the transparency of a steganographic algorithm designed specifically for the usage in an audio streaming environment with limited bandwidth and 8-Bit quantisation.

4.3 Improving LSB Steganalysis using marginal and joint probabilistic distributions

LSB steganography is obviously one of the simplest way to hide information in the host data because a great amount of bits can be ranged without causing perceptible degradation of the cover object (digital images are generally coded with eight bits by colour channel and embedding in the low significant bitplane is not visible).

This section gives a summary of the work presented by Roue et al. in [ROUE2004]. The goal of this study was to test the steganographic method presented in [DUMITRESCU2003] on an image database to draw conclusion of the performance of the method according to images features (1D and 2D histograms). The accuracy of the studied steganalysis scheme is very good for almost 70% of the database. Indeed the results obtain on the whole set of images show that the algorithm has very high estimation accuracy: the Mean Absolute Error (MAE) of the estimate of the embedding capacity is above 10% for only 2% of the images and the MAE is very low for 13% of the images. In the whole set of images the estimation errors are relatively low.

Even if the scheme yields to very accurate estimations of the message's length, it is interesting to point out the features of the image that lead to important estimation errors.

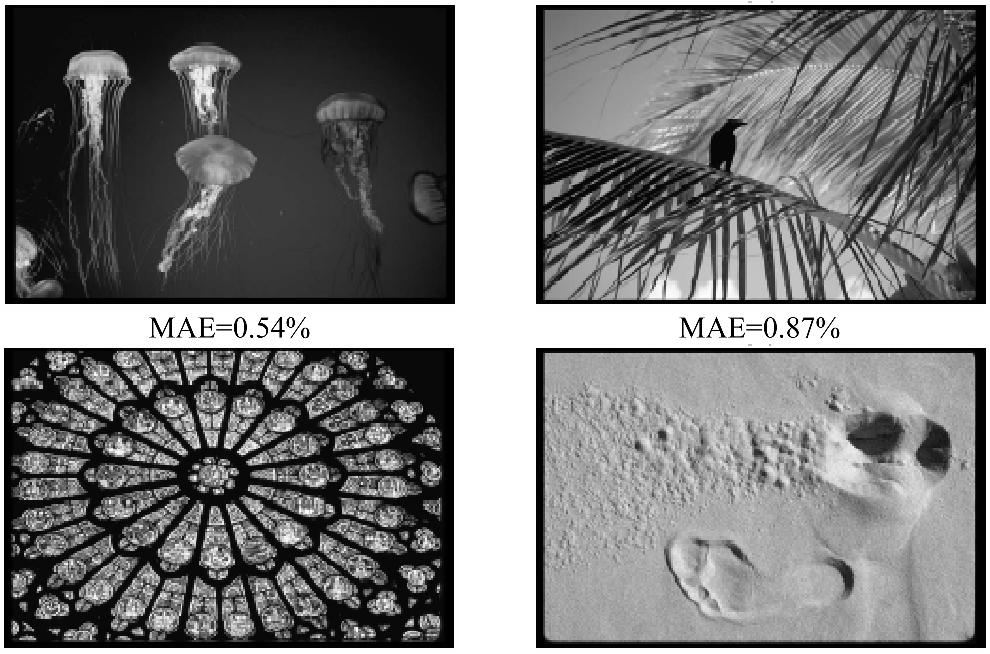

Figure 4-5: different images and different estimation error of the steganographic capity.

Our analysis leads to the fact that important estimation errors are due to two different drawbacks:

- a singular marginal distribution of the pixels in the image,
- a singular joint distribution of the pixels in the image.

To analyse joint distributions we have use the coocurrence matrix which represents the joint probability of two neighbouring pixels. Estimation of the coocurence matrix on stego and original images leads to important remarks:

- For images presenting a low estimation error, the distribution of the maxima of joint probabilities is concentrated along the diagonal.
- For images that present an high estimation error, the distribution of the maxima of joint probabilities is spread. It is better then to take into account only the sample pairs that are largely represented in the coocurrence matrix.

The choice of the threshold of the coocurrence matrix is important in this method. Therefore we have computed the MAE for several thresholds, in order to understand the meaning of this value. For several thresholds the estimation of the length of the message is not efficient at all. For example the threshold is too low, it is equivalent to use all the coefficients of the matrix of coocurrence. On an other side if the threshold is an important value, the estimate will rely on too few sample pairs of pixels.

We have tested this scheme on the three images of the Kodak database that have the worst results. For each image we have chosen the best threshold (i.e. the one which provides best estimations), and the estimation errors lie in the table below.

Image	MAE before processing (%)	MAE after processing (%)
Notredamewindow	12.01	4.23
Sandprints	15.18	4.2
Duneprints	10.14	3.2

Table 4-3-1: Estimation errors before and after the segmentation of the coocurrence matrix.

As a conclusion, this study enables both to detect images that are likely to counter steganalysis schemes, e.g. images that may contain a high ratio of undetectable hidden information and to improve the original scheme by reducing the outlined drawbacks.

5 Summary

In this report the WVL3 activities in the fields of benchmarking methods and tools for digital watermarks and steganography and steganalysis of the second ECRYPT year were indicated. Since some of the research results were already reported in other publications they were referred to.
One focus of WVL3s work is the evaluation and benchmarking of schemes and algorithms based on the work of WVL1 and WVL2. For this reliable frameworks for fair benchmarking have to identified or, if necessary, implemented. With SMBA, which is described in detail in section 3 of this document, a benchmarking suite is introduced which is concerned with the so far neglected audio watermarking domain. The results from the evaluations of watermarking algorithms provided for testing from other ECRYPT partners have not only shown the usefulness of SMBA but are also the foundation for a open and fair benchmarking scheme provided to all ECRYPT partners.
The steganography and steganalysis part of the work done in WVL3 is focused on basic research of available steganographic solutions to identify used approaches in a

rapidly evolving environment. This basic classification will be provided to the ECRYPT partners to act as a knowledgebase and foundation for the implementation of own steganographic and steganalytic tools. Steganographic evaluation frameworks could identify possible steganographic software tools for the testing of selected approaches by using this classification.
WVL3 will continue its research in watermarking and steganography benchmarking and in steganalysis using the research results identified here as a foundation for upcoming evaluations.

6 Bibliography

[AUDACITY] http://audacity.sourceforge.net/download/

[Comp03] Compris Intelligence GmbH, TextHide Overview, 2003
http://www.compris.com/TextHide/en/Overview.html#Umformulieren

[DITTMANN2004] Jana Dittmann, Martin Steinebach, Andreas Lang, Sascha Zmudizinski; Advanced audio watermarking benchmarking; Security, Steganography, and Watermarking of Multimedia Contents VI; Edward J. Delp III, Ping W. Wong (Eds.) SPIE Vol. 5306; SPIE and IS&T, pp. 224-235, Electronic Imaging Science and Technology, 19-22 Jan. 2004, San Jose, California, USA, ISBN 0-8194-5209-2, 2004

[DITTMANN2005] Jana Dittmann, Christian Kraetzer, Andreas Lang; *Attack tuning - Attack Transparency Models and their Impact to Geometric Attacks;* 1st Wavila Challenge, Barcelona, 8th-9th June 2005, ISBN 3-929757-89-3

[DUMITRESCU2003] S.Dumitrescu, X.Wu, and Z.Wang. Detection of LSB steganography via sample pair analysis. In IEEE transactions on Signal Processing, pages 1997- 2007, 2003.

[EGGERS2003] Eggers, J. J., Bäuml, R., Tzschoppe, R. and Girod, B., *Scalar Costa scheme for information embedding*, IEEE Trans. on Signal Processing, 2003

[FRIDRICH99] J. Fridrich, “Applications of data hiding in digital images” *Tutorial for the ISSPA 1999 conference in Brisbane, Australia,* 1999.

[GUELVOIT2005] Guelvouit, G. Le., *Trellis-coded quantization for public-key watermarking*, accepted for IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2005

[GUILLON2002] Guillon, P., Furon, T. and Duhamel, P., *Applied public-key steganography*, in Proc. SPIE, San Jose, CA, USA, 2002

[IFEACHOR2002] Emmanuel C. Ifeachor; Barrie W. Jervis: *Digital Signal Processing*, Prentice Hall, 2002, ISBN 0201-59619-9

[INOUE1999] Inoue, H., Miyazaki, A., Yamamoto, A., Katsura, T., *A Digital Watermarking Technique Based on the Wavelet Transform and Its Robustness on Image Compression and Transformation*, IEICE Trans. Fundamentals, vol. E82-A, no. 1, 1999

[JOHNSON2000] N. Johnson, Z. Duric and S. Jajodia, Information Hiding: Steganography and Watermarking Attacks and Countermeasures, Kluwer Academic Publishers, 2000

[Kim2003] Hyoung Joong Kim; Audio Watermarking Techniques, Pacific Rim Workshop on Digital Steganography 2003, Kyushu Institute of Technology, Kitakyushu, Japan, July 3-4, 2003

[Klimant2003] Herbert Klimant, Rudi Piotraschke, Dagmar Schönfeld, *Informations- und Kodierungstheorie*, TEUBER, 2. Eddition, ISBN 3-5192-3003-8, 2003

[KRAETZER2006A] Christian Kraetzer, Jana Dittmann, Andreas Lang; *Transparency benchmarking on audio watermarks and steganography*; to appear in SPIE conference, at the Security, Steganography, and Watermarking of Multimedia Contents VIII, IS&T/SPIE Symposium on Electronic Imaging, 15-19th January, 2006, San Jose, USA, 2006

[KRAETZER2006B] Christian Kraetzer, Jana Dittmann, Thomas Vogel, Reyk Hillert: Design and Evaluation of Steganography for Voice-over-IP, submitted to ISCAS 2006, Kos, Greece

[Kutter2000] M. Kutter, S. Voloshynovskiy and A. Herrigel; *Watermark copy attack*, In Ping Wah Wong and Edward J. Delp eds., IS&T/SPIE's 12th Annual Symposium, Electronic Imaging 2000: Security and Watermarking of Multimedia Content II, Vol. 3971 of SPIE Proceedings, San Jose, California USA, 23-28 January 2000

[LANG2003] Andreas Lang; Jana Dittmann; Martin Steinebach; *Psycho-akustische Modelle für StirMark Bechmark - Modelle zur Transparenzevaluierung* Rüdiger Grimm; Hubert B. Keller; Kai Rannenberg (eds.), Sicherheit - Schutz und Zuverlässigkeit, Informatik 2003 - Mit Sicherheit Informatik, pages 399–410, October 2003, Frankfurt/Main, ISBN 3-88579-365-2

[LANG2004] Andreas Lang, Jana Dittmann, StirMark and profiles: from high end up to preview scenarios, Reviewed Paper, IFIP/GI Workshop on Virtual Goods, Ilmenau (Germany), 28-29 May 2004, online publication available from http://virtualgoods.tu-ilmenau.de/2004/program.html

[LANG2005] Andreas Lang, Jana Dittmann, Ryan Spring, Claus Vielhauer; *Audio watermark attacks: from single to profile attacks*; Multimedia and security, MM & Sec'05 (Workshop New York, NY, USA August 1-2 2005); New York, NY : ACM, pp. 39 - 50, ISBN 1-59593-032-9, 2005

[Lang2005B] A. Lang, J. Dittmann, E. T. Lin, and E. J. Delp. "Application-oriented audio watermark benchmark service," in *Proceedings of the IS&T/SPIE's 17th Annual Symposium on Electronic Imaging*, 5681 - Security, Steganography, and Watermarking of Multimedia Contents VII, (San Jose, CA, US), January 2005.

[LANG2005C] A. Lang, J. Dittmann, E.T. Lin, E.J. Delp III, Application-oriented audio watermarking benchmark service [5681-28], page 275-297, SPIE-IS&T Electronic Imaging 2005, Vol. 5681, ISBN 0-8194-5654-3

[LANG2006] Andreas Lang, Jana Dittmann; Profiles for Evaluation - the Usage of Audio WET; to appear in SPIE conference, at the Security, Steganography, and Watermarking of Multimedia Contents VIII, IS&T/SPIE Symposium on Electronic

Imaging, 15-19th January, 2006, San Jose, USA, 2006

[libsamplerate] Secret Rabbit Code (aka libsamplerate), http://www.mega-nerd.com/SRC/, 2004

[Matev2005] Kiril Matev, Least Significant Bit Watermarking, University of Magdeburg internal report, 2005

[Megías2003] D. Megías, J. Herrera-Joancomartí, and J. Minguillón. "A robust audio watermarking scheme based on MPEG 1 layer 3 compression," in *Communications and Multimedia Security – CMS 2003, Lecture Notes in Computer Science 2828*, pages 226–238, Turin (Italy), October 2003. Springer-Verlag.

[Megías2004a] D. Megías, J. Herrera-Joancomartí, and J. Minguillón. "An audio watermarking scheme robust against stereo attacks," in *Proceedings of the Multimedia and Security Workshop*, pages 206–213, Magdeburg (Germany), September 2004. ACM.

[Megías2004b] D. Megías, J. Herrera-Joancomartí, and J. Minguillón. "Robust frequency domain audio watermarking: a tuning analysis," in *International Workshop on Digital Watermarking – IWDW 2004, Lecture Notes in Computer Science 3304*, pages 244–258, Seoul (Korea), November 2004. Springer-Verlag.

[Megías2005] D. Megías, J. Herrera-Joancomartí, and J. Minguillón. "Total disclosure of the embedding and detection algorithms for a secure digital watermarking scheme for audio," in *International Conference in Information and Communications Security* – ICICS 2005, Lecture Notes in Computer Science 3783, pages 427–440, Beijing (China), December 2005. Springer-Verlag.

[Megías2006] D. Megías, J. Herrera-Joancomartí, J. Serra and J. Minguillón. "A benchmark assessment of the WAUC watermarking algorithm," *Proceedings of the IS&T/SPIE's 18th Annual Symposium on Electronic Imaging*, 6072 - Security, Steganography, and Watermarking of Multimedia Contents VIII, January 2006.

[Publimark] Publimark, http://gleguelv.free.fr/soft/publimark/, 2005

[RIFE1989] D.D.Rife and J.Vanderkooy, Transfer-Function Measurement with Maximum-Length Sequences, JAES, June 1989

[ROUE2004]: B. Roue, P. Bas, J-M Chassery "Improving LSB Steganalysis using marginal and joint probabilistic distributions ", Multimedia and Security Workshop 2004, Magdeburg, Germany.

[Shapiro1993] Shapiro, J.M., *Embedded image coding using zerotrees of wavelet coefficients*, IEEE Trans. Signal Processing, vol. 41, no.12, pp. 3445-3462, 1993

[SMBA] StirMark Benchmark for Audio, http://amsl-smb.cs.uni-magdeburg.de, 2005

[SMITH1997] Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, 1997, ISBN 0966017633

[soundtouch] SoundTouch Sound Processing Library, http://sky.prohosting.com/oparviai/soundtouch/, 2004

[Wayn02] P. Wayner. Disappearing Cryptography. Information Hiding: Steganography and Watermarking. Second Edition , 2002

[XMMS] X Multimedia System, http://www.xmms.com/, 2004

Tool Resource Links

http://sourceforge.net/

http://packetstormsecurity.nl/

http://www.packetstormsecurity.org/

http://www.funet.fi/pub/crypt/steganography/

http://munitions.dotforge.net/software/steganography/

Appendix A - List of Investigated Steganographic Tools

- AppendX
- BlindSide
- Cameleon
- Camera/Shy
- Camouflage
- Ciphile Software's Steganography
- Cloak
- Contraband
- Contraband Hell Edition
- Cryptobola JPEG
- CryptoMX
- The Cryptographic Tool
- Data Stash
- DataStealth
- DC-Stego (DC-Steganograph)
- Encrypt Pic
- F5
- FFencode
- Gifitup
- Gifshuffle
- Gzsteg
- Hermetic Stego
- Hide
- Hide and Encrypt
- Hide and Seek

- Hide in Picture
- Hide4PGP
- Hydan
- Info Stego
- Invisible Secrets 4
- Invisible Secrets Pro (TM)
- Jpeg-Jsteg
- JpegX
- JPHide and JPSeek
- JSteg Shell
- MandelSteg and GIFExtract1.0
- Masker
- Mod_stego
- Mp3stego
- NicePCX
- Nicetext
- Outgess
- Paranoid
- PGMStealth
- Piilo
- PNGstego
- Pretty Good Envelope
- Proteus
- Secret
- SecurEngine
- Silan
- S-Mail
- Snow
- Sprytek
- Stash-It
- Steaghan
- Stealth Files
- Stegano Lite
- Stash-It
- Steaghan
- Stealth Files
- Stegano Lite
- Stegano Plug-In
- SteganoG
- SteganoGifPaletteOrder
- Steganografia
- Steganography
- Steganos Security Suite
- Steganosaurus
- Steganozorus
- Steggo
- Steggy
- Steghide
- Stego
- Stego WAV
- StegoDOS
- StegPage
- Stegparty
- StegSecurity Suite
- S-Tools
- TextHide
- Texto
- UnderMP3Cover
- Visual Cryptography
- Voices MP3 Stenography
- WBStego
- White Noise Storm
- Zfile Camouflage&Encryption System