Steganalysis in the Presence of Weak Cryptography and Encoding

Andreas Westfeld

Technische Universität Dresden
Institute for System Architecture
01062 Dresden, Germany
westfeld@inf.tu-dresden.de

Abstract. Steganography is often combined with cryptographic mechanisms. This enhances steganography by valuable properties that are originally left to cryptographic systems.

However, new problems for cryptographic mechanisms arise from the context of steganography. There are two sorts of steganographic tools: commercial tools with insecure or badly implemented cryptography and academic proof-of-concepts that abstain from the actual implementation of the cryptographic part.

Comparably to cryptography, steganography evolves in an iterative process of designing and breaking new methods. In this paper we examine the encoding properties and cryptographic functionality of steganographic tools to enable the detection of embedded information in steganograms even if the embedding part was otherwise secure.

1 Introduction

Digital steganography has evolved from an art in the early 1990's to a mature discipline in computer science. As a consequence, the gap between academic research and application has widened. The current situation is characterised by a huge and fast-growing number of publicly available tools offering steganographic functionality "out of the box." Apart from a poorly reflected use of deprecated and weak embedding functions, these tools also comprise all indispensable functions for pre- and post processing of message data and steganographic objects, respectively. These lateral processing steps are usually not covered in academic research. Since documentation is scarce and developers may tend to neglect possible security impacts, it was initially expected that a large number of tools would be vulnerable to simple (and thus avoidable) mistakes in the pre- and post-processing steps. We verified this assumption on a quantitative basis by scrutinising a number of arbitrarily chosen steganographic tools. This paper will give a survey on the security of current end-user steganography. Empirical evidence was gathered about weaknesses in steganographic tools, due to mistakes originating outside from the embedding function itself.

Y.Q. Shi and B. Jeon (Eds.): IWDW 2006, LNCS 4283, pp. 19–34, 2006.

The combination of steganography with cryptography delivers important and desirable security properties:

- Encrypting messages before embedding identifies authorised recipients by the knowledge of a secret key, or, combined with other key-dependent transformations, becomes a second line of defence.
- Permuting data values pseudo-randomly before embedding reduces the local embedding density and dilutes statistical measurements.
- Encoding of message bits can be used to alter distributions for better camouflage, or add redundancy for error correction and robustness.

Hence, cryptographic primitives and encoding principles are constitutional elements for secure steganographic systems; as they offer their desirable properties in other applications as well.

This paper is organised as follows: Sect. 2 describes our approach to find the steganographic tools, the cryptographic functionality of which we survey in Sect. 3. We encountered different types of weaknesses in steganographic tools. These are described in Sect. 4. For each type of weakness we developed at least one prototype, which we present in Sect. 5. The paper is concluded in Sect. 6.

2 Tool Search

Our search is based on a comprehensive collection of steganographic tools, which grew over several years and is updated from time to time. We observe other collections (e. g., Johnson [1], [2], and Petitcolas [3]) to gain search terms apart from generic ones like "steganographic tools" and "download." According to our definition, steganographic tools are programs targeted to the end-user, which embed data into carrier with the aim of secrecy of the fact that hidden data exists. This implies that watermarking algorithms, steganalysis tools, and pure cryptographic tools are not included. If we count multiple releases of the same tool or ports to different platforms as one tool, the collection comprises 96 tools.

All tools in the collection were installed, and the files shipped with the install packages were examined. Some quick tests were performed for the tools with undocumented algorithms, steganograms were produced with these tools and compared with the original to determine the embedding function, if it uses encryption, or straddles the steganographic changes over the whole carrier medium.

3 Employment of Cryptography in Steganographic Tools

65 % of the tools provide encryption of the message before embedding. The rest neglects encryption or assumes that the message is encrypted using extra software (see Table 1). Although it is hard or impossible to read encrypted messages without the correct key, the steganographic embedding can be detected in most cases. In addition, many encryption applications add header information to the encrypted message. If such header information can be extracted, this proves the

Table 1. Availability of encryption option

Cryptographic availability	Number of tools	Percentage of all tools
implemented	62	65 %
not implemented	28	29 %
not indicated	6	6 %
total	*96*	*100 %*

presence of the embedded message. Once this header and the encrypted message is extracted, a cryptanalyst can try to find out the message content.

We can divide the cryptography integrated in steganographic tools into two categories:

1. tools that employ widely used and standardised algorithms and
2. tools that use improvised or non-public cryptography, which are mostly not published and not reviewable.

Table 2 lists the number of implementations by their particular standardised cryptographic algorithms. Because a single tool can offer more than one encryption scheme, we count implementations instead of tools. The particular algorithms appear only in the table if three or more implementations are known. Schemes provided by two or fewer tools have been added to the group "Other." The table shows that the most popular encryption algorithm in steganographic software is Blowfish, closely followed by DES.

Finally, most of the improvised algorithms break Kerckhoffs' principle [4] because they are kept secretly. In contrast to open source algorithms (e. g., Blowfish, DES, etc.) they are not reviewed and evaluated by a wide community and should not be trusted.

4 Weaknesses

This section gives an overview of the weaknesses detected in our survey of steganographic tools (see Table 3). We found weaknesses in 18 tools, with some seeming to be systematic while others are very specific. We explicitly do not count broken embedding function as a weakness, because they are not caused in the cryptographic or encoding part.

The indicated weaknesses lead to more or less strong attacks. Similar to cryptography, we can define classes of different strengths.

Total break: One can separate carriers and steganograms *almost* completely. (In cryptography: recovery of the secret key)

Existential break: One can tell for some files that they are *almost* certainly a steganogram. (In cryptography: one can forge the signature for some text, but not all texts.)

Table 2. Types of encryption schemes

Cryptographic category	Encryption scheme	Number of implementations	Percentage of all implementations
Standardised		*86*	*80 %*
	Blowfish	15	13 %
	DES	10	8 %
	RC4	5	5 %
	Twofish	5	5 %
	ICE	4	4 %
	IDEA	4	4 %
	AES	3	3 %
	GOST	3	3 %
	PGP	3	3 %
	RSA	3	3 %
	Other	31	29 %
Improvised		*22*	*20 %*

Exclusion of innocuous files: One can tell for sure which files (usually many) have not been modified by the tool under attack.[1]

The definitions use the word "almost" because, in contrast to cryptography, one not only has to distinguish between "able" and "unable" to read some information, but also have to resolve its origin: was it deliberately there or by chance. In most cases, there is a very small probability for the latter.

4.1 Magic Prefix

One of the most common mistakes is the inclusion of unencrypted status information at a deterministic place in the file (e. g., the beginning of the file). An adversary can access this information and use it to separate steganographic objects from plain carrier. This degrades the affected tool to a simple cryptographic tool with a very bad payload to data ratio. The presence of a magic prefix leads to total breaks. Examples: BlindSide [5], Stegano [6]

4.2 Specification of the Length of the Hidden Message

Some tools embed the length of the hidden message as status information with a fixed number of bits. While this is not as easy to detect as deterministic redundancy (cf. Sect. 4.1), comparing the length information with the maximum

[1] In some cases, one cannot exclude post-processing. Then "exclusion" means that this tool was not the last one to process the image.

Table 3. Selected steganographic tools and their weaknesses

Tool name	Attack prototype	Magic prefix	Fixed length	Varying length	Attachment	Small key	Weak straddling	Improper encoding	Break
AppendX ..	apdet	—	—	—	×	—	—	—	excl.
BlindSide ..	bsdet	×	(×)	—	—	×	—	—	total
Camera/Shy	—	—	—	—	—	—	(×?)	(×)	(exist.)
Contraband	codet	—	×	—	—	(×)	×	—	excl.
Cryptobola.	—	—	—	—	—	—	(×?)	—	(exist.)
Encrypt Pic	epdet	—	×	—	—	—	(×)	—	excl.
Gif it up!...	giudet	—	×	—	—	—	(×)	×	excl.
Gifshuffle...	—	—	—	—	—	—	(×?)	—	(excl.)
Hermetic...	—	—	(×?)	—	—	—	(×?)	—	(excl.)
Invisible....	rdjpegcom	—	—	—	×	—	—	—	excl.
Jsteg.....	jsteglen.R	—	—	×	—	—	(×)	—	excl.
Masker.....	apdet	—	—	—	×	—	—	—	excl.
MP3Stego..	—	—	(×?)	—	—	—	(×)	—	(?)
Pngstego...	—	—	—	—	—	(×?)	—	—	(excl.)
Stegano....	stdet	×	×	—	—	—	×	(×?)	total
Steganos...	—	—	—	—	—	—	(×?)	—	(excl.)
Steggy.....	apdet	—	(×)	—	×	—	—	—	excl.
Ump3c.....	updet	—	—	×	—	—	×	—	excl.

×—weakness,
excl.—exclusion of innocuous files,
total—total break,
exist.—existential break,
?—slightly better than random guessing,
(...)—attack not implemented

capacity for a given file size can easily identify a large number of plain carriers. This ratio increases for smaller carrier files. Examples: BlindSide [5], Contraband [7], EncryptPic [8], Gif it up! [9], Stegano [6], Steggy [10].

A similar comparison between length information and capacity can be made for those tools that avoid reserving a fixed number of bits, but store a prefix of the size of the length information before the actual information. Here, the adversary on average gains a little less information compared to the case with

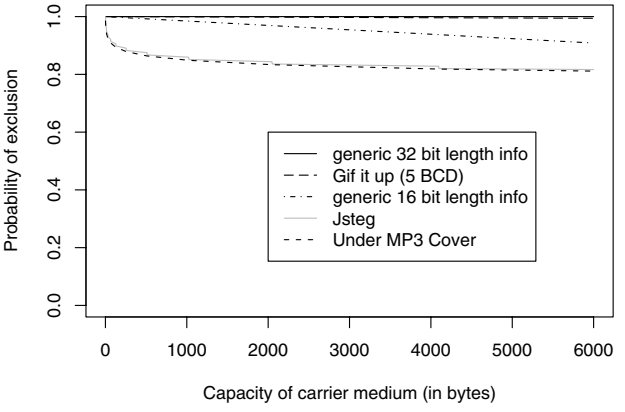

Fig. 1. Probability of exclusion by the attack under the assumption of uniformly distributed steganographic bits

fixed length information, but is still able to exclude innocuous files. Fig. 1 shows that the exclusion rate depends on the capacity of the carrier medium. Although the variable length field performs better, we can still exclude about 80 % of the carriers that provide less than 6 KB capacity. Examples: Jsteg [11], Under MP3 Cover [12].

4.3 Attached Message

We found some tools that do not actually alter the carrier data, but simply use special comment fields or undocumented areas at the file end to store the so-called "hidden" information. These tools provide almost no security. Surprisingly, quite a lot pursue this approach. However, because some commercial editors append their name to files or some executable files contain proprietary resources before file end, manual inspection is required for a total break. Machines can only exclude those files that don't have attachments. Examples: AppendX [13], Invisible [14], Masker [15], Steggy [10].

4.4 Small Keys

Another flaw is the insufficient length of the security parameter used for cryptographic functions. Some tools generate very short hashes from pass phrases to initialise random number generators. This enables an adversary to try the whole key space with brute force in reasonable time. Examples: BlindSide [5], Contraband [7], Pngstego [16].

4.5 Weak Straddling

A more specific vulnerability is the concentration of steganographic alterations in small parts of the carrier. Permutative straddling is a well-known and effective technique to avoid this problem. It is important to permute the alterations really

randomly, because an adversary can anticipate and exploit any deterministic step size (whether adaptive or not). This is not a severe weakness per se, but steganalytic attacks are much easier if the content is concentrated. Examples: Contraband [7], Encrypt Pic [8], Gif it up! [9], Jsteg [11], MP3Stego [17], Stegano [6], Under MP3 Cover [12].

4.6 Improper Encoding of the Embedded Message

Last but not least, a poor choice of source encoding—even after a cryptographic operation—introduces redundancies that can be detected by an adversary to identify steganograms, i. e., lead to existential breaks. Examples: Camera/Shy [18], Gif it up! [9].

5 Prototype Applications and Sample Sessions

All of the weaknesses classified in the previous sections have been successfully attacked with at least one prototype application. The prototypes are listed in Table 4. They are implemented in C/C++, and ready to download as source

Table 4. List of prototype applications

apdet	detects data appended to files
bsdet	determines a valid password for BlindSide
codet	excludes files not produced by Contraband
epdet	determines length information produced by EncryptPic
giudet	excludes files not produced by Gif it up!
stdet	detects data hidden with Stegano GIMP
updet	excludes files not produced by ump3c

code and Windows executables[19]. The usage is quite intuitive and does not require lengthy documentation: The programs are called from the command line with a filename of the medium under investigation as a parameter.

5.1 `apdet`—Detection of Appended Messages

The most primitive steganographic programs just append the message to the carrier medium. Although some of these programs are sold for a reasonable amount of money, their work can be done with simple commands that are already part of operating systems.

```
Unix:
    cat carrier.bmp readme.txt >steganogram.bmp
Windows:
    copy /b carrier.bmp+readme.txt steganogram.bmp
```

Such steganograms are more or less reliably detected by `apdet`. This tool currently supports the following carrier file formats: `.asf`, `.avi`, `.bmp`, `.dll`, `.exe`, `.gif`, `.jpg`, `.mid`, `.mov`, `.mp3`, `.mpg`, `.ocx`, `.snd`, `.tif`, and `.wav`. `apdet` needs a file name or directory that is automatically scanned (including subdirectories) for file names with the aforementioned suffixes. If additional bytes after the structurally expected end of the file are detected, the number of these bytes and the file name is written, otherwise `apdet` is silent.

```
$ apdet .
There are 333 bytes after the expected end of file!
... in file ./apstego.bmp
```

5.2 `bsdet`—Password Detection for BlindSide

`bsdet` detects steganograms created with BlindSide [5]. If the secret data are protected with a password, a string with four characters that is equivalent to the original password is derived and can be used to decrypt the embedded data. BlindSide hashes the password to a 16 bit key. Even worse: We don't have to search the whole key space. The structure of the embedded header (cf. Table 5) contains the known plaintext "OK," which can be used to determine the 16 bit key directly and convert it into an appropriate password. For instance, "cppe" is equivalent to the password "abc123," which was used to create the file `bsstego.encrypted.bmp`.

```
$ bsdet carrier.bmp
File does not contain any BlindSide hidden data.

$ bsdet bsstego.bmp
This file contains BlindSide hidden data.
Data is not encrypted, there is no password.

$ bsdet bsstego.encrypted.bmp
This file contains BlindSide hidden data.
Data is encrypted, a valid password is "cppe".
```

Table 5. Blindside message structure

Offset	Length (bytes)	unencrypted	encrypted
0	2	"BS"	"BE"
2	4	Length n	Encrypted length n
6	1	Version (0x01)	Encrypted version
7	2	"OK"	Encrypted "OK"
9	n	Message	Encrypted message

5.3 codet—Exclusion Despite Straddling

Contraband [7] uses a PIN[2] dependent straddling of the changes in the image. This PIN is needed to extract the length of the embedded message. This straddling is not very secure, and it is possible to extract the length with some errors. Nevertheless, `codet` is able to exclude a lot of carrier files from being steganographic. In the following example, we derive the number of leading zeros from the capacity. Since we know the earliest and latest bit positions of the (randomly straddled) length field, we can exclude files with less than 18 zeros in that part.

```
$ codet carrier.bmp
Analysing file carrier.bmp
capacity: 24000 bytes
--> 18 zeros expected in embedded length.
File does not contain any contraband hidden data.

$ codet costego.bmp
Analysing file costego.bmp
capacity: 24000 bytes
--> 18 zeros expected in embedded length.
min possible offset = 0
max possible offset = 3
There are 36 possible combinations for the first two digits of
the PIN.
possible sizes of embedded data: 1024, 2049, 4099, 8199 bytes

111000000000010000000000000000000010010001101000101001101 <-- extracted bits
123456789012345678901234567890123456789012345678901234567890 <-- ruler
```

5.4 epdet—Exclusion by Length Information

Encrypt Pic [8] embeds the length of the message in the first 8 pixels. The capacity offered by this tool can be determined by the number of pixels divided by 2 (in bytes; one byte fits in two pixels). The image was not created by Encrypt Pic if the potential length specification exceeds the capacity−8. The length and other header information use 8 bytes of the capacity. The tool `epdet` compares the length and capacity to decide if a file is a potential steganogram.

```
$ epdet carrier.bmp
Analysing file carrier.bmp
capacity: 31992+8 bytes
File does not contain any encpic hidden data. (length
exceeds capacity: 118386139 > 31992)
11011011101101100111000011100000 <-- 32 length bits (LSB ...  MSB)
```

[2] Personal identification number, here: a four digit numeric password.

```
$ epdet epstego.bmp
Analysing file epstego.bmp
capacity: 31992+8 bytes
length of embedded data: 18713
10011000100100100000000000000000 <-- 32 length bits (LSB ...  MSB)
First two blocks of embedded data (2 x 128 bits)

01000001010011101110100110010011010011111010001010111011110010010000011100 0...
11000101101101110010111110011001010001000000011000110001000000101011001110110...
12345678901234567890123456789012345678901234567890123456789012345678901234567890 <-- ruler
```

Encrypt Pic can also encrypt the message with an unknown encryption algorithm. `epdet` dumps the first two 128 bit blocks of the ciphertext to the screen. If the plain text is periodically repeated (e. g., a series of zeros only), the ciphertext blocks are all the same. So at least the mode of operation used here is weak.

```
$ epdet epstego.periodic.bmp
Analysing file epstego.periodic.bmp
capacity: 290832+8 bytes
length of embedded data: 262144
00000000000000000010000000000000 <-- 32 length bits (LSB ...  MSB)
First two blocks of embedded data (2 x 128 bits)

00001101100100110010110111000000010111000111010010010101001001011111000011001...
00001101100100110010110111000000010111000111010010010101001001011111000011001...
12345678901234567890123456789012345678901234567890123456789012345678901234567890 <-- ruler
```

5.5 `giudet`—Exclusion by Length Information

Gif it up! [9] uses the first 20 pixels of a GIF file for storing the length of the embedded message. The size of the message is limited to 99,999 bytes. For each of the five digits, four bits are used to store the binary representation of the numbers 0 to 9, i. e., the length is encoded as binary coded digits (BCD). Consequently, media can be excluded from being steganographic in two ways. First one can exclude all length specifications that exceed the capacity of the current image as described in Section 4.2. Second, all digits > 9 (1010...1111) can be excluded because they are not BCD.

`giudet` uses this principle to detect valid length specifications embedded with "Gif it up!"

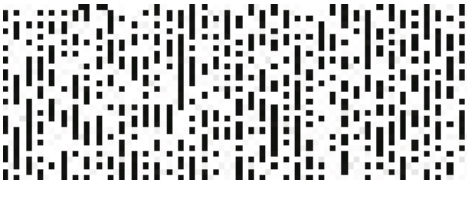

Fig. 2. Gif it up!: equidistant straddling (black: changed pixels)

```
$ giudet carrier.gif
Processing file carrier.gif
File does not contain any Gif-it-up hidden data.

$ giudet giustego.gif
Processing file giustego.gif
Possible length of embedded data: 2032
```

5.6 stdet—Detection by Magic Prefix

The embedding principle of Stegano [6] (a plug-in for GIMP) can be called LSB matching by increment (see Fig. 3). This is harder to detect than LSB replacement and the introduced error is smaller than with plus minus one steganography. Pixel values at the upper bound are only changed by decrement, because they cannot be incremented. Stegano is also the only known program that embeds in vertical direction, column by column from left to right (see Fig. 4).

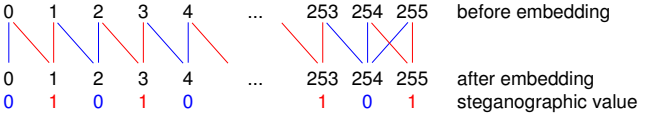

Fig. 3. Stegano: LSB matching by increment

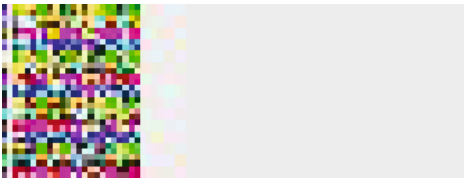

Fig. 4. Stegano: continuous embedding in vertical direction (coloured pixels are changed)

Stegano can be easily detected by the magic prefix `STG`, which is prepended to every embedded message. There is also information about the file name and length of the embedded message. Even if the embedded message is encrypted, which is possible only using an external tool, the prefix and header is still plaintext. This is exploited by `stdet`, which can reliably detect messages embedded with Stegano.

```
$ stdet carrier.bmp
Analysing file carrier.bmp
File does not contain any Stegano-GIMP hidden data.
```

```
$ stdet ststego.bmp
Analysing file ststego.bmp
Steganographic message detected.
embedded file name: /usr/users/mat02/s1924087/stegano/README
length of embedded data: 2140
```

5.7 updet—Exclusion by Length Information

Under MP3 Cover (ump3c [12]) is a tool that embeds one bit into every second block of an MP3 stream. As with some other tools, ump3c stores a length word in front of the actual data. Instead of reserving a fixed width, ump3c reserves 6 bits to specify the width. This makes the analysis interesting. Reasoning about the distributions of initial bits, under the assumption that normal MP3 files hold uniformly distributed bits in their first blocks[3], gives a theoretical ratio of arbitrary MP3 files that can be easily detected as non-steganographic.

```
$ updet carrier.mp3
Analysing file carrier.mp3
capacity: about 159 bytes
File does not contain any ump3c hidden data.

$ updet ump3c.stego.mp3
Analysing file ump3c.stego.mp3
capacity: about 159 bytes
Possible length specification found: 148
```

5.8 Analysis of Non-random Straddling Functions

MP3Stego [17] embeds into the parity of block lengths of MP3 streams. This tool is a good example of a specific straddling function: For each block a coin is tossed. On heads, the block is used for embedding, on tails not. As an embedding rate of 50 % was deemed too small by the author of the tool, MP3Stego ignores every third tail of the coin. This leads to an interesting probabilistic selection rule, which can be modelled as a finite state machine with deterministic initial state s_1 (see Fig. 5).

This finite state model is used to compute the expected embedding probabilities for the first n blocks of an MP3 stream and expected additional information for steganalysis. As shown in Fig. 5, the probability indeed vibrates around the steady state probability 0.6 in the first 8 blocks. Unfortunately, this convergence appears too fast, so that the additional information is marginal and does not lead to a significantly better ability to tell plain carriers apart. Nevertheless, this technique may lead to better results in cases where the parameters are chosen differently.

[3] Incidentally, we gain additional confidence from the fact that one of the leading bits is always zero, which seems to be a programming error.

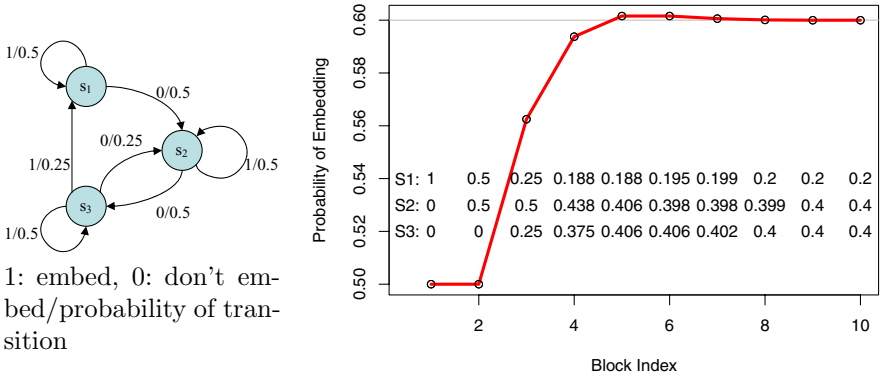

Fig. 5. Probability of embedding for the first 10 blocks with MP3Stego

5.9 Application of Jsteg Length Information to Jsteg Attacks

Jsteg length information can be used to reduce the detection power of targeted Jsteg attacks. We use two sets of 936 JPEG images to demonstrate—based on Jsteg length information—the generic improvement of the attacks of Zhang and Ping [20] and Lee et al. [21]. The first set C consists of the original carrier images from the CBIR database [22]. The second set S is derived from C with a Jsteg like algorithm (random embedding path) at 5 % embedding rate. Because of this small embedding rate, the attacks cannot perfectly separate both sets and will result in false positives or false negatives.

Jsteg embeds the length l of the embedded message (in bytes) in a variable length field at the beginning of the JPEG file. This is similar to ump3c (cf. Sect. 5.7). The first five bits tell the width of the length field ($\lceil \log_2 l \rceil = 0 \ldots 31$ bits). l is stored in these bits. Beside the length information one can determine also the capacity c of the JPEG image. This is the number of DCT coefficients that are neither 0 nor 1 minus the length of the length information ($5 + \lceil \log_2 l \rceil$). If

$$l = 0 \text{ or } l > \left\lfloor \frac{c - 5 - \lceil \log_2 l \rceil}{8} \right\rfloor,$$

Jsteg will not extract anything, and if the 6th bit in the extracted stream—the most significant bit of l—is 0 (in about 44 % of all cases), nothing has been embedded using Jsteg, because the length of the length field is always minimal. With these criteria we can find 717 (77 %) out of the 936 files in C that do not contain a proper Jsteg message. In 54 % the capacity was exceeded by the length specification. Fig. 6 shows the improved Receiver Operating Characteristic (ROC) of the attacks when they are combined with knowledge about the Jsteg length information. Table 6 shows the improved reliability $\rho = 2A - 1$ where A is the area under the ROC curve, the false positive rate (FPR) at 0.5 true positive rate (TPR) and the TPR for 1 % false positives.

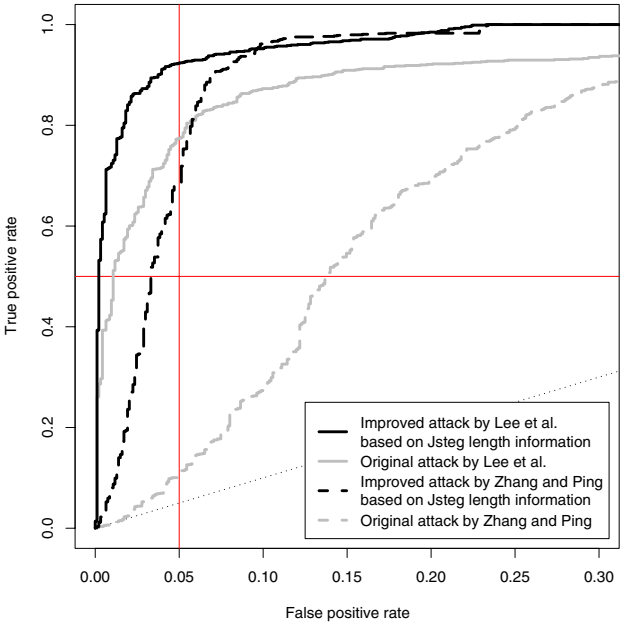

Fig. 6. ROC curves for generic improvement of targeted Jsteg attacks by exclusion of false positives based on Jsteg length information

Table 6. Reduced false positive rate and increased reliability ρ for targeted Jsteg attacks (5 % embedding rate)

Attack	ρ	FPR at 0.5 TPR	TPR at 0.01 FPR
Lee et al., original	0.8657	0.0107	0.4530
Lee et al., improved	0.9671	0.0021	0.7244
Zhang and Ping, original	0.6425	0.1389	0.0096
Zhang and Ping, improved	0.9163	0.0331	0.0801

6 Conclusion

The conclusions of this research are twofold. First, our prior belief that many end-user tools suffer from weaknesses in the encryption and encoding part has been confirmed. We presented a range of exemplary attacks against popular tools from the Internet, and the discovery of new weaknesses could be continued, although the academic outcome of analysing additional tools gets marginal.

Second, and more general, it remains the impression that the diversity of tools and carrier formats keep open the possibility for undetected steganographic communications, essentially because the resources to analyse every

tool and every exotic format[4] are limited. Even blind attacks cannot improve the situation substantially, because formats are not supported, features not developed, and training sets not available.

Acknowledgements

The author thanks Rainer Böhme for fruitful discussions, Benjamin Kellermann (né Scholz) for the implementation of `codet` and `apdet`, Kwangsoo Lee for providing his implementation of the category attack and 2 GB of test images, as well as Elke Franz for helpful comments on the paper. The work on this paper was supported by the Air Force Office of Scientific Research, Air Force Material Command, USAF, under the research grant numbers FA8655-04-1-3036 and FA8655-06-1-3046. The U. S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation there on.

References

1. Johnson, N.F.: Steganography tools (2002) Online available at `http://www.jjtc.com/Security/stegtools.htm`
2. Johnson, N.F.: Steganography and digital watermarking tool table (2003) Online available at `http://www.jjtc.com/Steganography/toolmatrix.htm`
3. Petitcolas, F.A.P.: Steganographic software (2005) Online available at `http://www.petitcolas.net/fabien/steganography/stego_soft.html`
4. Kerckhoffs, A.: La cryptographie militaire. Journal des sciences militaires **IX** (1883) 5–38, 161–191 Online available at `http://www.petitcolas.net/fabien/kerckhoffs/crypto_militaire_1.pdf`
5. Collomosse, J.: Blindside (2000) Online available at `http://www.blindside.co.uk`
6. Cotting, D.: Stegano (2000) Online available at `http://registry.gimp.org/plugin?id=314`
7. Thijssen, J., Zimmerman, H.: Contraband (1998) Online available at `http://www.xs4all.nl/~jult/4u/contrabd.exe`
8. Collin, F.D.: EncryptPic (2000) Online available at `ftp://ftp.elet.polimi.it/mirror/Winsite/win95/miscutil/encpic13.exe`
9. Nelson, L.: Gif it up! (2004) Online available at `http://digitalforensics.champlain.edu/download/Gif-it-up.exe`
10. Iceman81: Steggy (2003) Online available at `http://mesh.dl.sourceforge.net/sourceforge/steggy/steggy0.1rc1.tar.gz`
11. Upham, D.: Jsteg (2000) Online available at `0urls.txt:http://munitions.vipul.net/software/steganography/jpeg-jsteg-v4.diff.gz`
12. Platt, C.: Ump3c (2004) Online available at `http://mesh.dl.sourceforge.net/sourceforge/ump3c/UnderMP3Cover-1.1.tar.gz`
13. Bauer, M.: AppendX (2003) Online available at `http://www.unet.univie.ac.at/~a9900470/appendX/apX`

[4] For example, Stegogo (http://sourceforge.net/projects/stegogo) embeds information into GNU Go Games.

14. Neobyte Solutions: Invisible (2000) Online available at http://www.neobytesolutions.com/downloads/invsecr.zip
15. Zaretskiy, E.: Masker (2001) Online available at http://www.softpuls.com/masker/
16. HappyCactus: Pngstego (2003) Online available at http://mesh.dl.sourceforge.net/sourceforge/pngstego/pngstego-0.3.2.tar.gz
17. Petitcolas, F.A.P.: MP3Stego (2001) Online available at http://packetstorm.trustica.cz/crypt/stego/SourceCode/MP3Stego_1.0.14b1_src.tar.gz
18. Hacktivismo: Camera/Shy (2002) Online available at http://mesh.dl.sourceforge.net/sourceforge/camerashy/CameraShy.0.2.23.1.exe
19. Westfeld, A., Kellermann, B.: Detection utilities (2005) Online available at http://dud.inf.tu-dresden.de/~westfeld/detectors/
20. Zhang, T., Ping, X.: A fast and effective steganalytic technique against JSteg-like algorithms. In: Proc. of the 2003 ACM Symposium on Applied Computing, Melbourne, Florida (2003) 307–311
21. Lee, K., Westfeld, A., Lee, S.: Category attack for LSB steganalysis of JPEG images (2006) In these proceedings.
22. University of Washington: CBIR image database (2004) Online available at http://www.cs.washington.edu/research/imagedatabase/groundtruth