Practical invisibility in digital communication ⋆

Tuomas Aura

Helsinki University of Technology
Digital Systems Laboratory
FIN-02150 Espoo, Finland
Email: Tuomas.Aura@hut.fi

Abstract. This paper gives an overview of cryptographically strong mass application invisibility in digital communication. It summarizes principles and methodology, clarifies terminology, and defines some new concepts. A new algorithm for hiding bit selection in digital images is proposed and an experimental implementation of the algorithm is described. Finally, the paper closes with a discussion of the implications of the availability of invisible communication.

Keywords. Information hiding, invisibility, hiding bit selection in random access covers, pseudorandom permutation.

1 Introduction

The aim of encryption is to conceal the contents of secret messages. *Invisibility* goes yet further, it attempts to hide the fact that any messages even exist. The secret messages are hidden in ostensibly harmless data (*cover*) that can be casually stored and communicated without raising suspicion. While the designers of encryption algorithms assume that a resourceful enemy will do anything to decrypt the messages, the designer of an invisibility algorithm takes it granted that an enemy will try to reveal the existence of the secret message. Invisibility algorithms should be based on principles not unlike those of the encryption algorithms:

- Messages are hidden using a public algorithm and a secret key.
- Only a holder of the correct key can detect, extract or prove to a third party the existence of the hidden messages. Nobody else should be able to find any statistically significant evidence of their existence.
- Even if the enemy knows or is able to select the contents of the hidden messages, this should be no advantage in detecting other hidden messages.

⋆ An early version of this paper was presented at the HUT Seminar on Network Security in November 1995 [1].

– The algorithm must be cryptographically strong. That is, it is theoretically impossible or computationally infeasible to detect hidden messages.

There are several well-known methods for hiding secret messages in digital signatures and other well defined but scarce components of digital communication. In this paper, we look at similar techniques for bulk data, photographs and audio and video signals that are routinely transfered in large quantities. There are few references to such algorithms in literature, but a body of knowledge exists in the cryptographic community. Our goal is to survey the methods that are best suited for mass application, and to present a new algorithm.

In Section 2 we will overview hiding techniques and related terminology. Sec. 3 will go deeper into the the kinds of data that can have secrets hidden into it and the hiding algorithms. In Sec. 4 we present a practical algorithm for hiding data in digitized images. This is the main contribution of the paper. Sec. 5 briefly describes an implementation of the algorithm. We conclude with a brief discussion of the implications of the technology on organizations and society in Sec. 6.

2 Hiding data in digital communication

Practical invisibility algorithms are based on replacing a noise component of a digital message with a pseudorandom (usually encrypted) secret message. We call the noisy message *cover* and the bits carrying the noise *cover bits*. The bits of the pseudorandom secret message are *secret bits*. The cover bits actually substituted with the secret bits will be called *hiding bits*. We call this kind of a system *substitution method* of invisibility.

The cover data must be sufficiently noisy in order for a slight increase in its randomness not to be noticeable. The cover bits are typically the least significant bits (LSBs) of some inexact values. These can be results of a series of measurements. A digitized image or a sound track is, in fact, a huge array of measurement results, all with a significant random error component. The hiding bits can also be coded in the cover in more complicated ways than simply by replacing the LSBs. For instance, [3] describes how to find the least significant differences between data values in a palette colored image.

The least significant bits can, although they are noise from a measurement accuracy viewpoint, have some special statistical characteristics. Fig. 1 shows how the run lengths of LSBs in a signal from a low quality digital video camera differ from those in purely random noise. It has been often suggested that the encoding of the secret message should mimic the characteristics of the cover bits, a goal not easily achieved. One possibility is to generate a large number of alternative cover messages all in the same way and to choose one that happens to have the secret code in it. This is called *selection method* of invisibility. For example, we can digitize the same photograph again and again, every time producing a slightly different digital image. These images are then reduced with a message digest algorithm to, say, 8 bit numbers. If we want to hide an eight bit number, we pick the first scanner file that incidentally produces the particular number as a digest. This image has exactly the statistical properties that any other digitized image has. Selection invisibility can be viewed as the ultimately secure method, tantamount to one-time pad in encryption. The only problem with this approach is that, even if optimally organized, it can hide only small quantities of data with lots of work.

Another approach is to model the characteristics of the cover bit noise. A mimic function should be built that not only encodes the secret message, but also respects the model of the original noise. In the extreme case, the entire message is constructed according to the model. We call this *constructive* invisibility. The strategy has inherent dangers. Modelling the noise or an error component in data is not easy. Building the model requires significant effort, creative work to be done again for every communication channel or cover data source. It is likely that someone with greater resources and more time will be able to develop a better model of the noise and still differentiate between the original signal and the replacement. Furthermore, the patterns created by the model may reveal the secret message instead of increasing security. If the adversary knows the model, he can with little investment find flaws in it. The model of the noise is a part of the hiding algorithm and keeping the algorithm secret is in violation of good cryptographic practices as it is likely to leak to the hands of the unfriendly in any case.

Since attempts at mimicking the original noise either result in questionable security or far too low bandwidth for most applications, it is best to go back to the basic substitution procedure. Select a class of sufficiently noisy messages, cover messages, and identify the bits that carry the noise, cover bits. Then, approximate how large portion of the cover bits can be replaced with pseudorandom data without significantly changing the characteristics of the cover.

For instance, if the cover message is a digitized photograph, the cover bits are naturally the least significant bits of the grayscale or RGB values, or the Fourier coefficients in a JPEG compressed image. One might want

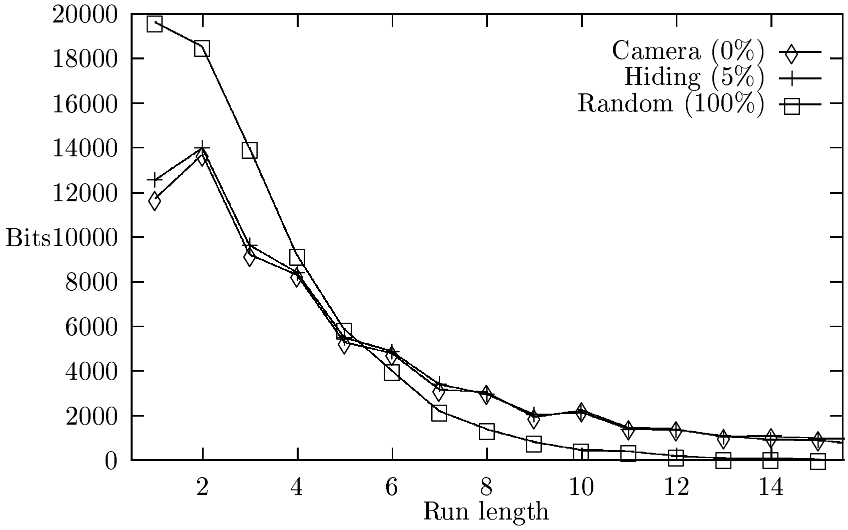


Fig. 1. Deviation of bits to runs of consecutive 0s or 1s.

to play safe and alter only every 100th pixel of the image. In that case, a one megabyte uncompressed image can hide about one kilobyte of secret data. In Fig. 1 we see that replacing as much as 5 % of the LSBs with pseudorandom bits does not significantly change the run length distribution. On the other hand, it is obvious that the popular practice of simply replacing all LSBs with secret bits is no option for serious cryptographers. As a consequence, secure invisibility inevitably depends on covers being at disposal in quantities.

The secret message should be encrypted with a strong cryptographic algorithm in order for it to look like random data. Substituting pseudorandom bits for a few of the noisiest bits of the cover will only slightly increase its noise level. On the other hand, inserting bits of a plaintext could drastically change the character of the cover. In theory, encryption is not essential to security if the invisibility algorithm is reliable. The pseudorandomness of the secret message could be achieved in other ways such as effective compression. The random permutation method presented in Sec. 4 does not even require pseudorandom secret bits, only ones with equal frequency of zeros and ones. In any case, we suggest that it is a good idea to both compress and encrypt the secret message before hiding. Compression decreases the number of secret bits thus improving security, and strong encryption will protect the contents of the secret message even if the much less mature invisibility techniques fail. This should be kept in mind because there is always a risk involved in cover selection, no matter how strong the invisibility algorithm is.

Moreover, the hiding bits must be selected in a pseudorandom fashion as a function of a secret key. Otherwise, an enemy having hold of the algorithm will be able to extract the secret bits. If these bits in a suspect message are completely random looking, as the products of encryption, but a similar selection of bits in another message has a some sort of statistical deviation different from purely random data, the enemy can conclude that the suspected image most probably contains a hidden data.

3 Covers and hiding bit selection

The cover messages can be divided into two types. The cover can be a continuous data stream, like a digital telephone connection, or a file, like a single bitmap image. We call the former a *stream cover* and the latter a *random access cover*.

In a stream cover, it may not be possible to tell in advance when the cover message begins, when it ends, or how long it will be. The cover may be continuous or very long, so that several secret messages need to be hidden in the same cover message. Furthermore, one may not know in advance what the next bits of the cover will be, but the secret message has to be inserted into the stream as it is transmitted, before the next bits of the stream have even been produced. The hiding bits have to be selected with a keystream generator that gives the distance between consecutive bits in the stream [6]. We call this *random interval method* of hiding bit selection.

In a continuous data stream, it is difficult for the receiver to tell when a hidden message begins. There can be no visible (or audible) synchronization sequence. Still, the receiver has to synchronize its keystream generator with the hidden message in order to pick the correct hiding bits from the stream. If the data stream has some inherent synchronization signals or packet boundaries, the hidden message should start right after one of them. The receiver will then try to synchronize its random number generator to the hidden message after those points. Usually, only garbage is found, but at times, an encrypted secret message is revealed. In the easiest case, if the data stream is of a finite length and reopened often, like telephone conversations, the secret message may always start at the opening of the session. For the sender, it may be a problem if he cannot trust the cover stream to be long enough for the entire secret message. With stream covers, it is also difficult to evenly spread the hiding bits all over the cover.

Invisibility in stream covers has a close relation to spread spectrum techniques, where the message is spread pseudorandomly over a large bandwidth. Spread spectrum communication is used mostly because of its resistance to interference and jamming but the same techniques can be used to hide that there is any communication at all [2].

Fixed length files as cover messages do not have the same disadvantages as stream covers. The sender knows in advance the size of the file and its contents. The hiding bits can be evenly selected with a suitable pseudorandom selection function from all parts of the message. Random access to the cover bits can be utilized to achieve this. Because of the random access, it is not necessary to insert the secret bits into the hiding bits in any particular order. For example, header info can be written after the secret message, whereas in a stream cover, the header has to be sent first. The main drawback of random access covers is that the cover size is often much shorter than a data stream, and it cannot easily be adjusted to fit the needs.

Despite us not giving much value to statistical models of cover noise above, they may still be helpful in detecting abnormally bad covers. Since random access covers are known in full before they are chosen for data hiding, good covers can be selected. For instance, in a series of digitized images well fit for hiding, there might be some that are completely black, not so noisy etc. These should be recognized and discarded. If covers are produced automatically in large quantities, statistical tests or visual inspection should be applied to ensure their quality before sending.

It should be noted that the the random interval method is not especially well suited for a hiding bit selection in random access covers. First of all, even distribution of hiding bits is only achieved probabilistically. That is, one cannot know in advance if all the secret bits will fit in the cover. Therefore, one has to play safe and make the average interval between bits so short that they will be all written before the file ends. Normally, the secret message will end far before the end of the message. It is then necessary to pad the secret message with random bits in order to cause an equal change in randomness in the beginning and end of the cover. Another disadvantage is that the distances between hiding bits are uniformly distributed between the shortest and longest allowed distances, while true random noise would have exponential distribution of interval lengths. It is, of course, possible to generate pseudorandom exponentially distributed numbers but this is usually too laborious. It is debatable whether the difference between exponential and uniform distribution of interval lengths will make any difference in the statistical characteristics of a noisy cover message. We tested the effect on run lengths in digital image data. When the hiding bit density is low enough compared to the run lengths, so that at most one bit in any run is likely to be altered, there is no difference in the resulting run length distribution. There are, however, other selection functions for random access covers that both escape the question of secret message padding and produce a random distribution of hiding bits: pseudorandom permutation functions. These will be discussed in detail in Sec. 4.

An interesting alternative to simple replacement of pseudorandomly selected cover bits with secret bits is to alter the parity of bit sequences. In a stream cover, the length of the next bit sequence is determined with a keystream generator, and in random access cover, a set of bits is picked with a selection function. If the parity of the selected bits is not the same as the next secret bit, the parity is changed by flipping any bit in the set. The bit to be altered can be chosen to cause minimum deviation from the original statistical characteristics of the cover. This approach has the same disadvantage as the constructive invisibility systems. Namely, the model according to which the choice of the altered bit is made may not be good enough, and the enemy might detect the secret message by looking at the places where a bit is most likely to be altered. Still, we believe that the parity method can be as strong as any of the pseudorandom substitution strategies if the model-based choice of the altered bit is made only to avoid the worst cases of random selection.

4 A secret key algorithm for hiding bit selection in random access covers

In this section, we describe a secret key method for pseudorandom selection of the hiding bits. The basic idea is to use a *pseudorandom permutation* of the cover bits. We have no knowledge of any previous invisibility algorithm or program where the choice of hiding bits is based on pseudorandom permutations. Let N be the number of cover bits available and let P_0^N be a permutation of the numbers $\{0, \ldots, N-1\}$. Then, if we have a secret n-bit message to hide, we can simply insert the secret bits into the cover bits $P_0^N(0), P_0^N(1), \ldots, P_0^N(n-1)$. The permutation function must be pseudorandom, i.e. it has to select bits in an apparently random order. Consequently, the hiding bits will be evenly spread all over the cover bits. No padding of the secret message with random data or other trickery is needed to assure even spreading.

For our purposes, the permutation function also has to depend on a secret key K. Therefore, we need a *pseudorandom permutation generator* P^N, a function which for every parameter K (secret key) produces a different pseudorandom permutation of $\{0, \dots, N-1\}$. We denote by P_K^N the permutation generator instantiated with key K. If the permutation P_K^N is computationally secure, that is, nobody can guess or reason the permutation without having access to the secret key K, it is impossible for anyone to guess which cover bits were chosen for hiding bits.

A secure pseudorandom permutation generator can be efficiently built from a *pseudorandom function generator* [5]. A pseudorandom function generator is like a pseudorandom permutation generator in that it produces a different unpredictable function for each secret key value, but the range of the function does not need to equal the domain. A pseudorandom function generator is easily constructed from any secure hash function H, such as SHS, by concatenating the argument i with a secret key K and feeding the resulting bit string to H.

$$f_K(i) = H(K \circ i),$$

where $K \circ i$ is the concatenation of the bit strings K and i. The result $f_K(i)$ is a pseudorandom function of i that depends on the parameter K.

The pseudorandom permutation generator of Luby and Rackoff [5] is constructed as follows. $a \oplus b$ denotes the bit-by-bit exclusive or of a and b and the result has the same length as a. Let i be a binary string of length $2l$. Divide i in two parts, Y and X, of length l and the key K into four parts K_1, K_2, K_3 and K_4. Compute

$$\begin{aligned}
&Y = Y \oplus f_{K_1}(X);\\
&X = X \oplus f_{K_2}(Y);\\
&Y = Y \oplus f_{K_3}(X);\\
&X = X \oplus f_{K_4}(Y);\\
&\text{return } Y \circ X;
\end{aligned}$$

For every key value K, the algorithm gives a pseudorandom permutation of $\{0, \dots, 2^{2l}-1\}$. Luby and Rackoff show that the permutation is as secure as the pseudorandom function generator. They also give a similar algorithm for permutation of $\{0, \dots, 2^{2l+1} - 1\}$. If the values of the function f_K are long enough bit strings, the same effect could be achieved simply by letting Y be the l first bits of i and X be the last $l + 1$ bits.

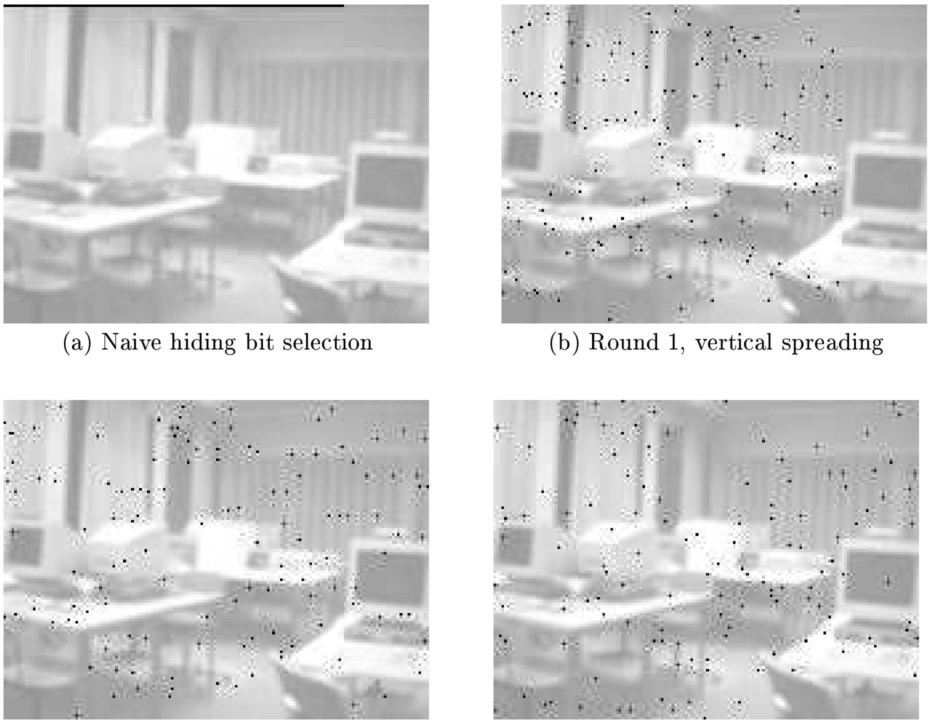
The above construction produces a permutation $P_K^{2^k}$ of $\{0, \dots, 2^k - 1\}$ for an arbitrary k. However, when the number of cover bits is N, we need a permutation P_K^N of $\{0, \ldots, N-1\}$. Our advantage here is that we can restrict ourselves to feeding arguments to P_K^N in the order $0, 1, 2, \ldots$. Let $k = \lceil log(N) \rceil$. Then, $2^{k-1} < N \leq 2^k$. Simply compute the values $P_K^{2^k}(0), P_K^{2^k}(1), P_K^{2^k}(2), \ldots,$ discarding from the sequence any numbers above $N-1$, and take the remaining values for $P_K^N(0), P_K^N(1), P_K^N(2), \ldots$. This is feasible when the permutation function is evaluated for increasing argument values starting from 0, as in our case. Thus, a permutation generator P^N for arbitrary N can be constructed from the Luby-Rackoff algorithm.

However, when N is known to be composite, there is a more convenient way to construct the pseudorandom permutation generator. The following algorithm is based on a block cipher with arbitrary block size [4]. The number of cover bits has to be a composite number with two factors of almost equal size, that is, $N = x * y$ for some x and y. When data is being hidden in the least significant bits of a digitized photograph, the factors x and y are, naturally, the dimensions of the image. To get the index of the ith ($i \in \{0, \ldots, N-1\}$) hiding bit, compute

$$
\begin{aligned}
&Y = i \text{ div } x; \\
&X = i \bmod x; \\
&Y = (Y + f_{K_1}(X)) \bmod y; \\
&X = (X + f_{K_2}(Y)) \bmod x; \\
&Y = (Y + f_{K_3}(X)) \bmod y; \\
&\text{return } Y * x + X;
\end{aligned}
$$

The two first rounds of the algorithm are needed for even spreading of hiding bits amongst the cover bits. Figure 2 illustrates how the first round randomizes the Y-coordinates of the hiding pixels and the second round randomizes the X-coordinates. The third round is necessary because of a chosen plaintext attack. With only two rounds, let $i = B * x + A$ and let $P_K^N(i)$ be the permuted value. If the cryptanalyst can guess A and can obtain a plaintext–ciphertext pair ($i' = C * x + A$, $P_K^N(i')$) for some C, he is able to deduce B. Even though we believe the algorithm to be secure with three rounds, it may well be that adding a fourth round would still significantly increase strength of the algorithm, at least when employed as a cipher.

Example. Assume that we have a grayscale image of size 800×600, an encrypted secret message of 1 kilobytes and a key $123, 456, 789$. We want to hide the secret message to the least significant bits of the image. $x = 800$,



(a) Naive hiding bit selection

(b) Round 1, vertical spreading

(c) Round 2, horizontal spreading

(d) Round 3, vertical spreading

Fig. 2. Spreading the hiding bits evenly all over the image

$y = 600$, and the number of cover bits equals the number of pixels in the image, $N = 800 \cdot 600 = 480000$. There are $1024 \cdot 8 = 8096$ secret bits. Less than 1 % of the pixels will be altered. The hiding pixels are selected with the above algorithm, starting from secret bit number 0. We now show how to compute the hiding pixel for the 1001st secret bit. (The values of the secure hash function H have been arbitrarily chosen.)

$$\begin{aligned}
Y &= 1001 \text{ div } 800 = 1, \\
X &= 1001 \bmod 800 = 201, \\
Y &= (1 + H(123 \circ 201)) \bmod 600 = (1 + 7377) \bmod 600 = 178, \\
X &= (201 + H(456 \circ 178)) \bmod 800 = (201 + 3854) \bmod 800 = 55, \\
Y &= (178 + H(789 \circ 55)) \bmod 600 = (178 + 1124) \bmod 600 = 102.
\end{aligned}$$

The result is $102 * 800 + 55$, meaning that the 1001th secret bit is inserted into the least significant bit of the cover pixel whose x-coordinate is 102 and y-coordinate 55.

This procedure for selecting the hiding bits is best in accord with digitized photographs as covers. In other file types, such as audio files, the number of cover bits may not always have a suitable factorization. Then, one is forced to go back to the previously described system of discarding some of the permutation function values.

In addition to a reasonable noise level, the algorithm makes some assumptions about the covers. *The same cover message must never be used again. The original cover file must be completely destroyed.* With two differing copies of the file, it is easy to see which bits are the hiding bits. This makes it plain and clear to the enemy that messages are being hidden. It would be best if the original cover message is never saved anywhere, but the secret message is inserted immediately after the cover has been produced.

Another question arises when the algorithm is applied in practice. If the secret key and cover size remain unchanged, the hiding bits will always be the same. Obviously, the same key should never be used for more than very few images. Fortunately, one can get around this problem by making the key dependent on the cover message. Let K_u be the secret key of the user and H' a random function on the user keys and cover messages that does not change when the hiding bits are altered. Then, take the key K to be

$$K = H'(K_u \circ cover).$$

The function H' has to depend so heavily on the cover that two different covers will not produce the same value, but it has to be independent of the choice and contents of the hiding bits. For example, in a bitmap image where the cover bits are the least significant bits, the function H' can be a message digest of the user key and the image after all LSBs are set to zero. It is extremely unlikely that there will ever be two digitized images only differing in the LSBs. By making the key K a function of the cover, the same user key K_u can be used for as many covers as needed. The same key cannot, of course, used again for the same cover, since the covers must never be reused.

One more practical concern is that the secret message length must be short compared to the cover size. What is the maximum ratio of secret message size to the cover size, remains to be determined separately for each cover source and application. We do advice to choose covers that are so abundant that there is not reason to exploit more than a few percent of the cover bits for hiding.

The strength of an invisibility algorithm will always depend heavily on the properties of the cover. If we select cover bits with a random error component and keep the hiding bits to cover bits ratio low, the security of the presented algorithm depends on the random function f_K being computationally secure. We are rather safe here since the pseudorandom nature of the secret message makes a brute force attack infeasible: The enemy would be trying to identify a pseudorandom bit sequence, but almost any choice of a key will give one. What the attacker can do is to erase the hidden message by adding his own noise to the cover. It is not even necessary to erase all random bits but only alter some of them, and the encrypted secret message cannot be recovered.

5 An experimental implementation

The algorithm of Sec. 4 was implemented to hide files in PGM grayscale bitmap images. The reasons for the choice are that the PGM is a widely accepted standard format that is easily converted to other non-lossy compression image formats. Masses of grayscale images are at hand, for instance, from surveillance cameras. Nonetheless, there is no reason why the algorithm could not be implemented on other file formats or color images.

The least significant bits were chosen as the cover bits. The secret message is encrypted with the IDEA cipher to make it pseudorandom. f_K is constructed from the secure hash function SHS. An SHS message digest is computed on the user key and the 7 most significant bits of the image. The digest is used as the encryption and hiding key K.

A magic number, and the name and length of the secret file are hidden along the secret message. It is important to note that they all are encrypted and hidden into the cover bits the same way as the secret message itself. The secret key is needed for extraction and decryption. If the secret message is short in comparison to the image size, it is impossible to detect or prove the existence of the secret message without the key.

An advantage of our algorithm in comparison to the random interval method is that it is very easy to implement. A disadvantage is the speed. The algorithm requires three evaluations of a random function per hidden bit, while the random interval method manages with one. SHS with its 160 bit range is not an optimal choice for the random function since only around 10 bits are needed. Computational costs of the algorithm are hidden in the random function, it being thus the most promising part for further optimization. Nevertheless, the implementation of the algorithm proved to be efficient enough for interactive hiding of data into persistent image files. It is questionable whether the speed is sufficient for on-the-fly hiding in a stream of images. As expected, the hiding does not change the visual appearance of digital photographs at all. When only a few percent of the image pixels were utilized for hiding, the run length distribution of the LSBs did not change significantly in comparison to the natural variation between images. The assumption that more advanced statistical methods cannot spot the hidden data remains to be tested.

6 Conclusions and discussion of the implications of invisibility

We have given an overview of mass application invisibility and proposed a new method for pseudorandom hiding bit selection in random access covers. An experimental implementation of the algorithm on grayscale images was described. All along, we have stressed cryptographic strength of the invisibility techniques. Apart from optimization of the algorithms, future research should concentrate on statistical methods for detecting the increased noise in real covers. A concrete measure would be needed for how much data it is safe to hide in a given cover. We still need to make some conclusions on how the described technology can be applied and what kind of effect it has on the society.

The availability to effective data hiding has serious practical implications. Any type of information can be saved and transfered invisibly where there is a sufficient storage or flow of noisy digital data. The most common types of data transfered in volumes, digital audio and video signals and digital image files, are exactly the ones most apt for hiding secret messages.

The establishments that are perhaps most dramatically affected by the danger of uncontrollable information flows are military organizations. Military security is traditionally based on keeping secret or confidential data in physically controlled locations, so that even the trusted personnel cannot move anything more than they can memorize over the physical boundaries. Special rules have been established for reviewing and formally downgrading data before it can be moved to a lower security area. The possibility of automatically hiding confidential documents in unclassified data flows presents a new challenge to the military information infrastructure. These organizations must, nevertheless, take full advantage of new information technology. On the bright side, information hiding can be utilized to frustrate traffic flow analysis attacks.

The inability to control the types of data transfered also has a profound influence on the commercial communications service providers. It would be in the interest of telecom operators to charge an optimum price for all different types of data. Bulk rates would apply when data is transfered in large quantities, as in video conferencing. Low bandwidth communication, such as electronic mail, would be more expensive per bit. It is, however, impossible to differentiate between the forms of communication. Where digital video is regularly transmitted, any amounts of mail can be hidden into the signal. This implies that the pricing of communication resources has to be based on bandwidth, and availability and reliability of the service, not the types of data transfered.

Probably the most controversial issue about widely available secure communications is that the same technology can be employed for legally and morally questionable purposes. It has been claimed, for example, that free application of cryptography enables drug traffickers and terrorists to communicate in secret, without the law enforcement officials being able to intercept their messages. In some countries, strong encryption has been banned or the keys have to be escrowed for government officials. With invisibility readily available to anyone with moderate programming skills, it is obvious that any such measures are ineffective. Restrictions on encryption cannot stop criminals from using it, but only hurt the law-abiding businesses and individuals who would greatly benefit from mass application of cryptographic techniques.

References

1. Tuomas Aura. Invisible communication. In *Proceedings of the HUT Seminar on Network Security '95*, Espoo, Finland, November 1995. Telecommunications Software and Multimedia laboratory, Helsinki University of Technology.
2. Hannes Federrath and Jürgen Thees. Schutz der Vertraulichkeit des Aufenhaltsorts von Mobilfunkteilnehmern. *Datenschutz und Datensicherung*, (6):338–348, June 1995.
3. Maxwell T. Sandford II, Jonathan N. Bradley, and Theodore G. Handel. The data embedding method. In *Proceedings of the SPIE Photonics East Conference*, Philadelphia, September 1995.
4. Paul C. Kocher. Personal communication, October 1995.
5. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, April 1988.
6. Steffen Möller, Andreas Pfitzmann, and Ingo Stierand. Rechnergstützte Steganographie: Wie sie funktioniert und warum folglich jede Reglementierung von Verschlüsselung unsinnig ist. *Dateschutz und Datensicherung*, (6):318–326, June 1994.

Other sources are the steganography mailing list archive at `http://www.thur.de/ulf/stegano/` and documentation and source files of several invisibility programs. Kaisa Nyberg made some helpful comments on the session key generation.