Invisible communication

Tuomas Aura
Helsinki University of Technology
Digital Systems Laboratory
Email: Tuomas.Aura@hut.fi

November 6, 1995

Abstract

An overview of practical mass application invisibility in digital communication is given. Existing terminology is clarified and some new concepts are defined. A new algorithm for hiding bit selection in digital images is proposed and an experimental implementation of the algorithm is described. Implications of the availability of invisible communication are discussed.

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 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 are based on principles not unlike those of the encryption algorithms: The secret messages are hidden with a secret key, and only the holder of the correct key will be able to tell if there actually is a secret message or not. If the enemy somehow knows about some secret messages, it must not be able to prove their presence to a third party or to discover any more messages in other data, as long as the key is kept secret.

Nor should the enemy have any advantage of knowing or being able to choose the contents of some secret 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 algorithms for bulk data, photographs and audio and video signals that are routinely transfered in large quantities.

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

2 Hiding data in digital communication

Practical invisibility algorithms are based on replacing a noise component of a digital message with an encrypted (pseudorandom) secret message. We call the noisy message a *wrapper message* and the bits carrying the noise *wrapper bits*. An encrypted secret message with perhaps some encrypted header info are *secret bits*. The wrapper bits actually substituted with the secret bits will be called *hiding bits*. The wrapper data must be sufficiently noisy in order for a slight increase in its randomness not ot be noticeable. We call this kind of a system *substitutive steganography*. The wrapper bits are typically the least significant bits of some inexact values. These can be results of a series of measurements. A digitized image or sound track is, in fact, a huge array of measurement results, all with a significant random error component.

The least significant bits can, although they are noise from a measurements accuracy viewpoint, have some special statistical characteristics. It has been often suggested that the encoding of the secret message should mimic the characteristics of the wrapper bits, a goal not easily achieved. One possibility is to generate a large number of alternative wrapper messages all in the same way and to choose one that happens to have the secret code in it. This is called *selective steganography*. 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 happens to produce the particular number as a digest. This image has exactly the statistical properties that any other digitized image has. 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 wrapper 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 steganography*. The strategy has many drawbacks. First of all, it is problematic to combine with strong encryption. Secondly, 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 wrapper 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 a replacement. Furthermore, the patterns created by the model may reveal the secret message, instead of increasing security. If the adversary knows our model, he can with little investment find flaws in it. The model of the noise is a part of the hiding algorithm. Keeping the algorithm secret is in violation of good cryptographic practices, for the reason that the algorithm 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 procedure. Select a class of sufficiently noisy messages, wrapper messages, and identify the bits that carry the noise, wrapper bits. Then, approximate how large portion of the wrapper bits can be replaced with pseudorandom data without significantly changing the characteristics of the wrapper.

For instance, if the wrapper message is a digitized photograph, the wrapper 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 to play safe and alter only ever 100th pixel of the image. In that case, a one megabyte uncompressed image can hide about one kilobyte of secret data. Secure invisibility inevitably depends on wrappers being at disposal in quantities.

The secret message must 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 wrapper message will only slightly increase the noise level of the message. On the other hand, inserting the bits of plaintext into the wrapper bits could drastically change the character of the wrapper.

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 message bits. If these bits in a suspect message are completely random looking, as the products of strong 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 Wrappers and hiding bit selection

The wrapper messages can be divided in two types. The wrapper can be a continuous data stream, like a digital telephone connection, or a file, like a bitmap image. We call the former a *stream wrapper* and the latter a *random access wrapper*, for reasons soon to be explained.

In a stream wrapper, it may not be possible to tell in advance when the wrapper message begins, when it ends, or how long it will be. The wrapper may be continuous or very long, so that that several secret messages need to be hidden in the same wrapper message. Furthermore, one may not know in advance what the next bits of the wrapper will be but has to insert the secret message into the stream as it is transmitted, before the next bits of the stream have been produced. The hiding bits have to be selected with a keystream generator that gives the distance between consecutive bits in the stream. [4] We call this *random interval method* of hiding bit selection.

In a continuous data stream the biggest difficulty for the receiver is to tell when a hidden message begins. There can be no visible (audible) synchronization sequence. 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 wrapper stream to be long enough for the entire secret message. With stream wrappers, it is also difficult to evenly spread the hiding bits all over the wrapper.

Invisibility in stream wrappers has a close relation to spread spectrum techniques. Spread spectrum communication is used mostly because of its resistance to interference and jamming. The same techniques can be used to hide that there is any communication at all or the location where it originates from. [3]

Fixed length files as wrapper messages do not have the same disadvantages as stream wrappers. 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 wrapper 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 wrapper, the header has to be sent first. The main drawback of random access wrappers is that the wrapper 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 wrapper noise above, they may still be helpful in detecting abnormally bad wrappers. Since a random access wrapper is known in full before it is chosen for data hiding, the wrapper can be selected to fit for hiding. For instance, in a series of digitized images well fit for hiding, there might be some that are not so good, completely black etc. These should be recognized and discarded. If wrappers 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 function in random access wrappers. 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 wrapper. 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. The secret message will normally 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 wrapper. 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 it 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 wrapper message. There are, however, other selection functions for random access wrappers 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 Section 4.

An interesting alternative to simple replacement of pseudorandomly selected wrapper bits with secret bits is to alter the parity of bit sequences. In a stream wrapper, the length of the next bit sequence is determined with a keystream generator, and in random access wrapper, 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 altered bit can be chosen to cause minimum deviation from the original statistical characteristics of the wrapper. This approach has the same disadvantage as the constructive steganographic 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 a 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 wrappers

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 wrapper bits. Let N be the number of wrapper 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 wrapper bits $P_0^N(0), P_0^N(1), \ldots, P_0^N(n)$. The permutation function must be pseudorandom, i.e. it has to select bits in a apparently random order. Then, the hiding bits will be evenly spread all over the wrapper bits. No padding of the secret message with random data or other trickery is needed in order 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 wrapper bits were chosen for hiding bits.

A secure pseudorandom permutation generator can be efficiently built from a *pseudorandom function generator.* [1] A pseudorandom function generator is like a pseudorandom permutation generator in that it produces a different pseudorandom function for each secret key value, but the function does not need to be a permutation. A such 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.

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

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

The pseudorandom permutation generator of Luby and Rackoff [1] is constructed as follows. $a \oplus b$ is 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, L and R, of length l and the key K into four parts K_1, K_2, K_3 and K_4. Compute

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

For every key value K, the algorithm gives a pseudorandom permutation of $\{0, \dots, 2^{2l} - 1\}$. Luby and Rackoff 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 L be the l first bits of i and R 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 wrapper bits is N, we need a permutation P_K^N of $\{0, \dots, N - 1\}$. Our advantage here is that we can restrict ourselves to giving the arguments to P_K^N in the order $0, 1, 2, \dots$.

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 basically a block cipher with arbitrary block size, suggested to the author by Paul C. Kocher. The number of wrapper bits has to be a composite number with two factors of almost equal size, that is, $N = x * y$ for some x and y. To get the index of the ith , $i \in \{0, \ldots, N-1\}$, hiding bit, compute

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

The two first rounds of the algorithm are needed for even spreading of hiding bits amongst the wrapper bits. 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.

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. Therefore, this procedure for selecting the hiding bits is best in accord with digitized photographs as wrappers. In other file types, such as audio files, the number of wrapper 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.

Example. Assume that we have a grayscale image of size 800×600, an encrypted secret message of 1 kilobytes and a key $123,456,678$. We want to hide the secret message to the least significant bits of the image. $x = 800$, $y = 600$, and the number of wrapper 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 found 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.)

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

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

In addition to a reasonable noise level, the algorithm makes some assumptions about the wrappers. *The same wrapper message must never be used again. The original wrapper 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 wrapper message is never saved anywhere, but the secret message is inserted immediately after the wrapper has been produced.

Another question arises when the algorithm is applied in practice. If the secret key and wrapper 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 wrapper message. Let K_u be the secret key of the user and f' a random function on the wrapper messages that does not change when the hiding bits are altered. Then, take the key K to be

$$K = K_u \oplus f'(wrapper).$$

The function f' has to depend so heavily on the wrapper that two different wrappers 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 wrapper bits are the least significant bits, the function f' can be a message digest of the image after all least significant bits are set to zero. It is extremely unlikely that there will ever be two digitized images only differing in the least significant bits. By making the key K a function of the wrapper, the same user key K_u can be used for as many wrappers as needed. The same key cannot, of course, used again for the same wrapper, since the wrappers must never be reused.

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

Having described a cryptographic algorithm, we should say a few words about its reliability. Firstly, the strength of an invisibility algorithm will always depend heavily on the properties of the wrapper. If we take the wrapper bits to have random error component and keep the hiding bits to wrapper bits ratio low, the security of the algorithm presented depends on the random function f_K being computationally secure. We are rather safe here since encrypting the secret message before insertion 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 wrapper. It is not 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 Section 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 any other file format or color images.

The least significant bits were chosen as the wrapper bits. The secret message is encrypted with the IDEA cipher and f_K is constructed from SHS. First, an SHS message digest is computed on the 7 most significant bits of the image. The key K is then obtained by xoring the digest with a user key K_u. The first 16 bytes of K are used for encryption key and the entire 20 bytes for the pseudorandom permutation. K_1, K_2 and K_3 are all equal to 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 wrapper 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 call to a PRNG. SHS may not be an optimal choice for the random function. Nevertheless, the implementation of the algorithm proved to be efficient enough for interactive use in hiding data to persistent 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. The assumpion that statistical tests cannot spot the hidden data remains to be tested.

6 Conclusions and discussion of the implications of invisibility

So far, we have given an overview of practical mass application invisibility. Existing terminology has been clarified and some new concepts defined. A new method for hiding bit selection in random access wrappers was proposed and an experimental implementation of the algorithm was described. We still need to make some conclusions on how the described technology can be applied and what kind of effect it has on organizations and society.

The availability to effective data hiding has many practical implications. Any type of data can be hidden and transfered invisibly where there is a sufficiently large storage or flow of noisy data. The most common types of data transfered in volumes, digital audio and video signals and digital image files, are exactly the ones that are 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 to lower security data flows presents a new challenge to the military information infrastructure. These organizations must, nevertheless, take full advantage of new information technology. The danger of information leaks is one of the main reasons why not all main stream civil technology is directly applicable in military and other high security environments.

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] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*,

17(2):373–386, April 1988.

[2] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 1994.

[3] von Hannes Federrath and Jürgen Thees. Schutz der vertraulichkeit des aufenhaltsorts von mobilfunkteilnehmern. *Datenschutz und Datensicherung*, (6):338–348, June 1995.

[4] von Steffen Möller, Andreas Pfitzmann, and Ingo Stierand. Rechnergstützte steganographie: Wie sie funktioniert und warun folglich jede reglementierung von verschlüsselung unsinnig ist. *Dateschutz und Datensicherung*, (6):318–326, June 1994.

Other sources include the steganography mailing list archive at `http://www.thur.de/ulf/stegano/` and documentation and source files from several freeware and shareware invisibility programs.