Steganographic Key Leakage Through Payload Metadata

Tomáš Pevný
Agent Technology Group
CTU in Prague
Prague 16627, Czech Republic
pevnak@gmail.com

Andrew D. Ker
Department of Computer Science
University of Oxford
Oxford OX1 3QD, UK
adk@cs.ox.ac.uk

ABSTRACT

The only steganalysis attack which can provide absolute certainty about the presence of payload is one which finds the embedding key. In this paper we consider refined versions of the key exhaustion attack exploiting metadata such as message length or decoding matrix size, which must be stored along with the payload. We show simple errors of implementation lead to leakage of key information and powerful inference attacks; furthermore, complete absence of information leakage seems difficult to avoid. This topic has been somewhat neglected in the literature for the last ten years, but must be considered in real-world implementations.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Information hiding*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*Information theory*

Keywords

Steganographic Security; Key Leakage; Brute-force Attack; Bayesian Inference

1. INTRODUCTION

A steganographic object does not only contain its covert payload: it will also contain a small amount of metadata about the payload. Typically this will be the length of the payload and/or some parameters that tell the recipient how to decode it: otherwise, the embedder is forced to use a fixed coding method which cannot exploit better embedding efficiency available for lower payloads, or adapt the embedding method to the cover. Does this metadata leak information about the embedding key? In this paper we consider details of implementation which are typically elided in the literature, and this paper is aimed at real-world practitioners [12]. We show how easy it is to make mistakes, with examples of a historic steganography scheme that does make obvious mistakes, and hypothetical implementations where the mistakes are more subtle. Indeed, it seems surprisingly difficult to avoid information leakage altogether.

In this paper we do not consider *statistical* steganalysis: a highly-refined discipline (particularly for still, grayscale, images) which can detect even small payloads with surprising accuracy. But the level of reliability is never certainty and never enough, for example, to convince a criminal court. The authors believe that steganalysis with false positive rates below, say, 10^{-9} is an important area for research, but no current statistical steganalysis method meets such a standard. (Even if one did, we would have no way to prove it.)

Instead, we turn to the older *brute force* attack, which tries all possible embedding keys in turn. Such attacks are certainly plausible, we argue more so than in traditional cryptography, and are the only methods by which one could truly pronounce guilt with certainty. The brute force attack is not, of course, new, but we suggest that the embedding metadata provides a way to speed up and refine exhaustion attacks, which can be mitigated (but perhaps not prevented) by careful implementation.

1.1 Typical Extraction Procedure

Consider a typical steganographic *extraction* process:

(i) The *embedding key* is derived from a password, using a key derivation function (KDF): $k = \mathrm{KDF}(p)$.

(ii) The embedding key locates and decodes a small block of metadata, the *decoding parameters*.

(iii) The decoding parameters and the embedding key are used to extract the *payload*.

In step (i) we distinguish a *password* p, the secret shared between the sender and receiver which we expect to have only a moderate amount of entropy (e.g. one of a few million possible words and modified words), from the quasi-cryptographic key k that parameterizes the extraction. In contemporary security systems, the relatively small entropy of passwords is often a weakness because it permits a brute-force attack [8]. Ideally, the sender and receiver would share a 64- or 128-bit secret key, but the practicalities of covert communication make this unlikely: if they can share such keys at will, they may have no need for steganography at all. Perhaps a protocol could be used to exchange embedding keys, but at the moment public-key steganography protocols have enormous computational costs [12].

By *(de)coding parameters* we mean the information necessary to en/decode the payload: it might simply be the

IH&MMSec'14, June 11–13, 2014, Salzburg, Austria.

http://dx.doi.org/10.1145/2600918.2600921.

payload length; in syndrome coding methods it will determine the parity-check matrix size [3] (the contents of the matrix will probably derive from the embedding key); in adaptive embedding methods it may determine the parameters of the trellis [4]. Most research literature assumes that decoding parameters are somehow already known to the receiver, e.g. YASS [22], or provide only simulators that do not encode a real message, e.g. UNIWARD [9] and HUGO [17].

In the real world, decoding parameters must be embedded in step (ii), using a fixed code, and stored securely using the embedding key (for example as a cryptographic key), otherwise the system does not survive Kerckhoffs' Principle. For example, OutGuess [19] has a header containing a 16-bit field storing payload length (in bytes); F5 [23] uses 8 bits to store the dimension of the Hamming code used in matrix embedding [3] and 23 bits to store payload length (in bytes); JpHide&Seek [15] stores the message length (in bytes) in 24 bits encrypted by the same password as the payload.

The distinction between steps (ii) and (iii) is important. Having step (iii) depend on the output of (ii) allows the embedder to vary the encoding parameters according to the payload size or cover characteristics. For example, using random linear codes, smaller relative payloads can be embedded with higher embedding efficiency by choosing codes with higher dimension [7]. Or in the paradigm of distortion minimization using syndrome trellis codes, the width of the generator matrix must be adjusted according to the inverse payload size, to approach the rate-distortion bound [9]. Without transmitting decoding parameters, the encoder would be unable to adjust their embedding to make the best use of the particular cover and payload combination.

In this paper, in sections 2–4, we examine increasingly refined versions of exhaustion attacks attempting to use step (ii) to determine the embedding key. We will assume that the embedder has created multiple stego images using the same key (with different payloads); this seems highly realistic in a covert communication scenario.

1.2 Prior Art on Brute Force

Recovering the embedding key seems to have received relatively little attention in the literature. Probably the first work was [20], which tried to detect the use of steganography on the internet. Downloaded images were first scrutinised by statistical steganalysis based on the χ^2-test, and if deemed suspicious the hidden content was further verified by guessing the embedding key from a dictionary. A key was deemed possible if it extracted a header compatible with known embedding algorithms. The approach resembles the Intersection Attack we present in section 3, but it only uses one stego image; contrary to this prior art, we can often determine the embedding key uniquely.

Ref. [6] used a statistical approach, generating the embedding path from each stego key. The attack, targeting OutGuess and F5, assumed that pixels/coefficients carrying the payload have different statistical properties from those that do not. A χ^2-test is used to measure this difference. Such an attack will not work directly with modern content-adaptive embedding schemes, which use all pixels with some probability, but could perhaps be adapted to them: it was demonstrated in [21] that knowledge of the embedding path makes them more vulnerable to modified weighted stego-image steganalysis [13] in the limited case of LSB Replacement.

2. RECOGNISABLE PLAINTEXT

An important concept in cryptography is the *recognisable plaintext*: the assumption that an attacker can tell when they have found the correct decryption key, by verifying the decoded plaintext in some way. Ciphertext-only attacks (as opposed to those with a known plaintext-ciphertext pair) are impossible without this assumption. The same assumption is plausible in steganography, for much the same reasons: if the payload has meaning, it most likely also has redundancy or structure, for which an attacker can look. Given this assumption, the simplest attack on steganography is:

The Exhaustion Attack. The attacker tests every possible key k, or alternatively every possible password p, decoding the plaintext and checking for known structure.

2.1 Countermeasures

Recognisable plaintext is a genuine problem for steganography, particularly if it is more difficult to convey good passwords in a covert communication setting than in traditional cryptography. Compressing redundancy, or encrypting plaintext with the embedding key, provides no additional security because a Kerckhoffs' attacker can reverse it.

One countermeasure is to ensure that the keyspace is at least 64 bits: in modern times, there is no excuse for using a 32-bit embedding key, because 32-bit keyspaces can be exhausted. Unfortunately this still allows an attacker possessing a suitably generous dictionary to attack the passwords instead. Increases in computational power were supposed to benefit cryptographers (who can increase key sizes with polynomially more work) more than cryptanalysts (who are forced into near-exponentially more work), but humans' ability to remember passwords has not kept pace [8].

We could borrow a technique from entity authentication mechanisms, and ensure that the KDF is slow to compute, for example by iterating a cryptographic hash at least 10^5 times [10]. This slows down the embedder, extractor, and attacker equally. However, slow KDFs can be defeated by attackers with plenty of parallel computational resources, and the key dictionary can be derived offline unless the keys are also salted [16] (and with what? – perhaps a sequence number, but this is vulnerable to prediction and fragile under desynchronization). Similarly, but less sensibly, we could try to ensure that the decoding process itself is very slow.

The best that we can suggest is a separate password encrypting the payload itself; a similar approach is assumed in [5]. Although doubling the password requirement, it squares the brute-force work for an attacker, who must exhaust for each embedding key and then again to recognise plaintexts.

3. IMPOSSIBLE PARAMETERS

Even a separate encryption password or a completely unrecognisable plaintext may not protect against exhaustion. We make the same observation as Provos [20]: that most embedding keys will imply decoding parameters which are not possible, and these can be discarded. With multiple images embedded under the same key, we can use:

The Intersection Attack. The attacker maintains a list of all possible keys k, or passwords p. For each image received, they remove all keys which produce impossible decoding parameters.

Consider the OutGuess [19] algorithm: it uses 16 bits to store the length of the payload. Its capacity can be esti-

mated as half the number of non-zero, non-one, non-DC, discrete cosine transform (DCT) coefficients. It is easy to compare the message length extracted from the header with the capacity (which is not changed by embedding) and, if exceeded, the key can be discarded. For incorrect keys, we expect to read uniformly distributed lengths of less than 2^{16} bytes, most of which will be too long for the stego image.

For F5 the same attack is slightly more difficult, because the embedding process reduces the capacity. However, the capacity can never be less than the number of nonzero AC coefficients, since if F5 changes a coefficient to zero it re-embeds using another; thus we obtain an upper bound on the capacity of the cover from the stego object. Furthermore, its implementation has even key higher leakage, because it also stores the parameters of the Hamming code. As well as discarding impossible message lengths, we can also verify that the Hamming coding is compatible with the message length and capacity. A similar mechanism can be expected for embedding algorithms using syndrome trellis codes.

This attack can be extremely powerful: when proportion α of all possible keys give rise to possible decoding parameters, after n images there will be on average only α^n possible keys left. Put another way, we expect the entropy of the keyspace to decrease linearly with n.

3.1 Experiment

The Intersection Attack is demonstrated using set of approximately 9000 images, where every image contains a payload of random length (uniformly chosen up to capacity) embedded using OutGuess with the same password 'Neil'. All images were JPEG images downloaded from one user of the photo-sharing site Flickr. We simulate a steganalyst with a list of more than 2 million passwords downloaded from [1] (the searchable keyspace therefore has an entropy of approximate 21 bits). The experiment was repeated each time with up to 50 images embedded with the same key, and after each image the steganalyst removes passwords implying a message length not compatible with capacity. The average- and worst-case (over 1000 repetitions) entropy of the remaining keyspace (the binary logarithm of the number of possible passwords) at each stage is shown in Figure 1. The results demonstrate key leakage, but not as much as we had expected: this is because the OutGuess implementation does not decode all parameter blocks to uniformly random payloads: some keys imply consistently small payloads, regardless of image, and cannot be eliminated. Thus 50 images are not sufficient to determine the password uniquely.

3.2 Countermeasures

Implementations which allow the Intersection Attack are flawed, but it can be tricky to remove the flaw entirely. The first step is to use the correct length of field: for example, if payload length can never be more than (say) 2^{23} bits (1 bit per pixel of an 8 megapixel image) then only 23 bits should be used to store payload length; if they must be padded to 32 bits, the other nine bits should be random and discarded by the extractor.

However, this still leaves plenty of impossible parameters for smaller images, and the exponential power of the Intersection Attack can exploit even small gaps. We propose a simple padding scheme that makes all parameter blocks possible: if the number to be stored (payload size, matrix parameters, etc.) lies in the range $0 \ldots (N-1)$, encrypt using k a uniformly-chosen random integer with the correct value (mod N).

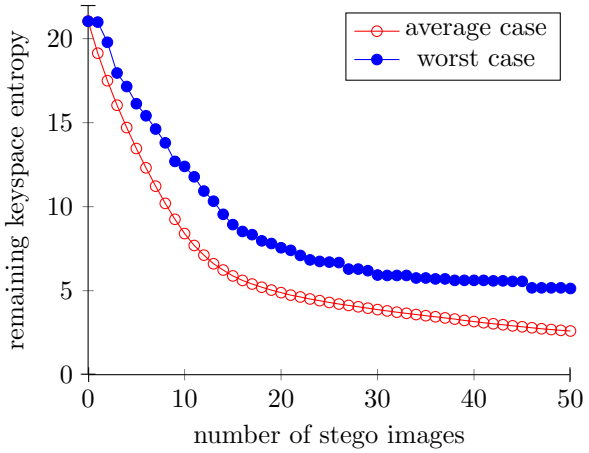

Figure 1: Entropy of keyspace after the Intersection Attack.

It is not always easy to choose such an N. If we are storing a payload size, ideally N should be the maximum capacity of the cover. But N has to be recoverable by the receiver, so that they can perform the same modular reduction. Unfortunately, for many embedding schemes the capacity is lower after embedding (e.g. F5 as we previously stated [5]). The sender may be forced to predict a *lower* bound based on the capacity of the *stego object* and use that N instead, which in turns limits their own capacity meaning that maximum-size payloads cannot be embedded. However, maximum-size payloads are more vulnerable to statistical steganalysis, so perhaps this does not matter. Analogous problems may arise in storing coding parameters such as matrix or trellis sizes.

Even random padding that makes all decoding parameters possible may leak information, if the parameters are not equally likely. This is the subject of the following section.

4. PAYLOAD SIZE ESTIMATION

We try to use some techniques from statistical steganalysis to strengthen the brute-force attack. Most embedding methods are vulnerable to *quantitative steganalysis*, which attempts to estimate the payload length (or its proxy, number of embedding changes) from a stego image: a form of regression. Most steganalysis classifiers can be converted to regressors [18]. Even though the estimates are subject to error, they make certain keys more likely than others.

For now, we will assume that the decoding parameters simply specify the payload length; we will revisit this in subsection 4.3. We assume that the regressor outputs an estimate of the payload length y and, by empirical simulation, we can obtain an estimate of the distribution of the estimate conditional on the true payload length x (typically by fitting the estimation error to a distribution, preferably one with heavy tails [2]). We denote this distribution $\mathrm{P}(y|x)$. Each possible embedding key k decodes a parameter block specifying that the payload length is $x(k)$. Now we can perform:

Bayesian Key Inference. If $p(k)$ represents the prior probability that the key is k, then the posterior $p(k|y)$ after one observation of a stego image with estimated payload y

is given by

$$p(k|y) = \frac{\mathrm{P}\big(y|x(k)\big)p(k)}{\sum_{k'} \mathrm{P}\big(y|x(k')\big)p(k')} \propto \mathrm{P}\big(y|x(k)\big)p(k).$$

We can ignore the denominator, because it is constant in k, and rescale the distribution (if necessary) at the end. Iterating for multiple observations $y_1, \ldots, y_n$, the unscaled posterior can therefore be written

$$\log p(k|y_1, \ldots, y_n) = \log p(k) + \sum_{i=1}^{n} \log \mathrm{P}\big(y_i|x(k)\big).$$

Regardless of the prior, this means that all keys can be scored by their log-likelihood.

We briefly analyse the performance of this algorithm (proof omitted). Let S denote the score for the true key k, and S' the score of an incorrect key k'. Then

$$\mathrm{E}[S] = n \int \mathrm{P}\big(y|x(k)\big) \log \mathrm{P}\big(y|x(k)\big)\, \mathrm{d}y,$$

and if $\tilde{\mathrm{P}}(y)$ represents the unconditional output of the regressor, the mixture over all true payloads (which may very well be uniform random) then

$$\mathrm{E}[S'] = n \int \mathrm{P}\big(y|x(k)\big) \log \tilde{\mathrm{P}}(y)\, \mathrm{d}y.$$

Thus the separation of the scores of true and false keys is linear in n, governed by the Kullback-Leibler divergence

$$\mathrm{D}_{KL}\Big(\mathrm{P}\big(y|x(k)\big) \,\Big\|\, \tilde{\mathrm{P}}(y)\Big), \tag{1}$$

which is larger when the regressor is more accurate, and (assuming independent scores between images) its variance is $O(n)$. This implies that the probability of the correct key will soon dominate the others.

4.1 Experiment

Bayesian Key Inference was evaluated using the same image set as in subsection 3.1. In each repetition we randomly picked 10 images (embedding using the same key) over which to perform Bayesian inference. The remaining 8990 images were split into two sets: 66% of them were used to train a linear quantitative steganalyzer by ordinary least-square regression on Cartesian-calibrated PF-584 features [14] the remaining 34% were used to estimate the quantitative steganalyzer's error, which was fitted to a Gaussian distribution to create $\mathrm{P}(y|x)$.

Figure 2 shows the entropy of the keyspace (the posterior key distribution) after applying the attack. We observe significantly faster convergence than for the Intersection Attack; at most eight images was sufficient to determine the correct password uniquely. Although the Gaussian model for regressor error is probably optimistic, it does not seem to hurt the accuracy of inference unless we care about the precise posterior probabilities; on the other hand, it could easily be swapped for a heavier-tailed distribution.

4.2 Countermeasures

To our knowledge, such an attack has not previously been described. It exploits an implementation rather than the method for putting the bits into the cover, and such topics have received relatively little attention in the literature.

It seems difficult to prevent this attack in any situation where the number of possible keys or passwords is exhaustible

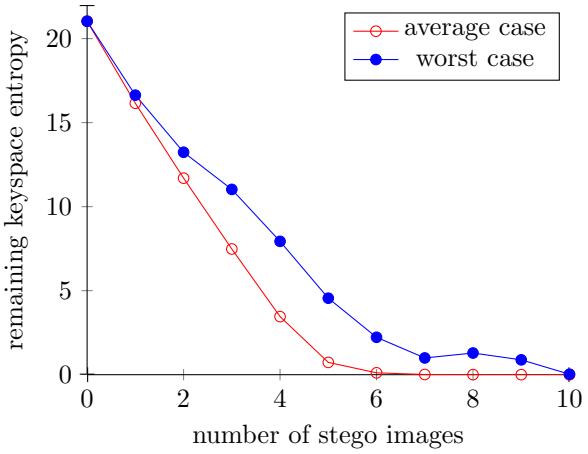

Figure 2: Entropy of keyspace after Bayesian Key Inference using message length metadata.

and a quantitative steganalyzer exists, but we suggest that some countermeasures may be possible. The exponential power of Bayesian Key Inference relies on the following observation: if $x_i(k)$ the payload size implied by key k in image i, and if k' is an incorrect guess for the true embedding key k then

$$|x_i(k') - x_i(k)| \text{ is independent of } |x_j(k') - x_j(k)| \tag{2}$$

for $i \neq j$. In other words, a key which is a 'near miss' for image i is unlikely also to be a 'near miss' for other images j, so the likelihood of incorrect keys diminishes very quickly.

If $b(k)$ specifies some sequence of bits from the stego image, depending on the key k, we have implictly assumed that $x(k) = D_k(b(k))$ or $x(k) = D_k(b(k)) \pmod N$, where $D_k(-)$ is some cryptographic decryption. It is precisely this cryptographic property that gives (2). Is it possible to mitigate the effect, introducing strong covariance between $|x_i(k') - x_i(k)|$ and $|x_j(k') - x_j(k)|$? One simple proposal to avoid (2) is

$$x(k) = b + k \pmod N \tag{3}$$

for some block b at fixed location.

We can measure the amount of leaked information using a similar experiment simulating an OutGuess-like embedding. The message length is stored in 16 bits, so the effective key space for one-time pad is $\mathcal{K} = \{0, 1, \ldots, 2^{16} - 1\}$. Background bits b are read from some fixed positions. The inference is done only against $\mathcal{K}$, and it is assumed that the capacity is $\frac{1}{16}M$ bytes, where M is the number of useable coefficients (recall that OutGuess reserves approximately 50% of them for statistical restoration). The number of images available for key-breaking was increased to 50, since we expect the inference to be weaker.

The results are summarised in Figure 3. Comparing to the previous case, the inference is much weaker: the keyspace was not very large to begin with, but only approximately half of its entropy is lost.

It is counter-intuitive to use a weak encoding such as (3), and indeed it does leak information. Although the payload size is secured by the one-time pad of modular addition, multiple images with similar payloads and the same embedding key will all have similar values stored in the block: this allows an attacker an easy target for statistical attacks. It

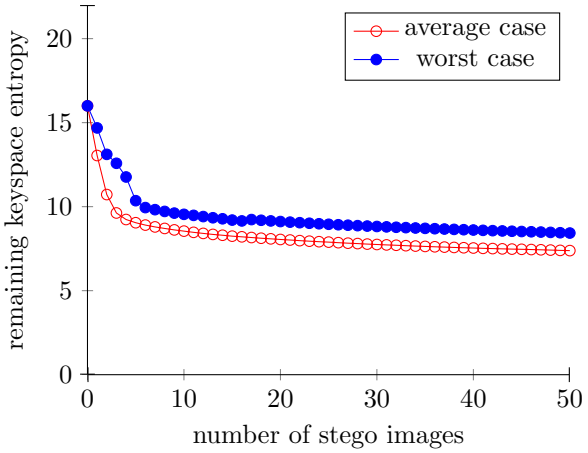

Figure 3: Entropy of keyspace after Bayesian Key Inference, where message length is protected by a one-time pad.

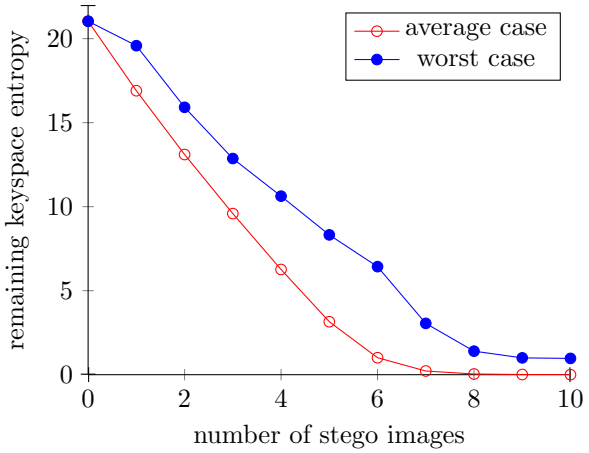

Figure 4: Entropy of keyspace after Bayesian Key Inference using the dimension of the code.

would appear that (2) forces such a weakness, because then

$$x_i(k) \approx x_j(k) \text{ implies } x_i(k') \approx x_j(k')$$

for every k'. We leave open the question as to whether there is a clever solution which prevents exponential key leakage from multiple stego images but does not introduce statistical dependence between those images.

4.3 Extension

Having read the preceding sections, any sensible steganographer should learn the lesson: do not embed the payload size at all. There is no absolute need for it, since the end-of-payload can be marked by an escape code, and the rest discarded after extraction. However, the same inference technique works even if the decoding parameters do not specify the payload precisely, as long as they imply some range of possible payloads. This is usually the case.

For example, take F5, which uses a Hamming syndrome code to increase embedding efficiency. For any integer m, it embeds m bits into blocks of size $2^m - 1$ while making at most one change. Thus larger m gives higher embedding efficiency but lower payload rates, and the embedder chooses the largest m such that the desired payload will fit. The only embedding parameter that must be communicated to the receiver is m, which we shall assume is perfectly padded so that the Intersection Attack is also impossible.

Even though the parameter block does not store the payload length, Bayesian inference is still possible. Each key k implies a value of m, which in turn implies a bound on the number of embedding changes in the image as follows. If the image has M usable locations then there can be at most $M/(2^m - 1)$ blocks used, so at most $M/(2^m - 1)$ embedding changes. And there cannot be much fewer than $M/(2^{m+1} - 1)$ changes, otherwise the payload size is (very likely) small enough that a larger value of m would have been used by the embedder. Given an estimator for the number of changes (e.g. [18]), we can assume a uniformly random number of changes x between these two limits, and still compute

$$P(y|m) = \sum_x P(y|x)P(x|m).$$

The conditional distribution $P(y|m)$ is flatter than the plain output of a regressor $P(y|x)$, so the divergence (1) is smaller, but inference can still happen and should still converge exponentially fast (reduce entropy linearly) if different keys imply independent decodings of m.

4.4 Experiment

We simulated the above scenario, using the estimator of the message length from subsection 4.1. Assuming Hamming codes used in F5, the probability of code parameter m, when the estimated length of the message is $\hat{y}$ and number of non-zero DCT coefficients is M is

$$P(y|m) = \int_{\frac{M(m-1)}{2^m-1}}^{\frac{Mm}{2^m-1}} \mathcal{N}(y \,|\, \hat{y}, \sigma^2)\, \mathrm{d}y, \tag{4}$$

where σ^2 is the estimated variance of the estimator. For every key, we assumed the most efficient code parameter

$$m = \max \left\{ m | y \leq \frac{Mm}{2^m - 1}, m \in 1, 2, \ldots, 8 \right\}. \tag{5}$$

Figure 4 shows the results, which are surprisingly powerful considering that the payload is only determined very approximately by the code parameters. At most 10 images were sufficient to determine the correct password uniquely.

5. CONCLUSION

This paper aims to bring steganography a little further from the laboratory into the real world [12], by focusing on the somewhat-neglected topic of key exhaustion, an attack that is likely to be practical. It is easy to dismiss as an 'implementation issue', but we have shown that it is difficult to include embedding parameters that are not subject to some kind of inference. Other key attacks [21, 6] could potentially be combined with the ideas presented here, but at present they are already sufficiently powerful for most purposes, reducing the potential keyspace in our experiments from 2 million down to a few hundred, or even down to one, given a few images embedded with the same key.

The entire attack could be avoided if the keyspace were 64 bits, and not derived from any password, because it would not be exhaustible. Furthermore, no two covers should ever

be embedded using the same key [11]. But the real world may preclude these good practices. Certainly, steganographers should avoid including the payload length in any kind of header, if possible. It is telling that most (perhaps all) widely-available steganography software makes this mistake. Any embedding parameters must be properly padded so that all decoded parameters are *possible*, but we have shown that this does not make them equally likely. Storing embedding parameters cryptographically is the root cause of the exponential power of inference attacks, because it makes incorrect key guesses independent over different images. Although we have suggested an alternative, it unlocks statistical attacks.

The alternative is to embed no metadata at all, use an escape code as an end-of-payload marker, and accept that embedding parameters must be fixed (or derived solely from the stego image). This reduces vulnerability to exhaustion, but probably makes statistical steganalysis more effective because of the suboptimal codes. Indeed, we conjecture that the embedder may be forced to choose between more susceptibility to exhaustion attacks or more susceptibility to statistical steganalysis.

Acknowledgments

This work was supported by European Office of Aerospace Research and Development under the research grant numbers FA8655-11-3035 and FA8655-13-1-3020. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation there on. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of EOARD or the U.S. Government.

6. REFERENCES

[1] Free list of 2 million popular ASCII passwords. `http://dazzlepod.com/site_media/txt/passwords.txt`.

[2] R. Böhme and A. D. Ker. A two-factor error model for quantitative steganalysis. In *Security, Steganography, and Watermarking of of Multimedia Contents VIII*, volume 6072 of *Proc. SPIE*, pages 59–74. SPIE, 2006.

[3] R. Crandall. Some notes on steganography. *Steganography Mailing List*, 1998. available from `http://os.inf.tu-dresden.de/~westfeld/crandall.pdf`.

[4] T. Filler, J. Judas, and J. Fridrich. Minimizing additive distortion in steganography using syndrome-trellis codes. *IEEE Trans. Information Forensics and Security*, 6(3):920–935, 2011.

[5] J. Fridrich, M. Goljan, and D. Hogea. Steganalysis of JPEG images: Breaking the F5 algorithm. In *Proc. 5th Information Hiding Workshop*, volume 2578 of *LNCS*, pages 310–323. Springer, 2002.

[6] J. Fridrich, M. Goljan, and D. Soukal. Searching for the stego key. In *Security, Steganography, and Watermarking of Multimedia Contents VI*, volume 5306 of *Proc. SPIE*, pages 70–82, 2004.

[7] J. Fridrich, M. Goljan, and D. Soukal. Wet paper codes with improved embedding efficiency. *IEEE Trans. Information Forensics and Security*, 1(1):102–110, 2006.

[8] D. Goodin. Why passwords have never been weaker and crackers have never been stronger. `http://arstechnica.com/security/2012/08/passwords-under-assault/`, 2012.

[9] V. Holub and J. Fridrich. Digital image steganography using universal distortion. In *Proc. 1st ACM Workshop on Information Hiding and Multimedia Security*, pages 59–68. ACM, 2013.

[10] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. Request for Comments 2898. Internet Engineering Task Force, 2000. `http://www.ietf.org/rfc/rfc2898.txt`.

[11] A. D. Ker. Locating steganographic payload via WS residuals. In *Proc. 10th ACM Workshop on Multimedia and Security*, pages 27–32. ACM, 2008.

[12] A. D. Ker, P. Bas, R. Böhme, R. Cogranne, S. Craver, T. Filler, J. Fridrich, and T. Pevný. Moving steganography and steganalysis from the laboratory into the real world. In *Proc. 1st ACM Workshop on Information Hiding and Multimedia Security*, pages 45–58. ACM, 2013.

[13] A. D. Ker and R. Böhme. Revisiting weighted stego-image steganalysis. In *Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, volume 6819 of *Proc. SPIE*, pages 0501–0517. SPIE, 2008.

[14] J. Kodovský and J. Fridrich. Calibration revisited. In *Proc. 11th ACM Workshop on Multimedia and Security*, pages 63–74. ACM, 2009.

[15] A. Latham. Implementation of the JPHide and JPSeek algorithms ver 0.3 (released August 1999). `http://linux01.gwdg.de/~alatham/stego.html`.

[16] R. Morris and K. Thompson. Password security: A case history. *Communications of the ACM*, 22:594–597, 1979.

[17] T. Pevný, T. Filler, and P. Bas. Using high-dimensional image models to perform highly undetectable steganography. In *Proc. 12th Information Hiding Conference*, volume 6387 of *LNCS*, pages 161–177. Springer, 2010.

[18] T. Pevný, J. Fridrich, and A. D. Ker. From blind to quantitative steganalysis. *IEEE Trans. Information Forensics and Security*, 7(2):445–454, 2012.

[19] N. Provos. Defending against statistical steganalysis. In *Proc. 10th Conference on USENIX Security Symposium - Volume 10*, SSYM, pages 323–335. USENIX Association, 2001.

[20] N. Provos and P. Honeyman. Detecting steganographic content on the internet. Technical Report CITI Technical Report 01-11, University of Michigan, 2001.

[21] P. Schottle, S. Korff, and R. Böhme. Weighted stego-image steganalysis for naive content-adaptive embedding. In *IEEE International Workshop on Information Forensics and Security (WIFS 2012)*, pages 193–198, 2012.

[22] K. Solanki, A. Sarkar, and B. Manjunath. YASS: Yet another steganographic scheme that resists blind steganalysis. In *Proc. 9th Information Hiding Conference*, volume 4567 of *LNCS*, pages 16–31. Springer, 2007.

[23] A. Westfeld. F5 – a steganographic algorithm. In *Proc. 4th Information Hiding Workshop*, volume 2137 of *LNCS*, pages 289–302. Springer, 2001.