Cripto-esteganografia: imagens inocentes podem transportar arquivos secretos

Vitor S. Costa[*] **Vinícius G. Pereira de Sá**
Universidade Federal do Rio de Janeiro
Caixa Postal 68530, CEP 21941-590, Rio de Janeiro, RJ, Brasil
E-mail: v2costa@gmail.com, vigusmao@dcc.ufrj.br

RESUMO

Há muito tempo se discute como armazenar ou transmitir informações confidenciais de forma segura. Essa questão é fundamental nos dias de hoje, sobretudo com a Internet servindo de meio de comunicação para os diversos fins. Recentemente, diversos casos recentes de quebra de privacidade colocaram o tema no centro das atenções. *Esteganografia* [3, 5] consiste no uso de técnicas para se ocultar uma mensagem dentro de outra, que funcionará como veículo. Este trabalho mostra não apenas como podemos guardar uma informação *qualquer* dentro de uma imagem, mas também como podemos deixá-la protegida através de criptografia. Alguns trabalhos correlatos foram publicados [1, 4]. Nosso método, no entanto, é original e de simples implementação, podendo ser utilizado tanto para uso pessoal quanto para a troca de mensagens e arquivos entre duas partes através de uma rede insegura.

O primeiro desafio para se realizar a esteganografia criptografada (ou *cripto-estenografia*) é como esconder qualquer tipo de arquivo dentro de uma imagem, seja ele texto puro, áudio, imagem ou até mesmo vídeo. Evidentemente, qualquer arquivo pode ser convertido em um vetor de bytes. Essa é a propriedade fundamental que permitirá o método. Tendo em mãos, portanto, um vetor de bytes, podemos quebrá-lo ainda mais em um *bit array*, isto é, em um vetor composto unicamente por valores 0 e 1. Não diferenciaremos, a princípio, qual o tipo ou extensão do arquivo secreto; apenas o converteremos para um grande vetor binário. Utilizaremos um padrão qualquer de t bits — por exemplo, "111111", com $t = 6$ — para identificar o fim do arquivo. Vamos chamá-lo de *terminador*. Para evitar que a existência, no conteúdo do arquivo original, de um padrão de bits idêntico ao terminador seja confundida com uma indicação de fim de arquivo, utilizamos a técnica de *bit stuffing*: para cada subsequência idêntica aos $t-1$ bits mais significativos do terminador, insere-se um bit que é o complemento do bit menos significativo do terminador. Isto é, cada ocorrência de "11111" no vetor de bits original se transformará em "111110". Esse último bit, sendo 0', será descartado durante a leitura; sendo '1', será interpretado como o último bit de um terminador, e então estará sinalizada a situação de fim do arquivo. Além disso, para que o arquivo possa ser de fato recuperado, a extensão do arquivo (que indica seu tipo) precisa ser também armazenada e recuperada. Para isso, essa informação é codificada também em binário e colocada no início do vetor de bits, separada do conteúdo do arquivo por um terminador.

Nosso segundo problema é como esconder esse vetor binário em uma imagem de forma que seja impossível recuperar os dados sem uma senha específica. Para isso utilizaremos uma senha privada, acordada entre as partes comunicantes, que servirá de *seed* para um gerador de números pseudo-aleatórios. É sabido que um tal gerador, quando alimentado com determinada *seed*, produzirá na verdade uma sequência infinita (e determinística!) de valores. O gerador de números pseudo-aleatórios será então utilizado para definir, de antemão, um arranjo aleatório de n pixels da imagem, onde n é a quantidade de bits do arquivo a ser escondido. Um tal arranjo pode ser obtido em tempo $O(n)$ por uma execução

[*] bolsista de Iniciação Científica PIBIC/CNPq

parcial do *Fisher-Yates shuffle* [2]. A permutação obtida indicará então a exata sequência dos pixels da imagem que serão utilizados para codificar cada bit do vetor.

A codificação de um bit em um pixel será conseguida pelo controle da paridade de um dos códigos de cor (R, G ou B) daquele bit. Digamos que a componente utilizada seja B, fixada previamente quando da implementação do algoritmo. No caso dessa componente B ser par, ela indicará que o bit codificado naquela coordenada é 0; caso contrário, o bit é 1. Pode ser necessário, portanto, alterar o valor dessa componente de cor, de forma a corrigir sua paridade conforme o bit que se deseja codificar. Isso é feito pela modificação do bit menos significativo daquela componente, o que não acarreta qualquer diferença visual perceptível pelo olho humano numa comparação da imagem modificada com a imagem original. Se quisermos aumentar o espaço disponível para o armazenamento de bits ocultos, é possível utilizar mais de uma componente de cor por pixel. Isso pode ser feito através de uma ordem pré-fixada de componentes (digamos, R, G, B), quando então cada sub-sequência de três bits do *bit array* será codificada em um único pixel, ainda sem qualquer alteração perceptível na imagem.

Para recuperar o arquivo original, seja qual for sua natureza, é indispensável o conhecimento da senha, para que a mesma sequência de pixels seja determinada pelo gerador. O processo é, então, análogo ao da codificação. Geramos os números aleatórios de acordo com a senha (*seed*) pré-acordada, e acessamos em cada pixel a componente de cor pré-combinada. A paridade do valor dessa componente indica o bit ali codificado. Cada bit assim obtido é então, acrescentado a um vetor, ao que se segue uma verificação dos últimos t bits decodificados. Se for detectado um padrão obtido por *bit stuffing*, o último bit é descartado; se for detectado um terminador, encerra-se a decodificação da extensão do arquivo, apaga-se o vetor de bits e passa-se, a partir do próximo bit, à decodificação do arquivo propriamente dito. Quando o próximo terminador for encontrado, encerra-se a decodificação do arquivo.

As principais características do método proposto são:

- é de simples implementação;
- permite o armazenamento de arquivos de qualquer tipo;
- permite o armazenamento de arquivos cujo tamanho, em bytes, seja menor ou igual a $3/8$ do número de pixels da imagem-veículo (para uma imagem de 8 megapixels, seria possível armazenar arquivos de até 3 MB);
- o tempo de codificação/decodificação é linear no tamanho do arquivo a ser armazenado.

Finalmente, é possível esconder diversos arquivos em uma mesma imagem apenas utilizando-se terminadores distintos para diferenciar "fim do arquivo corrente, com outro a seguir" (por exemplo, "000000") e "fim do último arquivo cripto-esteganografado" (por exemplo, "111111").

Palavras-chave: *Esteganografia, Imagem, Criptografia, Segurança*

Referências

[1] M. Chroni, A. Fylakis, S. D. Nikolopoulos (2013). Watermarking Images in the Frequency Domain by Exploiting Self-Inverting Permutations. *Journal of Information Security* **4**: 80–91.

[2] R. Durstenfeld (1964). Random permutation. *Communications of the ACM* **7**(7): 420.

[3] N. F. Johnson, S. Jajodia (1998). Exploring Steganography: Seeing the Unseen. *IEEE Computer Society*, February 1998, 26–34.

[4] A. Patidar, G. Jagnade, L. Madhuri, P. Mehta, R. Seth (2012). Data Security Using Cryptosteganography in Web Application, *Computer Engineering and Intelligent Systems* **3**(4): 74–79.

[5] N. Provos, P.Honeyman (2003). Hide and Seek: an Introduction to Steganography. *IEEE Computer Society*, May/June 2003, 32–44.