CryptoStego - A Novel Approach for Creating Cryptographic Keys and Messages

Damir Omerasevic
PBH Technologies
PrinTec Group of Companies
Sarajevo
Bosnia and Herzegovina
Email: d.omerasevic@printec.ba

Narcis Behlilovic
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo
Bosnia and Herzegovina
Email: nbehlilovic@etf.unsa.ba

Sasa Mrdovic
Faculty of Electrical Engineering
University of Sarajevo
Sarajevo
Bosnia and Herzegovina
Email: sasa.mrdovic@etf.unsa.ba

***Abstract*—This paper proposes a cryptographic key establishment method based on set of images shared by sender and receiver. The method is simple, fast and secure. We call it CryptoStego. Possible key sizes are virtually limitless. Proposed method is implemented in C programming language. The implementation is compared with (A)RC4 stream cipher, by comparing CPU time and memory occupation by both algorithms. The results of comparison are presented. Intel VTune Amplifier XE 2011 was used as measuring tool.**

***Index Terms*—ARC4, RC4, ARCFOUR, CryptoStego, cryptography, key generation, steganography.**

I. Introduction

Modern ciphers that are standardized, like AES or RSA, are well known. They do not have weaknesses that would enable someone to decrypt ciphertext without correct key within reasonable time. They all implement Kerckhoffs's principle [1] that security of the system is in security of secret key. Therefore secret key needs to be safe. There are two possible ways to attack cipher secret key. One is to try all possible values of the key until the correct key is guessed, brute force attack. To prevent this kind of attack key needs to be as long as possible. The other avenue of attack is to try to get hold of the secret key. To protect secret key various key establishment protocols have been developed. They all address the problem of how to securely make secret key available to all pairs that need to use it to encrypt messages.

Steganography tries to enable hiding the very existence of messages being exchanged. Messages are very often hidden within various multimedia files, like pictures, audio or video recordings. These types of files have higher information redundancy than ordinary data files and can suffer minor changes with very little observable impact when reproduced. This fact is used to embed messages through "invisible" changes. Various techniques have been developed to better hide changes.

Idea of this paper is to use multimedia files to establish secret key for encryption. With proposed approach key space, and therefore key length, is virtually limitless. In addition there is no need to exchange keys. Keys are generated from multimedia files that both sides have. Parties only occasionally have to exchange information on which set of files they will be using. This information can be updated dynamically using encrypted channel that has been established.

The paper is organized as follows. Related work is addressed in section 2. Section 3 explains our idea on how to establish encryption keys. Example implementation and its comparison with (A)RC4 stream cipher are presented in section 4. Conclusion and discussion as well as directions for future research work are in section 5.

II. Related work

Basic issues of key establishment with various key transport and key agreement protocols are well covered in books [2], [3].

Different ideas on combining cryptography with steganography have appeared. One idea is to hide cihertext within an image using steganography like it was proposed in [4]. To further complicate things [5] proposes encrypting plaintext twice before hiding it in an image. Paper [6] proposes doing encryption and hiding in one step saving time and resources. Our approach is fundamentally different, much simpler and solves a different problem.

Idea to use media files to generate cryptographic keys has been around for a while. Most of proposed solutions were to generate personalised keys based on biometric features like fingerprint [7], voice [8] or face [9]. Good recent overview of biometric key generation methods and issues can be found in [10]. Again, our method borrows some ideas from this area of research but does not propose permanent personal keys, rather one time session keys.

The most similar idea to the one we propose is expressed in [11]. Their method uses image features for key generation. Process of key generation is rather complicated, and requires time. They also use their own encryption algorithm. Our idea uses image bits directly and we do not invent new encryption algorithm. In the next section the method that we propose will be explained.

III. Proposed method

A brief explanation of basic idea will be provided first. Then a more detailed explanation of one possible implementation of idea will be given.

A sender and a receiver should have an ordered set of files that are, individually, much bigger then messages being exchanged. For each message to be encrypted the sender picks a file from the set and a position within that file. The bits of a plaintext message are XORed with the bits of the selected file from the selected position to generate a ciphertext. The ciphertext with an index of the selected file and the position within the file is sent to the receiver. Using the index and the position, receiver can transform the ciphertext back to plaintext by XORing it with the bits of the same file from the same position.

A. Security analysis

A third party that monitors the communication channel can capture the ciphertext, the index and the position. The index and the position are of no value without knowledge of the file set. The ciphertext is the result of XORing plaintext message with the key, the bits form the selected file, that is the same length as the message. Since each message is encrypted with a different key that has the same length as the message method resembles one time pads.

There are two important differences. The key bits are not completely random. They are bits of a file that has certain format that might limit their randomness. Also, one time pads should be destroyed after use. Depending on the selected set of files this might not be possible. In a case that the set of files used to encrypt the message is revealed in future it will be possible to decrypt the message for anyone with an access to the set and the encrypted message with the index and the position.

Mentioned differences from one time pad mean that proposed method is not perfectly secure. For practical purposes the keys might be considered random enough. This points to a direction for future research on randomness of bits in various file types and selection of the best ones. Also, if encrypted messages have a value for a limited period of time than a selected set of files used for encryption needs to stay secret only for that period of time.

It is obvious that security of proposed encryption method is in secrecy of a set of files. The set of files might be considered as a master key or some sort of key encryption key, while the bits of files used to encrypt messages have a role of session keys. Key size of this master key is practically limitless since the number of possible file sets is practically limitless. There are implementation issues regarding the size of the set and the size of the files that might limit the possible size of this ”master” key for a particular implementation.

There is also, a very important, issue of distribution of this ”master” key. Parties in secret communication need to agree on a set of files they are going to use for encryption. We do not address that problem. We believe that it is solvable. The parties need to exchange information on any set of ordered files available to both sides. It could, and should, be done using different channel from the one that will be used to send encrypted messages. To support our belief we give some examples of possible file sets that can be named in short telephone or even SMS message. ”A folder of distribution of some OS ordered by names (sizes, dates, ...) of files”. ”A public, or available to parties, online repository of images (songs, video material, ...) ordered by names (sizes, dates, ...) of files”.

B. Formal model

Formal model has the following notation:

- P - ordered set of files
- i - file index
- P_i - selected file
- p - starting position in bits in file P_i
- $bP_i(k)$ - bit k in file P_i
- M - plaintext
- L - length of the plaintext
- C - ciphertext of plaintext
- bM_j - bit j of plaintext
- bC_j - bit j of ciphertext

Using above notation encryption process can be expressed with:

$forj = 1toL$

$bC_j = bM_j \oplus bP_i(p + j - 1)$

Similarly, decryption can be expressed with:

$forj = 1toL$

$bM_j = bC_j \oplus bP_i(p + j - 1)$

We will present implementation of CryptoStego example and we will compare it with (A)RC4 stream cipher in the following section.

IV. Comparing CryptoStego and (A)RC4

In this section we will describe message format for CryptoStego, testing environment we used, testing procedure we established and results received from CryptoStego and (A)RC4 measurements.

An alleged implementation of RC4 was posted September 13, 1994 at anonymously on the Internet newsgroup sci.crypt without permission or verification from author Ron Rivest [12] [13]. The name RC4 is trademarked, so RC4 is very often referred to as ARC4 or ARCFOUR [14], in order to avoid trademark problems. We will use name (A)RC4 in our work.

A. Message format

Since messages with a ciphertext need to include file index ”i” and starting position ”p” we defined message format for the proof of concept implementation we created. The structure of message is given in Table I.

TABLE I
THE STRUCTURE OF MESSAGE

File index i	Position p in file P_i	Bits of ciphertext
1 byte	4 bytes	L - length of plaintext in bits

Above message format assumes that there are maximum of 256 files in set. Position is defined with four bytes that allows for 2^{32}, over 4 billion, positions.

Details of testing and matching results will be given in next subsections.

B. Testing environment

Testing environment was set on laptop, with the following hardware: CPU Intel Core i7-3610QM, CPU working frequency was 2.30GHz, and RAM memory was 12 GB.

The laptop had the following software: operating system Windows 7 Ultimate Edition with SP1, software Intel VTune Amplifier XE 2011 and compiler Borland C++ version 5.02.

As a source for our set of files, we used one CD with family pictures. All pictures were inside one folder. The folder had 215 pictures. We made a copy of all pictures, sorted by date, from CD to one folder on laptop.

C. Testing procedure

We will describe how we set up environment, in order to be able to make comparison and get matching results.

We used compiler Borland C++ and made CryptoStego algorithm for enryption.

The algorithm first reads plaintext from a file. At that time we also calculate plaintext length. We will use plaintext length later, when we make ciphertext.

Second step of the algorithm is to randomly select a file from a set of family pictures. Random selection of a file is done by one function written in CryptoStego algorithm.

Third step of the algorithm is to select position within the file chosen in second step, as starting point for making CryptoStego ciphertext. Position selection is done by one function written in CryptoStego algorithm.

Fourth step is reading number of bytes calculated in first step, in the file chosen in the second step, starting from the position chosen in third step, which is our ”master” key.

Fifth step is encryption process, where we will get ciphertext.

Sixth step is writing ciphertext to a file, with structure given in table I.

We also used compiler Borland C++ to compile (A)RC4 algorithm [12] [13] for encryption.

We used Intel VTune Amplifier XE 2011 software for measurements.

Every time, for every length of bytes, we made measurement cycle between 4 to 6 times and we measured CPU and RAM memory usage.

One measurement cycle we did for (A)RC4 and one measurement cycle we did for CryptoStego.

For example, for (A)RC4, for the first measurement cycle we took 8 bytes (plaintext was 12345678) for plaintext message and for key, then we start measurement cycle. During this measurement cycle we wrote down current results. For CryptoStego, we also took 8 bytes (plaintext was 12345678) for plaintext message and then we start measurement cycle again. During this measurement cycle we also wrote down current results.

If the same current results in the same measurement cycle repeated minimum 4 times, we wrote down to Excel this as final result. Otherwise, we calculated average value of current results for current measurement cycle and then we wrote it down as final result to Excel.

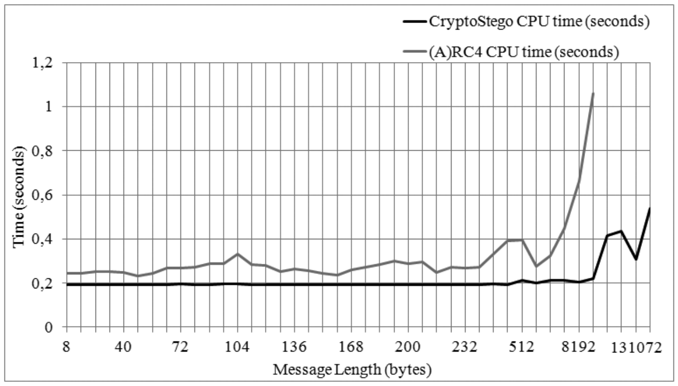

Fig. 1. CPU usage comparison

Maximum one current result in one measurement cycle is not taken into consideration, if it is deviated from the other current results.

For the next measurement cycle we took 16 bytes (plaintext was 1234567890123456) for plaintext message and for key on (A)RC4 and wrote down current results. For next measurement cycle on CryptoStego we took 16 bytes (plaintext was 1234567890123456) for plaintext message and wrote down current results.

After that, we wrote down final results for both (A)RC4 and CryptoStego measurements on 16 bytes to the Excel table.

We continued to increase number of plaintext bytes and key bytes to 24, 32, 40, etc. and we finished measurement cycles for (A)RC4 with 16384 bytes. However, we continued measurement cycles for CryptoStego, with increasing number of plaintext bytes to 32768, 65536, 131072 and 262144, respectively.

Ate the end of all measurement cycles, after we wrote down all final results to Excel table, we got one figure for CPU usage comparison and one table for RAM memory usage comparison.

The measurement is done only for encryption. We believe that decryption uses the same or less CPU time, because of the nature of measured algorithms, so we did not take it into consideration in this work.

Fig. 1 is having comparison of CPU usage for CryptoStego and (A)RC4. As we could see from fig. 1, CryptoStego is using less CPU time than (A)RC4.

Table II is having comparison of RAM memory usage for CryptoStego and (A)RC4, from where we see that CryptoStego use less RAM memory than (A)RC4.

We could see from Fig. 1 and Table II that CryptoStego have better performances than (A)RC4 in both CPU time and RAM memory consumption, at least for measured length of bytes.

V. CONCLUSION

CryptoStego key establishment and encryption method is simple and fast. It has been shown that it uses less resources than (A)RC4. CryptoStego resembles one time pads. Each message is encrypted with a different key. A length of the

TABLE II
MEMORY USAGE COMPARISON

Plain-text (bytes)	(A)RC4 memory (KB)	Cryp-toStego memory (KB)	Plain-text (bytes)	(A)RC4 memory (KB)	Cryp-toStego memory (KB)
8	7	7	176	8	7
16	7	7	184	8	7
24	7	7	192	8	7
32	7	7	200	8	7
40	7	7	208	8	7
48	7	7	216	8	7
56	7	7	224	8	7
64	7	7	232	8	7
72	7	7	240	8	7
80	7	7	248	8	7
88	7	7	256	8	7
96	7	7	512	8	7
104	7	7	1024	10	7
112	8	7	2048	12	7
120	8	7	4096	16	7
128	8	7	8192	24	7
136	8	7	16384	40	7
144	8	7	32768		9
152	8	7	65536		9
160	8	7	131072		9
168	8	7	262144		9

key is the same as a length of the message. Parties in secret communication need only to have an ordered set of files that are, individually, much bigger than messages being exchanged.

Future work is oriented towards the following open questions. An analysis of file types that would be the best as key generators. Files would need to have as random bits as possible, be much bigger than messages to be encrypted and organised in sets that are easy to name (and exchange if needed). Also protocols for agreement on set of files and ordering should be explored.

REFERENCES

[1] A. Kerckhoffs, “La cryptographie militaire - Partie I,” *Journal des sciences militaires*, vol. IX, pp. 5–83, Jan 1883.

[2] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1996.

[3] B. Schneier, *Applied cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. Wiley, 1996.

[4] P. Marwaha and P. Marwaha, “Visual cryptographic steganography in images,” in *Computing Communication and Networking Technologies (ICCCNT), 2010 International Conference on*, 2010, pp. 1–6.

[5] S. Usha, G. Kumar, and K. Boopathybagan, “A secure triple level encryption method using cryptography and steganography,” in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, vol. 2, 2011, pp. 1017–1020.

[6] S. Song, J. Zhang, X. Liao, J. Du, and Q. Wen, “A novel secure communication protocol combining steganography and cryptography,” *Procedia Engineering*, vol. 15, no. 0, pp. 2767 – 2772, 2011, ¡ce:title¿CEIS 2011¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877705811020224

[7] C. Soutar and G. Tomko, “Secure private key generation using a fingerprint,” in *Cardtech/Securetech Conference Proceedings*, vol. 1, 1996, pp. 245–252.

[8] F. Monrose, M. Reiter, Q. Li, and S. Wetzel, “Cryptographic key generation from voice,” in *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*, 2001, pp. 202–213.

[9] A. B. Teoh, D. C. Ngo, and A. Goh, “Personalised cryptographic key generation based on facehashing,” *Computers & Security*, vol. 23, no. 7, pp. 606 – 614, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404804001701

[10] L. Ballard, S. Kamara, and M. K. Reiter, “The practical subtleties of biometric key generation,” in *17th USENIX Security Symposium*, 2008.

[11] B. Santhi, K. Ravichandran, A. Arun, and L. Chakkarapani, “A novel cryptographic key generation method using image features,” *Research Journal of Information Technology*, vol. 4, no. 2, pp. 88–92, 2012.

[12] D. Sterndark, “Rc4 algorithm revealed,” https://groups.google.com/group/sci.crypt/msg/10a300c9d21afca0s, 1994, [Online; accessed 11.4.2013.].

[13] R. J. J. Jenkins, “Isaac and rc4,” http://burtleburtle.net/bob/rand/isaac.html, 1996, [Online; accessed 11.4.2013.].

[14] Wikipedia, “Rc4,” http://en.wikipedia.org/wiki/RC4, 2013, [Online; accessed 11.4.2013.].