A Novel Key Management for Virtually Limitless Key Size

Damir Omerasevic[1], Narcis Behlilovic[2] and Sasa Mrdovic[2]

[1] PBH Technologies, PrinTec Group of Companies,
Sarajevo, Bosnia and Herzegovina
d.omerasevic@printec.ba
[2] Faculty of Electrical Engineering, University of Sarajevo,
Sarajevo, Bosnia and Herzegovina
narcis.behlilovic,sasa.mrdovic@etf.unsa.ba

Abstract. The paper proposes key management between two parties, based on set of multimedia files shared by sender and recipient. The method is simple, fast, secure and robust. Possible key sizes are virtually limitless. One implementation, which uses YouTube website as a source of multimedia files, is presented.

Keywords: entropy, key creation, key exchange, key management, randomness.

1 Introduction

The goal of this paper is to propose simple, fast, secure and robust key management between two parties. We separated key management problem into three parts:

1. Creation of session-based type of keys, based on set of images, shared by sender and recipient,
2. Finding types of images with entropy suitable for session-based type of keys, and
3. Implementation with robust and secure key exchange properties, for creating key encryption key (KEK), for session-based type of keys.

1.1 Creation of session-based type of keys

All modern ciphers, like AES[1] or RSA[2], implement Kerckhoffs' principle [3] and Shannon's Maxim [4] that security of system is in security of secret key. Therefore, secret keys 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 way 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.

In this paper we use multimedia files to establish secret key for encryption, between two parties. There is no need to exchange keys. Keys are generated from multimedia files that both sides have. This is similar to session-based or one-time keys. Exchange parties have to exchange information on which set of files they use, from time to time, which is similar to KEK.

It is not good enough to use any set of multimedia files for session-based type of keys. Therefore, we had to discover adequate types of multimedia files for that purpose.

1.2 Discovering types of multimedia files suitable for session-based type of keys

Using entropy to measure randomness on series of data values is a well-accepted statistical practice in information theory [4].

In information theory, entropy is a measure of uncertainty. Under this term is commonly understood Shannon's entropy (although her origin comes from Pauli's [5] and von Neumann's quantum statistical mechanics [6]), which quantifies the expected value of the information contained in the message, usually in units such as bits.

According to Shannon, entropy H of any message or state is the following:

$$H = -K \sum_{i=1}^{n} p_i \log p_i \tag{1}$$

where p_i is probability of the state i from n possible states, and K is an optional constant.

By measuring entropy of different sets of multimedia files, we could also measure randomness on the sets, and therefore we could discover adequate types of multimedia files for session-based type of keys.

1.3 Implementation with secure and robust key exchange properties

GNU Privacy Guard (GnuPG or GPG) is a General Public Licence (GPL) alternative to the Pretty Good Privacy (PGP) suite of cryptographic software. The GNU General Public License (GNU GPL or GPL) is the most common used free software license today. GPL allows freedom to use, study, share (copy), charge and modify the software [7][8]. GPG is a part of the Free Software Foundation's GNU software project, and has received major funding from the German government.

The implementation with secure and robust exchange properties, described in this paper, proposes that the data set (i.e. sets and ordering of files) is first signed and encrypted by GNU Privacy Guard (GnuPG or GPG), which is compliant with RFC 4880[9], which is the current IETF standards track specification of

OpenPGP. After that, secure message is converter into Quick Response (QR) code.

GPG now supports RSA signing and encryption keys (inaddition to the older DSA for signing and ElGamal for encryption methods). DSA signing keys are limited to 1024 bit lengths, while RSA signing keys can be much longer (512 to 4096 bits are commonly used). In GnuPG version 2, the default is to create two RSA keys for the account now, one for encryption and one for signing.

We use QR codes error correction levels (L up to 7% of damage, M up to 15% of damage, Q up to 25% of damage and H up to 30% of damage) for including robustness in our proposition.

1.4 Our Contributions

Idea of this paper is to use a set of multimedia files, in order to establish secret key for encryption, between two parties. 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. Our idea is to use image bits directly from files. Exchange parties have to exchange information on which set of files they use, from time to time. This information can be updated dynamically, by using encrypted channel that has been established.

The main question here is how securely exchange information on which set of files parties use. The implementation of proposed key management, described in this paper, proposes that the data set of files parties use, is first signed and encrypted by GPG, and after that converted to QR code. GPG use RSA keys for signing and encryption. Robustness of presented implementation is due to QR code features, which are resistant to a certain level on errors.

1.5 Paper organization

The paper is organized as follows. Related work is addressed in section 2. Section 3 explains how to measure entropy in different video and audio media files. Section 4 explains our idea on how to establish encryption keys. Conclusion and discussion, as well as directions for future research work are in section 5.

2 Related work

We separated related work into two parts:

1. Creation of keys, and
2. QR codes, as a base for robustness.

2.1 Related work on creation of keys

Basic issues of key establishment with various key transport and key agreement protocols are well covered in books [11][12].

We use some concepts and ideas from both steganography and cryptography, when we use multimedia files to establish secret key for encryption, between two parties.

Steganography deals with ways of embedding secret messages on carrier media [13]. The characteristics of the carrier medium depend on the amount of secret information that can be hidden, on the perceptibility of carrier media and its robustness [14][15][16][17][18].

Different ideas on combining cryptography with steganography have appeared [19][20][21][22][23][24]. One idea is to hide ciphertext within an image using steganography, like it was proposed in [25]. To further complicate things [26] proposes encrypting plaintext twice before hiding it in an image. Paper [27] proposes doing encryption and hiding in one step, and saving time and resources.

According to authors' best knowledge and available data, the focus of the most of ideas is mainly on steganography, where cover medium is used as a carrier.

Idea to use different kind of media files to generate cryptographic keys is not new. Most of proposed solutions were to generate personalized keys based on biometric features like fingerprint [28], voice [29] or face [30]. Good recent overview of biometric key generation methods and issues can be found in [31]. However, all of ideas mentioned here requiring certain processing time, which prolongs total encryption time. 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 [32]. 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 is to use image bits directly.

2.2 Related work on QR codes

We use error correction levels embedded into QR code for improving robustness of our secure information (on data set).

In the last few years we experienced a very large application of QR codes in steganography, authentication and video watermarking. In [33], QR code and image processing techniques are used to construct a nested steganography scheme. A lossless data is embedded into a cover image. The data does not have any distortion, when comparing with the extracted data and original data. Since the extracted text is lossless, the error correction rate of QR encoding must be carefully designed. Authors of [33] found out that 25% error correction rate is suitable for the goal. This scheme is also robust to Joint Photographic Experts Group (JPEG) attacks.

In [34] authors proposed a geo-location based QR-code authentication scheme using mobile phone, to defeat against man-in-the-middle phishing attacks. The proposed scheme provides convenience, mobility, and security for the user.

Paper [35], proposes a video watermarking with text data (verification message) by using QR code. QR code is prepared to be watermarked by SVD (singular value decomposition) and DWT (Discrete Wavelet Transform). In addition to that, logo/watermark gives the authorized ownership of video document.

3 How to measure randomness in different video and audio media files

By using existing sets of already existing sources of media file types, which are good enough from randomness perspective to be used in everyday practice, we are shortening time for encryption/decryption, and therefore making the whole encryption/decryption process faster.

3.1 Randomness Tests

In order to test which media file types are good enough from randomness perspective to be used in everyday practice, we were using different statistical tests [36] [10], namely the following:

1. Entropy Test
   Entropy originally was introduced in thermodynamics and Shannon applied it on digital communications [4]. Entropy is a measure of the uncertainty in a random variable in information theory, so we could interpret entropy as the measurement of randomness. Shannon was interested in determining what was theoretical maximum amount for file compression, i.e. more entropy means less compression (and better quality of randomness) and vice versa. We tested entropy as percentage, which means that results which are the closest to 100% are the best.
2. Arithmetic Mean Test
   Arithmetic Mean Test is simply the result of summing all of bits in tested file and divide with the length of the file. If bits in tested file are close to random, the result should be close to 0.5.
3. Serial Correlation Test
   Serial Correlation Test measures coefficient or extent to which each byte in tested file depends on the previous byte [36]. If the coefficient in tested file are close to random, the result should be close to 0.
4. Lempel-Ziv Compression Test [37]
   The purpose of the test is to determine if and how much of testing sequence can be compressed. The sequence is considered to be random if it can not be significantly compressed. If the sequence in tested file is close to random, the result should be close to 0.

3.2 Testing environment for randomness tests

Testing environment for randomness tests was set on laptop, with the following hardware: CPU Intel Core i7-3610QM, CPU working frequency 2.30GHz, and RAM memory 12 GB. The laptop had the following software installed: operating system Windows 7 Professional Edition with SP1, and compiler Borland C++ version 5.02.

As a source for our testing sets of file types, we used the following sets: JPG, WAV, FLV WEBM and MP3 set of files.

3.3 Testing procedure

We used compiler Borland C++ and adopted source code from [36] and we created additional scripts for faster processing. Scripts are done in that way that we use [36] not only for one file, but for the whole folder, so we made efficiency and performance improvement for overall measurement process.

The measurement is done by running scripts, one time for each tested file type, and after that we collected results. We extracted all tables and comparisons, which are presented in next subsections of this paper, from collected results.

We used file indexes instead of real file names, due to space reduction and better table data clarity. File size is given in bits, for calculating purposes.

3.4 Test Results

Test results are presented for two best suited types of files:

- FLV set of files, and
- WEBM set of files.

FLV Testing Test results for FLV file types are given in Table 1. As we could see from the results, FLV files have very good test results, for the purpose of this work, for all of four tests, as the following:

- File entropy expressed as a percentage varies from 99.9531 to 100.0000, which is very close to 100.0000,
- Arithmetic mean varies from 0.4939 to 0.50004, which is very close to 0.5,
- Serial correlation varies from 0.001153 to 0.019413, which is very close to 0, and
- Reduction of compression is expressed as a percentage and is not varying, which means that is exactly equal to 0.

Table 1. Results of Comparison for FLV files

File Index	File Size	Entropy (%)	Arithmetic Mean	Serial Correlation	Compression (%)
FLV1	149177272	99.9531	0.4872	0.019413	0
FLV2	59895888	99.9994	0.4985	0.005974	0
FLV3	158340880	99.9984	0.4977	0.00546	0
FLV4	700971952	100.0000	0.5004	0.001153	0
FLV5	33027968	99.9891	0.4939	0.012764	0
FLV6	361880112	99.9993	0.4985	0.003405	0
FLV7	460491968	99.9983	0.4975	0.00545	0
FLV8	309196744	100.0000	0.4999	0.002019	0
FLV9	58047696	99.9982	0.4975	0.005092	0
FLV10	156009472	99.9983	0.4975	0.00545	0

WEBM Testing Test results for WEBM file types are given in Table 2. As we could see from the results, WEBM files have very good test results, very close to FLV results, for the purpose of this work, for all of four tests, as the following:

- File entropy expressed as a percentage varies from 99.9295 to 99.9998, which is very close to 100.0000,
- Arithmetic mean varies from 0.4844 to 0.4993, which is very close to 0.5,
- Serial correlation varies from 0.003173 to 0.014056, which is very close to 0, and
- Reduction of compression is expressed as a percentage and is not varying, which means that is exactly equal to 0.

4 Proposed implementation for key exchange

Parties in secret communication need to agree on a set of files (some kind of keys) they are going to use for encryption. Therefore, it is a very important issue of distribution of this "master" key. We address that problem in the following subsection.

4.1 Distribution of the "master" key

We proposed a solution to the issue, of distribution of "master" key.

We show by experiment that the best source for "'master"' key are FLV/WEBM files. Very good source of FLV files is YouTube website. All YouTube video files could be accessed by the following Uniform Resource Locator (URL) syntax:

https://www.youtube.com/watch?v=key

Table 2. Results of Comparison for WEBM files

File Index	File Size	Entropy (%)	Arithmetic Mean	Serial Correlation	Compression (%)
WEBM1	113135048	99.9295	0.4844	0.020691	0
WEBM2	1244183048	99.9998	0.4993	0.003173	0
WEBM3	311626752	99.9834	0.4924	0.005272	0
WEBM4	101210448	99.9960	0.4963	0.014056	0
WEBM5	365222584	99.9995	0.4986	0.011864	0
WEBM6	320629952	99.9969	0.4967	0.009266	0
WEBM7	228198976	99.9995	0.4987	0.006817	0
WEBM8	219042400	99.9909	0.4944	0.012517	0
WEBM9	314455432	99.9862	0.4931	0.006153	0
WEBM10	601979712	99.9976	0.4971	0.009781	0

where *key* is 11-alphanumeric YouTube video identification (ID), like, for example, *"voLNA8LdcCw"* (without quotes).

Our initial message (and the size of the set) has 256 *key*, i.e 256 file set is described with 256 lines of 11-alphanumeric YouTube IDs.

By using YouTube IDs, we could access all format of video files from one place and, depending of device and appropriate web browser, automatically show the best fitted video format for device which is currently used.

In order to get specific video format from specific YouTube IDs, we need to parse HyperText Markup Language (HTML) code for each of 256 YouTube IDs, and identify exact URL locations for FLV/WEBM files.

Considering the fact that we have all information about complete file set in one initial message, there is no need in this implementation to have separate messages for file sets and orders. We show processes from both sender and receiver side.

4.2 Distribution of the "master" key - sender side

In Fig.1 we described secure exchange process, initiated from sender side.

The process from sender side consists of eight steps:

1. Prepare initial message by sender,
2. Sign and encrypt initial message with GPG,
3. Prepare/encode QR code,
4. Send QR code to recipient,
5. Receive QR code by recipient,
6. Decode QR code,
7. Decrypt and verify signature with GPG, and
8. Prepare/calculate identical copy of initial message.

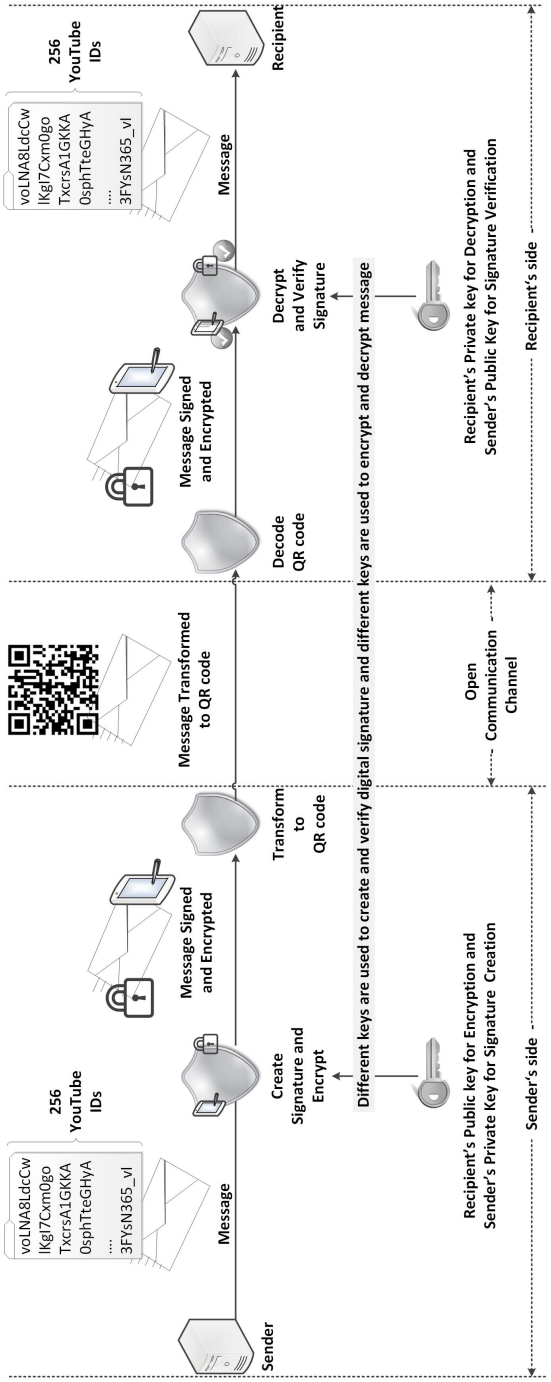

Fig. 1. Secure Key Exchange with QR code (sender side)

The most important parts from sender side are located in the first and the fourth step. We describe all steps in more details. After the last step is executed, we have to let sender know, that recipient received initial message.

Prepare initial message by sender Initial message file consist of 256 lines. In each line is 11-alphanumeric YouTube video identification (ID), plus additional end of line characters, like line feed (LF) and carriage return (CR). The total of 3,328 bytes is used.

We could manually type into initial message file all of 256 lines (YouTube IDs), but it not convenient. Although YouTube has a huge set of video files, it is not correct that we just randomly generate 256 of 11-alphanumeric IDs, because some of IDs generated on that way will no exist. We have to be sure that all of 11-alphanumeric IDs really exist on YouTube site.

Therefore, we suggest the following steps for creation of non-manual initial message file.

1. Randomly create 256 three-character (al least) strings,
2. Fetch from YouTube site pseudo-random chosen video, for each of three-character (al least) string, by using YouTube application programming interface (API),
3. Extract YouTube ID, for all of fetched YouTube videos.

Sign and encrypt initial message with GPG We used GPG4Win command-line utility *gpg.exe*, together with appropriate parameters, for signing and encrypting of initial message.

The syntax for signing and encrypting is the following:

gpg –armor –recipient Damir –encrypt –sign keys.txt

where *Damir* is recipient name, and *keys.txt* contains the initial message file.

Prepare/encode QR code For QR codes, we used compiler Microsoft Studio 2005, Visual C# part of the Studio, and adopted source code from [38], by making command-line applications. We were making additional batch scripts for easier usage. Scripts are done in that way that we write in advance parameters needed for command-line application, so we made efficiency and performance improvement for overall measurement process.

The syntax for creating QR code is the following:

QRCodeConsoleApp.exe keys.txt.asc jpg keys.jpg

where *QRCodeConsoleApp.exe* is name for QR code console application, *keys.txt.asc* contains GPG-signed and encrypted initial message file, *jpg* is type of graphical format used for QR code and *keys.jpg* if file name for created QR code.

Send QR code to recipient While sender is sending QR code to the recipient, an adversary could only listen (passive adversary), in order to try to learn more about messages exchanged. Adversary could try to put some noise into communication channel, or deliberately change some bits in the message, in order to prevent communication (active adversary).

Robustness of proposed key exchange is in the fact that we still can decode original (ciphertext) message, although QR code is damaged. Damage recovery is dependent on error level correction which we use during QR code encoding.

Receive QR code by recipient Recipient receives QR code and he does not know if QR code is sent by sender or not. In order to check it, recipient first has to decode QR code.

Decode QR code Result of decoding QR code should be (identical copy of) the initial message file. However, recipient still does not know if initial message is send by sender or not. In order to check it, recipient has to decrypt and verify signature with GPG.

Decrypt and verify signature with GPG The syntax for decrypting and verifying initial message is the following:

gpg –output decrypt-keys.txt –decrypt keys.txt.asc

where *keys.txt.asc* is GPG-signed and encrypted message, and *decrypt-keys.txt* contains (identical copy of) the initial message file, if sender signature is verified.

Prepare/calculate identical copy of initial message After confirming authenticity of (identical copy of) the initial message file, the result of previous step is creating *decrypt-keys.txt* file, which contains decrypted (identical copy of) the initial message file.

4.3 Distribution of the "master" key - recipient side

The process from recipient side consists of six steps:

1. Sign and encrypt (identical copy of) initial message with GPG,
2. Prepare/encode QR code,
3. Send QR code to sender,
4. Receive QR code by sender,
5. Decode QR code, and
6. Decrypt and verify signature with GPG.

It is important to stress here that it is not enough just to sign (identical copy of) initial message with GPG, but the message must be signed and encrypted, in order to preserve secrecy of the message.

As soon as sender decrypt and verify signature with GPG, which is send from recipient side as an acknowledgment of receiving ordered set of files, secret message communication could begin.

4.4 Secret message communication

As soon as sender and a recipient securely exchange an information of ordered set of files that are, individually, much bigger then messages being exchanged, secret message communication could start.

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 XOR-ed 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 recipient. Using the index and the position, recipient can transform the ciphertext back to plaintext by XOR-ing it with the bits of the same file from the same position.

We describe formal model in the following subsection.

Formal model Formal model of secret message has the following notation:

- k - key space (FLV files),
- P_k - ordered set of files P from key space k,
- i - file index i,
- P_i - selected file i from ordered set of files P,
- p - starting position p in bits in file P_i,
- $bP_i(j)$ - bit j in file P_i,
- m - plaintext message only,
- L_{UH} - length of unencrypted header,
- L_{EH} - length of encrypted header,
- L_m - length of the plaintext message,
- L_{EF} - length of encrypted footer,
- C - ciphertext message,
- bM_j - bit j of to-be-encrypted header, plaintext message and to-be-encrypted footer,
- bC_j - bit j of encrypted header, ciphertext message and encrypted footer.

Using above notation, encryption for part of secret message which is to-be-encrypted can be expressed with:

Algorithm 1 Encryption for part of secret message which is to-be-encrypted

1: **for** j = 1 to $(L_{EH} + L_m + L_{EF})$ **do**
2: $\quad bC_j = bM_j \oplus bP_i(p + j - 1)$
3: **end for**

Similarly, decryption for part of secret message which is encrypted can be expressed with:

Algorithm 2 Decryption for part of secret message which is encrypted

1: **for** j = 1 to $(L_{EH} + L_m + L_{EF})$ **do**
2: $bM_j = bC_j \oplus bP_i(p + j - 1)$
3: **end for**

Message format Since messages with a ciphertext need to include file index i and starting position p, within unencrypted part of the message, we defined message format, for implementation we created.

The structure of unencrypted part of the header of the message is given in Table 3.

Table 3. The structure of unencrypted part of the header of the message

Header field description	Length
File index i	1 byte
Position p in file P_i	4 bytes

The structure of encrypted part of the header of the message is in Table 4.

Table 4. The structure of encrypted part of the header of the message

Header field description	Length
Datetime stamp sender	8 bytes
Datetime stamp recipient	8 bytes

The structure of encrypted footer of the message is given in Table 5.

Table 5. The structure of encrypted footer of the message

Footer field description	Length
Secure Hash Algorithm-1 (SHA-1) of the whole message	20 bytes
Secure Hash Algorithm-1 (SHA-1) of the file used for encryption	20 bytes

The structure of the message is given in Table 6.

4.5 Security analysis

It is obvious that security of proposed key exchange method is in secrecy of a set of files. The set of files might be considered as a master key or some sort

Table 6. The structure of the message

Field description	Length
Unencrypted header	5 bytes
Encrypted header	16 bytes
Bits of ciphertext	L - length of plaintext/ciphertext in bits
Encrypted footer	40 bytes

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.

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 XOR-ing 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, our method resembles one-time pads.

Message format described assumes that there are maximum of 256 files in set (File index is 1 byte long), meaning 256! of permutations for selected file set. Position is defined with four bytes that allows for 2^{32}, over 4 billion, positions. Encrypted header contains encrypted time stamps for sender and recipient. Encrypted footer is SHA-1 hash for the complete message. The session key is selected from key space of randomly selected FLV files.

The attacker could only find out unencrypted part of the header of the message, i.e. file number and position in the file. The attacker can not decrypt the message, since he has no knowledge about KEK. However, the attacker/adversary could change unencrypted part of the header of the message, and therefore prevent communication between two parties.

Therefore, we use encrypted footer. Within encrypted footer we have two SHA-1 hash values. The first SHA-1 hash protects the whole message, including timestamps from sender and recipient, which could prevent replay attack. By calculating SHA-1 hash of received message on recipient side, we could also know if any other attempt was made, in order to change the message, by comparing calculated SHA-1 hash value with the first SHA-1 hash value in encrypted footer of the message received.

The second SHA-1 hash is calculated from the file which is used for encryption/decryption. The file is downloaded from YouTube site, at different time, on sender and recipient sides. In order to be sure that sender and recipient have the same file for encryption/decryption, we calculate SHA-1 hash and use it during secure communication.

We describe in Section 3 how to measure entropy in different video and audio media files. In a case that the set of files used to encrypt the message is revealed

in future, it is possible to decrypt the message for anyone with an access to the set and the encrypted message with the index and the position. However, considering the fact that we are using GPG with RSA keys for signing and encrypting, we consider our proposition safe and secure.

5 Conclusion

Key management presented in this paper is simple and fast. Presented solution resembles One-Time-Pads (OTP). Each message is encrypted with a different key. A length of the key is the same as the 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.

Easily available sets of already existing sources of media file types were tested on entropy. Files with content that is random could be the source for short lived cryptographic keys. Otherwise, key generation could take time. Using such files could make the whole encryption/decryption process faster.

Entropy/randomness measuring was performed using different statistical tests. Testing showed that FLV set of files, compared with all other above mentioned audio and video files, have the best results for all given statistical tests.

Each user is distributed with a unique and secret RSA key pair. Using RSA key pairs is reasonable, because of its general acceptance and safety checked during long time. However, it is only used in a minimum volume, not in full capacity, like in transport of symmetric keys, where we have to have keys longer than 2048 bits. In this paper is not an essence of RSA keys to protect keys, because, if that protection is broken, an adversary does not get key (which resembles OTP). In this paper, RSA has, except protective function, an important aspect in the phase of creating non-repudiation.

Key exchange implemented is not only secure, but also more robust. The most of well-known and widely-used cryptographic techniques are out of order, if we change a single bit of secret message. Therefore, we use QR code to add robustness/self-healing feature, up to a certain level, to our solution.

Robustness of presented implementation is due to QR code features, based on Reed-Solomon error correction codes, which are resistant to a certain level on errors. In our case we showed that we could use up to 25 percent error level correction, due to the length of the message (information on ordered set of files).

Our future work is oriented mostly towards transformation of the implementation to other platforms/operating systems, like Android, Windows Mobile platform or IOS, and compare performances from smartphone platform(s) to laptop/desktop platform based on Windows operating system.

References

1. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer Verlag, Berlin, Heidelberg, New York (2002)

2. Rivest, R. L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. In: Communications of the ACM, vol. 21(2), pp. 120–126 (1978)
3. Kerckhoffs, A.: La cryptographie militaire - Partie I. In: Journal des sciences militaires, pp. 5–83 (1883)
4. Shannon, C.E., Weaver, W.: A Mathematical Theory of Communication. University of Illinois Press, Champaign, IL, USA (1963)
5. Pauli, W.: Über das H-Theorem vom Anwachsen der Entropie vom Standpunkt der neuen Quantenmechanik. (German) [On the H-theorem of entropy increase from the standpoint of the new quantum mechanics]. In: Probleme der modernen Physik, Arnold Sommerfeld zum 60. Geburtstage, gewidmet von seinen Schülern. (German) [Problems of modern physics, Arnold Sommerfeld's 60th Birthday, dedicated by his students], pp. 30–45 (1928)
6. Petz, D.: Entropy, von Neumann and the von Neumann entropy. In: ArXiv Mathematical Physics e-prints, pp. 83–96 (2001)
7. GNU: GNU General Public License, version 3, `https://www.gnu.org/copyleft/gpl.html` (2007)
8. Wikipedia: GNU General Public License, `http://en.wikipedia.org/wiki/GNU_General_Public_License` (2007)
9. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, F.: RFC 4880 - OpenPGP Message Format, `http://tools.ietf.org/html/rfc4880` (2007)
10. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic application. National Institute of Standards and Technology, pp. 2–3 (2010)
11. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. CRC Press, Boca Raton, FL, USA (1996)
12. Schneier, B.: Applied cryptography (2nd ed.): Protocols, Algorithms, and Source Code in C. Wiley, New York, NY, USA (1996)
13. Anderson, R., Petitcolas, F.: On The Limits of Steganography. In: IEEE Journal of Selected Areas in Communications, vol. 16, pp. 474–481 (1998)
14. Amin, M.M., Salleh, M., Ibrahim, S., Katmin, M.R., Shamsuddin, M.Z.I.: Information hiding using steganography. In: 4th National Conference on Telecommunication Technology (NCTT), pp. 21–25 (2003)
15. Johnson, N.F., Jajodia, S.: Exploring Steganography: Seeing the Unseen. In: Computer, vol. 31(2), pp. 26–34 (1998)
16. Sahoo, G., Tiwari, R.K.: Some new methodologies for secured data coding and transmission. In: International Journal of Electronic Security and Digital Forensics, vol. 3(2), pp. 120–137 (2010)
17. Marvel, L.M., Retter, C.T., Boncelet, C.G.Jr.: A methodology for data hiding using images. In: IEEE Military Communications Conference, MILCOM 98., vol.3, pp. 1044–1047 (1998)
18. Cachin, C.: An Information-Theoretic Model for Steganography. In: Information Hiding. LCNS, vol. 1525, pp. 306–318, Springer, Heidelberg (1998)
19. Provos, N., Honeyman, P.: Hide and seek: an introduction to steganography. In: International Conference on Computational Intelligence and Communication Networks, vol. 1(3), pp. 32–44 (2003)
20. Krishna B, Anindya J.P., Geetam S.T., Sarkar P.P.: Audio Steganography Using GA. In: IEEE Security & Privacy, pp. 449–453. IEEE Computer Society, Los Alamitos (2010)

21. Sharp, T.: An Implementation of Key-Based Digital Signal Steganography. In: Proceedings of the 4th International Workshop on Information Hiding, pp. 13–26, Springer-Verlag , London (2001)
22. Chan, C.K., Cheng, L.M.: Hiding data in images by simple LSB substitution. In: Pattern Recognition, pp. 469–474 (2004)
23. Lin, I.C., Lin, Y.B., Wang, C.M.: Hiding data in spatial domain images with distortion tolerance. In: Computer Standards & Interfaces, vol. 31(2), pp. 458–464 (2009)
24. Wang, R.Z., Lin, C.F., Lin, J.C.: Image hiding by optimal LSB substitution and genetic algorithm. In: Pattern Recognition, pp. 671–683 (2001)
25. Marwaha, P., Marwaha, P.: Visual cryptographic steganography in images. In: 2010 International Conference on Computing Communication and Networking Technologies (ICCCNT), pp. 1–6 (2010)
26. Usha, S., Kumar, G.A.S., Boopathybagan, K.: A secure triple level encryption method using cryptography and steganography. In: 2011 International Conference on Computer Science and Network Technology (ICCSNT), vol 2, pp. 1017–1020 (2011)
27. Song S., Zhang J., Liao X., Du J., Wen Q.: A Novel Secure Communication Protocol Combining Steganography and Cryptography. In: Procedia Engineering, vol. 15, pp. 2767–2772 (2011)
28. Soutar, C., Tomko, G.J.: Secure private key generation using a fingerprint. In: Cardtech/Securetech Conference Proceedings, vol. 1, pp. 245–252 (1996)
29. Monrose, F., Reiter, M.K., Li, Q., Wetzel, S.: Cryptographic key generation from voice. In: IEEE Symposium on Security and Privacy, pp. 202–213 (2001)
30. Teoh A.B.J., Ngo D.C.L., Goh A.: Personalised cryptographic key generation based on FaceHashing. In: Computers & Security, vol. 23(7), pp. 606–614 (2004)
31. Ballard, L., Kamara, S., Reiter, M.K.: The practical subtleties of biometric key generation. In: 17th USENIX Security Symposium, pp. 61–74 (2008)
32. Santhi, B., Ravichandran, K.S., Arun, A.P., Chakkarapani, L.: A Novel Cryptographic Key Generation Method Using Image Features. In: Research Journal of Information Technology, vol. 4(2), pp. 88–92 (2012)
33. Chung, C.H., Chen, W.Y., Tu, C.M.: Image Hidden Technique Using QR-Barcode. In: Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), pp. 522–525 (2009)
34. Liao, K.C., Lee, W.H.: A Novel User Authentication Scheme Based on QR-Code. In: Journal of Networks, vol 5(8), pp. 937–941 (2010)
35. Prabakaran, G.,Bhavani, R., Ramesh, M.: A robust QR-Code video watermarking scheme based on SVD and DWT composite domain. In: 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering (PRIME), pp. 251–257 (2013)
36. Walker, J.: ENT - A Pseudorandom Number Sequence Test Program, `http://www.fourmilab.ch/random/` (2008)
37. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. In: IEEE Transactions on Information Theory, vol. 23(3), pp. 337–343 (1977)
38. Codeproject: Open Source QRCode Library, `http://www.codeproject.com/Articles/20574/Open-Source-QRCode-Library` (2007)