# FGP (Good Privacy) – Technical Description

A program for encryption and decryption of text based messages. FGP utilises a random number generator, which passes all the statistical diehard battery tests, in combination with a secure password. A secure password has more than 12 upper and lower case characters. These features makes brute forcing FGP take years, even with a super computer.

Table of Contents

[FGP (Fairly Good Privacy) 1](#_Toc439762761)

[1 What is Encryption? 3](#_Toc439762762)

[2 The Encryption Arms Race 3](#_Toc439762763)

[3 Efficient Encryption 3](#_Toc439762764)

[4 How FGP Works 4](#_Toc439762765)

[5 Why use FGP? 6](#_Toc439762766)

[6 How FGP Generates Good Random Numbers 7](#_Toc439762767)

[7 FGP Password recommendations 8](#_Toc439762768)

[8 Key Security 9](#_Toc439762769)

[9 Summary 10](#_Toc439762770)

## 1 What is Encryption?

Encryption has been used since Roman times to conceal the content of messages. Encryption uses a key to encode information so that it looks random to those not possessing the key. It is much like locking a letter in a box. Breaking the encryption involves discovering the key; either because it is leaked by the encryption algorithm or by brute force of the entire key space (trying every possible key). Encryption algorithms widely used today use complex mathematics to combine the key and plain text to produce a cipher text output that looks random, unless you possess the key. Brute force attacks on modern key spaces (longer than 256 bits) are touted as being impractical due to the time and computing power required.

## 2 The Encryption Arms Race

Encryption will continue to be part of an ongoing arms race between Cryptographers and Hackers. Cryptographers develop new encryption methods, these are weakened over time by more powerful computers and more sophisticated hacking techniques that allow discovery of the key. Each new method of encryption will eventually become readable to well-resourced hackers, with the exception of the one-time pad. The one-time pad, when correctly used, is secure for all time.

## 3 Current Encryption Methods

Modern encryption methods (particularly AES) are fast, efficient algorithms. This is convenient for the encryption process, but unnecessary for most users who do not require high speed streaming of encrypted data. However, efficient encryption is vital when it comes to breaking encryption. Well-resourced hackers can attempt 1012 keys per second during a brute force attack. If encryption is inefficient, and each key attempt takes at least a millisecond, those 1012 key attempts take over 30 years.

One of the key factors cited by the NIST in favour of AES, was efficiency. Faster encryption favours the hacker attempting to decipher a message, not the person encrypting it. Inefficient encryption takes longer to attack while the delay to the sender is comparably minimal. Inefficiency would be a desired characteristic of secure encryption.

AES is permitted to be exported across the globe. Export restrictions would apply to any new form of encryption. This implies that AES, while touted as secure, provides little security against well-resourced governments.

While brute force attacks on the key space of AES is theoretically impractical, it is difficult to know if the key spaces we are using are as large as they appear. Limited usable key spaces and limited key expansion schedules both simplify the attacking of AES. Discovering such limitations is hard to prove, and even harder to disprove.

## 4 How FGP Works

The one-time pad is the only proven secure encryption method available. However, transferring large one time pads between users is annoying and complicated. FGP uses the theory of one-time pads, with highly complex passwords to encrypt data. It should be reinforced that this method is *not* a one-time pad and security is only as good as the key. A long key strengthens the system against a brute force attack on the key and this encryption technique requires long keys.

There is an unwritten mantra that a block cipher should not expand the message. One byte in = one byte out. This is convenient – an encrypted file is the same size as the unencrypted file. Nice, but not actually necessary. This mantra limits the ciphers ability to secure data. With a choice of insecure encryption that doesn’t expand the message, or encryption that doubled the size of the message but was highly secure; which would you pick in this world of ever increasing bandwidth? FGP goes against standards and doubles the size of messages sent.

When using FGP, users need to create a secure password. ‘Secure’ passwords are defined as containing 12 or more upper and lower case characters. An easy way to create long passwords that can fool a dictionary attack while still being easy to remember is detailed in Section 6. The secure password has many random number expansions and lossy hashes applied to it before being used with a one time pad to secure the data. FGP has a very good true random number generator which is described in Section 7. These expansions and hashes require a large number of brute force computations for the user. A hacker has to brute force everything and consequently takes a much longer time. The FGP series of expansions and hashes is described in detail below.

The Process:

Secure Password → Root Key → Hardened Root Key → Session Key → Pseudo ‘One-time pad‘

Plain text **XOR** ‘One time pad’ → Cipher text

Different root keys are generated for each message sent, even if the same password is used. Session keys are used for 1024 bytes of data only. For larger messages, a new session key is generated from the hardened root key. This allows users to reuse passwords without seriously compromising security. Each of these keys, and how they are generated, are described below.

Root Key:

The user password is expanded by shuffling in additional true random numbers (commonly referred to as ‘salt’) using a Fisher-Yates shuffle – a known unbiased shuffle algorithm. The root key is then created by hashing the password multiple times. These hashes are then XORed together to create the root key. Each hash needs to occur to build the root key.

Hardened Root Key:

The root key then undergoes a process to produce a ‘hardened root key’. Two random numbers, a ‘target’ and ‘collision count’ are transmitted with the message. Then a series of hashes are done with the root key and additional salt. This hash is XORed down to a byte. When the byte equals the target, this is a collision and a counter increases. When the counter equals the collision count, all the previous hashes are XORed together to produce the hardened root key.

This process cannot be avoided by an attacker and takes about 2 seconds for a standard computer (2.66GHz, 32 bit, Dual Core) which knows the password. This greatly delays any hackers who need to brute for the entire key space. A new hardened root key is generated for each message.

Session Key:

The hardened root key is then XORed with true random numbers to create a ‘session key’. The session key is used for message blocks of 1024 bytes. For longer messages, multiple session keys are used (one every 1024 bytes). These are created from the same hardened root key. The session key is then used to generate a compressed 1024 byte one time pad which is XORed with the message to create the cipher text.

The security:

The fundamental security is provided by true random numbers. If the key is not known to an attacker, decryption requires a brute force attack on the key. In this system a password derived key can be re-used as it is only ever transmitted when encrypted with true random numbers overcoming the principle drawback of a one-time pad. Once again, this system is *not* a one-time pad and relies on the strength and security of the key.

FGP has these advantages over current block cipher algorithms such as AES:

* It is simple to implement and relies on good random numbers and key protection for security. If the random numbers are good, the key and message are both protected.
* It is difficult to secretly back-door this method. If the random numbers are good, the logic is simple and most developers, not just specialist cryptographers, can verify that the software is free of back doors.
* It is not dependent on Elliptical Curve Cryptography so remains secure in the event of a future (or past) solution to the discrete logarithm problem.
* The only feasible attack would be a brute force attack on all possible permutations of the key using the known key expansion algorithm. A long, unique password or key makes such an attack infeasible.

The disadvantage is that the cipher text is just over double the plain text message size.

## 5 Why use FGP?

FGP was designed to remain secure against powerful adversaries. When we designed FGP, we made these assumptions:

* Standard cryptographic libraries available to software developers today are likely to have weaknesses that can or are being exploited. Security must not depend on standard cryptographic functions alone.
* AES is either already insecure or unlikely to remain secure for long.
* Random number generators that are available to software developers are likely to be compromised.
* Encryption that is slow is, by definition, harder to brute force attack than fast, efficient encryption.

FGP uses these techniques to maintain message security:

* The user password is expanded by adding salt and hashing the result to produce a new random key for every message. Discovering a key could only break a single message.
* The plain text user password is immediately deleted after hashing.
* The hashed password is never saved to disk and it is different for every message.
* The decryption algorithm does not know the key but can deduce it only when the user password is known. It has to brute force the salt and compute the hashed key. This process is intentionally slow (several seconds), significantly hindering brute force attacks. There is no sideband attack that would allow this slow process to be shortened.
* The expanded key is not directly used to encrypt the message. Fresh random numbers are generated for each block of message providing forward secrecy. The key is effectively protected by a series of one-time pads.
* Each message block is protected by new random numbers (one-time pad) from the key preventing key leakage from chosen plain text attacks.
* Random numbers are generated from eight different sources, combined to produce a good random output that cannot reveal source random number sequences or patterns. This prevents known pseudo-random sequences being detected and used to decrypt the message.
* Local entropy (randomness) is added within the random number generator to ensure there is no practical way to replicate the random number sequences generated for encryption.
* The only feasible attack on FGP is a brute force attack the password. A password or pass-phrase of 12 lower case characters only would take months to brute force with a supercomputer. Lower and upper case characters raise that time to years. Long passwords or pass-phrases are more secure.

## 6 How FGP Generates Good Random Numbers

Random numbers are at the heart of most cryptographic algorithms. Good random numbers are unpredictable and statistically randomly distributed. It is relatively easy to make a random number generator that appears to be secure for cryptographic purposes but has a concealed back door, assuming you have a bank of supercomputers. A random number generator on a PC is typically a pseudo random sequence that *eventually* repeats. The start position in the sequence can indeed be random, using a little entropy from the computer. However, if the entire sequence is known, it doesn’t take much to find out where in the sequence the generator is and the numbers thus become predictable and non-random.

Fortunately, for those who want privacy, it is also relatively simple to enhance a compromised random number generator with unique local entropy from a PC or smart phone. This will increase randomness sufficiently for a good cryptographically secure random number source. On a PC or SmartPhone unique entropy cab be gathered from keystroke timing, processor activity, microphone noise, webcam images. This entropy can be combined with a cryptographically “secure” but compromised random number generator to close any back doors. Unique local entropy is the key. Converting an analog real-world signal into digital form always generates noise – this is what we need, random noise, or entropy. A back-doored random number generator may be statistically excellent, just predictable to those in the know. Unique entropy from the PC makes those random numbers really unpredictable – truly random.

For the developers among you, a more sophisticated way than adding or xoring unique entropy is to consider using an unbiased shuffle algorithm such as the Fisher–Yates shuffle (see Wikipedia for a good description and a three line of source code implementation). Shuffling a 256-byte array of every number from 0 – 255 based on several poor entropy sources and a standard (possibly compromised) cryptographic random number generator will give a very good, truly random result. Taking random numbers from vertical trajectories through multiple rows of shuffled and re-shuffled 256 byte arrays will prevent the introduction of any biases from the unique but imperfect entropy sources within the PC as each row contains the full range of possibilities for each byte or nibble. Shuffling 4–bit nibbles is an alternative that is more computationally efficient and uses less unique entropy.

The Fisher-Yates shuffle approach to combining different streams of random numbers has the advantage that it is not possible to reconstruct the original random sources from the output stream. This makes it harder to identify positions in pseudo-random generators. It also minimises the amount of unique entropy required to make a good random number stream.

Example:

1 [0, 1, 2 … 255] initial array of each number from 0 to 255

2 [34, 167 … 29] Shuffle with potentially insecure random number generator.

3 [245, 66 … 79] Shuffle again with unique entropy from PC (webcam).

4 [65, 45 … 176] Shuffle again with unique entropy from PC (microphone).

5 Repeat from step 2 to produce as many subsequent rows as required.

[68, 48 … 163] Steps 2-4 shuffle.

[28, 186 … 76] Steps 2-4 shuffle.

[118, 31 … 63] Steps 2-4 shuffle.

Take a vertical trajectory down column 1 to generate truly random numbers 65, 68, 28, 118.

## 7 FGP Password recommendations

Choosing a strong password does not need to be difficult if the concept of permutations is followed. A strong password should require an attacker to attempt an infeasibly large number of permutations.

Each language has a finite vocabulary. A rule of thumb is that most people have a vocabulary of about 10,000 words in their first language. Using a single word immediately limits the number of permutations to 10,000 or less. Using multiple words (preferably not obvious compounds like “daytime” or “maybe”) increases the number of permutations. Using words from other languages increases the number of permutations significantly.

Using a phrase that is easy for a human to remember, but is unlikely to come up soon in an automated computer search is the goal. If people can memorise Shakespeare’s Hamlet, then we should be able to remember a phrase of a few words.

We recommend:

* Length over complexity – the longer a password is, the more permutations an attacker must attempt to read your message. 16 characters is a good minimum to aim for.
* Multiple unrelated words – every language has a finite vocabulary. A rule of thumb is that most people have a working vocabulary of about 10,000 words.
* Keep it simple to remember – using upper and lower case characters, numbers and punctuation symbols can be a waste of time, particularly if they make the password hard to remember. A single capital letter does increase the password permutations, but if it is the first character of the first word, that will be the first thing an automated password search will try. Adding a three letter word to a 12 character password increases the permutation count more than using upper case characters.
* Use memory aids. Phrases that can be remembered easily with a unique twist such as switching one or two words to another language are ideal.

Sample passwords – do not use these but they show how to construct a good password.

Example 1 (bad):

Memory aid: “foundation” – easy to remember a single word

Password: “Foundation” – capitalization of first letter

Permutations: 9 characters, 1 word, about 1 x 104 word permutations to search

Negatives: A Too short

B Too few words, only one language

Example 2 (good):

Memory aid: “the quick brown fox” – easy to remember

Password: “the quick brown reynard” – easy to look up if you forget the French “reynard”

Permutations: 23 characters, 4 words, about 1.6 x 1012 word permutations to search

Negatives: A Phrase is well known

B It has been used here

Example 3 (better):

Memory aid: “in a hole in the ground” – easy to remember

Password: “in a hole in el suelo” – easy to look up if you forget the Spanish “el suelo”

Permutations: 21 characters, 6 words, about 6.4 x 1025 word permutations to search

Negatives: A Phrase is the beginning of a well known book.

B It has been used here

## 8 Key Security

One of the main problems with any form of information security is key material. Humans can conveniently remember a password that has a complexity (number of permutations) of roughly 1014. More than that gets difficult to remember easily or uses information that is not private or secure such as an email address. Brute forcing 1014 password permutations of a compact efficient encryption algorithm (recall that efficiency was the key NIST design criteria for AES) isn’t going to tax a supercomputer. Hashing a password doesn’t make the system any more secure against a brute force attack on the source password.

The problem is that we are unable to remember a password of sufficient length to be secure against government intrusion. The solution in FGP is a very inefficient one and in this case inefficiency is our friend. We know the password and our password expansion program adds about 20 random numbers to bulk it out. We don’t tell our recipient the numbers, but instead give them pointers that let them know with certainty when they have deduced the correct random number, after trying all possible permutations. Now it takes our recipient, who does know our password some time to deduce the random, number portion of the password – several seconds perhaps – relatively minor inconvenience for our recipient. All of this can be done seamlessly within a mail program. With a good 12 character password, however, the brute force approach for our own personal spook who does not know our password, even with a supercomputer is going to see him or her waiting years before reading your message.

Example:

* Salt an 8 character password with a predetermined 3 byte salt string.
* Pick two true random numbers, R1 and R2 which are transmitted with the message
  + Swap two bytes of the salt with the index of a loop counting up from 0 to 65535
  + Hash the salt + password + index
  + XOR the 32 bit digest down to one byte
  + When the XOR of the hash equals R1, this is a collision
  + When R2 collisions have been counted, the correct index has been reached.
* Repeat this process 10 times (generates 20 random numbers and 320 bytes of digest)
* Hash the entire 320 byte digest in sequential blocks meaning that a brute force attacker is faced with an all or nothing requirement for each password attempt – no short cuts.

The number of hashes required is on average 327680, taking about 2 seconds on a modern computer. For the attacker however, the problem is multiplied by the password complexity. Even with a supercomputer operating at 109 times the speed of our PC we are looking at 2 x 105 seconds (~ 2 days) for a basic lower case password of complexity 1014. Lengthen the password a few characters to say 12, throw in some upper case and special characters and decryption time goes up to many years for just one message.

The password can be reused as every message uses a fresh block of 20 random numbers to generate a new 320 byte hash block or key. The XOR hash collisions do not occur linearly so the numbers R1 and R2 cannot be reliably used to pick a starting point for the loop indices.

This method of password expansion fits well with the encryption technique described in Section 4 as it generates a large amount of key data which can be expanded further by a simple hash expansion.

## 9 Summary

The encryption we use today is probably broken. We have accepted efficient encryption, though efficiency was not our requirement but the requirement of those who wished to read our emails. If we really want private communication we need to move towards true random number based encryption that is simple, inefficient and can’t be secretly back-doored.