Sending Emails

The safe way

Kristian Fiskerstrand

Contents

CHAPTER 1

Introduction

CHAPTER 2

Understanding digital signatures

CHAPTER 3

Encryption

CHAPTER 4

OpenPGP

Chapter 1

Introduction

Talking to an acquaintance on the phone, it is generally easy to know whether you're talking to the one you expected. This is however not necessarily as easy when sending an email. As a result the credibility of email messages in general is lower and implicitly many would rather talk to someone one the phone over sending an email, despite email's advantages for efficient communication. And there are many advantages: Email is an asymmetrical form of communication which doesn't require the other party to be present at the exact time you yourself have the time for it. It gives both the sender and the recipient time to properly formulate the communique in an unambiguous way, as well as do the necessary research for the information contained to be as accurate as possible. Our focus here will however be on another aspect: Emails enables the possibility of secure communication. There is an old proverb stating: "*There is no use closing the door, once the horse has left the barn*". Sadly many ignore any security considerations, often on a basis of claiming it too difficult of a concept to grasp. This excuse is often used until, and sometimes even after, the issue is overdue escalation. It is about time for this to change. Since you are already reading this book I hope you don't do so because you just had a security breach, or if you do; I urge you to inform others of your reasoning in order to help them be pro-active in protecting their privacy and their data and hope that the results in your own case did not lead to any permanent ramifications. Because security is usually considered a secondary, or even tertian need, it increases the difficulty of educating people. We do not generally sit down in front of our computer wanting to manage our security. Rather we want to send emails, browse web pages, download software, and we want security in place to protect us while we do these things. A paper written in 1998 named “Usability of Security: A Case Study” by Alma Whitte and J.D. Tygar[9], where they call this element the unmotivated user property discusses this. It follows up by defining the abstraction property which states “*Computer security management often involves security policies, which are systems of abstract rules for deciding whether to grant accesses to resources. The creation and management of such rules is an activity that programmers take for granted, but which may be alien and unintuitive to many members of the wider user population.*”

Combining the effect of the abstraction property and the unmotivated user property can give scary results. The general user will not understand the basis for the policies put forth in security applications without education, but at the same time, the general user is not to be expected to be interested in learning about security.

This book is logically divided into two. A great deal of the content is a generic introduction to digital signatures and encryption on a general basis. The rest will, however be quite practical. Hopefully both parts are useful to you.

Bruce Schneier is often quoted with an expression stating: “*Security is a process, not a product*”. Hopefully this book can help you gaining both interest and knowledge into the process of securing your email communication as well as your data, and help you increase the credibility of emails, by sending emails the safe way.

Chapter 2

Understanding digital signatures

2.1 What is a digital signature

A digital signature is a part of the email that, when properly implemented, is mostly invisible to the user. It does however have some unique features that makes it very valuable to add credibility for those seeking it. As opposed to an analog signature of a standard letter, digital signatures for our purposes accomplishes two goals. The first use is to verify the sender (**authentication**). This is also done by an analogue signature, if you know the persons handwriting well enough and trust that the signature is not a copy. However, a digital signature also verifies that the content has not been tampered with (**data integrity**), because, opposed to analogue signatures, the digital signature is created based on the content it signs, using a digest algorithm as discussed later in this book. If anyone were to change the content of a digitally signed document, the signature would be invalidated.

Historically both of these functions were performed by the use of seals. In a time with limited resources and knowledge with regards to the art of deception it often served the purpose well. Today's attackers are however slightly more sophisticated. The thing is, an ordinary signature today means little or nothing at all. People change the handwriting over time, and how the signature gets depends on the context it is singed in; the available space to sign on, if it is rushed or not, what kind of pen is used et cetera. But probably the greatest problem is the recipient's ability to properly verify that the signature actually comes from the intended sender. Digital signatures are far superior to analogue signatures in each aspect, and this gives email an advantage over both ordinary letters and faxes, if used properly, which you hopefully will be guided to by reading this book.

2.2 How can a digital signature help you?

If you run any kind of business you want to be able to know that the sender of the email is the one that first ordered your services. For instance a web hosting company receiving an email asking for the password of one of the domain hosting services to be reset, or the file permissions assigned to a different user. You will want proof that the sender is whom he claims to be, and if the user submitted a public key e.g. when paying for the package, this can easily be handled.

Another good practical example where the use of digital signatures vital is signing of software packages or other files that are to be distributed, exempli gratia over the Internet. Presuming that you have gotten the public key through a trusted source, or when downloading the last version of the program, you can then use the signature to verify that the file has not been altered. A real world example where this would have helped is with regards to the Internet Relay Chat (IRC) client BitchX, that came under attack using Domain Name Server (DNS) poisoning. Without going into too many technical details, which would bore most, in summary people downloaded a copy of it that contained spyware, because the download got directed to another server than the official one.

Two-hundred and fifty users of the Swedish bank Nordea got their bank accounts tapped after first having been infected with a modified version of the trojan horse “haxdoor” resulting in a loss for the bank of about $1.2 Million USD.

> **Trojan Horse:**
> The term trojan horse was coined as a result of a historic event between the Greek and Trojans in the city of Troy. The Greek offered a wooden horse as a gift to the Trojans, allowing the greeks hidden in the horse access behind the walls of the city.
>
> The analogy of this is wildly used in computer terminology today, representing a virus or worm opens a backdoor to the system for malicious individuals to connect, and gain control of, the system.

Chapter 3

Encryption

Data/Content that can be read and understood without any special measures is called *plaintext*, or *cleartext*. When you go through the encryption process you get *ciphertext* as output, data that can appear garbled and has to be decrypted before it once again can be read as cleartext. *Cryptography* is the science of using mathematics to encrypt and decrypt data. It enables you to store sensitive information or transmit it across insecure channels (like the Internet) so that it cannot be read by anyone except the intended recipients. While cryptography is the science of securing data, *cryptanalysis* is the science of analysing and breaking secure communication. Classical cryptanalysis involves a combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination and luck. Cryptology covers both cryptography and cryptanalysis. Lets start off by a simple question: Do you write down sensitive data on the back of a postcard? If the answer is "no" and I hope it is, why not? Because you know, that anyone dealing with the mail under ways; postal workers, the delivery guy, or anyone peaking in your mailbox, can read it. The same goes for e-mail, except then it's done electronically in a matter of seconds. Why do you put mail in an envelope? Breaking a sealed envelope is a felony in most countries, and as such it adds a level of judicial protection to the content in addition to making it more difficult to read. The solution for putting emails in an envelope is even harder to break through, but not necessarily more difficult to apply; it is called encryption.

In Norway the need for encryption escalated into a political fiasco in 2006. The prime minister's office has the website *smk.dep.no* and as such email addresses of the employees ends with @smk.dep.no . In a quick glimpse a PR worker for the Norwegian labor party sent an email containing key points for a political debate later that day to @smk.no , hence leaving out the ”dep” part. This is of course a completely different domain name, and hence it ended up in the inbox (catch-all enabled to get <everything>@smk.no) of a Norwegian corporation. The one receiving the email, supporting another political party than the labor party quickly forwarded it to the party he supported, and the labor party got questioned about the content in a fairly humiliating context on a live TV debate later the same day. The end of this story was that the government purchased the smk.no domain name for 16,000 USD, a relatively small sum compared to other popular domain names, but still the grandest one known for the .no top level domain.

But it is important to remember that this can happen to others as well, and if only the parties involved used encryption it could all have been avoided as the recipient would have been unable to read the content of the email.

What about communication between a lawyer and a client? Or the communication between the business leader and consultant? The norwegian firm Synovate performed a survey amongst Norwegian business leaders, lawyers and consultants. The results were not surprising, but still dissapointing. 30% of the participants admits to sending confidential information through email, while at the same time 50% states that it is not safe to send confidential information using email as means.

3.1 History of encryption

The first recorded person to use secure communication was Julius Caesar. He used a shift cipher a variant of a substitution cipher, with a key of three. This means that an A get turned into a D, a B get turned into a E et cetera.

This method has forever since been referred to as the caesar's cipher and it works like this: Write down the alphabet , then write it again, but this time move each character a given number of positions, in this case three.

Plaintext	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
Cipher	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*

Now, if I want to write SECRET in plaintext it would read VHFUHW as ciphertext. It is unknown how effective the Caesar cipher was at the time, but it is likely to have been reasonably secure, not least because few of Caesar's enemies would have been literate, let alone able to consider cryptanalysis, that is the art of deciphering an enciphered message. Today, however, methods such as the Caesar's cipher are considered weak cryptography. Although it might be enough to protect your data from classmates in kindergarten, it won't protect it from any government or against industrial espionage. The focus throughout this book is therefore solely on strong cryptography. But as with the case of the Caesar's cipher, what is strong today, might not be strong tomorrow, as computer power increases and mathematics evolve. Thus one should not claim something to be impenetrable. A conservative view and carefully following the development will get you much further, and the strong encryption utilized by OpenPGP is the strongest available today.

3.2 Industrial espionage

Corporations goes out of their way to protect its proprietary interest. Yet many doesn't focus on securing the technical aspect of the day-to-day operations, mostly due to the lack of knowledge on the subject.

There are known examples where a company with armed guards at the front gate and thick steel doors placed an unprotected wireless access point in the window of the structure. As a result of this, the same data that was protected by armed guards was available to anyone in a car across the street with a wireless-enabled laptop.

There are several examples throughout the history of how important ensuring privacy can be for the strategical outcome. Look at the stock prices of some companies that are about to be merged, and how strongly trading regulations protect insider information.

But the clearest example is probably communication during wartime. It was, for instance of vital importance for the outcome of World War II that the German Enigma cipher was cryptanalyzed (broken). Today Bletchley Park outside London is a museum that can be visited as a reminder of this. There are many fascinating documentaries about Alan Turing and The Bombes that are considered the predecessors of today's computers.

However, as most doesn't want to associate its strategies with those of war let us look at a specific business related example. But first a little information about the Echelon system to get some background information.

3.3 The Echelon surveillance system

On 5 July 2000 the European Parliament decided to set up a temporary committee on the ECHELON system.[7]. The Echelon system is probably the greatest surveillance effort ever established. The US National Security Agency (NSA) has created a global spy system, which captures and analyses virtually every phone call, fax, email and telex message sent anywhere in the world. ECHELON is controlled by the NSA and is operated in conjunction with the Government Communications Head Quarters (GCHQ) of England, the Communications Security Establishment (CSE) of Canada, the Australian Defense Security Directorate (DSD), and the General Communications Security Bureau (GCSB) of New Zealand. These organizations are bound together under a secret 1948 agreement, UKUSA, whose terms and text remain under wraps even today.

The final report that was published show a list of examples in its chapter 10.7 (page 103), of which I want to focus on one of them. This is the Airbus versus Boeing case of 1994. The American National Security Agency used its network to intercept the communication between Airbus and a client both Boeing and Airbus was negotiating with in Saudi Arabia. They did this by intercepting faxes and telephone calls and forwarded the information to Boeing and McDonnell–Douglas. The end result being that the Americans won the 6 billion US dollars contract.

3.4 Possible legal requirements

3.4.1 HIPAA

The American Health Insurance Portability and Accountability Act is a set of rules with recommendations and requirements for entities such as health plans, doctors, hospitals and other health care providers. This regulation challenges all entities to be able to assure that all patients' account handling, billing and medical records should be protected. More information can be found at at http://hipaa.org

3.4.2 Gramm-Leach-Bliley Act

The Gramm-Leach-Bliley Act consists of regulations developed for financial institutions, it is also known as the Financial Modernization Act 1999.

This federal law enables the United States to control financial institutions and the manner in which they handle and process private information of individuals. The Privacy Rules apply to financial institutions and their activities. Affected institutions could also be non bank companies that deal with lending, brokering, auditing, transferring or safeguarding money, preparing return of tax payment, providing financial advice and credit, providing residential real estate settlement services, collecting consumer debts, and more. The Act consists of Privacy obligation policy which emphasizes protection of non-public personal information. More information can be found at http://banking.senate.gov/

Chapter 4

OpenPGP

4.1 Introduction to OpenPGP

OpenPGP is the most widely used email encryption standard in the world. The OpenPGP standard was originally derived from PGP (Pretty Good Privacy), first created by Phil Zimmermann in 1991, and is now maintained by the OpenPGP Working Group of the Internet Engineering Task Force.

One of the great advantages of the framework is that it is flexible in terms of which cryptographical methods is to be used, id est it doesn't depend on one specific digital signature algorithm, or digital encryption algorithm. As such it is sustainable throughout changing times.

Some of the content in this chapter, including information about different key types and digest algorithms is mostly for people with a special interest. Hence you can safely skip over the content and go to the next chapter. I do, however feel it is necessary to write something about it in order to present OpenPGP adequately.

4.2 Implementations

The most common implementations of the OpenPGP standard are the products GNU Privacy Guard that will be discussed at a later point and the commercially available PGP product found at http://www.pgp.com.

I will focus on GnuPG since it is open source software available for free, but there is no obstacle to communicate safely between the different implementations.

4.3 Public Key Infrastructure

The foundation for OpenPGP evolve around something called public key infrastructure, often abbreviated **PKI**. Using PKI, that the special key used to encrypt a message is not the same as used to decrypt it. When a different key is used for encryption and decryption, the scheme is referred to as asymmetrical. If, on the other hand the same key is used both for encryption and decryption, the scheme is referred to as symmetrical. An analogy would be a lock in the daily life. You can safely transmit the locking key, referred to as a public key in an asymmetrical scheme. While you still keep the unlocking key, the private (or secret) key. You can then make the public key available for everyone, but only you keep the private key yourself. So if a neighbour walks by your door to find it unlocked, he or she can lock the door, using the locking key, but still not be able to unlock it again.

4.4 Hybrid system

OpenPGP is a hybrid system. For each new message that is sent, the content of the message is first compressed, then encrypted using a symmetrical block cipher to a random session key of a given length. This session key again is encrypted using the Public Key Infrastructure (PKI) to the different recipients' public keys. Hence, only the session key has to be encrypted multiple times and not the message.

4.5 Asymmetrical key types

Although the default settings are a good choice for most, those who grow interested in knowing what it is doing will want to know about the different types of keys in existence. Although some key types can be used both for encryption and signing, it is generally recommended never to use the same key for both uses at the same time. As a result of this a keyset generally consists of a master signing key and an encryption subkey. More advanced users will also consider using a signing subkey. The advantage of this is that it can be revoked at any time and generate a new signing subkey, but still have signatures from others for the master key in order to keep your position in the web of trust. A signing subkey should always be cross-signed with the master signing key, otherwise you can't know if the subkey truly belongs to the master key without requiring a challenge from the one claiming be the key holder. Historically the PGP key type was RSA version 3, although version 4 keys are more common today. Following is a little information about RSA.

4.5.1 RSA

RSA is short for Rivest, Shamir, Adleman, the names of the creators of the algorithm. RSA keys can, in theory be of any size, although most will want a 2046 bit or a 4096 bit key. Its security is based on the factoring problem, id est it is easy to calculate a product of two primes, but not easy to go from the product to get the two factors again as well as the RSA problem.

Clifford Cocks, a British mathematician working for the UK intelligence agency GCHQ, described an equivalent system in an internal document in 1973, but given the relatively expensive computers needed to implement it at the time, it was mostly considered a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1997 due to its top-secret classification, and Rivest, Shamir, and Adleman devised RSA independently of Cocks' work. (http://en.wikipedia.org/wiki/RSA)

4.5.2 DSA

DSA is the Digital Signature Algorithm, or the Digital Signature Standard (DSS)[2, 3, 6]. Version 1 was adopted in 1994, where the DSA is originally defined with the key length 1024 bits, and a q-size of 160 bits. Support for larger key sizes has since been added, commonly referred to as DSA2. As a result of this older implementations of OpenPGP might lack the ability to properly use DSA2 keys.

As opposed to RSA, DSA requires a specified q-size to be used for signature generation. The relationship between the key sizes defined in DSA2 and the q-sizes, and as such the hashes that can be used is listed in the table below.

DSA key size	**q-size**	**Hashes that can be used**
1024	160	SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512 hash
2048	224	SHA-224, SHA-256, SHA-384 or SHA-512 hash
2048	256	SHA-256, SHA-384 or SHA-512 hash
3072	256	SHA-256, SHA-384 or SHA 512 hash

Using a hash that is greater than the q size will lead to truncation, and the q size will be used for signature generation.

4.5.3 ElGamal

ElGamal is named after the creator, Taher ElGamal and is baseed on the discrete logarithmic problem for security. ElGamal keys are only used for encryptions. Although it is technically possible to generate a signing key using this key type, there have been security reasons not to and the support for such keys are generally not included. The Digital Signature Algorithm is a variant of the ElGamal signature scheme that counters the issues that was discovered, but the method itself is not to be confused with ElGamal.

4.6 Digest Algorithms (hashes)

First of all, let me make very clear that there is a difference between a cryptographic hash, or a digest algorithm, and the procedure commonly referred to as encryption. And the difference is in the ability to reverse the action. When you encrypt something, you want to be able to decrypt it to read it at a later point. This is not true for digest algorithms, they serve a very specific purpose.

4.6.1 SHA-1

The Secure Hash Algorithm (**SHA**) family is a set of related cryptographic hash functions. The SHA algorithms were designed by the National Security Agency (**NSA**) and published as a US government standard. SHA-1 has a size of 160 bits and is traditionally the default digest algorithm.

This algorithm has however been cryptographically broken, in that the operations required to find a collision has been reduced from 2^{80} operations (while factoring in the birthday paradox of a 160 bit original digest length) to at least 2^{63} operations.

Most of the attacks are given plaintext attacks, say you have two different contracts, one that says $1,000 and one that says $1,000,000. You send off the one saying client is supposed to pay $1,000 and the signature is valid for the $1,000,000 too. The basis of this attack is that you had access to alter both messages beforehand by padding using NULL characters. If the client was smart he would have altered the contract slightly before signing to avoid such an attack.

It is however only a matter of time till a collision can be produced of an arbitrary given text, and one should move on to other, and stronger, hashes.

4.6.2 SHA-2 family of hashes

The SHA-2 family of hashes consists of several digest lengths. The most common are sha256 and sha512 The numbers represents the length of the digest algorithms, 256 bits and 512 bits respectively.

These were first released as an official standard[5] in 2002, and as such there are compatibility issues with prior OpenPGP implementations, and should be used accordingly. The other family members are sha384 and sha224, these are however only truncated versions of sha512 and sha256 respectively, and are not seen as much in practice.

4.6.3 RIPEMD160

RACE Integrity Primitives Evaluation Message Digest is a 160-bit message digest algorithm (cryptographic hash function) developed in Europe by Hans Dobbertin, Antoon Bosselaers and Bart Preneel, and first published in 1996. It is an improved version of RIPEMD, which in turn was based upon the design principles used in MD4, and is similar in performance to the more popular SHA-1.

There also exist 128, 256 and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320, respectively. The 128-bit version was intended only as a drop-in replacement for the original RIPEMD, which was also 128-bit, and which had been found to have questionable security. The 256 and 320-bit versions diminish only the chance of accidental collision, and don't have higher levels of security as compared to, respectively, RIPEMD-128 and RIPEMD-160.

RIPEMD-160 was designed in the open academic community, in contrast to the NSA-designed algorithm, SHA-1. On the other hand, RIPEMD-160 is a less popular and correspondingly less well-studied design.[10]

4.7 Symmetrical key types

4.7.1 Advanced Encryption Standard Process

The Advanced Encryption Standard (AES) is a process by the US government. Fifteen different symmetrical block ciphers were submitted to the process, in alphabetical order: CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent, and Twofish. Out of these the five block ciphers: MARS, RC6, Rijndael, Serpent, and Twofish got to the final round of selection, of which Rijndael got selected as AES.

The interesting factor might be that from a purely academic point of view, Rijndael was not the most secure algorithm, however it was the most resource-efficient choice. NIST states "Based on the security analysis performed to-date, there are no known security attacks on any of the five finalists, and all five algorithms appear to have adequate security for the AES. In terms of security margin, MARS, Serpent, and Twofish appear to have high security margins, while the margins for RC6 and Rijndael appear adequate. Some comments criticized Rijndael for its mathematical structure and Twofish for its key separation property; however, those observations have not led to attacks".[4]

4.7.2 AES

As written in the Advanced Encryption Standard Process, Rijndael got selected as the AES. AES is defined with key lengths of 128, 192 and 256 bits, referred to as AES128 (or simply AES), AES192 and AES256.

4.7.3 Twofish

Twofish was an AES candidate provided by Bruce Schneier. Although the candidate did not get selected for AES, he writes “Of course I am disappointed that Twofish didn’t win. But I have nothing but good things to say about NIST and the AES process”.[8]

Twofish utilizes 128 bit data blocks, and can use a key length up till 256 bits.

4.7.4 Blowfish

Blowfish, is as Twofish, designed by Bruce Schenier. It was originally designed to replace the aging Data Encryption Standard and can use a key size between 32 and 448 bits.

There have been concerns about the 64 bit block sizes used, as it can leak information about the cleartext with message lengths greater than 232 data blocks (accounting for the birthday attack). That said, this is very unlikely to affect email messages, but twofish is generally recommended in situations where larger plaintext is to be encrypted (more than 32 gigabytes).

4.7.5 IDEA

IDEA is short for International Data Encryption Algorithm. The cipher was designed under a research contract with the Hasler Foundation, which became part of Ascom-Tech AG. IDEA uses a 128 bit key on 64bit blocks, and is patented in a number of countries.

IDEA was used in Pretty Good Privacy (PGP) V2.0, and was incorporated after the original cipher used in v1.0 ("Bass-O-Matic") was found to be insecure. It is an optional algorithm in OpenPGP. IDEA is patented in at least Austria, France, Germany, Italy, Japan, The Netherlands, Spain, Sweden, Switzerland, The UK and The US. The patents will expire in 2010 - 2011. Today, IDEA is licensed worldwide by MediaCrypt.

IDEA is freely available for non-commercial use, as well as available outside the patent areas, and it is required for backwards compatibility with systems such as PGP 2.0. That said, I would urge the individual to upgrade to a newer system supporting more alternatives.

> **For advanced users only:**
> For GnuPG version 1 adding IDEA support is a relatively trivial

matter (for an advanced user in a situation that requires it in the first place, this should be very few). Windows users can download the pre-compiled binary (.dll) from http://ftp.sunet.se/wmirror/gnupg/documentation/faqs.en.html . Uncompress the archive and put the idea.dll file in `c:\lib\gnupg\` (create the necessary folders). After that add the configuration line "load-extension idea" to gpg.conf. Users of other operating systems will have to compile it from source, information about how this can be done is found in the .c source code.

For GnuPG version 2 adding IDEA support can be slightly more tricky, at least it was, since no compatible version existed for it. I wrote this myself, and made it available at http://www.kfwebs.net/articles/article/42.

Chapter 5

The Web of Trust

5.1 The social problem

Despite its many advantages, the primary disadvantage of being able to send emails in a safe manner is that, to be able to make proper use; as many as possible require compatible systems.

This is a social problem that can only be met by education with regards to safe communication. I hope that, in addition to configuring the system yourself, also will find others that can be interested and help them configure and use OpenPGP and hence start building your own Web of Trust.

This is a natural way to start using OpenPGP. First installing it on your own computer. Then help an acquaintance or a family member to install a compatible system and hence you two will be able to communicate securely. Other persons can then be added to the Web of Trust one by one, expanding it little by little.

As with telephone services, if only one person has a phone, its not worth anything to you. The network is what makes it valuable. Until people understands the necessity for safe communications others won't be able to make proper use of it either. Hopefully this book will make it easier for yourself and others to secure communication your communication.

I recommend advertising that you use OpenPGP yourself in order for others that use it to know that they can communicate with you securely. Personally I have the following in my email signature of every outgoing email:

> ----------------------------
> This email was digitally signed using the **OpenPGP**
> standard. If you want to read more about this, visit:
> `http://www.secure-my-email.com`
> ----------------------------
> Public PGP key 0x6B0B9508 at `http://www.kfwebs.net/pgp/`

5.2 Key Validity / Trust

Now, simply having the public key of a person isn't enough, as anyone can create a keyset with any data and upload it to a keyserver. This is why you should verify with the person that it indeed is their own key. If this is a friend it is generally fairly easy as you can call and verify the key id and fingerprint, and recognize by voice that it is the correct person. Or if it is a business associate and you get to see his driver's license, and note down the key id and fingerprint. But if it is someone more distant it gets more difficult.

That is where the Web of Trust comes into play. To show yourself and others that you have verified that another person's public key in fact belong to the person, and hence confirmed the identity of the owner, you digitally sign the other person's key. You may, or may not, trust that the other person properly verify and sign keys, but if you believe that the individual does, you can use the digital signature verification again to verify your own trust.

Why all this? Now, if someone trust you to properly verify keys, they can know that your friend is who he claims he is, by looking at your digital signature in his or her public key. They don't have to go through the work of verifying the key, you have already done so.

Ok, that got a little bit complex, so let us illustrate it with Figure 5.1 on page 20.

Now, it is important to understand that whether you decide to trust Alice to sign other keys or not is a personal matter, and other's won't see it. Hence if Alice trust Daniel to sign keys, you still won't trust Fredrik.

So, why does this matter? Well, since one of the goals is verifying that emails come from the person that it claim it come from, you need to be able to see if the key belong to the actual person.

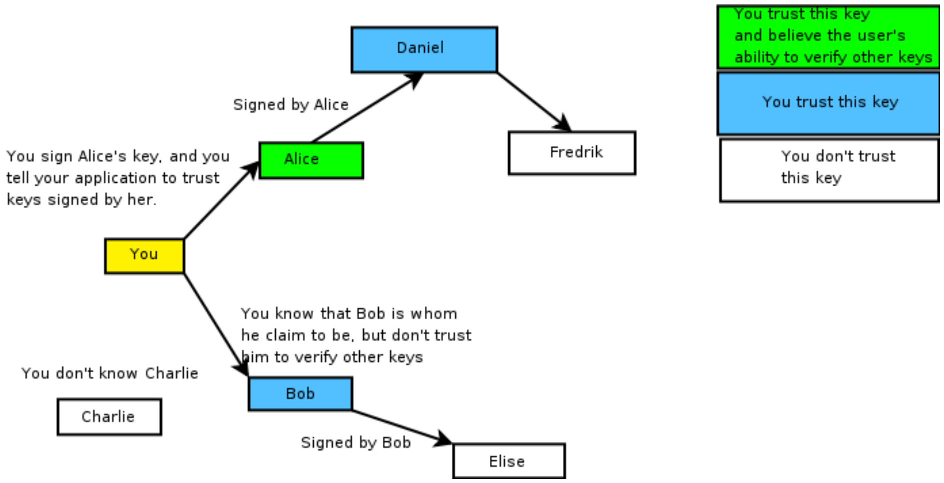

Daniel
Signed by Alice
Alice
You sign Alice's key, and you tell your application to trust keys signed by her.
Fredrik
You
You know that Bob is whom he claim to be, but don't trust him to verify other keys
You don't know Charlie
Charlie
Bob
Signed by Bob
Elise
You trust this key and believe the user's ability to verify other keys
You trust this key
You don't trust this key

Figure 5.1: Web of Trust

5.3 Keyservers

In order to make exchanging public keys easier, a series of key servers for OpenPGP have been configured around the world. These keyservers are storing massive amounts of keys, and synchronizes between each other. Instead of having to send your key to everyone already having your OpenPGP Public key if it changes, e.g. because someone new signs it, you upload it to a keyserver and people refreshes your key from there from time to time. At the time of writing the keyservers hosted by myself both holds about two and a half million keys.

Different keyservers can be found at `http://sks-keyservers.net` , and the round-robin DNS **x-hkp://pool.sks-keyservers.net** can be used to always find an available keyserver. In this context, **HKP** is an abbreviation for Horowitz Keyserver Protocol. As the protocol is not officially recognized by IETF it is often prefixed "x-", but hkp:// refers to the same protocol. HKP works on top of the standard web protocol, Hypertext transfer protocol (HTTP), defaulting to use port 11371 instead of 80, so x-hkp://pool.sks-keyservers.net is the same as specifying `http://pool.sks-keyservers.net:11371`

5.4 Using Certificate Authorities to extend the WoT

As maintaining the Web of Trust can require a great deal of work, there are situations where one wants to assign trust to a Certificate Authority (CA). This is done by default in other crypt systems, such as S/MIME and the HTTPS (SSL).

Using OpenPGP the choice whether or not to trust a certificate authority is given to the user. This gives OpenPGP an advantage over other systems that depends on CAs by default. The most known CAs today are Verisign and Thawte (a VeriSign company) but there are also attempts on communities such as `http://cacert.org` to be a generic certificate authority.

In a corporate setting it can be valuable to have a company signing key that is used when a trusted member of the company has verified a peer's credentials. This key can then be trusted by all employees to extend their Web of Trust, instead of all the employees having to check the credentials on their own.

Chapter 6

A balanced cryptosystem

A cryptosystem is only as strong as its weakest component, which brings up an interesting point regarding what would constitute a balanced cryptosystem. The basis is generally that you will want a minimum of security, generally defined with the basis in equivalence of symmetric key size. Such a relationship is shown in Table 6.1.

Symmetric key size	**Asymmetric Key Length**	**Hash size**
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Table 6.1: Shannon entropy

Say you want a minimum security of 128 bits. This would mean that the asymmetric keys would have to be at least 3072 bits, people would be required to use at least 128 bit symmetric encryption (e.g AES) and the hash used has to be at least 256 bits (e.g SHA256).

Of course, there is a limit to how far it is profitable to talk about the strength of the cryptosystem, as the technical aspects of the security goes over to physical attacks and/or monitoring as well as social engineering techniques at one point. Human behaviour often understate the threat in order to satisfy their own safety needs, often falsely. One example would be purchasing an expensive pick-proof lock at your home door, and thereby feeling comfortable that you have made the home safer for your family and yourself. However, the windows are still just as easy to break through.

Security is, as a result often merely an illusion, an illusion sometimes made even worse when gullibility, naivety, or ignorance come into play. Albert Einstein is quoted as saying: "*Only two things are infinite, the universe and human stupidity, and I'm not sure about the former.*"

Chapter 7

GNU Privacy Guard

7.1 Open Source – The GNU Project

The GNU Project was launched in 1984 to develop a complete UNIX-like operating system which is free software: the GNU system. Variants of the GNU operating system, which use the kernel called Linux, are now widely used; though these systems are often referred to as "Linux", they are more accurately called GNU/Linux systems.

One can read more about the GNU project at `http://gnu.org`

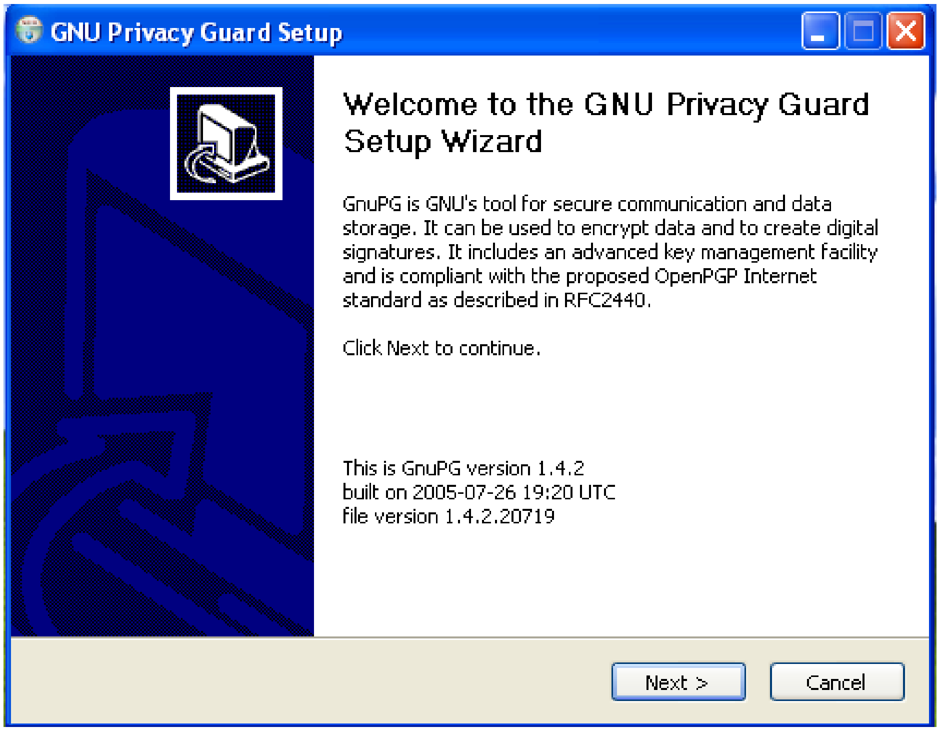
7.2 What is GNU Privacy Guard?

GnuPG is the GNU project's complete and free implementation of the OpenPGP standard that got introduced earlier in this book. GnuPG is Free Software (meaning that it respects your freedom). It can be freely used, modified and distributed under the terms of the GNU General Public License .

Version 1.0.0 was released on September 7th, 1999 after being developed since the first release 0.0.0 in 1997. At the time of writing the current stable versions are `1.4.11` and `2.0.18`. With regards to OpenPGP use, all that is needed is included in the 1.4 series, with version 2.0 adding extended support for S/MIME and features such as the gpg-agent.

7.3 Installing GnuPG

GnuPG can be obtained from http://www.gnupg.org . It already comes installed for most flavours or GNU/Linux, if not it can be installed using the package manager for your distribution. People using Microsoft's Windows, however, will have to install it. Pre-compiled binaries can be downloaded from the project's website. The installation procedure has improved magnificently over the past years, introducing a native windows installer which has helped getting a somewhat more wide-spread usage. Once downloaded you have the file gnupg-w32cli-X.Y.Z.exe. To start the installation simply double click the file, select the language, and press "next" until it completes. Another alternative is to use the package provided by http://www.gpg4win.org/, which include additional tools.



Welcome to the GNU Privacy Guard Setup Wizard

GnuPG is GNU's tool for secure communication and data storage. It can be used to encrypt data and to create digital signatures. It includes an advanced key management facility and is compliant with the proposed OpenPGP Internet standard as described in RFC2440.

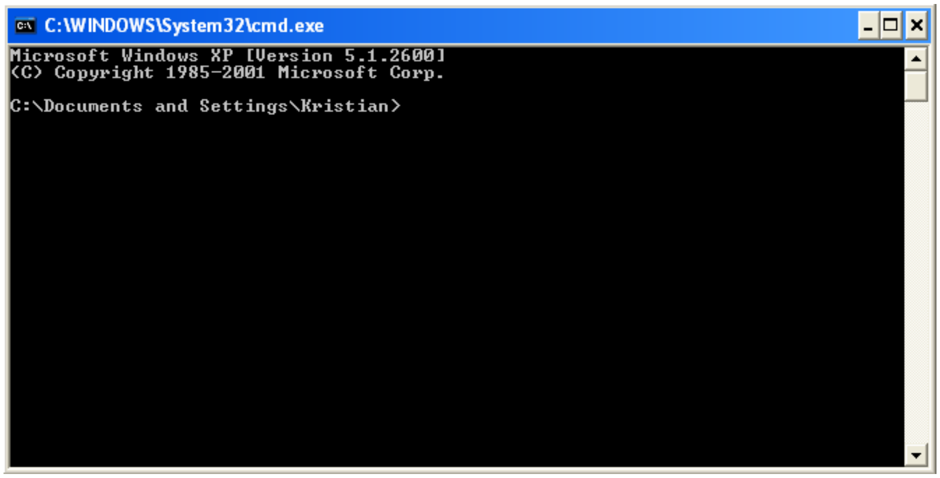
Click Next to continue.

This is GnuPG version 1.4.2
built on 2005-07-26 19:20 UTC
file version 1.4.2.20719

7.4 Generating your first key

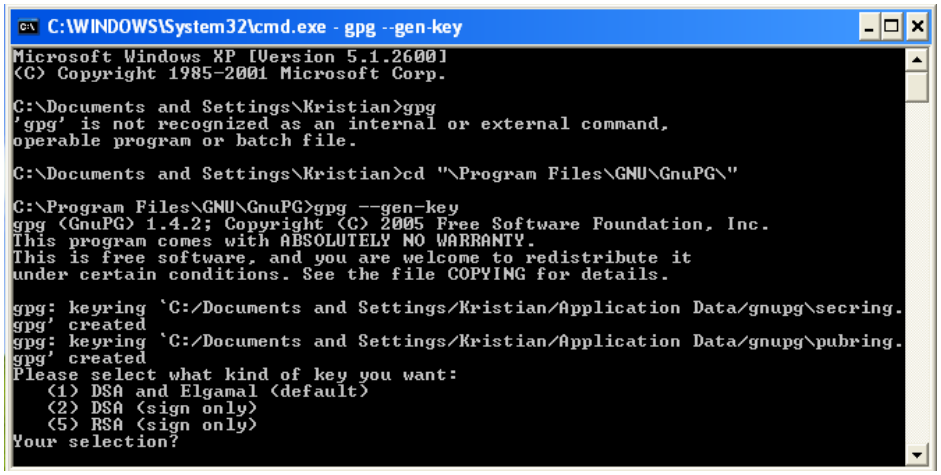
First a disclaimer: Here I'm going to go through the manual steps to generate your first key set in GnuPG, using the command-line interface. This is an operation that first-time wizards in different graphical front-ends will do for you, but it can give some insight into the workings of GnuPG for those interested.

But for those that like to do things manually, in order to have full flexibility, here we go: First you would open a command prompt. For windows one would start by pressing the start button then press run followed by writing "cmd". This will bring up a box that look like:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Kristian>
```

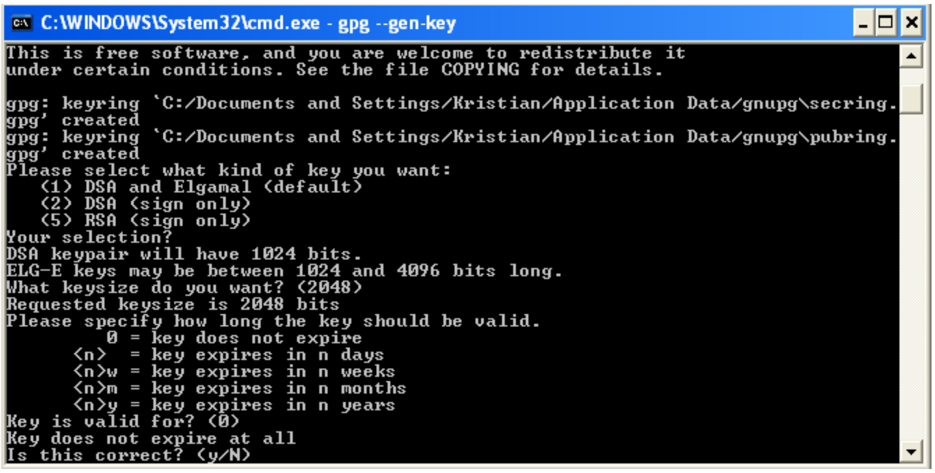
This is the command line, some basic commands are: cd – change directory, e.g. "cd dir1" to go into a folder named dir1 and "cd .." to go back one level. Now you want to navigate to where you installed GnuPG. In my case that is "`c:\Program Files\GNU\GnuPG`" Now you can execute the gpg.exe binary file using the command "gpg". The gpg binary contain everything you need to do. We want to generate a new set of keys. To do that we use "`gpg --gen-key`"



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Kristian>gpg
'gpg' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\Kristian>cd "\Program Files\GNU\GnuPG\"

C:\Program Files\GNU\GnuPG>gpg --gen-key
gpg (GnuPG) 1.4.2; Copyright (C) 2005 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: keyring `C:/Documents and Settings/Kristian/Application Data/gnupg\secring.
gpg' created
gpg: keyring `C:/Documents and Settings/Kristian/Application Data/gnupg\pubring.
gpg' created
Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection?
```

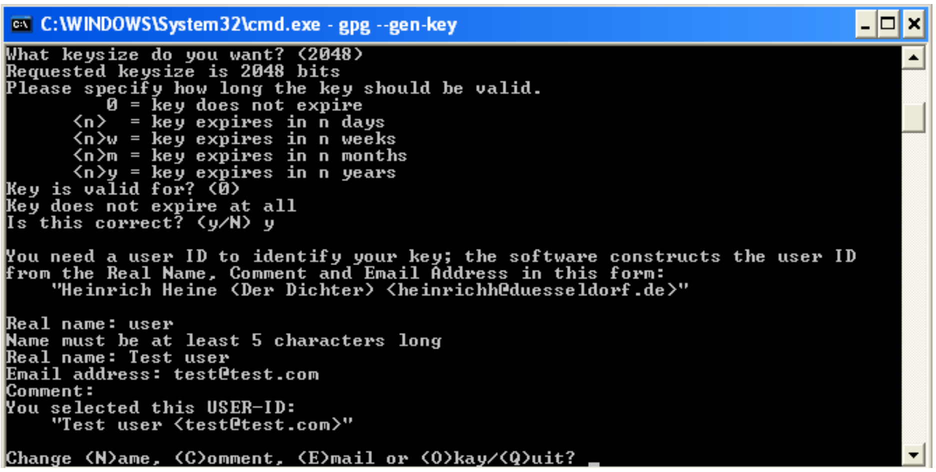
The default option here is a DSA and ElGamal keyset, which is okay for most users. Be aware that if you want to use a digest algorithm such as SHA256 you should select RSA signing key here. If you have such a need, however, you can probably manage from this point onwards without the help of this tutorial.

Pressing the Enter key here use the default option, and bring you to a screen where you can select the size, just press enter here again to use the default, a 2048 bit encryption key and a 1024bit signing key.



```
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: keyring `C:/Documents and Settings/Kristian/Application Data/gnupg\secring.
gpg' created
gpg: keyring `C:/Documents and Settings/Kristian/Application Data/gnupg\pubring.
gpg' created
Please select what kind of key you want:
   (1) DSA and Elgamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection?
DSA keypair will have 1024 bits.
ELG-E keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N)
```

Press enter one more time and you set the key to never expire, and confirm the action by pressing “y”. You will now be asked to write in your information, your full name and your email address.



```
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: user
Name must be at least 5 characters long
Real name: Test user
Email address: test@test.com
Comment:
You selected this USER-ID:
    "Test user <test@test.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? _
```

Once correct press "o" to proceed. You will then be asked to insert a passphrase that is used to protect your private key. Choose one at your will that you will remember.

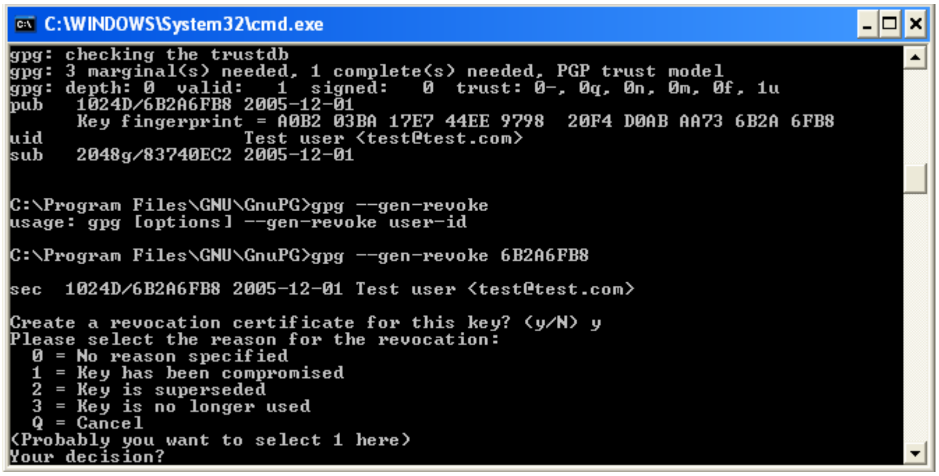
The next step take some time, but you don't have to do anything yourself. This is the actual key generation and require some noise in the system, so contrary to other tasks you might be familiar with, the more you use the computer for the faster it goes

7.5 Generating a revocation certificate

The next thing you want to do is generating a revocation certificate. A revocation certificate is used in the event that you loose your private key or the passphrase associated with it. Importing a revocation certificate into the public key (and subsequently distributing the modified version) will invalidate the key for further use, id est others won't be able to verify messages digitally signed using it, or encrypt messages to it.

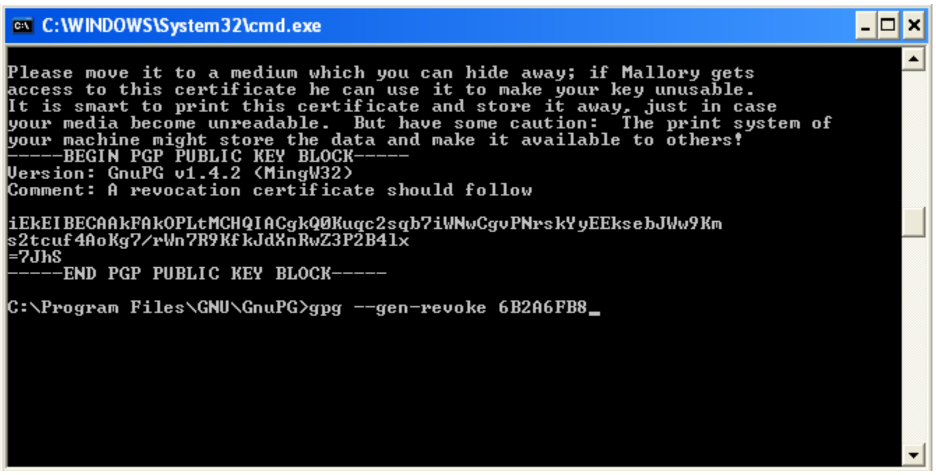
To generate a revocation certificate use the command "`gpg -gen-revoke key-id`"

The key id- as well as other information abut the key you got after the key was generated. In my case the key is 6B2AFB8 and I use "`gpg --gen-revoke 6B2AFB8`" to generate the revocation certificate.



```
gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   1024D/6B2A6FB8 2005-12-01
      Key fingerprint = A0B2 03BA 17E7 44EE 9798  20F4 D0AB AA73 6B2A 6FB8
uid                  Test user <test@test.com>
sub   2048g/83740EC2 2005-12-01


C:\Program Files\GNU\GnuPG>gpg --gen-revoke
usage: gpg [options] --gen-revoke user-id

C:\Program Files\GNU\GnuPG>gpg --gen-revoke 6B2A6FB8

sec  1024D/6B2A6FB8 2005-12-01 Test user <test@test.com>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision?
```

This produce the revocation certificate, in this case it look like



```
Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.2 (MingW32)
Comment: A revocation certificate should follow

iEkEIBECAAkFAkOPLtMCHQIACgkQ0Kuqc2sqb7iWNwCgvPNrskYyEEksebJWw9Km
s2tcuf4AoKg7/rWn7R9KfkJdXnRwZ3P2B41x
=7JhS
-----END PGP PUBLIC KEY BLOCK-----

C:\Program Files\GNU\GnuPG>gpg --gen-revoke 6B2A6FB8_
```

You now have your own keyset that you can use to communicate securely with others. Store the revocation certificate someplace safe, and even print a hard copy of it.

Congratulations you are now able to communicate safely.

7.6 Backing up the keyrings

On NT based Windows systems the two key rings, one containing the private keys and the other the public keys can be found at `\%appdata\%\gnu\gnupg` . In windows 9x based systems, you will most probably find the files pubring.gpg , secring.gpg and trustdb.gpg in the folder that contains the gpg executable. As for Unix-like systems the natural place to look would be $HOME/.gnupg .

No matter which operating system you use; you want to back up the appropriate files.

7.7 Verifying a key

In order to verify a key you will have to establish contact to the key holder in a trusted communication channel. As mentioned before this can be talking to a friend on the phone, or it can be meeting up with someone in person, that you recognize either by appearance or verify using already issued authentication credentials such as a driver's license.

What you want to do is to compare the User ID of the key, as well as the master key ID and the primary key fingerprint. Let me show you how you can get this information using GnuPG. Let us use my own key as an example, with key ID 0x6b0B9508.

By issuing the commandline command: ”`gpg --fingerprint 0x6b0b9508`” (two dashes). I get the following result:

```
[kristianf@kfc002 ~]$ gpg --fingerprint 0x6b0b9508
pub   4096R/6B0B9508 2005-02-21
      Key fingerprint = 65F1 73BE C045 0DA0 7A58  6197 16E0 CF8D 6B0B 9508
uid                  Kristian Fiskerstrand <kristian.fiskerstrand@kfwebs.net>
uid                  Kristian Fiskerstrand <kf@kfwebs.net>
uid                  [jpeg image of size 5187]
sub   4096g/FD83BAC5 2006-11-01 [expires: 2007-12-29]
```

Lets break this information down a bit. 4096R indicates a 4096 bit RSA signing key, the key ID that we looked for and got results for was 6B0B9508. We want to make a note of this key id, as well as the primary key fingerprint we got here, 65F1 73BE C045 0DA0 7A58 6197 16E0 CF8D 6B0B 9508. The key fingerprint is a checksum (using the sha1 digest algorithm) of the public key. We use this to confirm that the key has not been altered in any way from what it was intended.

4096g indicates a 4096 bit ElGamal Encryption Key. There are in fact three of these for my key, as I tend to generate a new one each year, but as the two former are expired, and this one is the last one to be generated it is the only one to show in this window.

There are three different "uid", or User ID fields for my key. The two first are different email addresses, the third is a photo ID, a JPEG stored with the public key in order to do facial recognition of myself.

7.8 How to sign a key using GnuPG

Although different graphical user interfaces (GUI) will allow give you a way to sign keys, it happens that you want to do so using gnupg directly from the command line.

If you were to sign my key you would issue the command "`gpg -edit-key 0x6b0b9508`" (again double dashes is used). You would then write "sign" and press enter. This should give you a question whether you want to sign all user id's which you can answer yes to,if that is, in fact what you want to do, if it isn't you can select which user IDs you want to sign by using "uid 1-3" (as I have three different user ID's). The selected user ids will be selected with an asterisk (*). Follow the steps and exit using "save".

7.9 How to assign trust to a key using GnuPG

As discussed above, trust is vital to how to interpret your web of trust. To construct an example, let us say I am a good friend of yours that you trust to verify other people's public keys. Hence, every key that is signed by my key 0x6b0B9508 is to be trusted by yourself.

After you have signed my key using the information in the section above, you can assign my key a full trust. Let us get back in the edit key prompt by again issuing "`gpg --edit-key 0x6b0B9508`" (again, double dashes are used). But this time let us write "trust" instead of "sign".

This brings up several choices, of which the important ones to us are:

Marginal Trust By default three marginally trusted keys are required to have signed a key before it is considered valid / trusted by yourself.

Full Trust This is the option we want to use in our example, as you fully trust myself to verify other keys. Full trust indicates that all keys signed by this person are also to be considered trusted by yourself.

Ultimate Trust Ultimate trust should only be used on your own keys.

7.10 Configuring GnuPG

GnuPG offers a great deal of configuration options for those wanting to customize the usage. The default settings are sufficient for most but let us look at some possible tweaks nevertheless. The gnupg configuration file is called gpg.conf and if it exist it is found in the same data dir as the key files are stored (ref. the chapter on backing up gnupg). Don't be alarmed if you do not find this file, however, as it is not generated by default. You will simply have to create it.

The natural configuration directive to start looking at is the default-key option . Simply adding a line to the file that, in my case looks like: "`default-key 6B0B9508`" makes this the default key to use for signatures when another isn't specified.

Chapter 8

Organization in a grander scale

As a business or an organization increase in size it gets cumbersome for each member to verify that the public keys really belongs to the different individuals. At this point it is beneficial to introduce a Certificate Authority. This can be the manager of a political team that signs each of the team members keys, or a designated task within the business unit that is to be trusted to properly verify keys.

For a business it would also make sense to have a different signing key for customers and employees. But at that point one adds the keys more importance than just that the user is who he or she claims to be.

At this point, let me emphasizes on the need to revoke signatures of keys whenever an employee leaves the firm, hence invalidate the trust it for later verification.

Chapter 9

Configuring your email client

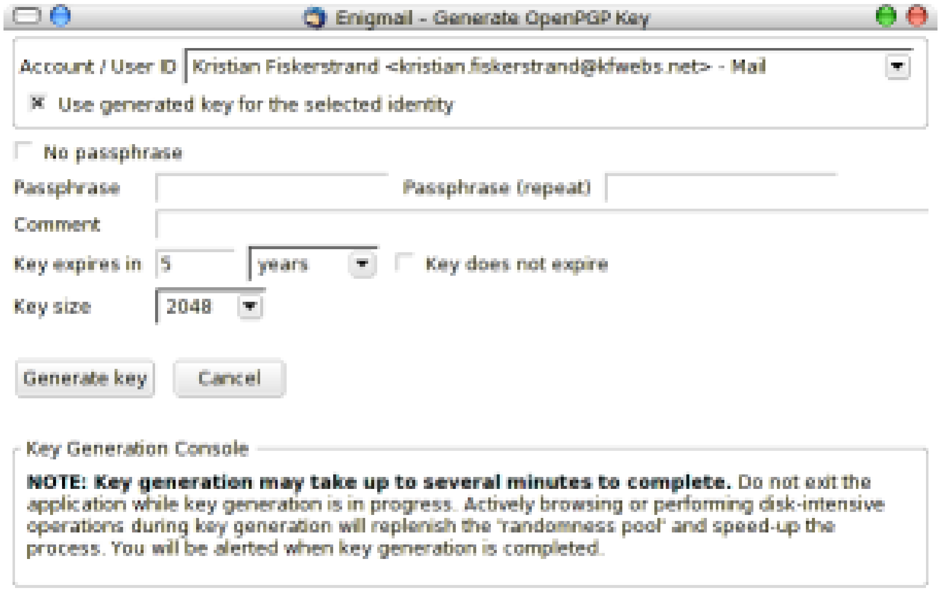
9.1 Mozilla Thunderbird and Suite

Mozilla Thunderbird is the stand-alone, cross-platform, email client provided by The Mozilla Foundation. It use XUL for its interface and mbox for storing files on disk. Mozilla Thunderbird can be downloaded from. http://www.mozilla.com . The primary focus over the Mozilla suite is being small but extensible, and the extensibility support is used by Enigmail.

Enigmail is an extension to Mozilla that add support for OpenPGP as a front-end for the GNU Privacy Guard. Enigmail can be installed using the addon interface in Mozilla Thunderbird. The extension file can be downloaded at http://enigmail.mozdev.org . Full installation instructions are available at http://enigmail.mozdev.org/documentation/quickstart.php.html.

9.1.1 First time use

Starting to use Enigmail is very easy, assuming you have gpg installed according to instructions. Enigmail include an own key management feature, and a first-time wizard is in the making. To start using Enigmail you will first have to generate a key. This can be done using the following procedure. Enigmail>OpenPGP Key Management>Generate>New key pair.



Enigmail - Generate OpenPGP Key

Account / User ID: Kristian Fiskerstrand <kristian.fiskerstrand@kfwebs.net> - Mail

☒ Use generated key for the selected identity

□ No passphrase

Passphrase:

Passphrase (repeat):

Comment:

Key expires in: 5 years □ Key does not expire

Key size: 2048

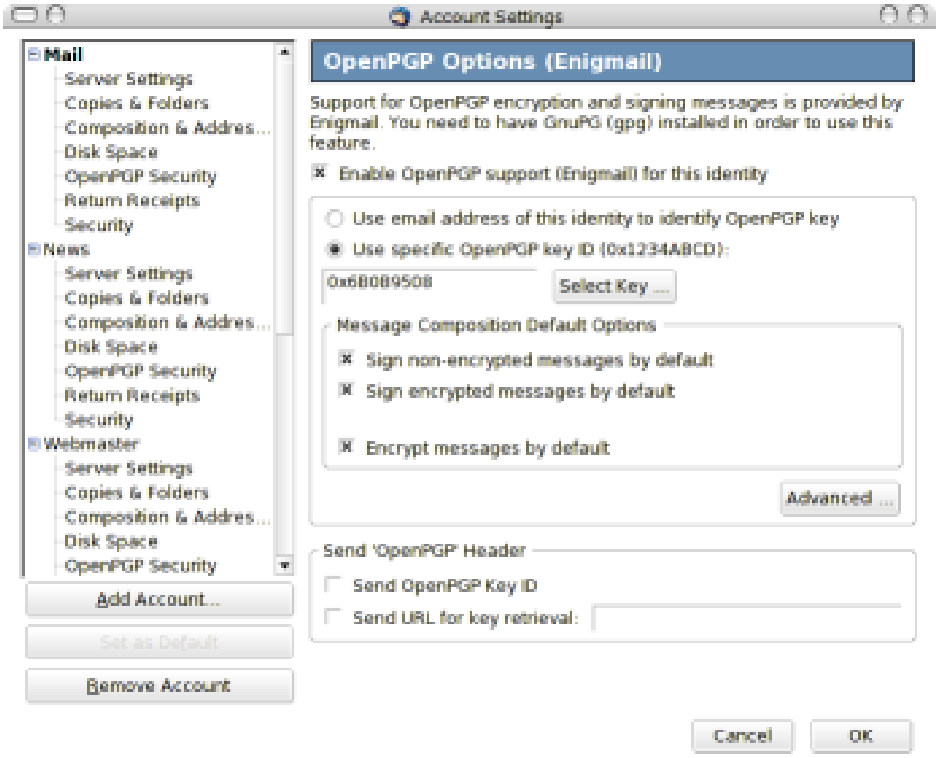
Key Generation Console

NOTE: Key generation may take up to several minutes to complete. Do not exit the application while key generation is in progress. Actively browsing or performing disk-intensive operations during key generation will replenish the 'randomness pool' and speed-up the process. You will be alerted when key generation is completed.

You will be presented with the key generation window. Here you can select which account you want it assigned to, and if you want it to be enabled as soon as the key has been generated. We will have a look at the account configuration afterwards. You're name and email address will be gathered from the account specified. You will and should specify a passphrase for the private key. I suggest setting the key to not expire. As you get more accustomed to GPG you can change the expiration date of the subkey used for encryption only, while the primary key doesn't expire. The default setting is a 2048 bit key, which will produce a 1024 bit DSA signing key and a 2048 bit ElGamal encryption key. These are sane settings for most common usage.

9.1.2 Account settings

You can access the account settings by right-clicking on an account and select properties. After installing Enigmail you will see an OpenPGP Security entry. I prefer to sign all messages by default and also to encrypt all messages by default. Ordinarily this will pop up a dialog every time you try to send an email to anybody you don't have a public key for, but we'll tweak that in the Engimail configurations afterwards.



OpenPGP Options (Enigmail)

Support for OpenPGP encryption and signing messages is provided by Enigmail. You need to have GnuPG (gpg) installed in order to use this feature.

☒ Enable OpenPGP support (Enigmail) for this identity

○ Use email address of this identity to identify OpenPGP key

● Use specific OpenPGP key ID (0x1234ABCD):

0x68089508

Message Composition Default Options

☒ Sign non-encrypted messages by default

☒ Sign encrypted messages by default

☒ Encrypt messages by default

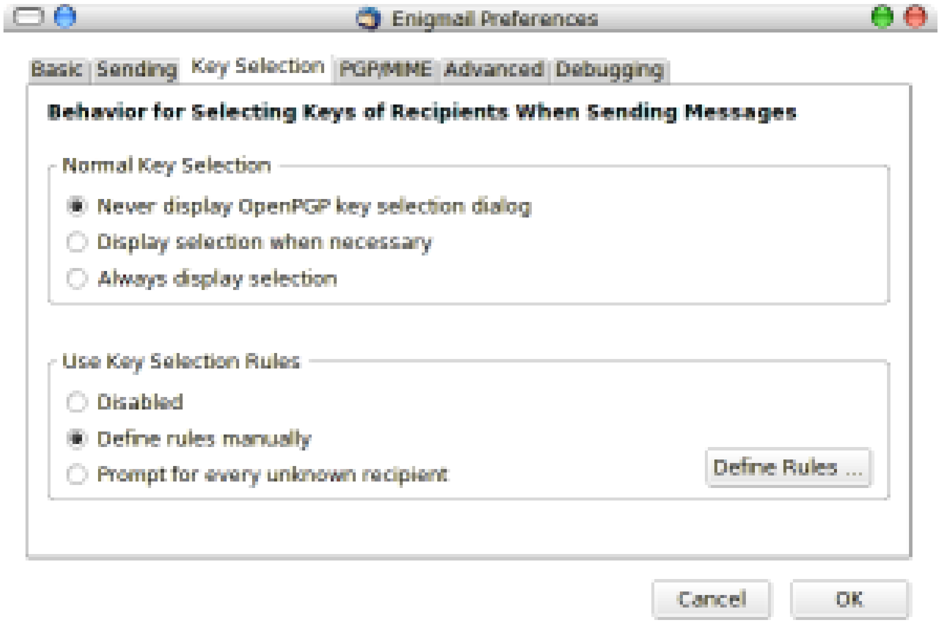
Send 'OpenPGP' Header

□ Send OpenPGP Key ID

□ Send URL for key retrieval:

9.1.3 Enigmail settings

From the previous step we can press advanced to get the Enigmail settings. These can also be found in the Enigmail menu > preferences. In the Key selection tab I prefer to have never show dialog enabled. In addition to the changes in the account settings this will automatically encrypt to everybody you have a public key for, if not, send the email unencrypted without warning.



Behavior for Selecting Keys of Recipients When Sending Messages

Normal Key Selection

● Never display OpenPGP key selection dialog
○ Display selection when necessary
○ Always display selection

Use Key Selection Rules

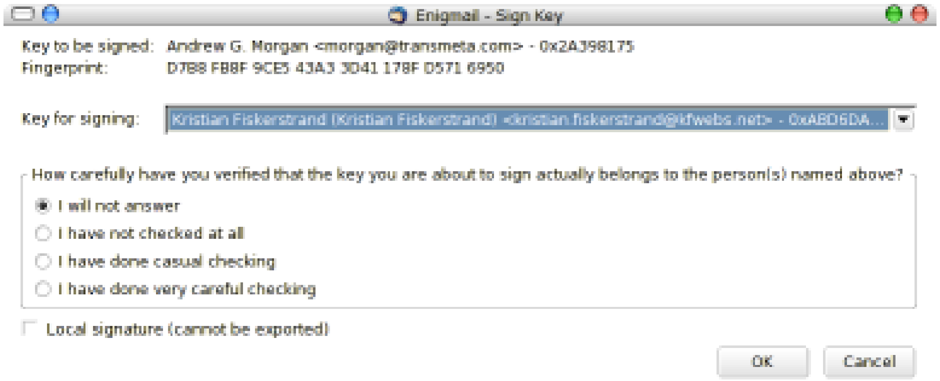
○ Disabled
● Define rules manually
○ Prompt for every unknown recipient

9.1.4 Retrieving keys

You should now be ready to send signed messages, however to encrypt to anybody else, or to verify a signature you need the other persons public key. If the other user has first sent an email, and the key is on a key server this is a rather simple task, involving clicking the pen icon and accepting that it download a new key. If you on the other hand don't have an email message available you would use the Key Management Window. This is also a very simple task involving opening the key management dialog, selecting key server and search for key. Then you can type in the name or email address of the person you want to retrieve the key form and in most cases it will exist

9.1.5 Signing keys

Key signing is vital to OpenPGP. As mentioned earlier in this book, you sign a key to authenticate that you know that the key belong to the person it claim to belong to.



Key to be signed: Andrew G. Morgan <morgan@transmeta.com> - 0x2A398175
Fingerprint: D788 FB8F 9CE5 43A3 3D41 178F D571 6950

Key for signing: Kristian Fiskerstrand (Kristian Fiskerstrand) <kristian.fiskerstrand@kfwebs.net> - 0xABD6DA...

How carefully have you verified that the key you are about to sign actually belongs to the person(s) named above?

● I will not answer
○ I have not checked at all
○ I have done casual checking
○ I have done very careful checking

□ Local signature (cannot be exported)

The Enigmail key management window provide for an easy way to sign keys; You simply right-click on a key listed and select sign key. You will be met by a dialog allowing you to select which key to use to sign, how carefully you have authenticated the key, and a check box where you can mark the signature as non-exportable or local. This implies that the key won't be sent to a keyserver if you send a key.

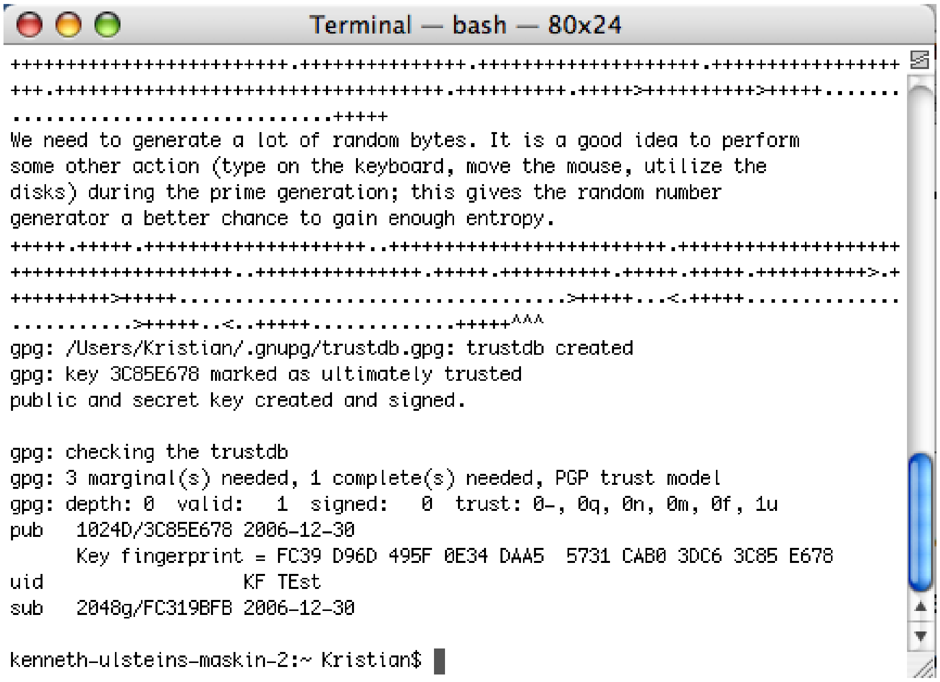
9.1.6 Sending and refreshing keys

For others to see that you have signed a key you will need to distribute the altered public key some way (with your signature on it). The most common way to do so is by using a keyserver. To upload a key to a keyserver you merely right click on the key and select upload keys to keyserver. To refresh a key you will do the same, right clicking and selecting refresh key from keyserver. Refreshing implies downloading the key again so that you can get updated related to the key. New signatures, new subkeys, or in the case the key has been revoked you will be notified. I therefore recommend that you do this once in a while.

9.2 OS X's Mail.app

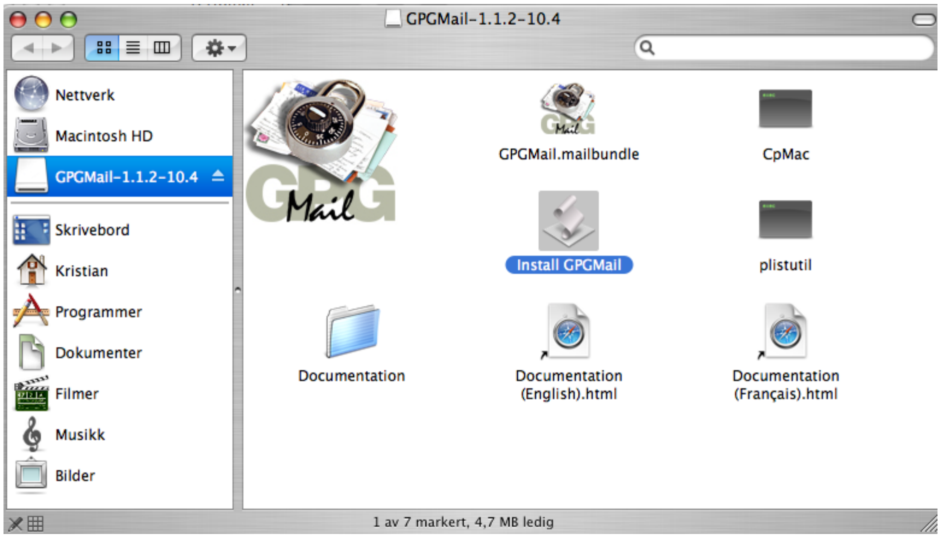
There is a distribution of GnuPG for the Mac platform, using the native installer available at http://macgpg.sourceforge.net.

After MacGPG has been installed, generate your first OpenPGP key. The procedure to do this is found in the chapter GNU Privacy Guard of this book. To get to the console open Terminal in the Utilities folder of Applications.

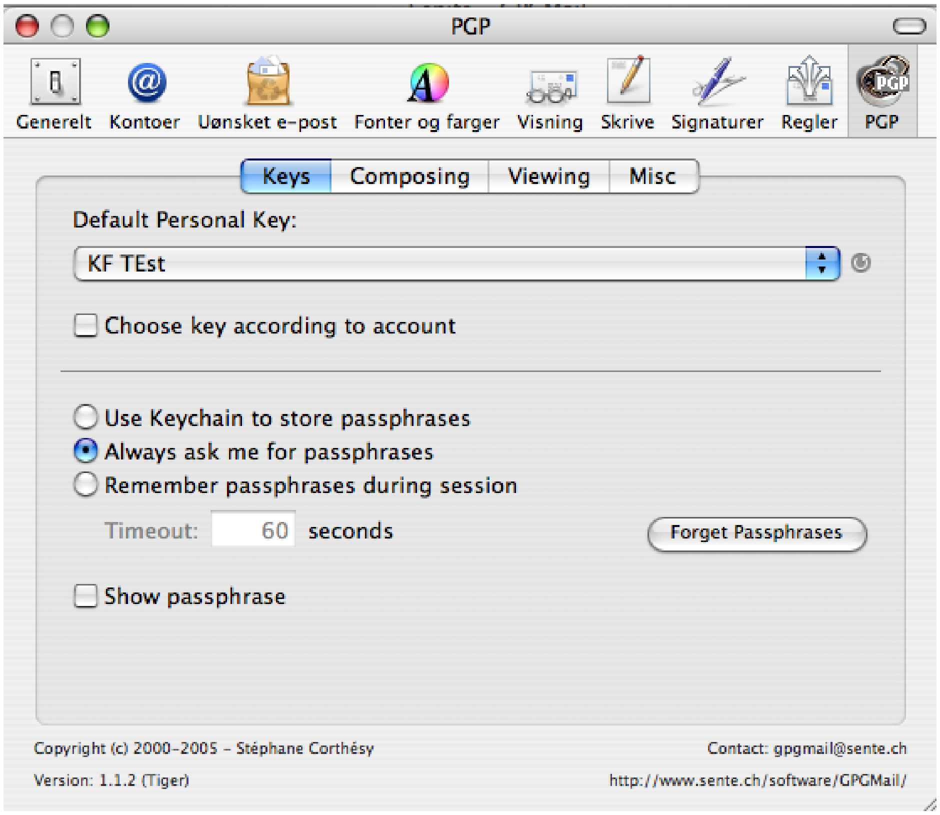


```
+++++++++++++++++++++++++.+++++++++++++++.++++++++++++++++++++.+++++++++++++++++
+++.+++++++++++++++++++++++++++++++++++.++++++++++.+++++>++++++++++>+++++.......
.............................+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++.+++++.++++++++++++++++++++..+++++++++++++++++++++++++.++++++++++++++++++++
++++++++++++++++++++..+++++++++++++++.+++++.++++++++++.+++++.+++++.++++++++++>.+
+++++++++>+++++...................................>+++++...<.+++++..............
...........>+++++..<..+++++.............+++++^^^
gpg: /Users/Kristian/.gnupg/trustdb.gpg: trustdb created
gpg: key 3C85E678 marked as ultimately trusted
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
pub   1024D/3C85E678 2006-12-30
      Key fingerprint = FC39 D96D 495F 0E34 DAA5  5731 CAB0 3DC6 3C85 E678
uid                  KF TEst
sub   2048g/FC319BFB 2006-12-30

kenneth-ulsteins-maskin-2:~ Kristian$ █
```

You can now quit Mail.app if you have it open, and execute the installer of GPGMail.



After it has completed, restart Mail.app and you should find a couple new windows in the configuration section under the PGP tab.



Default Personal Key:

KF TEst

□ Choose key according to account

○ Use Keychain to store passphrases
● Always ask me for passphrases
○ Remember passphrases during session

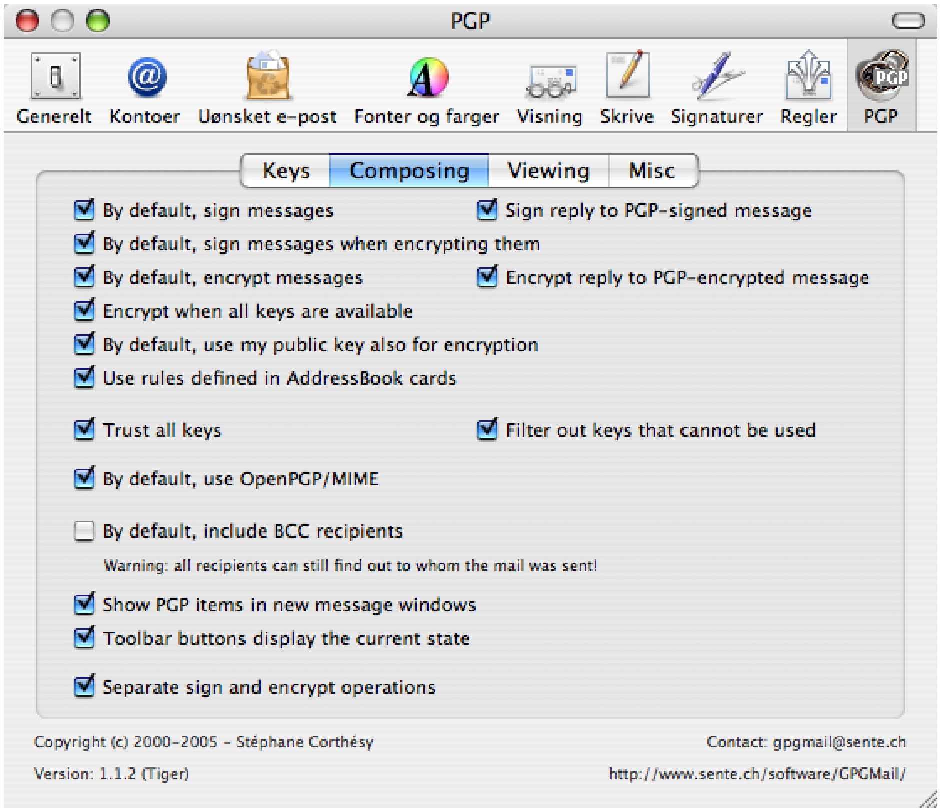
Timeout: 60 seconds

□ Show passphrase

Copyright (c) 2000-2005 - Stéphane Corthésy
Version: 1.1.2 (Tiger)

Contact: gpgmail@sente.ch
http://www.sente.ch/software/GPGMail/

The first window you will be presented with is the Keys tab. If you have multiple secret keys you can select which one to use by default here, or if you want to use one key per account. If you have followed this book and started from scratch you will have only one secret key, so the default here should be ok. So time to move to the second tab, Composing.



☑ By default, sign messages ☑ Sign reply to PGP-signed message
☑ By default, sign messages when encrypting them
☑ By default, encrypt messages ☑ Encrypt reply to PGP-encrypted message
☑ Encrypt when all keys are available
☑ By default, use my public key also for encryption
☑ Use rules defined in AddressBook cards

☑ Trust all keys ☑ Filter out keys that cannot be used

☑ By default, use OpenPGP/MIME

□ By default, include BCC recipients
Warning: all recipients can still find out to whom the mail was sent!

☑ Show PGP items in new message windows
☑ Toolbar buttons display the current state

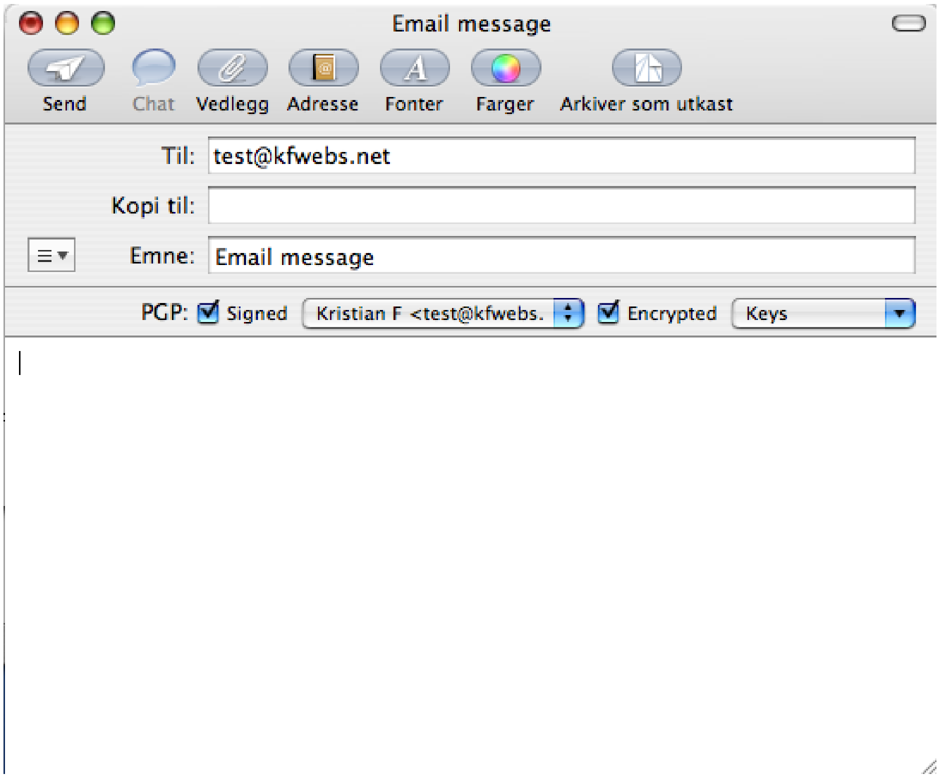
☑ Separate sign and encrypt operations

Copyright (c) 2000-2005 - Stéphane Corthésy
Version: 1.1.2 (Tiger)
Contact: gpgmail@sente.ch
http://www.sente.ch/software/GPGMail/

This tab has quite a lot more options to select from.

A message signed can still be read by others, even though they don't have compatible systems to verify the signature. Therefore I recommend signing all outgoing messages, to enable people that have compatible systems to have added security.

Encryption requires the other party's public keys. The combination of by default, encrypt messages and encrypt when all keys are available is a good one. A compose window is shown below, here you see both PGP signed and encrypted. As soon as you enter an email address that doesn't have a corresponding user id in a public key, it will be deselected by default, and if you want to encrypt the message you will have to select the keys manually.



Til: test@kfwebs.net

Kopi til:

Emne: Email message

PGP: ☑ Signed Kristian F <test@kfwebs. ☑ Encrypted Keys

This email is ready to be sent, both signed and encrypted. So go ahead, press send.

9.3 Using GPG Relay for mail clients in Windows

GPG Relay is an Open Source transparent proxy that operate between your email client and your email server. This mean that you can use OpenPGP even though your email client doesn't have support for it directly. The project website is http://sites.inka.de/tesla/gpgrelay.html

Before installing GPGRelay you should have the GNU Privacy Guard installed, information can be found in the chapter GNU Privacy Guard.

The GPG Relay is a simple windows installation file, and is just to double click to install. After installation you will be presented the option to run GPGRelay. If you don't have a keypair and/or keyring installed from before, you can let GPG-Relay create one for you. If none is detected you will be presented with the following window

You can download the installation file from GPGRelay's website. If you want GPGRelay to handle Secure Socket Layer (SSL), you will also download http://sites.inka.de/tesla/download/OpenSSL-0.9.7e.dll.zip . This is an ordinary zip-file, that can be unzipped e.g. using WinZip or the un-archiver in Windows. The unarchived files can be copied into the directory where you chose to install GPGRelay.

More information on use of GPGRelay is available at http://secure-my-email.com/clients_gpgrelay.php

9.4 Evolution

Evolution is an integrated solution for email, contacts and calendar. It was originally developed by Novell but is now an Open Source project. It is first and foremost for the GNOME2 desktop system, but it work well for KDE3 as well, of which the screenshots of this article is based on.

Evolution include several features, including SpamAssassin for junk filtering. I'll focus most on the feature they call "Security and encryption". Evolution include support for both S/MIME and OpenPGP, we will however only focus on the OpenPGP support.

Evolution use GNU Privacy Guard5 for its OpenPGP support. If you have this configured already it will start using Evolution with it straight away, as it automatically verifies signatures and decrypts messages. If you don't have GnuPG configured, please read the chapter GNU Privacy Guard before proceeding.

After that, the direct support makes it very easy to use.

9.4.1 Small drawback

One thing I encountered when testing Evolution is that it (confirmed with at least versions 2.2.3 and 2.4.2.1) require the encrypted data to be RFC 1847 Encapsulated. It does not support the other method of both digitally signing and encrypting defined in RFC 3156[1, Ch. 6.1-6.2].

The ramifications of this is that although Evolution will decrypt an OpenPGP/MIME message sent this way, it will not show the box for a valid signature. Other email clients, such as Mozilla Thunderbird with Enigmail as an extension to handle the security use the combined method by default when sending a message using OpenPGP/MIME. It will be able to properly verify both methods. The workaround to this quirk is to use Inline messages instead of OpenPGP/MIME.

More information on use of Evolution is available at `http://secure-my-email.com/clients_evolution.php`

References

[1] Internet Engineering Task Force. Mime security with openpgp, Aug 2001. URL http://www.ietf.org/rfc/rfc3156.txt.

[2] National Institute of Standards and Technology. Digital signature standard (DSS) v1, 1994. URL http://www.itl.nist.gov/fipspubs/fip186.htm.

[3] National Institute of Standards and Technology. Digital signature standard (DSS) v2, 2000. URL http://www.cs.haifa.ac.il/~orrd/IntroToCrypto/online/fips186-2.pdf.

[4] National Institute of Standards and Technology. Report on the development of the advanced encryption standard (AES), 2000. URL http://csrc.nist.gov/archive/aes/round2/r2report.pdf.

[5] National Institute of Standards and Technology. Secure hash standard, 2002. URL http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[6] National Institute of Standards and Technology. Digital signature standard (DSS) v3, 2009. URL http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[7] European Parliament. Report on the existence of a global system for the interception of private and commercial communications (echelon interception system) (2001/2098(ini)), 2001. URL http://cryptome.org/echelon-ep-fin.htm.

[8] Bruce Schneier. Crypto-gram newsletter, october 15, 2000. URL http://www.schneier.com/crypto-gram-0010.html#8.

[9] Alma Whitten and J.D. Tygar. Usability of security: A case study, Dec 1998. URL http://reports-archive.adm.cs.cmu.edu/anon/1998/abstracts/98-155.html.

[10] Wikipedia. Ripemd-160. URL http://en.wikipedia.org/wiki/RIPEMD.