Can We Trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3

Phong Q. Nguyen

CNRS/École normale supérieure
Département d'informatique
45 rue d'Ulm, 75230 Paris Cedex 05, France.
Phong.Nguyen@ens.fr
http://www.di.ens.fr/~pnguyen

Abstract. More and more software use cryptography. But how can one know if what is implemented is good cryptography? For proprietary software, one cannot say much unless one proceeds to reverse-engineering, and history tends to show that bad cryptography is much more frequent than good cryptography there. Open source software thus sounds like a good solution, but the fact that a source code can be read does not imply that it is actually read, especially by cryptography experts. In this paper, we illustrate this point by examining the case of a basic Internet application of cryptography: secure email. We analyze parts of the source code of the latest version of GNU Privacy Guard (GnuPG or GPG), a free open source alternative to the famous PGP software, compliant with the OpenPGP standard, and included in most GNU/Linux distributions such as Debian, MandrakeSoft, Red Hat and SuSE. We observe several cryptographic flaws in GPG v1.2.3. The most serious flaw has been present in GPG for almost four years: we show that as soon as one (GPG-generated) ElGamal signature of an arbitrary message is released, one can recover the signer's private key in less than a second on a PC. As a consequence, ElGamal signatures and the so-called ElGamal sign+encrypt keys have recently been removed from GPG. Fortunately, ElGamal was not GPG's default option for signing keys.

Keywords: Public-key cryptography, GnuPG, GPG, OpenPGP, Cryptanalysis, RSA, ElGamal, Implementation.

1 Introduction

With the advent of standardization in the cryptography world (RSA PKCS [20], IEEE P1363 [14], CRYPTREC [15], NESSIE [8], *etc.*), one may think that there is more and more good cryptography. But as cryptography becomes "global", how can one be sure that what is implemented in the real world is actually good cryptography? Numerous examples

(such as [4, 2, 21]) have shown that the frontier between good cryptography and bad cryptography is very thin. For proprietary software, it seems difficult to make any statement unless one proceeds to the tedious task of reverse-engineering. If a proprietary software claims to implement 2048-bit RSA and 128-bit AES, it does not say much about the actual cryptographic security: which RSA is being used? Could it be textbook RSA [5] (with zero-padding) encrypting a 128-bit AES key with public exponent 3? Are secret keys generated by a weak pseudo-random number generator like old versions of Netscape [9]? Who knows if it is really RSA–OAEP which is implemented [21]? With proprietary software, it is ultimately a matter of trust: unfortunately, history has shown that there is a lot of bad cryptography in proprietary software (see for instance [28, 12] for explanations). Open source software thus sounds like a good solution. However, the fact that a source code can be read does not necessarily imply that it is actually read, especially by cryptography experts.

The present paper illustrates this point by examining the case of "perhaps the most mature cryptographic technology in use on the Internet" (according to [1]): secure email, which enables Internet users to authentify and/or encrypt emails. Secure email became popular in the early 90s with the appearance of the now famous Pretty Good Privacy (PGP) [27] software developed by Phil Zimmermann in the US. Not so long ago, because of strict export restrictions and other US laws, PGP was unsuitable for many areas outside the US. Although the source code of PGP has been published, it is unknown whether future versions of PGP will be shipped with access to the source code.

GNU Privacy Guard [10] (GnuPG, or GPG in short) was developed in the late 90s as an answer to those PGP issues. GPG is a full implementation of OpenPGP [26], the Internet standard that extends PGP. GPG has been released as free software under the GNU General Public License (GNU GPL): As such, full access to the source code is provided at [10], and GPG can be viewed as a free replacement for PGP. The German Federal Ministry of Economics and Technology granted funds for the further development of GPG. GPG has a fairly significant user base: it is included in most GNU/Linux distributions, such as Debian, MandrakeSoft, Red Hat and SuSE. The first stable version of GPG was released on September 7th, 1999. Here, we review the main public-key aspects of the source code of v1.2.3, which was the current stable version (released on August 22nd, 2003) when this paper was submitted to Eurocrypt '04. Our comments seem to also apply to several previous versions of GPG.

However, we stress that our analysis is not claimed to be complete, even for the public-key aspects of GPG.

We observe several cryptographic flaws in GPG v1.2.3. The most serious flaw (which turns out to have been present in GPG for almost four years) is related to ElGamal signatures: we present a lattice-based attack which recovers the signer's private key in less than a second on a PC, given any (GPG-generated) ElGamal signature of a (known) arbitrary message and the corresponding public key. This is because both a short private exponent and a short nonce are used for the generation of ElGamal signatures, when the GPG version in use is between 1.0.2 (January 2000) and 1.2.3 (August 2003). As a result, GPG–ElGamal signing keys have been considered compromised [19], especially the so-called primary ElGamal sign+encrypt keys: with such keys, one signature is always readily available, because such keys automatically come up with a signature to bind the user identity to the public key, thus leaking the private key used for both encryption and signature. Hence, ElGamal signatures and ElGamal sign+encrypt keys have recently been removed from GPG (see [10] for more information).

We notice that GPG encryption provides no chosen-ciphertext security, due to its compliance with OpenPGP [26], which uses the old PKCS #1 v1.5 standard [17]: Bleichenbacher's chosen-ciphertext attack [4] applies to OpenPGP, either when RSA or ElGamal is used. Although the relevance of chosen-ciphertext attacks to the context of email communications is debatable, we hope that OpenPGP will replace PKCS #1 v1.5 to achieve chosen-ciphertext security. The other flaws do not seem to lead to any reasonable attack, they only underline the lack of state-of-the-art cryptography in GPG and sometimes OpenPGP. It is worth noting that the OpenPGP standard is fairly loose: it gives non-negligible freedom over the implementation of cryptographic functions, especially regarding key generation. Perhaps stricter and less ambiguous guidelines should be given in order to decrease security risks.

The only published research on the cryptographic strength of GPG we are aware of is [18, 16], which presented chosen-ciphertext attacks with respect to the symmetric encryption used in PGP and GPG. The rest of the paper is organized as follows. In Section 2, we give an overview of the GPG software v1.2.3. In Section 3, we review the GPG implementation of ElGamal and present the attack on ElGamal signatures. In Section 4, we review the GPG implementation of RSA. There is no section devoted to the GPG implementation of DSA, since we have not found any noteworthy weakness in it. In Appendix A, we give a brief introduction to lattice

theory, because the attack on ElGamal signatures uses lattices. In Appendix B, we provide a proof (in an idealized model) of the lattice-based attack on ElGamal signatures.

2 An Overview of GPG v1.2.3

GPG v1.2.3 [10] supports ElGamal (signature and encryption), DSA, RSA, AES, 3DES, Blowfish, Twofish, CAST5, MD5, SHA-1, RIPE-MD-160 and TIGER. GPG decrypts and verifies PGP 5, 6 and 7 messages: It is compliant with the OpenPGP standard [26], which is described in RFC 2440 [6].

GPG provides secrecy and/or authentication to emails: it enables users to encrypt/decrypt and/or sign/verify emails using public-key cryptography. The public-key infrastructure is the famous web of trust: users certify public key of other users.

GPG v.1.2.3 allows the user to generate several types of public/private keys, with the command `gpg --gen-key`:

- Choices available in the standard mode:
  - (1) DSA and ElGamal: this is the default option. The DSA keys are signing keys, while the ElGamal keys are encryption keys (type 16 in the OpenPGP terminology).
  - (2) DSA: only for signatures.
  - (5) RSA: only for signatures.
- Additional choices available in the expert mode:
  - (4) ElGamal for both signature and encryption. In the OpenPGP terminology, these are keys of type 20.
  - (7) RSA for both signature and encryption.

In particular, an ElGamal signing key is also an encryption key, but an ElGamal encryption key may be restricted to encryption. In GPG v1.2.3, ElGamal signing keys cannot be created unless one runs the expert mode: however, this was not always the case in previous versions. For instance, the standard mode of GPG v1.0.7 (which was released in April 2002) proposes the choices (1), (2), (4) and (5).

2.1 Encryption

GPG uses hybrid encryption to encrypt emails. A session key (of a symmetric encryption scheme) is encrypted by a public-key encryption scheme: either RSA or ElGamal (in a group $\mathbb{Z}_p^*$, where p is a prime number). The

session key is formatted as specified by OpenPGP (see Figure 1): First, the session key is prefixed with a one-octet algorithm identifier that specifies the symmetric encryption algorithm to be used; Then a two-octet checksum is appended which is equal to the sum of the preceding session key octets, not including the algorithm identifier, modulo 65536.

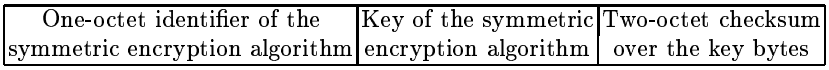

Fig. 1. Session key format in OpenPGP.

This value is then padded as described in PKCS#1 v1.5 block type 02 (see [17] and Figure 2): a zero byte is added to the left, as well as as many non-zero random bytes as necessary in such a way that the first two bytes of the final value are 00 02 followed by as many nonzero random bytes as necessary, and the rest. Note that this formatting is applied to both RSA and ElGamal encryption, even though PKCS#1 v1.5 was only designed for RSA encryption. The randomness required to generate nonzero random bytes is obtained by a process following the principles of [11].

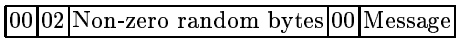

Fig. 2. PKCS#1 v1.5 encryption padding, block type 02.

2.2 Signature

GPG supports the following signature schemes: RSA, DSA and ElGamal. The current GPG FAQ includes the following comment: *As for the key algorithms, you should stick with the default (i.e., DSA signature and ElGamal encryption). An ElGamal signing key has the following disadvantages: the signature is larger, it is hard to create such a key useful for signatures which can withstand some real world attacks, you don't get any extra security compared to DSA, and there might be compatibility problems with certain PGP versions. It has only been introduced because at the time it was not clear whether there was a patent on DSA.* The `README` file of GPG includes the following comment: *ElGamal for signing is available,*

*but because of the larger size of such signatures it is strongly deprecated (Please note that the GnuPG implementation of ElGamal signatures is *not* insecure).* Thus, ElGamal signatures are not really recommended (mainly for efficiency reasons), but they are nevertheless supported by GPG, and they were not supposed to be insecure.

When RSA or ElGamal is used, the message is first hashed (using the hash function selected by the user), and the hash value is encoded as described in PKCS#1 v1.5 (see [17] and Figure 3): a certain constant (depending on the hash function) is added to the left, then a zero byte is added to the left, as well as as many FF bytes as necessary in such a way that the first two bytes of the final value are 00 01 followed by the FF bytes and the rest. With DSA, there is no need to apply a signature padding,

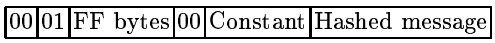

Fig. 3. PKCS#1 v1.5 signature padding, block type 01.

as the DSS standard completely specifies how a message is signed.

The randomness required by ElGamal and DSA is obtained by a process following the principles of [11].

3 The Implementation of ElGamal

GPG uses the same key generation for signature and encryption. It implements ElGamal in a multiplicative group $\mathbb{Z}_p^*$ (where p is a large prime) with generator g. The private key is denoted by x, and the corresponding public key is $y = g^x \pmod{p}$.

3.1 Key generation

The large prime number p is chosen in such a way that the factorization of $p-1$ is completely known and all the prime factors of $(p-1)/2$ have bit-length larger than a threshold q_{bit} depending on the requested bit-length of p. The correspondance between the size of p and the threshold is given by the so-called Wiener table (see Figure 4): notice that $4q_{bit}$ is always less than the bit-length of p.

Once p is selected, a generator g of $\mathbb{Z}_p^*$ is found by testing successive potential generators (thanks to the known factorization of $p-1$), starting with the number 3: If 3 turns out not to be a generator, then one tries with

Bit-length of p	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
q_{bit}	119	145	165	183	198	212	225	237	249	259	269	279	288	296

Fig. 4. The Wiener table used to generate ElGamal primes.

4, and so on. The generation of the pair (p, g) is specified in the procedure `generate_elg_prime` of the file `cipher/primegen.c`. The generation of the pair of public and private keys is done in the procedure `generate` of the file `cipher/elgamal.c`. Although the generator g is likely to be small, we note that because all the factors of $(p-1)/2$ have at least $q_{bit} \geq 119$ bits, and $g > 2$, Bleichenbacher's forgery [3] of ElGamal signatures does not seem to apply here.

The private exponent x is not chosen as a pseudo-random number modulo $p-1$, although GPG makes the following comment: *select a random number which has these properties:* $0 < x < p-1$. *This must be a very good random number because this is the secret part.* Instead, x is chosen as a pseudo-random number of bit-length $3q_{bit}/2$, which is explained by the following comment (and which somehow contradicts the previous one): *I don't see a reason to have a* x *of about the same size as the* p. *It should be sufficient to have one about the size of* q *or the later used* k *plus a large safety margin. Decryption will be much faster with such an* x. Thus, one chooses an x much smaller than p to speed-up the private operations. Unfortunately, we will see that this has implications on the security of GPG–ElGamal signatures.

3.2 Signature

Description. The signature of a message already formatted as an integer m modulo p (as decribed in Section 2.2), is the pair (a, b) where: $a = g^k \bmod p$ and $b = (m - ax)k^{-1} \pmod{p-1}$. The integer k is a "random" number coprime with $p-1$, which must be generated at each signature. GPG verifies a signature (a, b) by checking that $0 < a < p$ and $y^a a^b \equiv g^m \pmod{p}$. We note that such a signature verification does not prevent malleability (see [30] for a discussion on malleability): if (a, b) is a valid signature of m, then $(a, b + u(p-1))$ is another valid signature of m for all integer u, because there is no range check over b. This is a minor problem, but there is worse.

Theoretically, k should be a cryptographically secure random number modulo $p-1$ such that k is coprime to $p-1$. Recent attacks on discrete-log signature schemes (see [22, 2, 13]) have shown that any leakage of information (or any peculiar property) on k may enable an attacker to recover

the signer's private key in a much shorter time than what is usually required to solve discrete logarithms, provided that sufficiently many message/signature pairs are available. Intuitively, if partial information on k is available, each message/signature pair discloses information on the signer's private key, even though the signatures use different k's: when enough such pairs are gathered, it might become possible to extract the private key. Unfortunately, the GPG generation of k falls short of such recommendations.

The generation of k is described in the procedure `gen_k` of the file `cipher/elgamal.c`. It turns out that k is first chosen with $3q_{bit}/2$ pseudo-random bits (as in the generation of the private exponent x, except that k may have less than $3q_{bit}/2$ bits). Next, as while as k is not coprime with $p-1$, k is incremented. Obviously, the final k is much smaller than p, and therefore far from being uniformly distributed modulo $p-1$: the bit-length of k should still be around $3q_{bit}/2$, while that of p is at least $4q_{bit}$. This is explained in the following comment: *IMO using a k much lesser than p is sufficient and it greatly improves the encryption performance. We use Wiener's table and add a large safety margin.* One should bear in mind that the same generation of k is used for both encryption and signature. However, the choice of a small k turns out to be dramatic for signature, rather than for encryption.

Attacking GPG–ElGamal signatures. Independently of the choice of the private exponent x, because k is much smaller than $p-1$, one could apply the lattice-based attack of Nguyen-Shparlinski [22] with very slight modifications, provided that a few signatures are known. However, because x is so small, there is even a simpler attack, using only a single signature! Indeed, we have the following congruence:

$$bk + ax \equiv m \pmod{p-1}. \tag{1}$$

In this congruence, only k and x are unknowns, and they are unusually small: From Wiener's table (Figure 4), we know that k and x are much smaller than $\sqrt{p}$.

Linear congruences with small unknowns occur frequently in public-key cryptanalysis (see for instance the survey [24]), and they are typically solved with lattice techniques. We assume that the reader is familiar with lattice theory (see Appendix A and the references of [24]). Following the classical strategy described in [24], we view the problem as a closest vector problem in a two-dimensional lattice, using lattices defined by

a single linear congruence. The following lemma introduces the kind of two-dimensional lattices we need:

Lemma 1. *Let $(\alpha, \beta) \in \mathbb{Z}^2$ and n be a positive integer. Let d be the greatest common divisor of α and n. Let e be the greatest common divisor of α, β and n. Let L be the set of all $(u, v) \in \mathbb{Z}^2$ such that $\alpha u + \beta v \equiv 0 \pmod{n}$. Then:*

1. *L is a two-dimensional lattice in $\mathbb{Z}^2$.*
2. *The determinant of L is equal to n/e.*
3. *There exists $u \in \mathbb{Z}$ such that $\alpha u + (\beta/e)d \equiv 0 \pmod{n}$.*
4. *The vectors $(n/d, 0)$ and $(u, d/e)$ form a basis of L.*

Proof. By definition, L is a subgroup of $\mathbb{Z}^2$, hence a lattice. Besides, L contains the two linearly independent vectors $(n, 0)$ and $(0, n)$, which proves statement 1. Let f be the function that maps $(u, v) \in \mathbb{Z}^2$ to $\alpha u + \beta v$ modulo n. f is a group morphism between $\mathbb{Z}^2$ and the additive group $\mathbb{Z}_n$. The image of f is the subgroup (of $\mathbb{Z}_n$) spanned by the greatest common divisor of α and β: it follows that the image of f has exactly n/e elements. Since L is the kernel of f, we deduce that the index of L in $\mathbb{Z}^2$ is equal to n/e, which proves statement 2. Statement 3 holds because the greatest common divisor of α and n is d, which divides the integer $(\beta/e)d$. By definition of u, the vector $(u, d/e)$ belongs to L. Obviously, the vector $(n/d, 0)$ belongs to L. But the determinant of those two vectors is n/e, that is, the determinant of L. This proves statement 4. □

We use the following lattice:

$$L = \{(u, v) \in \mathbb{Z}^2 : bu + av \equiv 0 \pmod{p-1}\}. \tag{2}$$

By Lemma 1, a basis of L can easily be found. We then compute an arbitrary pair $(k', x') \in \mathbb{Z}^2$ such that $bk' + ax' \equiv m \pmod{p-1}$. To do so, we can apply the extended Euclidean algorithm to express the greatest common divisor of a, b and $p-1$ as a linear combination of a, b and $p-1$. This gcd must divide m by (1), and therefore, a suitable multiplication of the coefficients of the linear combination gives an appropriate (k', x').

The vector $\boldsymbol{l} = (k' - k, x' - x)$ belongs to L and is quite close to the vector $\boldsymbol{t} = (k' - 2^{3q_{bit}/2-1}, x' - 2^{3q_{bit}/2-1})$. Indeed, k has about $3q_{bit}/2$ bits and x has exactly $3q_{bit}/2$ bits, therefore the distance between $\boldsymbol{l}$ and $\boldsymbol{t}$ is about $2^{(3q_{bit}-1)/2}$, which is much smaller than $\det(L)^{1/2}$, because from Lemma 1:

$$\det(L) = \frac{p-1}{\gcd(a, b, p-1)}.$$

From the structure of $p-1$ and the way a and b are defined, we thus expect $\det(L)$ to be around p. Hence, we can hope that $\boldsymbol{l}$ is the closest vector of $\boldsymbol{t}$ in the lattice L, due to the huge size difference between $2^{(3q_{bit}-1)/2}$ and $\sqrt{p}$, for all the values of q_{bit} given by Figure 4: this heuristic reasoning is frequent in lattice-based cryptanalysis, here it can however be proved if we assume that the distribution of a and b is uniform modulo $p-1$ (see Appendix B). If $\boldsymbol{l}$ is the closest vector of $\boldsymbol{t}$, $\boldsymbol{l}$ and therefore the private exponent x can be recovered from a two-dimensional closest vector computation (we know t and a basis of L). And such a closest vector computation can be done in quadratic time (see for instance [23]), using the classical Gaussian algorithm for two-dimensional lattice reduction. Figure 5 sums up the attack, which clearly runs in polynomial time.

Input: The public parameters and a GPG–ElGamal signature (a, b) of m.
Expected output: The signer's private exponent x.

1. Compute a basis of the lattice L of (2), using statement 4 of Lemma 1.
2. Compute $(k', x') \in \mathbb{Z}^2$ such that $bk' + ax' \equiv m \pmod{p-1}$, using the Euclidean algorithm.
3. Compute the target vector $\boldsymbol{t} = (k' - 2^{3q_{bit}/2-1}, x' - 2^{3q_{bit}/2-1})$.
4. Compute the lattice vector $\boldsymbol{l}$ closest to $\boldsymbol{t}$ in the two-dimensional lattice L.
5. Return x' minus the second coordinate of $\boldsymbol{l}$.

Fig. 5. An attack using a single GPG–ElGamal signature.

Alternatively, if one wants to program as less as possible, one can mount another lattice-based attack, by simply computing a shortest vector of the 4-dimensional lattice L' spanned by the following row vectors, where K is a large constant:

$$\begin{pmatrix} (p-1)K & 0 & 0 & 0 \\ -mK & 2^{3q_{bit}/2} & 0 & 0 \\ bK & 0 & 1 & 0 \\ aK & 0 & 0 & 1 \end{pmatrix}$$

This shortest vector computation can be done in quadratic time using the lattice reduction algorithm of [23]. The lattice L' contains the short vector $(0, 2^{3q_{bit}/2}, k, x)$ because of (1). This vector is expected to be a shortest lattice vector under roughly the same condition on q_{bit} and p as in the previous lattice attack (we omit the details). Thus, for all values of Wiener's table (see Figure 4), one can hope to recover the private

exponent x as the absolute value of the last coordinate of any shortest nonzero vector of L'.

We implemented the last attack with Shoup's NTL library [29], using the integer LLL algorithm to obtain short vectors. In our experiments, the attack worked for all the values of Wiener's table, and the total running time was negligible (less than a second).

Practical impact. We have shown that GPG's implementation of the ElGamal signature is totally insecure: an attacker can recover the signer's private key from the public key and a single message/signature pair in less than a second. Thus, GPG–ElGamal signing keys should be be considered compromised, as announced by the GPG development team [19]. There are two types of ElGamal signing keys in GPG:

- The primary ElGamal sign+encrypt keys. When running the command `gpg --list-keys`, such keys can be spotted by a prefix of the form `pub 2048G/` where `2048` can be replaced by any possible keylength. The prefix `pub` specifies a primary key, while the capital letter `G` indicates an ElGamal sign+encrypt key.
- The ElGamal sign+encrypt subkeys. When running the command `gpg --list-keys`, such keys can be spotted by a prefix of the form `sub 2048G/` where `2048` can be replaced by any possible keylength. The prefix `sub` indicates a subkey.

The primary keys are definitely compromised because such keys automatically come up with a signature to bind the user identity to the public key, thus disclosing the private key immediately. The subkeys may not be compromised if no signature has ever been generated. In both cases, it is worth noting that the signing key is also an encryption key, so the damage is not limited to authentication: a compromised ElGamal signing key would also disclose all communications encrypted with the corresponding public key.

The mistake of using a small k and a small x dates back to GPG v1.0.2 (which was released in January 2000), when the generation of k and x was changed to improve performances: the flaw has therefore been present in GPG for almost four years. A signing key created prior to GPG v1.0.2 may still be compromised if a signature using that key has been generated with GPG v1.0.2 or later.

Nobody knows how many ElGamal sign+encrypt keys there are. What one knows is the number of ElGamal sign+encrypt keys that have been registered on keyservers. According to keyserver statistics (see [19]), there

are 848 registered primary ElGamal sign+encrypt keys (which is a mere 0.04% percent of all primary keys on keyservers) and 324 registered ElGamal sign+encrypt subkeys: of course, GPG advised all the owners of such keys to revoke their keys. These (fortunately) small numbers can be explained by the fact that ElGamal signing keys were never GPG's default option for signing, and their use was not really advocated.

As a consequence, ElGamal signatures and ElGamal sign+encrypt keys have recently been removed from GPG, and the GNU/Linux distributions which include GPG have been updated accordingly.

3.3 Encryption

Let m be the message to be encrypted. The message m is formatted in the way described in Section 2.1. The ciphertext is the pair (a, b) where: $a = g^k \bmod p$ and $b = my^k \bmod p$. The integer k is a "random" number coprime with $p-1$. Theoretically, k should be a cryptographically secure random number modulo $p-1$ such that k is coprime to $p-1$. But the generation of k is performed using the same procedure `gen_k` called by the ElGamal signature generation process. Thus, k is first selected with $3q_{bit}/2$ pseudo-random bits. Next, as while as k is not coprime with $p-1$, k is incremented. Hence, k is much smaller than $p-1$.

The security assumption for the hardness of decryption is no longer the standard Diffie-Hellman problem: instead, this is the Diffie-Hellman problem with short exponents (see [25]). Because the key generation makes sure that all the factors of $(p-1)/2$ have bit-length $\geq q_{bit}$, the best attack known to recover the plaintext requires at least $2^{q_{bit}/2}$ time, which is not a real threat.

However, the session key is formatted according to a specific padding, PKCS#1 v1.5 block type 02, which does not provide chosen-ciphertext security (see [4]). If we had access to a validity-checking oracle (which is weaker than a decryption oracle) that tells whether or not a given ciphertext is the ElGamal encryption of a message formatted with PKCS#1 v1.5 block type 02, we could apply Bleichenbacher's attack [4] to decrypt any ciphertext. Indeed, even though Bleichenbacher's attack was originally described with RSA, it also applies to ElGamal due to its homomorphic property: if (a, b) and (a', b') are ElGamal ciphertexts of respectively m and m', then $(aa' \bmod p, bb' \bmod p)$ is an ElGamal ciphertext of $mm' \bmod p$. One could argue that a validity-checking oracle is feasible in the situation where a user has configured his software to automatically decrypt any encrypted emails he receives: if an encrypted email turns out

not to be valid, the user would inform the sender. However, Bleichenbacher's attack require a large number of oracle calls, which makes the attack debatable in an email context. Nevertheless, it would be better if OpenPGP recommended a provably secure variant of ElGamal encryption such as ACE-KEM selected by NESSIE [8].

4 The Implementation of RSA

4.1 Key generation

To generate the parameters p, q, n, e, d, GPG implements the process described in Figure 6. Although the process does not lead to any realistic

Input: Bit-length k of the RSA modulus.
1. Repeat
2. Generate a pseudo-random prime p of $k/2$ bits.
3. Generate a pseudo-random prime q of $k/2$ bits.
4. If $p > q$, swap p and q.
5. $n \longleftarrow pq$.
6. Until the bit-length of n is equal to k.
7. If 41 is coprime with $\varphi(n)$, then $e \longleftarrow 41$
8. Else if 257 is coprime with $\varphi(n)$, then $e \longleftarrow 257$
9. Else
10. $e \longleftarrow 65537$
11. While e is not coprime with $\varphi(n)$, $e \longleftarrow e + 2$
12. Let d be the inverse of e modulo $\varphi(n)$.

Fig. 6. The RSA key generation in GnuPG.

attack, it is worth noting that the process leaks information on the private key. Indeed, the value of the RSA public exponent e discloses additional information on $\varphi(n)$. For instance, if we see a GPG–RSA public key with $e \geq 65539$, we know that $\varphi(n)$ is divisible by the prime numbers 41, 257 and 65537: we learn a 30-bit factor of $\varphi(n)$, namely $41 \times 257 \times 65537$. However, the probability of getting $e \geq 65539$ after the process is very small. To our knowledge, efficient attacks to factor n from partial knowledge of $\varphi(n)$ require a factor of $\varphi(n)$ larger than approximately $n^{1/4}$. Thus, this flaw does not lead to a serious attack, since the probability of getting a factor $\geq n^{1/4}$ after the process is way too small.

Nevertheless, any leakage on $\varphi(n)$ (apart from the fact that e is coprime with $\varphi(n)$) is not recommended: if one really wants a small public

exponent, one should rather select e first, and then generate the primes p and q until both $p-1$ and $q-1$ are coprime with e.

4.2 Encryption

As already mentioned in Section 2, GPG implements RSA encryption as defined by PKCS#1 v1.5. This is not state-of-the-art cryptography: like with ElGamal, Bleichenbacher's chosen-ciphertext attack [4] can decrypt any ciphertext. But, as mentioned in 3.3, the relevance of such attacks to the email world is debatable, in part because of the high number of oracle calls. We hope that future versions of the OpenPGP standard, will recommend better RSA encryption standards (see for instance PKCS#1 v2.1 [20] or NESSIE [8]).

4.3 Signature

GPG implements RSA signatures as defined by PKCS#1 v1.5. Again, this is not state-of-the-art cryptography (no security proof is known for this padding), but we are unaware of any realistic attack with the GPG setting, as opposed to some other paddings (see [7]). The RSA verification does not seem to check the range of the signature with respect to the modulus, which gives (marginal) malleability (see [30]): given a signature s of m, one can forge another signature s' of m. As with encryption, we hope that future versions of the OpenPGP standard will recommend a better RSA signature standard (see for instance PKCS#1 v2.1 [20] or NESSIE [8]).

References

1. D. M. Bellovin. Cryptography and the Internet. In *Proc. of Crypto '98*, volume 1462 of *LNCS*. IACR, Springer-Verlag, 1998.
2. D. Bleichenbacher. On the generation of one-time keys in DSS. Manuscript, February 2001. Result presented at the Monteverita workshop of March 2001.
3. D. Bleichenbacher. Generating ElGamal signatures without knowing the secret key. In *Proc. of Eurocrypt '96*, volume 1070 of *LNCS*, pages 10–18. IACR, Springer-Verlag, 1996.
4. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Proc. of Crypto '98*, volume 1462 of *LNCS*, pages 1–12. IACR, Springer-Verlag, 1998.
5. D. Boneh, A. Joux, and P. Q. Nguyen. Why textbook ElGamal and RSA encryption are insecure. In *Proc. of Asiacrypt '00*, volume 1976 of *LNCS*, pages 30–43. IACR, Springer-Verlag, 2000.

6. J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP message format: Request for Comments 2440. Available as `http://www.ietf.org/rfc/rfc2440.txt`.
7. J.-S. Coron, D. Naccache, and J. P. Stern. On the security of RSA padding. In *Proc. of Crypto '99*, volume 1666 of *LNCS*, pages 1–18. IACR, Springer-Verlag, 1999.
8. European Union. European project IST-1999-12324: New European Schemes for Signatures, Integrity, and Encryption (NESSIE). `http://www.cryptonessie.org`.
9. I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr Dobb's*, January 1996.
10. GPG. The GNU privacy guard. `http://www.gnupg.org`.
11. P. Gutmann. Software generation of practically strong random numbers. In *Proc. of the 7th Usenix Security Symposium*, 1998.
12. P. Gutmann. Lessons learned in implementing and deploying crypto software. In *Proc. of the 11th Usenix Security Symposium*, 2002.
13. N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Design, Codes and Cryptography*, 23:283–290, 2001.
14. IEEE. P1363: Standard specifications for public-key cryptography. Available at `http://grouper.ieee.org/groups/1363/`.
15. IPA. Cryptrec: Evaluation of cryptographic techniques. Available at `http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html`.
16. K. Jallad, J. Katz, and B. Schneier. Implementation of chosen-ciphertext attacks against PGP and GnuPG. In *Proc. of ISC '02*, volume 2433 of *LNCS*. Springer-Verlag, 2002.
17. B. Kaliski. PKCS #1: RSA encryption version 1.5: Request for Comments 2313. Available as `http://www.ietf.org/rfc/rfc2313.txt`.
18. J. Katz and B. Schneier. A chosen ciphertext attack against several E-Mail encryption protocols. In *Proc. of the 9th Usenix Security Symposium*, 2000.
19. W. Koch. GnuPG's ElGamal signing keys compromised. Internet public announcement on November 27th, 2003.
20. RSA Labs. PKCS #1: RSA cryptography standard. Available at `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html`.
21. J. Manger. A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS #1 v2.0. In *Proc. of Crypto '01*, volume 2139 of *LNCS*, pages 230–231. IACR, Springer-Verlag, 2001.
22. P. Q. Nguyen and I. E. Shparlinski. The insecurity of the Digital Signature Algorithm with partially known nonces. *Journal of Cryptology*, 15(3), 2002.
23. P. Q. Nguyen and D. Stehlé. Low-dimensional lattice basis reduction revisited. In *Algorithmic Number Theory – Proc. of ANTS-VI*, LNCS. Springer-Verlag, 2004.
24. P. Q. Nguyen and J. Stern. The two faces of lattices in cryptology. In *Proc. Workshop on Cryptography and Lattices (CALC '01)*, volume 2146 of *LNCS*, pages 146–180. Springer-Verlag, 2001.
25. P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman key agreement with short exponents. In *Proc. of Eurocrypt '96*, volume 1070 of *LNCS*, pages 332–343. IACR, Springer-Verlag, 1996.
26. OpenPGP. `http://www.openpgp.org`.
27. PGP. Pretty good privacy. `http://www.pgp.com`.
28. B. Schneier. Security in the real world: How to evaluate security technology. *Computer Security Journal*, XV(4), 1999.

29. V. Shoup. Number Theory C++ Library (NTL) version 5.3.1. Available at `http://www.shoup.net/ntl/`.
30. J. Stern, D. Pointcheval, J. Malone-Lee, and N. P. Smart. Flaws in applying proof methodologies to signature schemes. In *Proc. of Crypto '02*, volume 2442 of *LNCS*, pages 93–110. IACR, Springer-Verlag, 2002.

A Lattices in a nutshell

We recall basic facts about lattices. To learn more about lattices, see [24] for a list of references. Informally speaking, a lattice is a regular arrangement of points in n-dimensional space. In this paper, by the term lattice, we actually mean an integral lattice.

An integral lattice is a subgroup of $(\mathbb{Z}^n, +)$, that is, a non-empty subset L of $\mathbb{Z}^n$ which is stable by subtraction: $\boldsymbol{x} - \boldsymbol{y} \in L$ whenever $(\boldsymbol{x}, \boldsymbol{y}) \in L^2$. The simplest lattice is $\mathbb{Z}^n$. It turns out that in any lattice L, not just $\mathbb{Z}^n$, there must exist linearly independent vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in L$ such that:

$$L = \left\{ \sum_{i=1}^{d} n_i \boldsymbol{b}_i \mid n_i \in \mathbb{Z} \right\}.$$

Any such d-uple of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ is called a basis of L: a lattice can be represented by a basis, that is, a matrix. Reciprocally, if one considers d integral vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{Z}^n$, the previous set of all integral linear combinations of the $\boldsymbol{b}_i$'s is a subgroup of $\mathbb{Z}^n$, and therefore a lattice.

The *dimension* of a lattice L is the dimension d of the linear span of L: any basis of L has exactly d elements. It turns out that the d-dimensional volume of the parallelepiped spanned by an arbitrary basis of L only depends on L, not on the basis itself: this volume is called the *determinant* (or *volume*) of L. When the lattice is full-rank, that is, when the lattice dimension d equals the space dimension n, the determinant of L is simply the absolute value of the determinant of any basis. Thus, the volume of $\mathbb{Z}^n$ is 1.

Since our lattices are subsets of $\mathbb{Z}^n$, they must have a shortest nonzero vector: In any lattice $L \subseteq \mathbb{Z}^n$, there is at least one nonzero vector $\boldsymbol{v} \in L$ such that no other nonzero lattice vector has a Euclidean norm strictly smaller than that of $\boldsymbol{v}$. Finding such a vector $\boldsymbol{v}$ from a basis of L is called the shortest vector problem. When the lattice dimension is fixed, it is possible to solve the shortest vector problem in polynomial time (with respect to the size of the basis), using lattice reduction techniques. But the problem becomes much more difficult if the lattice dimension varies. In this article, we only deal with low-dimensional lattices, so the shortest vector problem is really not a problem.

The lattice determinant is often used to estimate the size of short lattice vectors. In a typical d-dimensional lattice L, if one knows a nonzero vector $\boldsymbol{v} \in L$ whose Euclidean norm is much smaller than $\det(L)^{1/d}$, then this vector is likely to be the shortest vector, in which case it can be found by solving the shortest vector problem, because any shortest vector would be expected to be equal to $\pm\boldsymbol{v}$. Although this can sometimes be proved, this is not a theorem: there are counter-examples, but it is often true with the lattices one is faced with in practice, which is what we mean by a typical lattice.

Another problem which causes no troubles when the lattice dimension is fixed is the closest vector problem: given a basis of $L \subseteq \mathbb{Z}^n$ and a point $\boldsymbol{t} \in \mathbb{Q}^n$, find a lattice vector $\boldsymbol{l} \in L$ minimizing the Euclidean norm of $\boldsymbol{l} - \boldsymbol{t}$. Again, in a typical d-dimensional lattice L, if one knows a vector $\boldsymbol{t}$ and a lattice vector $\boldsymbol{l}$ such that the norm of $\boldsymbol{t} - \boldsymbol{l}$ is much smaller than $\det(L)^{1/d}$, then $\boldsymbol{l}$ is likely to be the closest lattice vector of $\boldsymbol{t}$ in L, in which case $\boldsymbol{l}$ can be found by solving the closest vector problem. Indeed, if there was another lattice vector $\boldsymbol{l'}$ close to $\boldsymbol{t}$, then $\boldsymbol{l} - \boldsymbol{l'}$ would be a lattice vector of norm much smaller than $\det(L)^{1/d}$: it should be zero.

B Proving the GPG–ElGamal attack

We use the same notation as in Section 3.2. Let (a, b) be an GPG–ElGamal signature of m. If we make the simplifying assumption that both a and b are uniformly distributed modulo $p-1$, then the attack of Figure 5 can be proved, using the following lemma (which is not meant to be optimal, but is sufficient for our purpose):

Lemma 2. *Let $\varepsilon > 0$. Let p be a prime number such that all the prime factors of $(p-1)/2$ are $\geq 2^{q_{bit}}$. Let a and b be chosen uniformly at random over $\{0, \ldots, p-2\}$. Let L be the lattice defined by (2). Then the probability (over the choice of a and b) that there exists a non-zero $(u, v) \in L$ such that both $|u|$ and $|v|$ are $< 2^{3q_{bit}/2+\varepsilon}$ is less than:*

$$\frac{2^{7q_{bit}/2+5+3\varepsilon} \log_2 p}{(p-1)q_{bit}}.$$

Proof. This probability P is less than the sum of all the probabilities $P_{u,v}$, where the sum is over all the $(u, v) \neq (0, 0)$ such that both $|u|$ and $|v|$ are $< 2^{3q_{bit}/2+\varepsilon}$, and $P_{u,v}$ denotes the probability (over the choice of a and b) that $(u, v) \in L$. Let $(u, v) \in \mathbb{Z}^2$ be fixed and nonzero. If $v = 0$, there are at most $2\gcd(u, (p-1)/2)$ values of b in the set $\{0, \ldots, p-2\}$ such that:

$$bu + av \equiv 0 \pmod{(p-1)/2} \tag{3}$$

It follows that:

$$P_{u,0} \leq \frac{2\gcd(u,(p-1)/2)}{p-1}.$$

If $v \neq 0$: for any b, there are at most $2\gcd(v,(p-1)/2)$ values of a in the set $\{0,\ldots,p-2\}$ which satisfy (3), therefore:

$$P_{u,v} \leq \frac{2\gcd(v,(p-1)/2)}{p-1}.$$

Hence:

$$P \leq S + 2^{3q_{bit}/2+1+\varepsilon} S \leq 2^{3q_{bit}/2+2+\varepsilon} S,$$

where

$$S = \sum_{0<|u|<2^{3q_{bit}/2+\varepsilon}} \frac{2\gcd(u,(p-1)/2)}{p-1} = \sum_{0<|v|<2^{3q_{bit}/2+\varepsilon}} \frac{2\gcd(v,(p-1)/2)}{p-1}.$$

To bound S, we split the sum in two parts, depending on whether or not $\gcd(u,(p-1)/2) > 1$. If $\gcd(u,(p-1)/2) > 1$, then $\gcd(u,(p-1)/2) \leq |u| < 2^{3q_{bit}/2+\varepsilon}$ and u must be divisible by a prime factor of $(p-1)/2$ which is necessarily $\geq 2^{q_{bit}}$: the number of such u's is less than $2^{q_{bit}/2+1+\varepsilon}(\log_2 p)/q_{bit}$ because the number of prime factors of $(p-1)/2$ is less than $(\log_2 p)/q_{bit}$. We obtain:

$$\begin{aligned} S &\leq 2^{3q_{bit}/2+1+\varepsilon} \times \frac{2}{p-1} + 2^{q_{bit}/2+1+\varepsilon}(\log_2 p)/q_{bit} \times \frac{2^{3q_{bit}/2+1+\varepsilon}}{p-1} \\ &\leq \frac{2^{2q_{bit}+3+2\varepsilon}\log_2 p}{(p-1)q_{bit}} \end{aligned}$$

This completes the proof since $P \leq 2^{3q_{bit}/2+2+\varepsilon} S$. □

Because p is always much larger than $2^{4q_{bit}}$, the lemma shows that if ε is not too big, then with overwhelming probability, there is no non-zero $(u,v) \in L$ such that both $|u|$ and $|v|$ are $< 2^{3q_{bit}/2+\varepsilon}$. If $\boldsymbol{l}$ was not the closest vector of $\boldsymbol{t}$ in L, there would be another lattice vector $\boldsymbol{l'} \in L$ closer to $\boldsymbol{t}$: the distance between $\boldsymbol{l'}$ and $\boldsymbol{t}$ would be less than $2^{(3q_{bit}-1)/2}$. But then, the lattice vector $(u,v) = \boldsymbol{l} - \boldsymbol{l'}$ would contradict the lemma, for some small ε. Hence, $\boldsymbol{l}$ is the closest vector of $\boldsymbol{t}$ in L with overwhelming probability, which proves the attack. However, the initial assumption that both a and b are uniformly distributed modulo $p-1$ is an idealized model, compared to the actual way a and b are generated by GPG. In this sense, the lemma explains why the attack works, but it does not provide a complete proof.