The ElGamal Cryptosystem

Andreas V. Meier

June 8, 2005

Taher Elgamal first described the ElGamal Cryptosystem [6] in an article published in the proceedings of the CRYPTO '84, a conference on the advances of cryptology.
The proposed algorithm belongs to the family of public key cryptographic algorithms. Therefore it makes use of a key separated into a public and a private part. A fundamental aspect of this system is, that the knowledge of the private part makes the decryption easy. If the private key is unknown, it is virtually impossible to decrypt the message in acceptable time.
The security of ElGamal is based on the discrete logarithm problem. To encrypt and respectively decrypt a message, a discrete power is executed. This operation is efficient to compute. An attacker that seeks to decrypt an intercepted message may try to recover the private key. To this end a logarithm needs to be computed. No actual method exists for this, given certain requirements on the initial group are met. Under these circumstances, the encryption is secure.
Today the ElGamal algorithm is used in many cryptographic products. The open-source software GnuPG uses ElGamal as standard for signatures. On behalf of this software and its problems with ElGamal [10] discovered in late 2003 we will show the importance of correct implementation of cryptographic algorithms.
This paper will present the ElGamal Cryptosystem and will show how it is used for the encryption and decryption of secret messages as well as for signing messages in order to make them authentic. Variants of the algorithm will be shown that aim to make the algorithm more secure. Furthermore, issues in the design and implementations of the algorithm will be discussed.

1 Public Key Cryptography

The first section of this paper explains the history of public key cryptosystems. The algorithms proposed in the first paper on this family of cryptography algorithms will be displayed. Furthermore, the shortcomings of the system and common ways to avoid them are described.

Up to the time when the paper [5] by Diffie and Hellman offered new directions in cryptography, only private key cryptography systems were used.

Unfortunately, private keys are only secure as long as they are kept private. But they have to be known to both sides of an encrypted communication channel. This makes it necessary to preshare the key by other means. Presharing could be done by sending sealed envelopes with trusted messengers to the other side, by personally meeting with the other side or similar. Anyway, the transaction process is the weak point of the whole cryptosystem.

Another shortcoming of cryptosystems based on private keys is that building up the secure channel is quite extensive due to the problem mentioned before. If the communication should take place between two people that had no contact before, and even worse that are maybe on opposite sides of the globe, a presharing would be unfeasible.

Diffie and Hellman now offered a possibility to establish a secure channel between two randomly picked entities, that previously generated a special, for other channels reusable key-pair. Generation and use of the key-pair for encryption will be explained later in this section. In beforehand we will define some basic notations to have a common language about cryptosystems with the reader.

1.1 Definition of a Cryptosystem

To examine a cryptosystem from the scientific point of view, we first need a formal definition. This definition and variables introduced with it can then be used in the following to have a clear common language.

The basic setup is the desire of a sending entity to transmit information in form of a message to a receiving entity. The information is considered to be secret, therefore the possibility of a third entity to read the message should be as low as possible.

To distinguish the involved they are denoted by the names Alice (or the letter A) for the sender, Bob (or the letter B), as well as Eve (or the letter E) for the eavesdropping entity. Although we use names of persons here, it should be always kept in mind that entities can also be represented by autonomous computing systems. The method of using names for the distinct entities in cryptosystems is a common pattern in cryptography and is called "Alice and friends" [8].

In this paper we use the definition of a cryptosystem as the 5-tuple (M, C, K, E, D) where $M \in \Sigma^*$ is the plaintext message Alice seeks to transmit to Bob. The plaintext message can be split in multiple blocks denoted as m_i. The ciphertext $C \in \Sigma^*$ is the result of an encryption function $E_K(P) : C$, which takes the plaintext and an additional key K, and computes the ciphertext from it using an adequate algorithm. Bob, who receives C, uses it with a decryption function $D_K(C) : P$, which under application of the key K results in the plaintext. The message has been transferred, and Eve was not able to read it.

So far our cryptosystem can describe securing of communication using a single, preshared key K. But as we will see in the following, for a public key cryptosystem we need to split the key into two parts. The first part is the one which gave the system its name: the public key. The second part is the private key.

Since this so called "keypair" is personalized, we will denote the private key with a persons or entities initial letter in lower case. Alice therefore owns the private key a. The public key will be written as it is computed, leaving the modulus operation implizit. Thus, Alice has a public key identified by g^a. Both the private and the public key are strongly connected and therefore called the "keypair".

Using keypairs, it is also possible to sign a not necessarily encrypted document. The signature S is computed using the private key and the message M. Then, the signature is sent together with the message as the pair (M, S). A receiver can verify the signature by using the sender's public key.

Since we now have a notation for cryptosystems, we will present the original algorithm as proposed by Diffie and Hellman in the next section.

1.2 The Diffie-Hellman PK system

We mentioned in the introduction section that up to the pioneering paper of Diffie and Hellman the encryption systems only used symmetric keys.

This approach has two severe shortcomings. The first has been mentioned in the introduction section about public key cryptography. It is the fact that the symmetric key has to be known to both parties. To this end it has to be preshared by other means because the secure channel has not been established so far. The transmission is a weak point, as a third party could listen in and gain the key maybe even unnoticed.

The second problem is apparent in nowadays worldwide telecommunication systems and related to the first problem. If two random user of a n user group want to communicate securely, they would have to preshare a key. Not only is it intricate to create this channel if the two persons do not know each other and are on different sides of the globe. Even more, the big amount of preshared keys in the system, which can raise to a maximum of $(n1^2 - n)/2$ keys for all possible communications calls for a simpler solution.

Diffie and Hellman proposed two fundamental changes with respect to the keys:

- Personalize the keys. Each entity in the system has its own key.
- Split the keys. Each key is divided into a public key part and a private key part, the so called keypair.

The first step reduces the amount of keys in the system by an exponent to only n. But for this reason those keys can not be used in a symmetric way. On that account the second step introduces asymmetric keys.

The public key is published by open means, like a dedicated keyserver or simply sent to the other party through an insecure channel. The private key is always kept private, the security of the communication is based on the privacy of the private key.

As in our model Bob is the receiving entity, he first has to generate his keypair. For this he chooses a sufficiently large prime number p and a generator g, such that

$$\forall n \in [1; p-1] \exists k : n = g^k \bmod p$$

The generator and the prime are known to both parties and can be communicated publicly. Out of the group generated by g, Alice and Bob independently chose a random integer, which will be their private key part. Following the notation agreed on above, Alice's private key is a and Bob's private key is b.

Alice and Bob now generate their public key parts by exponentiation of the generator with the privately chosen integer and taking the result modulo p. This is the public keypart and needs to be sent to the other side. Figure 1 summarizes the involved keyparts. Now both parties can compute a shared secret key k:

Person	Private Key	Public Key
Alice	a	$g^a \bmod p$
Bob	b	$g^b \bmod p$

Figure 1: Entities and their keyparts

$$k = g^{ab} = (g^a)^b = (g^b)^a = g^{ba}$$

This is possible since Alice received the public key of Bob g^b) and her private key a. She can therefore can compute $k = (g^b)^a = g^{ba}$. For Bob, this holds in a similar way. He knows the public keypart of Alice, g^a and can use his private keypart b to compute $k = (g^a)^b = g^{ab}$. The shared secret k can now be used for a suitable symmetric encryption system.

The features of the key exchanging system explained in this section are obvious. Two entities are able to set up a shared secret using only public communication. The justification for calling this algorithm secure is the hardness of the Discrete Logarithm Problem, which will be explained in the next section.

2 Assigned Complexity Theoretic Problems

The security of actual cryptographic systems relies on the hardness of problems of complexity theory. The system we looked at in the last section is based on the problem of calculating the discrete logarithm, as will be explained in the following.

2.1 DL - Discrete Logarithm problem

The basic problem of the presented algorithm is the Discrete Logarithm problem. It describes that it is hard to compute the exponent given an power in a known multiplicative group.

An instance of a DL Problem consists of the following:

- A multiplicative group $(G, \dot{})$,
- a generator g of G and
- an element $x \in G$

The question now is to find the unique integer a, $0 \leq a \leq n-1$ such that $g^a = x$. The a in this question can be seen as the $\log_g x$, thus the search for the required exponent for g to retrieve x.

2.2 DH - Diffie Hellman problem

Within their work, Diffie and Hellman define the problem of finding the shared secret key given both public keys the DH problem. It is strongly connected to the Discrete Logarithm problem. The instance of a DH problem consists of the following:

- A multiplicative group $(G, \dot{})$,
- a generator g of G and
- two public key parts g^a and g^b.

The question to this problem is "Find the common key g^{ab} given an instance of DH.".

The creation of the large prime p does not affect the security of the algorithm directly. The logarithm can not be computed efficiently directly. But for small p it might be feasible to use a brute-force attack, i.e. test all possibilities. The higher the prime p gets, the larger the group G gets. Thus with high prime numbers p the security of the algorithm is strengthened.

2.3 DDH - Decision Diffie Hellman problem

Related to the Diffie Hellman is the Decision Diffie Hellman problem [2]. An instance of it contains of the following:

- A multiplicative group $(G, \dot{})$,
- a generator g of G and
- the triplet g^a, g^b and g^c

The question that can be asked to a machine solving the Decision Diffie Hellman problem is: "Is $c \equiv ab \pmod{p}$?"

2.4 Algorithms for DL/DH/DDH

The complexity of DDH is obviously not higher than DH, and DH is also not harder than DL. This can be seen from the fact that using a DL oracle machine, the DH problem can be resolved in constant time. There are only two requests to the oracle necessary to receive a and b, from which g^{ab} can then computed without effort.

The DDH problem can be similarly be solved using an DH oracle machine. Here the oracle is asked for the common key that is derived from g^a and g^b. The result is compared with the given g^c. If the comparison failed, the DDH machine returns false, and vice versa for true.

There exist a few approaches to reduce the time necessary to break a DL problem, if a weak group is chosen. For cyclic subgroups of $\mathbb{F}_p^*$ it is possible to apply a so called "index calculus attack", a newer variant being the "number field sieve". These algorithms proceed in two steps [1]:

1. Find the DL of elements in a factor base, which is a set of elements chosen in a way that any element can be decomposed into a subset of this base.
2. Operate on the requested element to find a related element that decomposes into elements in the factor base.

The number sieve [4] has a time and space complexity of the form:

$$L_p(a, c) = exp(c\,(\log p)^a\,(\log\log p)^{1-a})$$

where the constant c is depending on the algorithm chosen.

A more direct method for solving the DL problem is to optimize the brute-force attack. The basic approach is to compute all possible powers for a given g as $g^1, g^2, g^3, \ldots$. This has to be done until the requested value g^a has been found. The complexity for this approach is constant in space O(1) and linear in time O(n), assumed that raising the base g to the power of the exponent, modulo p is calculable in constant time O(1).

An optimization in time for this approach would be precalculating all possible powers and sorting them into a list. The calculation would consume O(n) time, the sorting with an efficient algorithm takes about O($n \log n$) steps. Searching for the power in the sorted list is possible in O($\log n$). Neglecting the logarithmic factors [11] we are at a precomputation complexity of O(n), a memory requirement of O(n) and are able to solve the DL problem in O(1) time.

Other algorithms as Shanks', the Pollard ρ or the Pohling-Hellman algorithm are presented in [11]. Their presentation would exceed the scope of this thesis.

3 The ElGamal Cryptosystem

In 1984 Taher ElGamal presented a cryptosystem which is based on the Discrete Logarithm Problem discussed in the last section [6]. It relies on the assumption that the DL cannot be found in feasible time, while the inverse operation of the power can be computed efficiently.

The original public key system proposed by Diffie and Hellman requires interaction of both parties to calculate a common private key. This poses problems if the cryptosystem should be applied to communication systems where both parties are not able to interact in reasonable time due to delays in transmission or unavailability of the receiving party.

Thus ElGamal simplified the Diffie-Hellman key exchange algorithm by introducing a random exponent k. This exponent is an replacement for the private exponent of the receiving entity. Due to this simplification the algorithm can be used to encrypt in one direction, without the necessity of the second party to take actively part. The key advance here is that the algorithm can be used for encryption of electronic messages, which are transmitted by the means of public store-and-forward services.

In this section, the ElGamal cryptosystem will be introduced to the reader.

3.1 Key Generation

As discussed above, the basic requirement for a cryptographic system is at least one key for symmetric algorithms and two keys for asymmetric algorithms.

The key generation steps are similar to the general scheme explained above. With ElGamal, only the receiver needs to create a key in advance and publish it. Following our naming scheme from above, we will now follow Bob through his procedure of key generation.

Bob will take the following steps to generate his keypair:

1. Prime and group generation
   First Bob needs to generate a large prime p and the generator g of a multiplicative group Z_p^* of the integers modulo p.

2. private key selection
   Now Bob selects an integer b from the group Z by random and with the constraint $1 \leq b \leq p - 2$. This will be the private exponent.

3. Public key assembling
   From this we can compute the public key part $g^b \bmod p$. The public key of Bob in the ElGamal cryptosystem is the triplet (p, g, g^b) and his private key is b.

4. Public key publishing
   The public key now needs to be published using some dedicated keyserver or other means, so that Alice is able to get hold of it.

3.2 Encryption Procedure

To encrypt a message M to Bob, Alice first needs to obtain his public key triplet (p, g, g^b) from a key server or by receiving it from him via unencrypted electronic mail. There is no security issue involved in this transmission, as the only secret part, b, is sent in g^b. Since the core assumption of the ElGamal cryptosystem says that it is infeasible to compute the discrete logarithm, this is safe.

For the encryption of the plaintext message M, Alice has to follow these steps:

1. Obtain the public key
   As described above, Alice has to acquire the public key part (p, g, g^b) of Bob from an official and trusted keyserver.

2. Prepare M for encoding
   Write M as set of integers $(m_1, m_2, \ldots)$ in the range of $\{1, \ldots, p - 1\}$. These integers will be encoded one by one.

3. Select random exponent
   In this step, Alice will select a random exponent k that takes the place of the second party's private exponent in the Diffie-Hellman key exchange. The randomness here is a crucial factor as the possibility to guess the k gives a sensible amount of the information necessary to decrypt the message to the attacker.

4. Compute public key
   To transmit the random exponent k to Bob, Alice computes $g^k \bmod p$ and combines it with the ciphertext that shall be sent to Bob.

5. Encrypt the plaintext
   In this step, Alice encrypts the message M to the ciphertext C. For this, she iterates over the set created in step 2 and calculates for each of the m_i:

   $$c_i = m_1 * (g^b)^k$$

   The ciphertext C is the set of all c_i with $0 < i \leq |M|$.

The resulting encrypted message C is sent to Bob together with the public key $g^k \bmod p$ derived from the random private exponent.

Even if an attacker would listen to this transmission, and in a second step would also acquire the public key part g^b of bob from a keyserver, he would still not be able to derive g^{b*k} as can be seen from the Discrete Logarithm problem.

Elgamal advises to use a new random k for each of the single message blocks m_i. This greatly improves security, as knowledge of one message block m_j does not lead the attacker to the knowledge of all other m_i. The reason for this ability is that if $c_1 = m_1 * (g^b)^k \bmod p$ and $c_2 = m_2 * (g^b)^k \bmod p$, from knowing only m_1 the next part of the message m_2 can be calculated by the following formula:

$$\frac{m_1}{m_2} = \frac{c_1}{c_2}$$

3.3 Decryption Procedure

After receiving the encrypted message C and the randomized public key g^k, Bob has to use the encryption algorithm to be able to read the plaintext M. This algorithm can be divided in a few single steps:

1. Compute shared key
   The ElGamal cryptosystem helped Alice to define a shared secret key without Bobs interaction. This shared secret is the combination of Bobs private exponent b and the random exponent k chosen by Alice. The shared key is defined by the following equation:
   $$(g^k)^{p-1-b} = (g^k)^{-b} = b^{-bk}$$

2. Decryption
   For each of the ciphertext parts c_i Bob now computes the plaintext using

   $$m_i = (g^k)^{-b} * c_i \bmod p$$

After combining all of the m_i back to M he can read the message sent by Alice.

3.4 ElGamal Signatures

The ElGamal cryptosystem does not only support encryption and decryption, but also the electronically signing of messages M.

A signature scheme has three main characteristics:

- Creation
  Alice needs to be able to find the signature for M by using her private key a. She will then send the message together with the signature as the pair (M, S) to Bob.

- Verification
  Bob has to be able to verify the signature by using the public key g^a. The verification of the signature assures Bob that Alice has signed the message as he received it. It does not deliver information about if Alice wrote the message herself or if she intended to send it at all. The second information Bob can draw from the verification is that the message has not been altered on the transmission path between him and Alice.

- Forgery prevention
  It should not be possible for a malicious user to use the public key g^a of Alice to create a signature for an arbitrary message.

A signature in the ElGamal cryptosystem is the pair (r, s) with $0 \leq r, s < p-1$ defined by the equation

$$g^M \equiv (g^a)^r r^s \bmod p \tag{1}$$

The procedure of signing follows similar steps as the encryption procedure:

1. Choose random $k \in G$

2. Compute $r \equiv g^k \bmod p$

3. Fill the signature equation from above as $g^M \equiv g^{ar} gks \bmod p$ and solve it for s using $m \equiv ar + ks \bmod (p-1)$. This has a solution for s if k is chosen such that $gcd(k, p-1) = 1$.

Bob received (M, r, s) and wants to verify the signature now. For this, he only needs to compute both sides of the equation 1 and check for equality.

3.5 ElGamal Summary

As has been shown, the ElGamal cryptosystem is as secure as it is hard to solve the Discrete Logarithm problem, given no weak random exponents or primes are chosen. It further prevents a chosen plaintext attack by using a randomized encryption exponent k. As this k is chosen uniformly before encryption, the same plaintext can result in $p-1$ different ciphertexts, one of which is chosen uniformly by choosing a k.

One shortcoming of ElGamal is the Message Expansion. The size of the transferred message increases by the factor 2. This comes from choosing a new random k for each block of the plaintext message m_i. The public key derived from this k has to be sent together with the c_i as the pair (c_i, g^k), the set of all these pairs giving the ciphertext C.

Another problem that can arise is that Alice relies on the authenticity of the public key she retrieves from Bob. This is a lesser problem if the public key is handed over in direct contact, but using the system on a large network like the Internet other means have to be found. The most common installation is a central keyserver. Users send their public keys there after generation so others can download them and use them for encryption or signature validation. If a malicious attacker manages to supply Alice with his key and make her believe it is the key of Bob, she would encrypt the message to the attacker and not to the intended receiver Bob. If the message is not signed, the attacker can then encrypt the message again, this time with Bob's key and send it to Bob. This is called the Man-In-The-Middle attack. At least the man in the middle would be detected if Alice would both sign and encrypt the message, since Bob would notice the changed contents when verifying the signature.

To overcome the problem of forged signatures, webs of thrust can be built up where users sign their keys against each other thereby assuring the authenticity of the key. Another possibility is a central certification authority that signs or even gives out the keys only after validating the owner of the key.

In this section, we have presented the ElGamal cryptosystem as presented in 1984 and followed the algorithm through the intended steps. The next section will explain the problems that can arise during the implementation of cryptography software and show an example where an implementation error has long lived undetected.

4 Importance Of Correct Implementation

In this paper we have looked at public key cryptosystems in general and the ElGamal cryptosystem in particular. These cryptosystems are only secure, if some given requirements are met:

- there exists no algorithm to solve NP complete problems efficiently
- the prime p is sufficiently high
- no weak exponents are chosen

This section will talk about a problem in cryptography, that goes beyond scientific analysis and research. The next stage for such algorithms, if proven to be secure and usable, is going into production. But, as we show here, this hurdle is often harder to take than the scientific one.

OpenSource software means the user has the possibility to look at the source code and verify it, if he has the intention and skills to do so. This basic idea was the key to the success of the Linux operating system and the associated software products. One of the applications included in all Linux distributions is the "GNU Privacy Guard" GPG. This cryptographic software includes many different crypto-algorithms that are not restricted

```
/* 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.
 */
nbits = wiener_map( orig_nbits ) * 3 / 2;
nbytes = (nbits+7)/8;
```

Listing 1: GPG v1.2.3 ciphers/elgamal.c Source Excerpt

$\|p\|$	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	...
q_{bit}	119	145	165	183	198	212	225	237	249	259	269	...

Figure 2: Wiener Table for min. bitlength of prime factors of $(p-1)/2$

in use by patents. The OpenSource license, under which it is published, often has to hold as an argument for the correctness of the implementation. Although the principle of many eyes has truly been guarantor for successful development of many OpenSource software products, a software working as specified might incorporate slight errors in the implementation of algorithms. Phong Q. Nguyen et. al. discovered in 2004, that the implementation of the ElGamal algorithm and it's use for signing messages was flawed [10]. Due to these weaknesses, it was possible to recover the signers private key in less than a second.

As a consequence of these discovers, all GPG ElGamal signing keys since the year 2000 had to be considered compromised. This shows, that even in a many eye OpenSource software system still exist parts of the code that have not been reviewed. The more people work on a project, the fewer such parts might exist, but with specialised algorithms like cryptographic routines it needs a expert in both cryptography and software development to spot these.

In Listing 1 an excerpt from the GPG source code is shown. It is the place, where the the minimal bitlength for the private exponent k is determined. He should be co-prime to $(p-1)$. To determine the bitlength for the private exponent, the prime factors of p, $(p-1)/2$ is looked up in the Wiener table (see Figure 2). The Wiener table is based on assumptions about the currently available computing power [9]. In GPG, it is used to choose an exponent high enough to be secure, but small enough to keep the computation time reasonable.

Deriving the bitlength for k using the Wiener table and generating a prime p with that specification, and then incrementing k until it is co-prime to $p-1$ keeps k far from being equally distributed [10]. Because k is chosen much smaller than p, the following congruence leads to the possibility to compute the exponents using the following congruence:

$$ks + ar \equiv m \bmod (p-1) \tag{2}$$

In equation 2, only k and a are unknown. But we know, that they are much smaller than $\sqrt{p}$. A lattice algorithm can now be used to compute the pair (a, k) in polynomial time [10, 7].

Further problems can arise with bad implementations that are described in [3]. This paper explains that without a special preprocessing, it is possible to break an ElGamal encryption for weak choices of p and g.

From this section we have seen that even if our cryptographic system is based on a hard problem of complexity theory, a bad implementation of the system can make the whole security assumption collapse. It is thus required to use special organizational or technical means for such sensitive parts of applications to ensure their correctness and flawlessness.

5 Summary

In this paper we have first agreed on a common language and notation for cryptosystems, partially using established patterns in cryptography. We have further presented the originator of public key cryptosystems, the Diffie-Hellman key exchange algorithm. In the following we looked at the complexity theoretic problems this algorithm was based on. Then we explained a cryptosystem that was derived from the one presented before, but could be seen as an improvement. The ElGamal Cryptosystem can be used for encryption and signing messages without direct interaction. In the last section, we showed the importance of correct implementation in software. We gave an example where a flawed implementation rendered the private keys of thousands of users compromised.

But not only the cryptosystem itself can be compromised. As public key systems are designed for the populace, a bigger security issue is in handling the private keys. Many people do not secure their private keys as would be necessary. And to be correctly applied, a private key that was accessible by a malicious user, even if it was not, has to be declared compromised. This is due to attackers being able to cover his tracks and steal the private key undetected.

Unfortunately, the complexity of the issues involved in dealing with cryptographic systems and software keeps many people from using them. In nowadays world, where all Internet communication is supervised by several authorities, encryption offers the possibility to get at least some privacy back.

References

[1] Ian F. Blake and Theo Garefalakis. On the complexity of the discrete logarithm and diffie-hellman problems. *J. Complex.*, 20(2-3):148–170, 2004.

[2] Dan Boneh. The decision Diffie-Hellman problem. *Lecture Notes in Computer Science*, 1423:48–63, 1998.

[3] Dan Boneh, Antoine Joux, and Phong Q. Nguyen. Why textbook elgamal and rsa encryption are insecure. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pages 30–43, London, UK, 2000. Springer-Verlag.

[4] Matthew Edward Briggs. An introduction to the general number field sieve. Master's thesis, Virginia politechnic institute and state university, 1998.

[5] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[6] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18. Springer-Verlag New York, Inc., 1985.

[7] N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001.

[8] S. Lehtonen and J. Prssinen. A pattern language for cryptographic key management, 2002.

[9] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001.

[10] Phong Q. Nguyen. Can We Trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3, May 2004.

[11] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, Inc., Boca Raton, FL, USA, 1995.