Why Textbook ElGamal and RSA Encryption Are Insecure

(Extended Abstract)

Dan Boneh[1], Antoine Joux[2], and Phong Q. Nguyen[3]

[1] Stanford University, Computer Science Department
Stanford, CA 94305, USA
dabo@cs.stanford.edu
http://crypto.stanford.edu/∼dabo/
[2] DCSSI, 18 rue du Docteur Zamenhof,
92131 Issy-les-Moulineaux Cedex, France
joux@ens.fr
[3] École Normale Supérieure, Département d'Informatique,
45 rue d'Ulm, 75005 Paris, France
pnguyen@ens.fr
http://www.di.ens.fr/∼pnguyen/

Abstract. We present an attack on plain ElGamal and plain RSA encryption. The attack shows that without proper preprocessing of the plaintexts, both ElGamal and RSA encryption are fundamentally insecure. Namely, when one uses these systems to encrypt a (short) secret key of a symmetric cipher it is often possible to recover the secret key from the ciphertext. Our results demonstrate that preprocessing messages prior to encryption is an essential part of both systems.

1 Introduction

In the literature we often see a description of RSA encryption as $C = \langle M^e \rangle \bmod N$ (the public key is $\langle N, e \rangle$) and a description of ElGamal encryption as $C = \langle My^r,\ g^r \rangle \bmod p$ (the public key is $\langle p, g, y \rangle$). Similar descriptions are also given in the original papers [17,9]. It has been known for many years that this simplified description of RSA does not satisfy basic security notions, such as semantic security (see [6] for a survey of attacks). Similarly, a version of ElGamal commonly used in practice does not satisfy basic security notions (even under the Decision Diffie-Hellman assumption [5]) [1]. To obtain secure systems using RSA and ElGamal one must apply a preprocessing function to the plaintext prior to encryption,

[1] Implementations of ElGamal often use an element $g \in \mathbb{Z}_p^*$ of prime order q where q is much smaller than p. When the set of plaintexts is equal to the subgroup generated by g, the Decision Diffie Hellman assumption implies that ElGamal is semantically secure. Unfortunately, implementations of ElGamal often encrypt an m-bit message by viewing it as an m-bit integer and directly encrypting it. The resulting system is not semantically secure – the ciphertext leaks the Legendre symbol of the plaintext.

T. Okamoto (Ed.): ASIACRYPT 2000, LNCS 1976, pp. 30–43, 2000.

or a conversion to the encryption function (see [10,16,13] for instance). Recent standards for RSA [15] use Optimal Asymmetric Encryption Padding (OAEP) which is known to be secure against a chosen ciphertext attack in the random oracle model [4]. Currently, there is no equivalent preprocessing standard for ElGamal encryption, although several proposals exist [1,10,16,13]. Unfortunately, many textbook descriptions of RSA and ElGamal do not view these preprocessing functions as an integral part of the encryption scheme. Instead, common descriptions are content with an explanation of the plain systems.

In this paper we give a simple, yet powerful, attack against both plain RSA and plain ElGamal encryption. The attack illustrates that plain RSA and plain ElGamal are fundamentally insecure systems. Hence, any description of these cryptosystems cannot ignore the preprocessing steps used in full RSA and full ElGamal. Our attack clearly demonstrates the importance of preprocessing. It can be used to motivate the need for preprocessing in introductory texts.

Our attack is based on the fact that public key encryption is typically used to encrypt session-keys. These session-keys are typically short, i.e. less than 128 bits. The attack shows that when using plain RSA or plain ElGamal to encrypt an m-bit key, it is often possible to recover the key in time approximately $2^{m/2}$. In environments where session-keys are limited to 64-bit keys (*e.g.* due to government regulations), our attack shows that both plain RSA and plain ElGamal result in a completely insecure system. We experimented with the attack and showed that it works well in practice.

1.1 Summary of Results

Suppose the plaintext M is m bits long. For illustration purposes, when $m = 64$ we obtain the following results:

- For any RSA public key $\langle N, e\rangle$, given $C = M^e \bmod N$ it is possible to recover M in the time it takes to compute $2 \cdot 2^{m/2}$ modular exponentiations. The attack succeeds with probability 18% (the probability is over the choice of $M \in \{0, 1, \ldots, 2^m - 1\}$). The algorithm requires $2^{m/2}m$ bits of memory.
- Let $\langle p, g, y\rangle$ be an ElGamal public key. When the order of g is at most $p/2^m$, it is possible to recover M from any ElGamal ciphertext of M in the time it takes to compute $2 \cdot 2^{m/2}$ modular exponentiations. The attack succeeds with probability 18% (over the choice of M), and requires $2^{m/2}m$ bits of memory.
- Let $\langle p, g, y\rangle$ be an ElGamal public key. Suppose $p - 1 = qs$ where $s > 2^m$ and the discrete log problem for subgroups of $\mathbb{Z}_p^*$ of order s is tractable, i.e. takes time T for some small T. When the order of g is $p - 1$, it is possible to recover M from any ciphertext of M in time T and $2 \cdot 2^{m/2}$ modular exponentiations. The attack succeeds with probability 18% (over the choice of M), and requires $2^{m/2}m$ bits of memory.
- Let $\langle p, g, y\rangle$ be an ElGamal public key. Suppose again $p - 1 = qs$ where $s > 2^m$ and the discrete log problem for subgroups of $\mathbb{Z}_p^*$ of order s takes time T for some small T. When the order of g is either $p-1$ or at most $p/2^m$,

it is possible to recover M from any ciphertext of M in time T plus one modular exponentiation and $2 \cdot 2^{m/2}$ additions, provided a precomputation step depending only on the public key. The success probability is 18% (over the choice of M). The precomputations take time $2^{m/2}T$ and $2^{m/2}$ modular exponentiations. The space requirement can optionally be decreased to $2^{m/4} \log_2 s$ bits without increasing the computation time, however with a loss in the probability of success.

All attacks can be parallelized, and offer a variety of trade-offs, with respect to the computation time, the space requirement, and the probability of success. For instance, the success probability of 18% can be raised to 35% if the computation time is quadrupled. Note that the first result applies to RSA with an arbitrary public exponent (small or large). The attack becomes slightly more efficient when the public exponent e is small. The second result applies to the usual method in which ElGamal is used in practice. The third result applies when ElGamal encryption is done in the entire group, however $p-1$ has a small smooth factor (a 64-bit smooth factor). The fourth result decreases the on-line work of both the second and the third results, provided an additional precomputation stage. It can optionally improve the time/memory trade-off. The third and fourth results assume that $p-1$ contains a smooth factor: such a property was used in other attacks against discrete-log schemes (see [2,14] for instance).

1.2 Splitting Probabilities for Integers

Our attacks can be viewed as a meet-in-the-middle method based on the fact that a relatively small integer (e.g., a session-key) can often be expressed as a product of much smaller integers. Note that recent attacks on padding RSA signature schemes [7] use related ideas. Roughly speaking, these attacks expect certain relatively small numbers (such as hashed messages) to be smooth. Here, we will be concerned with the size of divisors. Existing analytic results for the bounds we need are relatively weak. Hence, we mainly give experimental results obtained using the Pari/GP computer package [3].

Let M be a uniformly distributed m-bit integer. We are interested in the probability that M can be written as:

- $M = M_1 M_2$ with $M_1 \leq 2^{m_1}$ and $M_2 \leq 2^{m_2}$. See table 1 for some values.
- $M = M_1 M_2 M_3$ with $M_i \leq 2^{m_i}$. See table 2 for some values.
- $M = M_1 M_2 M_3 M_4$ with $M_i \leq 2^{m_i}$. See table 3 for some values.

The experimental results given in the tables have been obtained by factoring a large number of randomly chosen m-bit integers with uniform distribution. Some theoretical results can be obtained from the book [11]. More precisely, for $1/2 \leq \alpha < 1$, let $P_\alpha(m)$ be the probability that a uniformly distributed integer M in $[1 \ldots 2^m - 1]$ can be written as $M = M_1 M_2$ with both M_1 and M_2 less or equal to $2^{\alpha m}$. It can be shown that $P_{1/2}(m)$ tends (slowly) to zero as m grows to infinity. This follows (after a little work) from results in [11][Chapter 2] on the number $H(x, y, z)$ of integers $n \leq x$ for which there exists a divisor d such that

$y \leq d < z$. More precisely, the following holds (where log denotes the neperian logarithm):

$$P_{1/2}(m) = O\left(\frac{\log\log m \cdot \sqrt{\log m}}{m^{\delta}}\right), \tag{1}$$

where $\delta = 1 - \frac{1+\log\log 2}{\log 2} \approx 0.086$. On the other hand, when $\alpha > 1/2$, $P_\alpha(m)$ no longer tends to zero, as one can easily obtain the following asymptotic lower bound, which corrects [8, Theorem 4, p 377]:

$$\liminf P_\alpha(m) \geq \log(2\alpha), \tag{2}$$

This is because the probability must include all numbers that are divisible by a prime in the interval $[2^{m/2}, 2^{\alpha m}]$, and the bound follows from well-known smoothness probabilities.

Our attacks offer a variety of trade-offs, due to the freedom in the factorization form, and in the choices of the m_i's: the splitting probability gives the success probability of the attack, the other parameters determine the cost in terms of storage and computation time.

Table 1. Experimental probabilities of splitting into two factors.

Bit-length m	m_1	m_2	Probability
40	20	20	18%
	21	21	32%
	22	22	39%
	20	25	50%
64	32	32	18%
	33	33	29%
	34	34	35%
	30	36	40%

Table 2. Experimental probabilities of splitting into three factors.

Bit-length m	$m_1 = m_2 = m_3$	Probability
64	22	4%
	23	6.5%
	24	9%
	25	12%

Table 3. Experimental probabilities of splitting into four factors.

Bit-length m	$m_1 = m_2 = m_3 = m_4$	Probability
64	16	0.5%
	20	3%

1.3 Organization of the Paper

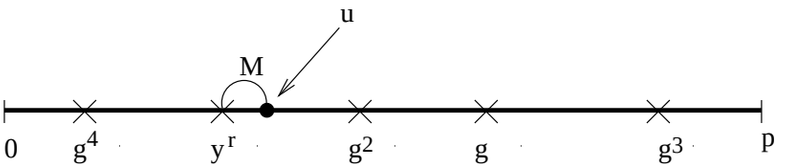
In Section 2 we introduce the subgroup rounding problems which inspire all our attacks. In Section 3 we present rounding algorithms that break plain ElGamal encryption when g generates a "small" subgroup of $\mathbb{Z}_p^*$. Using similar ideas, we present in Section 4 an attack on plain ElGamal encryption when g generates all $\mathbb{Z}_p^*$, and an attack on plain RSA in Section 5.

2 The Subgroup Rounding Problems

Recall that the ElGamal public key system [9] encrypts messages in $\mathbb{Z}_p^*$ for some prime p. Let g be an element of $\mathbb{Z}_p^*$ of order q. The private key is a number in the range $1 \leq x < q$. The public key is a tuple $\langle p, g, y \rangle$ where $y = g^x \bmod p$. To encrypt a message $M \in \mathbb{Z}_p$ the original scheme works as follows: (1) pick a random r in the range $1 \leq x < q$, and (2) compute $u = M \cdot y^r \bmod p$ and $v = g^r \bmod p$. The resulting ciphertext is the pair $\langle u, v \rangle$. To speed up the encryption process one often uses an element g of order much smaller than p. For example, p may be 1024 bits long while q is only 512 bits long.

For the rest of this section we assume $g \in \mathbb{Z}_p^*$ is an element of order q where $q \ll p$. For concreteness one may think of p as 1024 bits long and q as 512 bits long. Let G_q be the subgroup of $\mathbb{Z}_p^*$ generated by g. Observe that G_q is extremely sparse in $\mathbb{Z}_p^*$. Only one in 2^{512} elements belongs to G_q. We also assume M is a short message of length much smaller than $\log_2(p/q)$. For example, M is a 64 bits long session-key.

To understand the intuition behind the attack it is beneficial to consider a slight modification of the ElGamal scheme. After the random r is chosen one encrypts a message M by computing $u = M + y^r \bmod p$. That is, we "blind" the message by *adding* y^r rather than multiplying by it. The ciphertext is then $\langle u, v \rangle$ where v is defined as before. Clearly y^r is a random element of G_q. We obtain the following picture:



The $\times$ marks represent elements in G_q. Since M is a relatively small number, encryption of M amounts to picking a random element in G_q and then slightly

moving away from it. Assuming the elements of G_q are uniformly distributed in $\mathbb{Z}_p^*$ the average gap between elements of G_q is much larger than M. Hence, with high probability, there is a unique element $z \in G_q$ that is sufficiently close to u. More precisely, with high probability there will be a unique element $z \in G_q$ satisfying $|u - z| < 2^{64}$. If we could find z given u we could recover M. Hence, we obtain the additive version of the subgroup rounding problem:

Additive subgroup rounding: let z be an element of G_q and Δ an integer satisfying $\Delta < 2^m$. Given $u = z+\Delta \bmod p$ find z. When m is sufficiently small, z is uniquely determined (with high probability assuming G_q is uniformly distributed in $\mathbb{Z}_p$).

Going back to the original multiplicative ElGamal scheme we obtain the multiplicative subgroup rounding problem.

Multiplicative subgroup rounding: let z be an element of G_q and Δ an integer satisfying $\Delta < 2^m$. Given $u = z \cdot \Delta \bmod p$ find z. When m is sufficiently small z, is uniquely determined (with high probability assuming G_q is uniformly distributed in $\mathbb{Z}_p$).

An efficient solution to either problem would imply that the corresponding *plain* ElGamal encryption scheme is insecure. We are interested in solutions that run in time $O(\sqrt{\Delta})$ or, even better, $O(\log \Delta)$. In the next section we show a solution to the multiplicative subgroup rounding problem.

The reason we refer to these schemes as "plain ElGamal" is that messages are encrypted *as is*. Our attacks show the danger of using the system in this way. For proper security one must pre-process the message prior to encryption or modify the encryption mechanism. For example, one could use DHAES [1] or a result due to Fujisaki and Okamoto [10], or even more recently [16,13].

3 Algorithms for Multiplicative Subgroup Rounding

We are given an element $u \in \mathbb{Z}_p$ of the form $u = z \cdot \Delta \bmod p$ where z is a random element of G_q and $|\Delta| < 2^m$. Our goal is to find Δ, which we can assume to be positive. As usual, we assume that m, the length of the message being encrypted, is much smaller than $\log_2(p/q)$. Then with high probability Δ is unique. For example, take p to be 1024 bits long, q to be 512 bits long and m to be 64. We first give a simple meet-in-the-middle strategy for multiplicative subgroup rounding. By reduction to a knapsack-like problem, we will then improve both the on-line computation time and the time/memory trade-off of the method, provided that p satisfies an additional, yet realistic, assumption.

3.1 A Meet-in-the-Middle Method

Suppose Δ can be written as $\Delta = \Delta_1 \cdot \Delta_2$ where $\Delta_1 \leq 2^{m_1}$ and $\Delta_2 \leq 2^{m_2}$. For instance, one can take $m_1 = m_2 = m/2$. We show how to find Δ from u in space $O(2^{m_1})$ and $2^{m_1} + 2^{m_2}$ modular exponentiations. Observe that

$$u = z \cdot \Delta = z \cdot \Delta_1 \cdot \Delta_2 \bmod p.$$

Dividing by Δ_2 and raising both sides to the power of q yields:

$$(u/\Delta_2)^q = z^q \cdot \Delta_1^q = \Delta_1^q \bmod p.$$

We can now build a table of size 2^{m_1} containing the values $\Delta_1^q \bmod p$ for all $\Delta_1 = 0, \ldots, 2^{m_1}$. Then for each $\Delta_2 = 0, \ldots, 2^{m_2}$ we check whether $u^q/\Delta_2^q \bmod p$ is present in the table. If so, then $\Delta = \Delta_1 \cdot \Delta_2$ is a candidate value for Δ. Assuming Δ is unique, there will be only be one such candidate, although there will probably be several suitable pairs (Δ_1, Δ_2).

The algorithm above requires a priori $2^{m_2} + 2^{m_1}$ modular exponentiations and $2^{m_1} \log_2 p$ bits of memory. However, we do not need to store the complete value of $\Delta_1^q \bmod p$ in the table: A sufficiently large hash value is enough, as we are only looking for "collisions". For instance, one can take the $2\max(m_1, m_2)$ least significant bits of $\Delta_1^q \bmod p$, so that the space requirement is only $2^{m_1+1}\max(m_1, m_2)$ bits instead of $2^{m_1} \log_2 p$. Less bits are even possible, for we can check the validity of the (few) candidates obtained. Note also that the table only depends on p and q: the same table can be used for all ciphertexts. For each ciphertext, one needs to compute at most 2^{m_2} modular exponentiations. For each exponentiation, one has to check whether or not it belongs to the table, which can be done with $O(m_1)$ comparisons once the table is sorted.

It is worth noting that Δ_1 and Δ_2 need not be prime. The probability that a random m-bit integer (such as Δ) can be expressed as a product of two integers, one being less than m_1 bits and the other one being less than m_2 bits, is discussed in Section 1.2.

By choosing different values of m_1 and m_2 (not necessarily $m/2$), one obtains various trade-offs with respect to the computation time, the storage requirement, and the success probability. For instance, when the system is used to encrypt a 64-bit session key, if we pick $m_1 = m_2 = 32$, the algorithm succeeds with probability approximately 18% (with respect to the session key), and it requires on the order of eight billion exponentiations, far less than the time to compute discrete log in $\mathbb{Z}_p^*$.

We implemented the attack using Victor Shoup's NTL library [19]. The timings should not be considered as optimal, they are meant to give a rough idea of the attack efficiency, compared to exhaustive search attacks on the symmetric algorithm. Running times are given for a single 500 MHz 64-bit DEC Alpha/Linux. If $m = 40$ and $m_1 = m_2 = 20$, and we use a 160-bit q and a 512-bit p, the precomputation step takes 40 minutes, and each message is recovered in less than 1 hour and 30 minutes. From Section 1.2, it also means that, given only the public key and the ciphertext, a 40-bit message can be recovered in less than 6 hours on a single workstation, with probability 39%.

3.2 Reduction to Knapsack-like Problems

We now show how to improve the on-line computation time ($2^{m/2}$ modular exponentiations) and the time/memory trade-off of the method. We transform the multiplicative rounding problem into a linear problem, provided that p satisfies

the additional assumption $p-1 = qrs$ where $s \geq 2^m$ is such that discrete logs in subgroups of $\mathbb{Z}_p^*$ of order s can be efficiently computed. For instance, if $p_1^{e_1} \cdots p_k^{e_k}$ is the prime factorization of s, discrete logs in a cyclic group of order s can be computed with $O(\sum_{i=1}^{k} e_i(\log s + \sqrt{p_i}))$ group operations and negligible space, using Pohlig-Hellman and Pollard's ρ methods (see [12]). Let ω be a generator of $\mathbb{Z}_p^*$. For all $x \in \mathbb{Z}_p^*$, x^{qr} belongs to the subgroup G_s of order s generated by ω^{qr}.

The linear problem that we will consider is known as the k-table problem: given k tables $T_1, \ldots, T_k$ of integers and a target integer n, the k-table problem is to return all expressions (possibly zero) of n of the form $n = t_1 + t_2 + \cdots + t_k$ where $t_i \in T_i$. The general k-table problem has been studied by Schroeppel and Shamir [18], because several NP-complete problems (e.g., the knapsack problem) can be reduced to it. We will apply (slightly modified) known solutions to the k-table problems, for $k = 2, 3$ and 4.

The Modular 2-Table Problem Suppose that Δ can be written as $\Delta = \Delta_1 \cdot \Delta_2$, with $0 \leq \Delta_1 \leq 2^{m_1}$ and $0 \leq \Delta_2 \leq 2^{m_2}$, as in Section 3.1. We have $u^q = \Delta_1^q \Delta_2^q \bmod p$ and therefore:

$$u^{qr} = \Delta_1^{qr} \Delta_2^{qr} \bmod p,$$

which can be rewritten as

$$\log(u^{qr}) = \log(\Delta_1^{qr}) + \log(\Delta_2^{qr}) \bmod s,$$

where the logarithms are with respect to ω^{qr}.

We build a table T_1 consisting of $\log(\Delta_1^{qr})$ for all $\Delta_1 = 0, \ldots, 2^{m_1}$, and a table T_2 consisting of $\log(\Delta_2^{qr})$ for all $\Delta_2 = 0, \ldots, 2^{m_2}$. These tables are independent of Δ. The problem is now to express $\log(u^{qr})$ as a modular sum $t_1 + t_2$, where $t_1 \in T_1$ and $t_2 \in T_2$. The number of targets $t_1 + t_2$ is $2^{m_1+m_2}$. Hence, we expect this problem to have very few solutions when $s \geq 2^{m_1+m_2}$. The problem involves modular sums, but it can of course be viewed as a 2-table problem with two targets $\log(u^{qr})$ and $\log(u^{qr}) + s$. The classical method to solve the 2-table problem with a target n is the following:

1. Sort T_1 in increasing order;
2. Sort T_2 in decreasing order;
3. Repeat until either T_1 or T_2 becomes empty (in which case all solutions have already been found):
   (a) Compute $t = \text{first}(T_1) + \text{first}(T_2)$.
   (b) If $t = n$, output the solution which has been found, and delete first(T_1) from T_1, and first(T_2) from T_2;
   (c) If $t < n$ delete first(T_1) from T_1;
   (d) If $t > n$ delete first(T_2) from T_2;

It is easy to see that the method outputs all solutions of the 2-table problem, in time $2^{\min(m_1,m_2)+1}$. The space requirement is $O(2^{m_1} + 2^{m_2})$.

Since the original problem involves modular sums, it seems at first glance that we have to apply the previous algorithm twice (with two different targets). However, we note that a simple modification of the previous algorithm can in fact solve the modular 2-table problem (that is, the 2-table problem with modular additions instead of integer additions). The basic idea is the following. Since T_2 is sorted in descending order, $n - T_2$ is sorted in ascending order. The set $(n - T_2) \bmod s$ though not necessarily sorted, is almost sorted. More precisely, two adjacent numbers are always in the right order, to the exception of a single pair. This is because $n - T_2$ is contained in an interval of length s. The single pair of adjacent numbers in reverse order corresponds to the two elements a and b of T_2 surrounding $s - n$. These two elements can easily be found by a simple dichotomy search for $s - n$ in T_2. And once the elements are known, we can access $(n - T_2) \pmod s$ in ascending order by viewing T_2 as a circular list, starting our enumeration of T_2 by b, and stopping at a.

The total cost of the method is the following. The precomputation of tables T_1 and T_2 requires $2^{m_1} + 2^{m_2}$ modular exponentiations and discrete log computations in a subgroup of $\mathbb{Z}_p^*$of order s, and the sort of T_1 and T_2. The space requirement is $(2^{m_1} + 2^{m_2}) \log_2 s$ bits. For each ciphertext, we require one modular exponentiation, one efficient discrete log (to compute the target), and $2^{\min(m_1,m_2)+1}$ additions. Hence, we improved the on-line work of the method of Section 3.1: loosely speaking, we replaced modular exponentiations by simple additions. We now show how to decrease the space requirement of the method.

The Modular 3-Table Problem The previous approach can easily be extended to an arbitrary number of factors of Δ. Suppose for instance Δ can be written as $\Delta = \Delta_1 \cdot \Delta_2 \cdot \Delta_3$ where each Δ_i is less than 2^{m_i}. We obtain

$$\log(u^{qr}) = \sum_{i=1}^{3} \log(\Delta_i^{qr}) \bmod s,$$

where the logarithms are with respect to ω^{qr}. In a precomputation step, we compute in a table T_i all the logarithms of $\Delta_i^{qr} \bmod p$ for $0 \leq \Delta_i < 2^{m_i}$. We are left with a modular 3-table problem with target $\log(u^q r)$. The modular 3-table problem with target n modulo s can easily be solved in time $O(2^{m_1+\min(m_2,m_3)})$ and space $O(2^{m_1} + 2^{m_2} + 2^{m_3})$. It suffices to apply the modular 2-table algorithm on tables T_2 and T_3, for all targets $(n - t_1) \bmod s$, with $t_1 \in T_1$.

Hence, we decreased the space requirement of the method of Section 3.2, by (slightly) increasing the on-line computation work and decreasing the success probability (see Section 1.2 for the probability of splitting into three factors). More precisely, if $m_1 = m_2 = m_3 = m/3$, the on-line work is one modular exponentiation, one discrete log in a group of order s, and $2^{2n/3}$ additions. Since an addition is very cheap, this might be useful for practical purposes.

The Modular 4-Table Problem Using 3 factors did not improve the time/memory trade-off of the on-line computation work. Indeed, for both modular 2-table and modular 3-table problems, our algorithms satisfy $TS = O(2^m)$, where

T is the number of additions, and S is the space requirement. Surprisingly, one can obtain a better time/memory tradeoff with 4 factors.

Suppose Δ can be written as $\Delta = \Delta_1 \cdot \Delta_2 \cdot \Delta_3 \cdot \Delta_4$ where each Δ_i is less than 2^{m_i}. For instance, one can take $m_1 = m_2 = m_3 = m_4 = m/4$. We show how to find Δ from $\log(u^{qr})$ in time $O(2^{m_1+m_2} + 2^{m_3+m_4})$ and space $O(\sum_{i=1}^{4} 2^{m_i})$, provided a precomputation stage of $\sum_{i=1}^{4} 2^{m_i}$ modular exponentiations and discrete log computations in a group of order s.

We have $\log(u^{qr}) = \sum_{i=1}^{4} \log(\Delta_i^{qr}) \bmod s$. Again, in a precomputation step, we compute in a table T_i all the logarithms of $\Delta_i^{qr} \bmod p$ for $0 \leq \Delta_i < 2^{m_i}$. We are left with a modular 4-table problem, whose solutions will reveal possible choices of Δ_1, Δ_2, Δ_3 and Δ_4. Schroeppel and Shamir [18] proposed a clever solution to the basic 4-table problem, using the following idea. An obvious solution to the 4-table problem is to solve a 2-table problem by merging two tables, that is, considering sums $t_1 + t_2$ and $t_3 + t_4$ separately. However, the algorithm for the 2-table algorithm described in Section 3.2 accesses the elements of the sorted supertables sequentially, and thus there is no need to store all the possible combinations simultaneously in memory. All we need is the ability to generate them quickly (on-line, upon request) in sorted order. To implement this idea, two priority queues are used :

- Q' stores pairs (t_1, t_2) from $T_1 \times T_2$, enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pairs with the smallest $t_1 + t_2$ sum accessible in constant time.
- Q'' stores pairs (t_3, t_4) from $T_3 \times T_4$, enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pairs with the largest $t_3 + t_4$ sum accessible in constant time.

This leads to the following algorithm for a target n:

1. Precomputation:
   - Sort T_2 into increasing order, and T_4 into decreasing order;
   - Insert into Q' all the pairs $(t_1, \text{first}(T_2))$ for $t_1 \in T_1$;
   - Insert into Q'' all the pairs $(t_3, \text{first}(T_4))$ for $t_3 \in T_3$.
2. Repeat until either Q' or Q'' becomes empty (in which case all solutions have been found):
   - Let (t_1, t_2) be the pair with smallest $t_1 + t_2$ in Q';
   - Let (t_3, t_4) be the pair with largest $t_3 + t_4$ in Q'';
   - Compute $t = t_1 + t_2 + t_3 + t_4$.
   - If $t = n$, we output the solution, and apply what is planned when $t < n$ or $t > n$.
   - If $t < n$ do
     - delete (t_1, t_2) from Q';
     - if the successor t_2' of t_2 in T_2 is defined, insert (t_1, t_2') into Q';
   - If $t > n$ do
     - delete (t_3, t_4) from Q'';
     - if the successor t_4' of t_4 in T_4 is defined, insert (t_3, t_4') into Q'';

At each stage, a $t_1 \in T_1$ can participate in at most one pair in Q', and a $t_3 \in T_3$ can participate in at most one pair in Q''. It follows that the space complexity of the priority queues is bounded by $O(|T_1| + |T_3|) = O(2^{m_1} + 2^{m_3})$. Each possible pair can be deleted from Q' at most once, and the same holds for Q''. Since at each iteration, one pair is deleted from Q' or Q'', the number of iterations cannot exceed the number of possible pairs, which is $O(2^{m_1+m_2} + 2^{m_3+m_4})$.

Finally, as in the 2-table case, we note that this algorithm can be adapted to modular sums, by changing the starting points in T_2 and T_4 to make sure that the modular sets are enumerated in the correct order. Hence, it is not necessary to apply the 4-table algorithm on 4 targets. If $m_1 = m_2 = m_3 = m_4 = m/4$, we obtain a time complexity of $O(2^{m/2})$ and a space complexity of only $O(2^{m/4})$, which improves the time/memory tradeoff of the methods of Sections 3.2 and 3.2. The probability that a random m-bit integer (such as Δ) can be expressed as a product of four integers Δ_i, where Δ_i has less than m_i bits, is given in Section 1.2. Different values of m_1, m_2, m_3 and m_4 (not necessarily $m/4$), give rise to different trade-offs with respect to the computation time, the storage requirement, and the success probability.

Our experiments show that, as expected, the method requires much less computing power than a brute-force attack on the 64-bit key using the symmetric encryption algorithm. We implemented the attack on a PII/Linux-400 MHz. Here is a numerical example, using DSS-like parameters:

$q = 762503714763387752235260732711386742425586145191$

$p = 124452971950208973279611466845692849852574447655208586550576344180427926821830$
$386338947599247842658333549269645045449033209411448963415127034470249728876 81$

The 160-bit number q divides the 512-bit number $p - 1$. The smooth part of $p - 1$ is $4783 \cdot 1759 \cdot 1627 \cdot 139 \cdot 113 \cdot 41 \cdot 11 \cdot 7 \cdot 5 \cdot 2^7$, which is a 69-bit number. Our attack recovered the 64-bit secret message 14327865741237781950 in only 2 hours and a half (we were lucky, as the maximal running time for 64 bits should be around 14 hours).

4 An Attack on ElGamal Using a Generator of $\mathbb{Z}_p^*$

So far, our attacks on ElGamal encryption apply when the public key $\langle p, g, y\rangle$ uses an element $g \in \mathbb{Z}_p^*$ whose order is much less than p. Although many implementations of ElGamal use such g, it is worth studying whether a "meet-in-the-middle attack" is possible when g generates all of $\mathbb{Z}_p^*$. We show that the answer is positive, although we cannot directly use the algorithm for subgroup rounding.

Let $\langle p, g, y\rangle$ be an ElGamal public key where g generates all of $\mathbb{Z}_p^*$. Suppose an m-bit message M is encrypted using plain ElGamal, i.e. the ciphertext is $\langle u, v\rangle$ where $u = M \cdot y^r$ and $v = g^r$. Suppose s is a factor of $p - 1$ so that in the subgroup of $\mathbb{Z}_p^*$ or order s the discrete log problem is not too difficult (as in Section 3.2), i.e. takes time T for some small T. For example, s may be an integer with only small prime divisors (a smooth integer).

We show that when $s > 2^m$ it is often possible to recover the plaintext from the ciphertext in time $2^{m/2}m$ plus the time it takes to compute one discrete log

in the subgroup of $\mathbb{Z}_p^*$ of order s. We refer to this subgroup as G_s. Note that when M is a 64-bit session key the only constraint on p is that $p-1$ have a 64 bit smooth factor.

Let $u = M \cdot y^r$ and $v = g^r$ be an ElGamal ciphertext. As before, suppose $M = M_1 \cdot M_2$ where both M_1 and M_2 are less than $2^{m/2}$. Let $q = (p-1)/s$ then: $M_1 y^r = u/M_2 \bmod p$. Hence,

$$M_1^q (y^r)^q = u^q / M_2^q \bmod p$$

We cannot use the technique of Section 3.1 directly since we do not know the value of y^{rq}. Fortunately, y^{rq} is contained in G_s. Hence, we can compute y^{rq} directly using the public key y and $v = g^r$. Indeed, suppose we had an integer a such that $y^q = (g^q)^a$. Then $y^{rq} = g^{rqa} = v^{qa}$. Computing a amounts to computing a single discrete log in G_s. Once a is found the problem is reduced to finding $\langle M_1, M_2 \rangle$ satisfying:

$$M_1^q v^{qa} = u^q / M_2^q \bmod p \tag{3}$$

The techniques of Section 3.1 can now be used to find all such $\langle M_1, M_2 \rangle$ in the time it takes to compute $2^{m/2}$ exponentiations. Since the subgroup G_s contains at least 2^m elements the number of solutions is bounded by m. The correct solution can then be easily found by other means, *e.g.* by trying all m candidate plaintexts until one of them succeeds as a "session-key".

Note that all the techniques of Section 3.2 can also be applied. The on-line work of $2^{m/2}$ modular exponentiations is then decreased to $2^{m/2}$ additions, provided the precomputation of many discrete log in G_s. Indeed, by taking logarithms in (3), one is left with a modular 2-table problem. Splitting the unknown message M in a different number of factors leads to other modular k-table problems. One can thus obtain various trade-offs with respect to the computation time, the memory space, and the probability of success, as described in Section 3.2.

To summarize, when g generates all of $\mathbb{Z}_p^*$ the meet-in-the-middle attack can often be used to decrypt ElGamal ciphertexts in time $2^{m/2}$ as long as $p-1$ contains an m-bit smooth factor.

5 A Meet-in-the-Middle Attack on Plain RSA

To conclude we remark that the same technique used for the subgroup rounding problem can be used to attack plain RSA. This was also mentioned in [8]. In its simplest form, the RSA system [17] encrypts messages in $\mathbb{Z}_N$ where $N = pq$ for some large primes p and q. The public key is $\langle N, e \rangle$ and the private key is d, where $e \cdot d = 1 \bmod \phi(N)$ with $\phi(N) = (p-1)(q-1)$. A message $M \in \mathbb{Z}_N$ is then encrypted into $c = M^e \bmod N$. To speed up the encryption process one often uses a public exponent e much smaller than N, such as $e = 2^{16} + 1$.

Suppose the m-bit message M can be written as $M = M_1 M_2$ with $M_1 \leq 2^{m_1}$ and $M_2 \leq 2^{m_2}$. Then:

$$\frac{c}{M_2^e} = M_1^e \bmod N.$$

We can now build a table of size 2^{m_1} containing the values $M_1^e \bmod N$ for all $M_1 = 0, \ldots, 2^{m_1}$. Then for each $M_2 = 0, \ldots, 2^{m_2}$, we check whether $c/M_2^e \bmod N$ is present in the table. Any collision will reveal the message M. As in Section 3.1, we note that storing the complete value of $M_1^e \bmod N$ is not necessary: for instance, storing the $2\max(m_1, m_2)$ least significant bits should be enough. The attack thus requires $2^{m_1+1}\max(m_1, m_2)$ bits of memory and takes 2^{m_2} modular exponentiations (we can assume that the table sort is negligible, compared to exponentiations).

Using a non-optimized implementation (based on the NTL [19] library), we obtained the following results. The timings give a rough idea of the attack efficiency, compared to exhaustive search attacks on the symmetric algorithm. Running times are given for a single 500 MHz 64-bit DEC Alpha/Linux. If $m = 40$ and $m_1 = m_2 = 20$, and we use a public exponent $2^{16}+1$ with a 512-bit modulus, the precomputation step takes 3 minutes, and each message is recovered in less than 10 minutes. From Section 1.2, it also means that, given only the public key and the ciphertext, a 40-bit message can be recovered in less than 40 minutes on a single workstation, with probability at least 39%.

6 Summary and Open Problems

We showed that plain RSA and plain ElGamal encryption are fundamentally insecure. In particular, when they are used to encrypt an m-bit session-key, the key can often be recovered in time approximately $2^{m/2}$. Hence, although an m-bit key is used, the *effective* security provided by the system is only $m/2$ bits. Theses results demonstrate the importance of adding a preprocessing step such as OAEP to RSA and a process such as DHAES to ElGamal. The attack presented in the paper can be used to motivate the need for preprocessing in introductory descriptions of these systems.

There are a number of open problems regarding this attack:

Problem 1: Is there a $O(2^{m/2})$ time algorithm for the multiplicative subgroup rounding problem that works for all Δ?

Problem 2: Is there a $O(2^{m/2})$ time algorithm for the additive subgroup rounding problem?

Problem 3: Can either the multiplicative or additive problems be solved in time less than $\Omega(2^{m/2})$? Is there a sub-exponential algorithm (in 2^m)?

Acknowledgments

We thank Paul van Oorschot and David Naccache for several conversations on this problem. We thank Adi Shamir for informing us of reference [18]. We thank Igor Shparlinski for providing us (1) and informing us of reference [11]. We thank Carl Pomerance for providing us (2) and helpful information on splitting probabilities.

References

1. M. Abdalla, M. Bellare, P. Rogoway, "DHAES: An encryption scheme based on the Diffie-Hellman problem", manuscript, 1998.
2. R. J. Anderson, S. Vaudenay, "Minding your p's and q's", Proc of Asiacrypt '96, LNCS 1163, Springer-Verlag, pp. 26–35, 1996.
3. C. Batut, K. Belabas, D. Bernardi, H. Cohen, M. Olivier, "Pari/GP computer package version 2", available at `http://hasse.mathematik.tu-muenchen.de/ntsw/pari/Welcome`.
4. M. Bellare, P. Rogaway, "Optimal asymmetric encryption — how to encrypt using RSA", Proc. Eurocrypt '94, LNCS 950, Springer-Verlag, 1995.
5. D. Boneh, "The Decision Diffie-Hellman Problem", Proc. ANTS-III, LNCS 1423, Springer-Verlag, 1998.
6. D. Boneh, "Twenty Years of Attacks on the RSA cryptosystem", Notices of the AMS, 46(2):203–213, 1999.
7. J.-S. Coron, D. Naccache, J. P. Stern, "On the Security of RSA Padding", Proc. of Crypto '99, LNCS 1666, Springer-Verlag, pp. 1–18, 1999.
8. J.-S. Coron, M. Joye, D. Naccache, P. Paillier, "New Attacks on PKCS#1 v1.5 Encryption", Proc. of Eurocrypt '2000, LNCS 1807, Springer-Verlag, pp. 369–381, 2000.
9. T. ElGamal, "A public key cryptosystem and a signature scheme based on the discrete logarithm", IEEE Trans. on Information Theory, 31(4):469–472, 1985.
10. E. Fujisaki, T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes", Proc. of Crypto '99, LNCS 1666, Springer-Verlag, pp. 537–554, 1999.
11. R. R. Hall, G. Tenenbaum, "Divisors", Cambridge University Press, 1988.
12. A. Menezes, P. v. Oorschot, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1997.
13. T. Okamoto and D. Pointcheval, "PSEC-3: Provably Secure Elliptic Curve Encryption Scheme", Submission to IEEE P1363a, 2000.
14. P. v Oorschot, M. J. Wiener, "On Diffie-Hellman Key Agreement With Short Exponents", Proc. Eurocrypt '96, LNCS 1070, Springer-Verlag, 1996.
15. PKCS1, "Public Key Cryptography Standard No. 1 Version 2.0", RSA Labs.
16. D. Pointcheval, "Chosen-Ciphertext Security for any One-Way Cryptosystem", Proc. PKC '2000, LNCS 1751, Springer-Verlag, 2000.
17. R. L. Rivest., A. Shamir, L. M. Adleman " A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2):120–126, 1978.
18. R. Schroeppel, A. Shamir, "A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems", SIAM J. Comput., 10(3):456–464, 1981.
19. V. Shoup, "Number Theory C++ Library (NTL) version 3.7", available at `http://www.shoup.net/`.