Exposing an RSA Private Key Given a Small Fraction of its Bits

Dan Boneh*
dabo@cs.stanford.edu
Stanford University

Glenn Durfee†
gdurf@cs.stanford.edu
Stanford University

Yair Frankel
yfrankel@cryptographers.com
TechTegrity LLC

Abstract

We show that for low public exponent RSA, given a quarter of the bits of the private key an adversary can recover the entire private key. Similar results (though not as strong) are obtained for larger values of e. For instance, when e is a prime in the range $[N^{1/4}, N^{1/2}]$, half the bits of the private key suffice to reconstruct the entire private key. Our results point out the danger of partial key exposure in the RSA public key system.

1 Introduction

Let $N = pq$ be an RSA modulus and let e, d be encryption/decryption exponents, i.e., $ed \equiv 1 \bmod \phi(N)$. We study the following question: how many bits of d does an adversary require in order to reconstruct all of d? Surprisingly, we show that for low public exponent RSA, given only a quarter of the least significant bits of d, an adversary can efficiently recover all of d. We obtain similar results, summarized in the next subsection, for larger values of e as well. Our results show that RSA, and particularly low public exponent RSA, are vulnerable to *partial key exposure*. We refer to this class of attacks as *partial key exposure attacks*.

To motivate this problem consider a computer system which has an RSA private key stored on it. An adversary may attempt to attack the system in a variety of ways in order to obtain the private key. Some attacks (e.g. a timing attack [13]) are able to reveal some bits of the key, but may fail to reveal the entire key [8]. Our results show that attacks, such as the timing attack on RSA, need only be carried out until a quarter of the least significant bits of d are exposed. Once these bits are revealed the adversary can efficiently compute all of d. Another scenario where partial key exposure comes up is in the presence of covert channels. Such channels are often slow or have a bounded capacity. Our results show that as long as a fraction of the private exponent bits can be leaked, the remaining bits can be reconstructed.

It is natural to ask the analogous question in the context of discrete log schemes. For instance, given a fraction of the bits of the private key in the ElGamal public key system [11], can one efficiently recover the entire key? There is no known method for doing so. Furthermore, the common belief is that no such efficient algorithm exists. This resistance to partial key exposure is an interesting distinction between RSA and discrete log schemes.

Let $N = pq$ be an n-bit RSA modulus. Throughout the paper we view the private exponent

*Supported by DARPA contract #F30602-97-C-0326.
†Supported by a NSF Graduate Research Fellowship and a Stanford Graduate Fellowship.

d as an n-bit string. When referring to the t most significant bits of d we refer to the t leftmost bits of d when viewed canonically as an n-bit string. For instance, it is possible that the t most significant bits of d are all zero, for some t. Similarly, a quarter of the bits of d always refers to $n/4$ bits.

1.1 Summary of Results

We summarize our results in the following two theorems. The proofs are given in the body of the paper. The first theorem applies to low public exponent RSA. The second applies to larger values of e. Throughout we assume $N = pq$ is an RSA modulus with $\sqrt{N}/2 < q < p < 2\sqrt{N}$.

Theorem 1.1 *Let $N = pq$ be an n-bit* RSA *modulus with $N = 3 \bmod 4$. Let $1 \leq e, d \leq \phi(N)$ satisfy $ed \equiv 1 \bmod \phi(N)$ and $e < 2^{(n/4)-3}$. Then there is an algorithm that given $\langle N, e\rangle$ and the $n/4$ least significant bits of d computes all of d in polynomial time in n and e.*

As we shall see, the running time of the attack algorithm in the above theorem is in fact *linear* in $e \log_2 e$. Consequently, as long as e is not "too large" the attack can be efficiently mounted. For a very small value of e such as $e = 3$ we will show in Section 5 that the attack runs in a reasonable amount of time. For larger values, such as $e = 65537$, the attack is still feasible, though clearly takes much longer.

Theorem 1.2 *Let $N = pq$ be an n-bit* RSA *modulus. Let $1 \leq e, d \leq \phi(N)$ satisfy $ed \equiv 1 \bmod \phi(N)$.*

1. *Suppose e is a prime in the range $\{2^t, \ldots, 2^{t+1}\}$ with $n/4 \leq t \leq n/2$. Then given the t most significant bits of d there is a polynomial time (in n) algorithm to compute all of d.*

2. *More generally, suppose $e \in \{2^t, \ldots, 2^{t+1}\}$ is the product of at most r distinct primes with $n/4 \leq t \leq n/2$. Then given the factorization of e and the t most significant bits of d there is an algorithm to compute all of d in time polynomial in n and 2^r.*

3. *When the factorization of e is unknown, we obtain a weaker result. Suppose e is in the range $\{2^t, \ldots, 2^{t+1}\}$ with $t \in 0 \ldots n/2$. Further, suppose $d > \epsilon N$ for some $\epsilon > 0$. Then there is a polynomial time (in n and $1/\epsilon$) algorithm that given the $n - t$ most significant bits of d, computes all of d.*

Theorem 1.2 applies to public exponents e in the range $2^{n/4} \leq e \leq 2^{n/2}$. Unlike the previous theorem, Theorem 1.2 makes use of the *most* significant bits of d. When e is prime, at most half the bits of d are required to mount the attack. Fewer bits are needed when e is smaller. Indeed, if e is close to $N^{1/4}$ only a quarter of the MSB bits of d are required. The same result holds when e is not prime, as long as we are given the factorization of e and e does not have too many distinct prime factors. The last part of the theorem applies to $e < N^{1/2}$ when the factorization of e is not known. To mount the attack, at least half the MSB bits of d are required. More bits are necessary, the smaller e is. The attack algorithm works for most e, but may fail if d is significantly smaller than N.

One may refine Theorem 1.2 in many ways. It is possible to obtain other results along these lines for public exponents $e < N^{1/2}$. For instance, consider the case when the factorization of e is unknown. If the adversary is given half the most significant bits of d and a quarter of the least significant bits then we show the adversary can recover all of d. When $e < N^{1/4}$ this is better than the results of Theorem 1.2 part (3). However, we view attacks that require non-consecutive bits of d as artificial. We briefly sketch these variants in Section 4.3.

1.2 Related Work

We note that Wiener [24] showed that RSA is insecure whenever the private exponent d is less than $N^{1/4}$ (the bound was recently improved by Boneh and Durfee [3] to $N^{(1-1/\sqrt{2})} \approx N^{0.292}$). In other words, Wiener's bound shows that when the 3/4 most significant bits of d are zero an adversary can efficiently recover the remaining quarter. These results do not apply to our problem: Wiener's continued fractions approach, as well as the method used by Boneh and Durfee, does not work efficiently when the most significant bits of d are given to the adversary but they are non-zero.

Maurer [17] studies the problem of factoring an RSA modulus N with the help of an (infinitely powerful) oracle that answers arbitrary yes/no questions. He shows that ϵn queries suffice to factor N in probabilisitic polynomial time with failure probability at most $N^{-\epsilon/2}$. One may view our results as a method for factoring N using between $n/4$ and $3n/4$ queries of an oracle, but one based on a weaker (and seemingly more natural) assumption. Namely, when viewed in this way, our attacks use an oracle that is restricted to revealing certain bits of an RSA private exponent.

There has been recent work on protecting against partial key exposure using *exposure resilient cryptography*. Initial progress was made several years ago by Chor et al. [5], who develop the notion of *t-resilient functions* and discuss an application to protection against partially leaked cryptographic keys. More recent work has focused on *exposure resilient functions*, introduced by Canetti et al. [4], which can be used to protect against partial key exposure for a variety of cryptographic primitives. These ideas were later refined by Dodis [9] and Dodis, Sahai, and Smith [10]. For the RSA public key cryptosystem, Steinfeld and Zheng [23] have shown that by carefully choosing p and q (instead of selecting them at random as is typical in practice) the low public exponent partial key exposure attacks presented in this paper can be prevented.

1.3 Notation

Throughout the paper we let $N = pq$ denote an n-bit RSA modulus. We assume the primes p and q are distinct and close to $\sqrt{N}$. More precisely, we assume

$$4 < \sqrt{N}/2 < q < p < 2\sqrt{N} \tag{1}$$

Our results also apply to RSA moduli $N = pq$ where p is much larger than q, but we do not give the details here.

Notice that equation (1) implies $p + q < 3\sqrt{N}$. For convenience, throughout the paper we set

$$s := p + q.$$

Under the assumption $p > q$ this implies:

$$p = \frac{1}{2}(s + \sqrt{s^2 - 4N}). \tag{2}$$

Furthermore, it follows by equation (1) that

$$N/2 < N - 4\sqrt{N} < \phi(N) < N, \tag{3}$$

where $\phi(N) = N - p - q + 1$ is the Euler totient function.

Let $1 \leq e, d \leq \phi(N)$ be encryption/decryption exponents. Then $ed \equiv 1 \pmod{\phi(N)}$. Throughout the paper we denote by k the unique integer such that:

$$ed - k\phi(N) = ed - k(N - s + 1) = 1. \tag{4}$$

Since $\phi(N) > d$ we know that $k < e$.

2 Finding Small Solutions to Bivariate Polynomials

Our results make heavy use of seminal results due to Coppersmith. Using the lattice basis reduction algorithm of Lenstra, Lenstra, and Lovasz [14], Coppersmith [6] shows how to find small solutions (x_0, y_0) to a bivariate polynomial $f(x, y)$, provided appropriate bounds on x_0 and y_0 are known in advance.

Theorem 2.1 (Coppersmith[6]) *Let $f(x, y)$ be a polynomial in two variables over $\mathbb{Z}$, of maximum degree δ in each variable separately, and assume the coefficients of f are relatively prime as a set. Let X, Y be bounds on the desired solutions x_0, y_0. Define $\tilde{f}(x, y) := f(Xx, Yy)$ and let D be the absolute value of the largest coefficient of $\tilde{f}$. If $XY < D^{2/(3\delta)}$, then in time polynomial in $\log D$ and 2^δ, we can find all integer pairs (x_0, y_0) with $p(x_0, y_0) = 0, |x_0| < X, |y_0| < Y$.*

We make use of an immediate consequence of this theorem, which is a slight generalization of a result in [6].

Corollary 2.2 *Let $N = pq$ be an n-bit* RSA *modulus. Let $r \geq 2^{n/4}$ be given and suppose $p_0 := p \bmod r$ is known. Then it is possible to factor N in time polynomial in n. (We denote this running time $T_C(n)$).*

Proof Without loss of generality, assume p and r are relatively prime (otherwise the problem is trivial). Then, p_0 and r are also relatively prime. From $p_0 := p \bmod r$ we may find $q_0 := q \equiv N/p_0 \bmod r$. We seek a solution (x_0, y_0) to $f(x, y) = (rx + p_0)(ry + q_0) - N$, where $0 \leq x_0 < X = 2^{n/2+1}/r$ (similarly $y_0 < Y = 2^{n/2+1}/r$). Notice, however, that the greatest common divisor of the coefficients of the polynomial $f(x, y)$ is r, so to use Theorem 2.1, we must divide through by r to get a new polynomial $g(x, y) = f(x, y)/r$. Now notice that the largest coefficient of $\tilde{g}(x, y) = g(Xx, Yy)$ is at least $2^{n+2}/r$. So, to use Theorem 2.1 we require

$$XY = r^{-2}2^{n+2} < (2^{n+2}/r)^{2/3},$$

which is satisfied whenever $r > 2^{(n+2)/4}$. By exhaustive search on the first two bits of x_0 and y_0 this can be reduced to $r \geq 2^{n/4}$. ∎

Throughout the remainder of the paper, we denote by $T_C(n)$ the running time of the algorithm of Corollary 2.2. We remark that $T_C(n) = \Omega(n^3)$, so the running time of this algorithm dominates other arithmetic operations in our attacks.

3 Partial key exposure attack on low exponent RSA

In this section we consider attacks on the RSA cryptosystem with a "small" exponent e. For our purposes, "small" implies that exhaustive search on all values less than e is feasible. In particular, since $k \leq e$ holds, our attack algorithm can exhaustively search all possible candidates k' for k (recall that k is the unique integer satisfying $de - k\phi(N) = 1$). We can now prove Theorem 1.1, restated more precisely as follows.

Theorem 3.1 *With the notation as in Section 1.3, suppose $N \equiv 3 \bmod 4$ and $e \leq 2^{(n/4)-3}$. Then given the $n/4$ least significant bits of d, we can factor N in time $O(T_C(n) \cdot e \log_2 e)$.*

Proof Suppose we are given the least-significant $n/4$-bit block of d; that is, we know some d_0 such that $d_0 \equiv d \bmod 2^{n/4}$. Reducing equation (4) modulo $2^{n/4}$ and substituting $s = p + N/p$, we see that $p \bmod 2^{n/4}$ is a solution for x in the equation

$$ed_0 \equiv 1 + k(N - x - N/x + 1) \pmod{2^{n/4}}.$$

This leads to the following quadratic equation in x:

$$kx^2 + (ed_0 - k(N+1) - 1)x + kN \equiv 0 \pmod{2^{n/4}}. \tag{5}$$

The attack works as follows: for every candidate value k' for k in the range $\{1, \ldots, e\}$ do:
1. Use the algorithm outlined below to compute all solutions for $x \bmod 2^{n/4}$ to the equation

$$k'x^2 + (ed_0 - k'(N+1) - 1)x + k'N \equiv 0 \pmod{2^{n/4}}. \tag{6}$$

2. For each solution $x_0 \bmod 2^{n/4}$ so obtained use the algorithm of Corollary 2.2 to attempt to factor N. Since one of these solutions will satisfy $x_0 \equiv p \bmod 2^{n/4}$ the attack will succeed in factoring N.

To complete the description of this attack we must describe the algorithm to obtain solutions to equation (6) and we must bound the total number of solutions as k' ranges over all candidates for k. Consider first the case $k' = k$. Notice that k divides the coefficients of the quadratic term and the constant term of equation (5); to see that k divides the coefficient of the linear term, observe $ed_0 - k(N+1) - 1 = -k(p+q)$. Suppose $k = 2^{t_k}m$ for some integer t_k and odd integer m. Then every solution x to equation (5) satisfies

$$mx^2 - m(p+q)x + mN \equiv 0 \pmod{2^{(n/4)-t_k}}.$$

Furthermore, m is odd so $m^{-1} \bmod 2^{(n/4)-t_k}$ is well-defined, thus every such solution x satisfies

$$x^2 - (p+q)x + N \equiv 0 \pmod{2^{(n/4)-t_k}}. \tag{7}$$

We may complete the square to obtain[1]

$$(x - (p+q)/2)^2 \equiv ((p-q)/2)^2 \pmod{2^{(n/4)-t_k}}. \tag{8}$$

[1]We note that the fractions in this expression represent divison over the integers, which is well-defined since $p \equiv q \bmod 2$.

D. Redmond [20] describes an algorithm to find a solutions to equations of the form $y^2 \equiv c \bmod 2^u$, when $c \equiv 1 \bmod 8$ and $u \geq 3$. Since $N \equiv 3 \bmod 4$, we have $p \not\equiv q \bmod 4$, and it follows that $((p-q)/2)^2 \equiv 1 \bmod 8$. Furthermore, $t_k \leq \log_2 e \leq (n/4) - 3$, so $(n/4) - t_k \geq 3$. Thus equation (8) is in precisely the required form. For completeness, we summarize this algorithm in Lemma A.1 in Appendix A.

The above produces one solution to equation (7). Lemma A.3 in Appendix A shows that there are exactly four solutions to this equation and that given one of these solutions the other three are easy to obtain. Call these four solutions $\{x_1, x_2, x_3, x_4\}$. It follows that the solutions to equation (5) are $x_i + j2^{(n/4)-t_k}$ for $i \in \{1, 2, 3, 4\}$ and $j \in \{0, \ldots, 2^{t_k} - 1\}$. Thus, there are 2^{2+t_k} solutions to equation (5).

In summary, when $k' = k$, we expect equation (6) to have at most 2^{2+t_k} solutions. Suppose k' is an arbitrary candidate for k, and let $t_{k'}$ be the largest integer such that $2^{t_{k'}}$ divides k'. If the attacker finds any of the required conditions to use the aforementioned approach violated (for instance, that the number of solutions to equation (6) is greater than $2^{2+t_{k'}}$), then it may immediately conclude that $k' \neq k$, and proceed to the next candidate k'. So $2^{2+t_{k'}}$ is an upper bound on the number of solutions that need to be checked for any candidate k'.

From the above discussion we may conclude that when k' is odd, there are at most four solutions to equation (6) to check; if $k' \equiv 2 \bmod 4$, then there are at most eight solutions to check, and so on. Since k' is exhaustively searched in the range $\{0, \ldots, e\}$, this implies that at most $4e\lceil\log_2 e\rceil$ solutions to equation (6) will be tested before the correct value of $x \equiv p \bmod 2^{n/4}$ is obtained, exposing the factorization of N. Each solution is tested using one run of the algorithm of Corollary 2.2; since $T_C(n)$ dominates the running time of the other arithmetic operations (including root-finding and use of Lemma A.1), this leads to a total running time that is $O(T_C(n) \cdot e \log_2 e)$. ■

Remark 1. The case $N \equiv 1 \bmod 4$ requires careful analysis, and R. Steinfeld and Y. Zheng show that a modification of the above attack has an expected running time of $O(T_C(n) \cdot n \cdot e \log_2 e)$ over random choice of $n/2$-bit primes p and q [23]. They go on to describe how to choose p and q to resist the above partial key exposure attack; this is achieved by forcing $p \equiv q \bmod 2^u$ for large values of u.

We did not employ the full generality of Corollary 2.2, as mod r bits could have been used for any $r \geq 2^{n/4}$. This will be used in the next section where we consider more sophisticated key exposure attacks.

One may wonder whether a similar partial key exposure attack is possible using the most significant bits of d. The answer is no. The reason is that low public exponent RSA leaks half the most significant bits of d. In other words, the adversary may obtain *half* the most significant bits of d from e and N alone. Consequently, revealing the most significant bits of d does not help the adversary in exposing the rest of d. This is stated more precisely in the following fact.

Fact 3.2 *With the notation as in Section 1.3, suppose there exists an algorithm $\mathcal{A}$ that given the $n/2$ MSB bits of d discovers all of d in time $t(n)$. Then there exists an algorithm $\mathcal{B}$ that breaks RSA in time $et(n)$.*

Proof Observe that by equation (4), we have $d = (1 + k(N + 1 - p - q))/e$. Let $\tilde{d}$ be

$$\tilde{d} = \left\lfloor \frac{1 + k(N+1)}{e} \right\rfloor$$

Then

$$0 \leq \tilde{d} - d \leq k(p+q)/e \leq 3k\sqrt{N}/e < 3\sqrt{N}$$

It follows that $\tilde{d}$ matches d on the $n/2$ most significant bits of d. Hence, once k is known, the half most significant bits of d are exposed. With this observation, algorithm $\mathcal{B}$ can work as follows: try all possible values k' for k in the range $\{1, \ldots, e\}$. For each candidate k', compute the corresponding value $\tilde{d}$ using k' as a guess for k. Run algorithm $\mathcal{A}$ giving it half the most significant bits of $\tilde{d}$. Once the $k' = k$, the entire private key is exposed. ■

Remark 2. Fact 3.2 demonstrates that computing the exponentiation associated with the half high-order bits of d can be performed by an untrusted server. This may be of use in server-aided RSA systems where techniques using the Chinese Remainder Theorem cannot be employed. For instance, in threshold RSA [12], the factorization of the modulus is not known to any party, so Chinese Remainder techniques are of no help in accelerating computation. Fact 3.2 suggests that the half high-order bits of d can be revealed to the server with no additional compromise in security, allowing accelerated computation by off-loading that portion of the exponentiation operation.

Fact 3.2 explains why for low exponent RSA one cannot mount a partial key recovery attack given the most significant bits of d. It is natural to ask whether one can expose all of d given a quarter of the low order bits of d that are not necessarily the least significant ones. For instance, the following attack uses $n/4$ bits of d in bit positions $n/4$ to $n/2$, outlined below as Theorem 3.3. Is it possible to generalize this result to *every* subsequence of $n/4$ bits from the $n/2$ low-order bits of d? At the moment this is an open question.

Theorem 3.3 *With the notation as in Section 1.3, suppose* $|p-q| \geq 2^{(n/2)-2}$ *and* $e \leq 2^{(n/4)-3}$. *Then given* $n/4$ *bits of* d *in positions* $n/4$ *to* $n/2$*, we can factor* N *in time* $O(T_C(n) \cdot e^2)$.

Proof Using the approximation $\tilde{d}$ developed in Fact 3.2 and small exhaustive search, we may compute bits $n/2$ to n of d. We now know bits $n/4$ to n of d; we denote this by d_1, so that $|d - d_1| < 2^{n/4}$. The attack tries every candidate value k' for k in the range $\{1, \ldots, e\}$. Define

$$s_1 := 1 + N - \frac{ed_1 - 1}{k'}.$$

Let us consider the case $k' = k$. We have

$$|s_1 - s| = \left| \frac{e}{k}(d - d_1) \right| < 2^{(n/4) + \log_2 e}.$$

Equation (2) suggests we take

$$p_1 := \frac{1}{2}(s_1 + \sqrt{s_1^2 - 4N}),$$

as an approximation to p. It turns out that this is a very good approximation, so that

$$|p_1 - p| \leq 2^{(n/4)+5+\log_2 e}.$$

This is proved as Lemma B.1 in Appendix B. So in at most $32e$ steps, exhaustive search on the high-order bits of p_1 will discover p_1^* such that $|p_1^* - p| \leq 2^{n/4}$. Each candidate for p_1^* is run through the algorithm of Corollary 2.2 to attempt to factor N.

In summary, once $k' = k$, the attack algorithm will discover the factorization of N. The testing of each candidate k' runs in time $32eT_C(n)$; there are at most e candidates to test, so the total running time is $O(T_C(n) \cdot e^2)$. ■

4 Partial key exposure attack on medium exponent RSA

We describe several attacks on the RSA system that can be employed when the public key e is in the range $2^{n/4}$ to $2^{n/2}$. Unlike the previous section, these attacks require the *most* significant bits of d to be given. We mount the attack by carefully studying equation (4):

$$ed - k(N - s + 1) = 1$$

Recall that $s = p + q$.

The key to mounting these attacks is in finding k. Searching for k by brute force is infeasible, since k is an arbitrary element in the range $\{1, \ldots, e\}$. Fortunately, given sufficiently many MSB's of d, we may compute k directly, eliminating it as an unknown from equation (4). Once k is revealed, we are left with two unknowns, d and s which we recover using various methods. The main tool for discovering k is presented in the following theorem. It shows that as long as $e < \sqrt{N}$ we can find k given only $\log_2 e$ MSB bits of d. The theorem produces a small constant size interval containing k. As always, we try all possible values of k in the interval until our attack algorithm succeeds.

Theorem 4.1 *With the notation as in Section 1.3, let t be an integer in the range $\{0, \ldots, n/2\}$. Suppose $2^t < e < 2^{t+1}$ and we know the t most significant bits of d. Then we can efficiently compute the unique k satisfying equation (4) up to a constant additive error.*

The proof of Theorem 4.1 relies on the following lemma, which provides general conditions under which k can be deduced by rounding.

Lemma 4.2 *Suppose d_0 is given such that the following two conditions hold:*

(i) $|e(d - d_0)| < c_1 N$, *and*

(ii) $ed_0 < c_2 N^{3/2}$.

Then the unique k satisfying $ed - k\phi(N) = 1$ is an integer in the range $[\tilde{k} - \Delta, \tilde{k} + \Delta]$ where $\tilde{k} = (ed_0 - 1)/N$ and $\Delta = 2(8c_2 + 2c_1)$.

Proof Let $\tilde{k} = (ed_0 - 1)/N$. Then

$$\left|\tilde{k} - k\right| = \left|(ed_0 - 1)\left(\frac{1}{\phi(N)} - \frac{1}{N}\right) + \frac{e(d - d_0)}{\phi(N)}\right| < c_2 N^{3/2}\left(\frac{N - \phi(N)}{\phi(N)N}\right) + c_1 \frac{N}{\phi(N)}$$

Since $N - \phi(N) < 4\sqrt{N}$ and $\phi(N) > N/2$ it follows that

$$\left|\tilde{k} - k\right| < 8c_2 + 2c_1.$$

Consequently, k is an integer in the range $\{\tilde{k} - \Delta, \ldots, \tilde{k} + \Delta\}$ as required. ∎

We are now prepared to prove Theorem 4.1.

Proof of Theorem 4.1
The t most significant bits of d enable us to construct an integer d_0 satisfying $|d - d_0| < 2^{n-t}$. We use Lemma 4.2 to compute k. By the restriction on e, condition **(i)** is satisfied with $c_1 = 2$. Since $d_0 < N$, condition **(ii)** holds with $c_2 = 2$. Hence k is an integer in a known interval of width 40. ∎

4.1 Prime public key

We are now ready to prove part (1) of Theorem 1.2. Theorem 4.1 enables us to find k. Once k is found we reduce equation (4) modulo e. This removes d from the equation. We can then solve for $s \bmod e$. Given $s \bmod e$ we are able to factor the modulus.

Theorem 4.3 *With the notation of Section 1.3, let t be an integer in the range $n/4 \le t \le n/2$. Suppose e is a prime in the range $\{2^t, \ldots, 2^{t+1}\}$. Furthermore suppose we are given the t most significant bits of d. Then we can factor N in time $O(T_C(n))$.*

Proof The assumptions of the theorem satisfy the conditions of Theorem 4.1. Consequently, k is known to be an integer in a constant size range. We try all candidate values k' for k. For each one we do the following:

1. Compute $s' \equiv N + 1 - k'^{-1} \pmod{e}$. This is well-defined since $\gcd(e, k') = 1$.
2. Compute a root $p' \bmod e$ for x in the quadratic
$$x^2 - s'x + N \equiv 0 \pmod{e} \tag{9}$$
This can be done efficiently (in probabilistic polynomial time) since e is prime. Indeed, once $s' \equiv s \equiv p + q \bmod e$, then $p' \equiv p \bmod e$.
3. Use Corollary 2.2 to find p given $p \bmod e$. This is possible since $e \ge 2^{n/4}$.

Once the correct candidate $k' = k$ is found (after a constant number of attempts) the factorization of N is exposed. Each attempt runs the algorithm of Corollary 2.2 once, leading to a total running time that is $O(T_C(n))$. ∎

A surprising consequence of this theorem is that, when e is prime and is roughly $\cong 2^{n/4}$, only the first $n/4$ MSB's of d are needed to mount the attack. This attack is as strong as the one on low public exponent RSA. In any case, for prime $e \in \{2^{n/4}, \ldots, 2^{n/2}\}$, the first $n/2$ most significant bits of d always suffice.

The proof shows that it is not necessary for e to be prime. As long as we can solve the quadratic in step (2) the proof can be made to work. In order to solve the quadratic we must be given the factorization of e. Unfortunately, modulo a composite, the quadratic may have many roots. We must try them all. If e has r distinct prime factors, there are at most 2^r solutions to consider. As a result, we must also bound the number of prime factors of e. We obtain part (2) of Theorem 1.2.

Corollary 4.4 *As in Theorem 4.3 suppose e is an integer in the range $\{2^t, \ldots, 2^{t+1}\}$. If e has at most r distinct prime factors, and its factorization is known, then given the t most significant bits of d we can factor time $O(T_C(n) \cdot 2^r)$.*

We point out that when e is close to $2^{n/2}$ the same attack can be mounted even if the factorization of e is unknown. In other words, for all e sufficiently close to $2^{n/2}$, half the MSB's of d are sufficient to reconstruct all of d. Indeed, the range $\{1, \ldots, \lceil 2^{n/2+2}/e \rceil\}$ can be searched exhaustively to find $\lceil s/e \rceil$. Given the value of $\lceil s/e \rceil$ we can obtain s (since $s \bmod e$ is already known.) Since s is now known in the integers, we can directly find p using equation (2).

4.2 Public key with unknown factorization

We now turn to proving part (3) of Theorem 1.2. We consider the case when e is in the range $\{2^t, \ldots, 2^{t+1}\}$ with $0 \leq t \leq n/2$. The factorization of e is unknown. The following result establishes that we can still find all of d, given some of its MSB's. Our attack works as long as k is not significantly smaller than e. At the end of the section we note that the attack heuristically works for almost all e in the range $\{2^t, \ldots, 2^{t+1}\}$.

Theorem 4.5 *With the notation as in Section 1.3, let t be an integer in the range $\{0, \ldots, n/2\}$. Suppose e is in the range $\{2^t, \ldots, 2^{t+1}\}$. Further suppose $k > \epsilon \cdot e$ for some $\epsilon > 0$. Then there is an algorithm that given the $n-t$ most significant bits of d finds all of d. The algorithm runs in time $O(n^3/\epsilon)$.*

Proof Given the $n-t$ most significant bits of d we can construct a d_0 such that $0 \leq d - d_0 < 2^t$. Since $e < 2^{n/2}$ we can use d_0 and Theorem 4.1 to limit k to a constant size interval. For each candidate k' for k we do the following:

1. Compute $d_1 \equiv e^{-1} \bmod k'$. This is possible since e and k are relatively prime, so only candidates k' relatively prime to e are worth considering. Since $ed - k\phi(n) = 1$ we know that $d_1 \equiv d \bmod k'$ when $k' = k$.

2. By assumption $k' > \epsilon 2^t$. Note that at this point we know $d \bmod k'$ as well as the $n-t$ MSB's of d. We determine the rest of the bits by an exhaustive search. More precisely, write
$$d = k' d_2 + d_1$$
Then $d_2 = d_0/k' + (d - d_0)/k' - d_1/k'$. The only unknown term in this sum is $v = (d - d_0)/k'$. Since $k' > \epsilon 2^t$ we know that $v = (d - d_0)/k' < 1/\epsilon$. To find v, we try all possible candidates v' in the range $\{0, \ldots, 1/\epsilon\}$. For each pair (k', v') of candidates we compute the corresponding value of d and test it.

3. Once the correct values $v' = v$ and $k' = k$ are found, d is exposed. Testing each pair (k', v') of candidates takes $O(n^3)$ time, and there are $O(1/\epsilon)$ candidate pairs to test. ∎

Theorem 4.5 works without having to know the factorization of e. Unfortunately, the results are not as strong as in the previous section. When e is close to $N^{1/4}$, Theorem 4.5 implies that $3/4$ of the bits of d are needed to reconstruct d. This is much worse than the corresponding bound achieved in the previous section, where only $1/4$ the bits were required. When e is close to $N^{1/2}$ the theorem produces results similar to the previous section.

Theorem 4.5 can only be applied when $k > \epsilon \cdot e$. Intuitively, k behaves roughly as a random integer in the range $\{1, \ldots, e\}$. As such, we should have $k > e/10$ for about 90% of the $e \in \{2^t, \ldots, 2^{t+1}\}$. Hence, heuristically the attack works efficiently for almost all e.

4.3 Further Results

What if the factorization of e is unknown and e was not randomly chosen? Although it may be computationally infeasible, it is possible for e, d to be specifically chosen as factors of $1 + k\phi(N)$ for very small k, violating the conditions of Theorem 4.5. We stress that this is particularly unlikely, as not only would the rather large value of $1 + k\phi(N)$ would need to be factored into ed, but a factor e in the range $\{2^{n/4}, \ldots, 2^{n/2}\}$ would need to be obtained, and one that itself cannot easily be factored (making it vulnerable to Corollary 4.4). However, under these circumstances, the above attacks would not apply. We conclude with the following general result which holds for all $e < 2^{n/2}$. Unfortunately, the result requires non-consecutive bits of d.

Theorem 4.6 *With the notation as in Section 1.3, let t be an integer in $\{1, \ldots, n/2\}$ and e in $\{2^t, \ldots, 2^{t+1}\}$. Suppose we are given the t most significant bits of d and the $n/4$ least significant bits of d. Then we can factor N in time $O(T_C(n))$.*

Proof Sketch Using Theorem 4.1 we may compute a constant size interval I containing k. Observe that the proof of Theorem 3.1 applies for all e, as long as k and the $n/4$ least significant bits of d are known. To recover d, run the algorithm of Theorem 3.1 on all candidate values of k in I. ∎

5 Experiments

To test the effectiveness of our attacks, we implemented them using the C++ programming language with the Number Theory Library package by V. Shoup [22]. The algorithm of Corollary 2.2 dominates the running time of our attacks (all other steps are efficient arithmetic operations), so its running time is the primary object of discussion in this section.

It turns out that the most efficient way to implement Corollary 2.2 is not to use the technique of Coppersmith to find solutions to bivariate polynomials (as in Theorem 2.1), but instead to use the Lattice Factoring Method developed by Boneh, Durfee, and Howgrave-Graham for factoring integers of the form $p^\alpha q$. [2]. We observe that a slight extension of the Lattice Factoring Method, described briefly here, can be used when we are given least significant bits of one of the factors of an RSA modulus $N = pq$. (See [2, Lemma 3.3].) Suppose we are given p_0 such that $p_0 \equiv p$ modulo $2^{(n/4)+\delta}$. We seek a solution x_0 for x in the equation

$$2^{(n/4)+\delta} x + p_0 \equiv 0 \pmod{p}. \qquad (10)$$

If we set $A := (2^{(n/4)+\delta})^{-1} \bmod N$, then solutions to equation (10) are solutions to

$$A(2^{(n/4)+\delta}x + p_0) \equiv x + Ap_0 \equiv 0 \pmod p,$$

where $|x_0| < N^{1/4-\delta}$. This equation is in the form required by the LFM algorithm.

Following the proof of Theorem 3.1, we see that knowing k and the $n/4$ least significant bits of d allows us to compute two candidates p_0, p_1, one of which is equal to $p \bmod 2^{(n/4)-1}$ and the other of which is equal to $q \bmod 2^{(n/4)-1}$. Since the LFM requires $n/4+\delta$ known bits of p (or q), this implies an exhaustive search of size $2^{\delta+1}$ is required to use the Lattice Factoring Method. This search can be easily parallelized in the obvious way. In practice, we found that δ can be very small, say $\delta \leq 6$.

The running time of the Lattice Factoring Method is essentially that of the LLL algorithm on a lattice of dimension Δ with basis coordinates of size no larger than $\Delta \log N$; for this application, $\Delta \leq 30$ usually suffices. Such lattices can be reduced by LLL in a matter of minutes on a typical workstation. We give a few examples below for $e = 3$. Note that in ths case, k is forced to be 2, so the total attack time is equal to $T_C(n)$ times exhaustive search of size less than $2^{\delta+1}$; our measurements reflect the running time for those steps only. For larger e, these times would be multiplied by approximately $4e \log_2 e$.

N	bits of d exposed	lattice dim. Δ	LLL runtime ($\approx T_C(n)$)	δ	observed total runtime
512 bits	128 bits	16	13 seconds	6	11 minutes
768 bits	192 bits	24	6.3 minutes	5	2.8 hours
1024 bits	256 bits	24	26.8 minutes	6	21 hours

These experiments were run on a 500MHz Intel Pentium III processor running Solaris.

Further optimization is possible if additional bits of d beyond the least $n/4$ are exposed: each additional bit of d reduces δ by one, halving the exhaustive search required; once $\delta = 0$, additional bits of d benefit the algorithm by giving closer approximations to p, allowing a decrease in the size of the lattice.

For medium public exponent e, where e is prime, the attack proceeds similarly: with the right guess for k, the value $p \bmod e$ can be determined, which (unless e is very close to $2^{n/4}$) will reveal enough information to run the above algorithm with $\delta = 0$. Thus, the running time of an attack on medium, prime public exponent e will be equal to $T_C(n)$ in the table above times the number of candidates for k that must be checked, which is always less than or equal to 40. With e of known factorization and r factors, there is a multiplicative increase of 2^r in running time to compute and check the distinct solutions to equation (9).

When the factors of e are unknown and the $n - \log_2 e$ bits of d are given, the attack is immediate – the algorithm suggested in Theorem 4.5 involves only a few simple arithmetic operations and a very small search.

6 Conclusions

We study RSA's vulnerability to partial key exposure. We showed that for low exponent RSA, a quarter of the least significant bits of d are sufficient for efficiently reconstucting all of d. We obtain similar results for larger values of e as long as $e < \sqrt{N}$. For instance, when e is close to $\sqrt{N}$, half the most significant bits of d suffice.

The results presented demonstrate the danger of leaking a fraction of the bits of d. We note that discrete log schemes (e.g. DSS, ElGamal) do not seem vulnerable to partial key exposure. A fraction of the bits of the private key in a discrete log based system does not seem to enable the adversary to efficiently break the system.

There are a number of related open problems:

- Does there exist a polynomial time algorithm which enables one to break the RSA system for values of e substantially larger than $\sqrt{N}$, given only a subset of the bits of d?
- Our strongest result with respect to the fewest bits of d required to break the system is for an e in the range $\{N^{1/4}, \ldots, N^{1/2}\}$ with known factorization. For an e with unknown factoriziation and in the same range, does there exist a polynomial time algorithm which provides as strong a result?

Acknowledgements

The authors would like to thank Phong Nguyen, Ron Steinfeld, Yuliang Zheng, and Michael Jason Hinek for several helpful suggestions regarding the preliminary version of this paper.

References

[1] D. Boneh, G. Durfee, and Y. Frankel. An attack on RSA given a small fraction of the private key bits. In proceedings *Asiacrypt '98*, Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, pp. 25–34, 1998.

[2] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^r q$ for large r. In proceedings *Crypto '99*, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, pp. 326–337, 1999.

[3] D. Boneh and G. Durfee. Cryptanalysis of RSA with Private Key d Less Than $N^{0.292}$. *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1339–1349, July 2000. Extended abstract in Proceedings of *Eurocrypt '99*, LNCS vol. 1592, Springer-Verlag, pp. 1–11, 1999.

[4] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, A. Sahai. Exposure-Resilient Functions and All-or-Nothing Transforms. In proceedings *Eurocrypt 2000*, Lecture Notes in Computer Science, vol. 1807, Springer-Verlag, pp. 453–469, 2000.

[5] B. Chor, J. Friedman, O. Goldreich, J. Høastad, S. Rudich, R. Smolensky. The bit extraction problem or t-resilient functions. In proceedings *26th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 396–407, 1985.

[6] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, vol. 10, pp. 233–260, 1997.

[7] C. Coupé, P. Nguyen, and J. Stern. The effectiveness of lattice attacks against low-exponent RSA. In proceedings *Public Key Cryptography '99*, Lecture Notes in Computer Science, vol. 1751, Springer-Verlag, 1999.

[8] J-F. Dhem, F. Koeune, P. Leroux, P. Mestré, J-J. Quisquater, and J-L. Willems. A practical implementation of the timing attack. In proceedings *CARDIS '98* – Third smart card research and advanced application conference, UCL, Louvain-la-Neuve, Belgium, Sep. 14-16, 1998.

[9] Y. Dodis. Exposure-Resilient Cryptography. Ph.D. Thesis, MIT, 2000.

[10] Y. Dodis, A. Sahai, A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In proceedings *Eurocrypt 2001*, Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, pp. 301–324, 2001.

[11] T. ElGamal. A public key cryptosystem and a signature scheme based on the discrete logarithm. *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[12] Y. Frankel. A practical protocol for large group oriented networks. In proceedings *Eurocrypt '89*, Lecture Notes in Computer Science, vol. 434, Springer-Verlag, pp. 56–61, 1989.

[13] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In proceedings *Crypto '96*, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, pp. 104–113, 1996.

[14] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, vol. 261, pp. 515–534, 1982.

[15] H. Lenstra. Divisors in Residue Classes. *Mathematics of Computation*, vol. 42, no. 165, pp. 331-340, 1984.

[16] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In proceedings *Cryptography and Coding*, Lecture Notes in Computer Science, vol. 1355, Springer-Verlag, pp. 131–142, 1997.

[17] U. Maurer. On the Oracle Complexity of Factoring Integers. *Computational Complexity*, vol. 5, pp. 237-247, 1996.

[18] P. Nguyen. Private communications, November 1998.

[19] I. Niven, H. Zuckerman, and H. Montgomery. An Introduction to the Theory of Numbers. Jon Wiley & Sons, Fifth Edition, pp. 87–88, 1991.

[20] D. Redmond. Number Theory: An Introduction. *Monographs and Textbooks in Pure and Applied Mathematics*, no. 201, Marcel Dekker, 1996.

[21] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[22] V. Shoup. Number Theory Library (NTL), `http://www.shoup.net/ntl/`.

[23] R. Steinfeld and Y. Zheng. An advantage of low-exponent RSA with modulus primes sharing least significant bits. In Proceedings *RSA Conference 2001, Cryptographer's Track*, Lecture Notes in Computer Science, vol. 2020, Springer-Verlag, pp. 52–62, 2001.

[24] M. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 553–558, 1990.

A Appendix A: Solutions to Modular Quadratic Equations

This appendix considers the problem of finding solutions for y in equations of the form

$$y^2 \equiv c \pmod{2^u}, \tag{11}$$

where $u \geq 3$ and $c \equiv 1 \bmod 8$.

We begin with an algorithm to find a solution to this equation presented in the following lemma. The proof appears in [20, p. 184]. For convenience we first define the following polynomial (with coefficients in $\mathbb{Q}$):

$$P_\nu(z) := \sum_{i=0}^{\nu} (-1)^{(i-1)} \frac{1}{4^i(2i-1)} \binom{2i}{i} z^i.$$

Lemma A.1 *Suppose $u \geq 3$ and $c \equiv 1$ mod 8. Then $y_0 := P_{u-3}(c-1)$ is an integer which satisfies $(y_0)^2 \equiv c$ mod 2^u.*

This provides one solution to equation (11). The following shows how to convert any solution to equation (11) into any other solution. We provide a proof different from what appears in [20].

Lemma A.2 *Assume y_0, y_1 are in the range $\{0, \ldots, 2^u - 1\}$, and suppose y_0 is a solution for y in equation (11). Then y_1 is a solution to (11) if and only if*

$$y_1 \equiv \sigma y_0 + \tau 2^{u-1} \pmod{2^u}$$

for some $\sigma \in \{-1, 1\}$ and $\tau \in \{0, 1\}$.

Proof Let $\sigma \in \{-1, 1\}$ and $\tau \in \{0, 1\}$ be arbitrary, and let y_0 be any solution to equation (11). If we define $y_1 := \sigma y_0 + \tau 2^{u-1}$ then we see

$$(y_1)^2 \equiv \sigma^2 (y_0)^2 + \sigma\tau 2^u y_0 + \tau^2 2^{2u-2} \pmod{2^u}.$$

Since $u \geq 3$, we have $2u - 2 \geq u$, so reducing modulo 2^u yields

$$(y_1)^2 \equiv \sigma^2 (y_0)^2 + \sigma\tau 2^u y_0 + \tau^2 2^{2u-2}. \equiv \sigma^2 (y_0)^2 \equiv (y_0)^2 \equiv c \pmod{2^u},$$

so y_1 is a solution for y in equation (11).

Now suppose that y_0 and y_1 are arbitrary soultions to equation (11). Since $c \equiv 1$ mod 8, it follows that $y_0^2 \equiv y_1^2 \equiv 1$ mod 4. So $y_1 \equiv \sigma y_0$ mod 4, where $\sigma \in \{-1, 1\}$. Thus there exists some integer v such that $y_1 = \sigma y_0 + 4v$. From equation (11) we see

$$y_1^2 \equiv (\sigma y_0 + 4v)^2 \equiv y_0^2 + 8\sigma y_0 v + 16 v^2 \equiv c \pmod{2^u},$$

implying $8\sigma y_0 v + 16v^2 \equiv 0 \bmod 2^u$. So 2^u divides $8\sigma y_0 v + 16v^2 = 8v(\sigma y_0 + 2v^2)$. Now, $2v^2$ is even and y_0 is odd (recall $y_0^2 \equiv 1 \bmod 8$), so it must be the case that 2^{u-3} divides v. Hence, $y_1 = \sigma y_0 + 4v = \sigma y_0 + 2^{u-1}\tau$. Since we are interested in solutions modulo 2^u, all choices of τ are equivalent to either $\tau = 0$ or $\tau = 1$. ∎

We are now ready to prove the main result of the Appendix.

Lemma A.3 *There are exactly four solutions for y in equation (11), and these solutions can be computed in time $O(u^3)$.*

The proof of the first clause is an immediate consequence of Lemmas A.1 and A.2. The running time analysis follows from the description of Lemma A.1. Note that it is especially efficient to evaluate $P_{u-3}(c-1)$ when only its result modulo 2^u is desired.

B Appendix B: Approximate Square Roots

This appendix considers the problem of bounding the error of the square roots of approximated values. The main tool is the following lemma, needed for Theorem 3.3.

Lemma B.1 *With the notation as in Section 1.3, suppose $|p-q| \geq 2^{(n/2)-2}$ and $e \leq 2^{(n/4)-3}$. Furthermore suppose the quantities d_1, s_1, p_1 satisfy $|d_1 - d| < 2^{n/4}$,*

$$|s_1 - s| \leq 2^{(n/4)+\log_2 e}, \qquad p_1 = \frac{1}{2}(s_1 + \sqrt{s_1^2 - 4N}).$$

Then

$$|p_1 - p| \leq 2^{(n/4)+5+\log_2 e}.$$

The proof of this lemma follows a suggestion made by P. Nguyen [18].

Proof Without loss of generality, assume $s_1 \geq s$, so that $p_1 \geq p$. Observe

$$\begin{aligned} p_1 - p &= \frac{1}{2}(s_1 - s) + \frac{1}{2}(\sqrt{s_1^2 - 4N} - \sqrt{s^2 - 4N}) \\ &= \frac{1}{2}(s_1 - s) + \frac{(s_1 + s)(s_1 - s)}{2(\sqrt{s_1^2 - 4N} + \sqrt{s^2 - 4N})}. \end{aligned}$$

Since $s_1 \geq s$ we have $s_1^2 - 4N \geq s^2 - 4N = (p-q)^2$, and we know $(p-q) \geq 2^{(n/4)-2}$. The bound on e tells us $s_1 \leq 2s$, so that

$$s_1 \leq 2s = 2(p+q) \leq 4p \leq 2^{3+(n/2)}.$$

Using these facts we may derive the following bound:

$$\begin{aligned} p_1 - p &\leq \frac{1}{2}(s_1 - s) + \frac{(s_1 + s)(s_1 - s)}{4(p-q)} \\ &\leq 2^{(n/4)-1+\log_2 e} + \frac{(6p)(2^{(n/4)+\log_2 e})}{2^{n/2}} \\ &\leq 2^{(n/4)-1+\log_2 e} + 2^{(n/4)+4+\log_2 e} \\ &\leq 2^{(n/4)+5+\log_2 e} \end{aligned}$$

as desired. ∎