Factoring RSA Modulus Using Prime Reconstruction from Random Known Bits

Subhamoy Maitra, Santanu Sarkar, and Sourav Sen Gupta

Applied Statistics Unit, Indian Statistical Institute,
203 B T Road, Kolkata 700 108, India
subho@isical.ac.in, santanu_r@isical.ac.in, sg.sourav@gmail.com

Abstract. This paper discusses the factorization of the RSA modulus N (i.e., $N = pq$, where p, q are primes of same bit size) by reconstructing the primes from randomly known bits. The reconstruction method is a modified brute-force search exploiting the known bits to prune wrong branches of the search tree, thereby reducing the total search space towards possible factorization. Here we revisit the work of Heninger and Shacham in Crypto 2009 and provide a combinatorial model for the search where some random bits of the primes are known. This shows how one can factorize N given the knowledge of random bits in the least significant halves of the primes. We also explain a lattice based strategy in this direction. More importantly, we study how N can be factored given the knowledge of some blocks of bits in the most significant halves of the primes. We present improved theoretical result and experimental evidences in this direction.

Keywords: Factorization, Prime reconstruction, Random known bits, RSA.

1 Introduction

The RSA [12] public key cryptosystem uses two primes p, q (usually of the same bit size, i.e., $q < p < 2q$ or $p < q < 2p$). The RSA modulus is $N = pq$. The factorization of N cannot be done efficiently on classical computational model without the knowledge of p, q, and this provides the security of RSA. However, there may be different possibilities to know partial information regarding the secret parameters (through side channel attacks, say) and it is necessary to study how that can affect the security of a cryptosystem. In addition, the basic problem of integer factorization is of great interest in literature.

An extensive amount of research has been done in RSA factorization and we refer the reader to the survey papers by Boneh [1] and May [10] for a complete account. One major class of RSA attacks exploit partial knowledge of the RSA secret keys or the primes. Rivest and Shamir [11] pioneered these attacks using Integer Programming and factored RSA modulus given two-third of the LSBs of a factor. This result was improved in the seminal paper [4] by Coppersmith, where factorization of the RSA modulus could be achieved given half of the MSBs of

D.J. Bernstein and T. Lange (Eds.): AFRICACRYPT 2010, LNCS 6055, pp. 82–99, 2010.

a factor. His method used LLL [9] lattice reduction technique to solve for small solutions to modular equations. This method triggered a host of research in the field of lattice based factorization, e.g., the works by Howgrave-Graham [7], Jochemsz and May [8].

These attacks require knowledge of contiguous blocks of bits of the RSA secret keys or the primes. In a different model, one may not get contiguous blocks, but may gain the knowledge of random bits of the RSA secret keys through cold boot or other side channel attacks. In [6], it has been shown how N can be factored with the knowledge of a random subset of the bits (distributed over small contiguous blocks) in one of the primes. Later, a similar result has been studied by Heninger and Shacham [5] to reconstruct the RSA private keys given a certain fraction of the bits, distributed at random. This is the work [5] where the random bits of both the primes are considered unlike the earlier works (e.g., [4,2,6]) where knowledge of the bits of a single prime have been exploited.

This paper studies how the least (respectively most) significant halves of the RSA primes can be completely recovered from some amount of randomly chosen bits from the least (respectively most) significant halves of the same. Thereafter one can exploit the existing lattice based results towards factoring the RSA modulus $N = pq$ when p, q are of the same bit size. It is possible to factor N in any one of the following cases in poly($\log N$) time: (i) when the most significant half of any one of the primes is known [4, Theorem 4], (ii) when the least significant half of any one of the primes is known [2, Corollary 2.2].

Road Map. In Section 2, we analyze the algorithm of [5] using a combinatorial model for reconstruction. The knowledge of random prime bits and existing lattice based method [2] allows us to factor N efficiently given certain fraction of the bits of p and q, namely about 0.5 fraction of the bits from the least significant halves of the primes when N is 1024 bits. In certain cases, the strategy presented in Section 2 does not work well. To overcome this, we present a lattice based strategy in Section 3. More importantly, we propose in Section 4 an idea to reconstruct the upper half of a prime using the knowledge of certain fraction of bits in p and q. Once one obtains the top half of any one of the primes, the factorization of N is possible using existing lattice based method [4]. Theoretical results as well as extensive experimental evidences are presented to corroborate our claims.

2 The LSB Case: Combinatorial Analysis of [5]

In this section, we analyze the reconstruction algorithm by Heninger and Shacham [5, Section 3] from combinatorial point of view. Though the algorithm extends to all relations among the RSA secret keys, we shall concentrate our attention to the primary relation $N = pq$ for the sake of factorization. The algorithm is a smart brute-force method on the total search space of unknown bits of p and q, which prunes the solutions those are infeasible given the knowledge of N and some random bits of the primes. Henceforth, we shall denote by l_N the bit size of N, i.e, $l_N = \lceil \log_2 N \rceil$.

2.1 The Reconstruction Algorithm

Definition 1. *Let us define $X[i]$ to be the i-th bit of X with $X[0]$ being the LSB. Also define X_i to be the partial approximation of X through the bits 0 to i.*

Then Algorithm 1 (described below) creates all possible pairs (p_i, q_i) by appending $(p[i], q[i])$ to the partial solutions (p_{i-1}, q_{i-1}) and prunes the incorrect ones by checking the validity of the available relation. A formal outline of Algorithm 1, which retrieves the least significant t many bits of both p, q, is as follows. It is easy to see that the correct partial solution till the t many LSBs will exist in the set of all pairs (p_{t-1}, q_{t-1}) found from Algorithm 1.

```
   Input: N, t and p[i], q[j], for some random values of i, j
   Output: Contiguous t many LSBs of p, q
 1 Initialize: i = 1 and p_0 = p[0] = 1, q_0 = q[0] = 1 (as both are odd);
 2 for all (p_{i-1}, q_{i-1}) do
 3 |   for all possible (p[i], q[i]) do
 4 |   |   p_i := APPEND(p[i], p_{i-1});
 5 |   |   q_i := APPEND(q[i], q_{i-1});
 6 |   |   if N ≡ p_i q_i (mod 2^{i+1}) then
 7 |   |   |   ADD the pair (p_i, q_i) at level i;
   |   |   end
   |   end
   end
 8 if i < t − 1 then
 9 |   i := i + 1;
10 |   GOTO Step 2;
   end
11 REPORT all (p_{t-1}, q_{t-1}) pairs;
```

Algorithm 1. The search algorithm

As one may notice, there are at most 4 possible choices for $(p[i], q[i])$ branches at any level i. Algorithm 1 works with all possible combinations of the bits $p[i], q[i]$ at level i and hence one may want to obtain a relation between $p[i]$ and $q[i]$ in terms of the known values of N, p_{i-1}, q_{i-1} so that it poses a constraint on the possibilities. Heninger and Shacham [5, Section 4] uses Multivariate Hensel's Lemma to obtain such a relation

$$p[i] + q[i] \equiv (N - p_{i-1}q_{i-1})[i] \pmod 2. \tag{1}$$

Now, this linear relation between $p[i], q[i]$ restricts the possible choices for the bits. Thus, at any level i, instead of 4 possibilities, the number cuts down to 2.

If we construct the search tree, then these possibilities for the bits at any level give rise to new branches in the tree. The tree at any level i contains all the partial solutions p_i, q_i up to the i-th LSB (the correct partial solution is one

among them). It is quite natural to restrict the number of potential candidates (i.e., the partial solutions) at any level so that the correct one can be found easily by exhaustive search among all the solutions and the space to store all these solutions is within certain feasible limit. This calls for restricting the width of the search tree at each level. Let us denote the width of the tree at level i by W_i. Now we take a look at the situations (depending on the knowledge of the random bits of the primes) that control the branching behavior of the tree.

2.2 Growth of the Search Tree

Consider the situation where we have a pair of partials (p_{i-1}, q_{i-1}) and do not have any information of $(p[i], q[i])$ in Step 3 of Algorithm 1. Naturally there are 4 options, $(0,0), (0,1), (1,0)$ and $(1,1)$ for getting (p_i, q_i). However, Equation (1) and the knowledge of N, p_{i-1}, q_{i-1} impose a linear dependence between $p[i], q[i]$ and hence restrict the number of choices to exactly 2. If $(N - p_{i-1}q_{i-1})[i] = 0$ then we have $p[i] + q[i] \equiv 0 \pmod 2$ and $(N - p_{i-1}q_{i-1})[i] = 1$ implies $p[i] + q[i] \equiv 1 \pmod 2$. Hence the width of the tree at this level will be twice the width of the tree at the previous one, as shown in Figure 1.

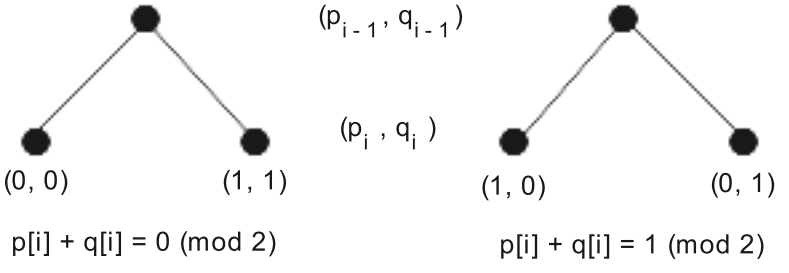

Fig. 1. Branching when both the bits $p[i], q[i]$ are unknown

Next, let us have a look at the situation when exactly one of $p[i], q[i]$ is known. First, the number of branches restricts to 2 by Equation (1), as discussed before. Moreover, the knowledge of one bit fixes the other in this relation. For example, if one knows the value of $p[i]$ along with N, p_{i-1}, q_{i-1} in Equation (1), then $q[i]$ gets determined. Thus the number of choices for $p[i], q[i]$ and hence the number of p_i, q_i branches reduces to a single one in this case. This branching, which keeps the tree-width fixed, may be illustrated as in Figure 2 ($p[i] = 0$ is known, say).

Though the earlier two cases are easy to understand, the situation is not so simple when both $p[i], q[i]$ are known. In this case, the validity of Equation (1) comes under scrutiny. If we fit in all the values $p[i], q[i], N, p_{i-1}, q_{i-1}$ in Equation (1) and it is satisfied, then we accept the new partial solution p_i, q_i at level i and otherwise we do not. In the case where neither of the possibilities for p_i, q_i generated from p_{i-1}, q_{i-1} satisfy the relation, we discard the whole subtree

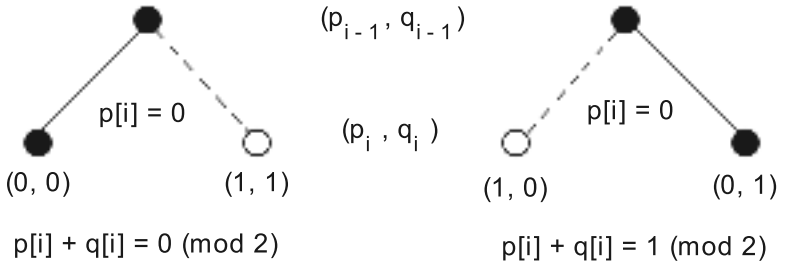

Fig. 2. Branching when exactly one bit of $p[i], q[i]$ is known

rooted at p_{i-1}, q_{i-1}. Thus, the pruning procedure not only discards the wrong ones at level i, but also discards subtrees from level $i-1$, thereby narrowing down the search tree. An example case ($p[i] = 0$ and $q[i] = 1$ are known, say) may be presented as in Figure 3.

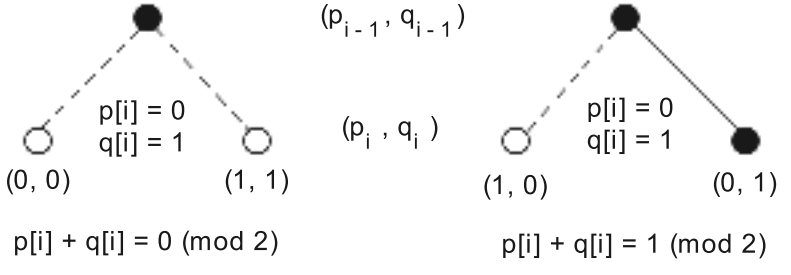

Fig. 3. Branching when both the bits $p[i], q[i]$ are known

Based on our discussion so far, let us try to model the growth of the search tree following Algorithm 1. As both p, q are odd, we have $p[0] = 1$ and $q[0] = 1$. Thus the tree starts from $W_0 = 1$ and the expansion or contraction of the tree at each level can be modeled as follows.

- $p[i]$ = UNKNOWN, $q[i]$ = UNKNOWN: $W_i = 2W_{i-1}$.
- $p[i]$ = KNOWN, $q[i]$ = UNKNOWN: $W_i = W_{i-1}$.
- $p[i]$ = UNKNOWN, $q[i]$ = KNOWN: $W_i = W_{i-1}$.
- $p[i]$ = KNOWN, $q[i]$ = KNOWN: $W_i = \gamma_i W_{i-1}$.

Here, we assume that the tree narrows down to a γ_i fraction ($0 < \gamma_i \leq 1$) from the earlier level if both the bits of the primes are known. One may note that Heninger and Shacham [5, Conjecture 4.3] conjectures the average value of γ_i (call it γ) to be $\frac{1}{2}$. We shall discuss this in more details later.

Suppose that randomly chosen α fraction of bits of p and β fraction of bits of q are known (by some side channel attack, e.g., cold boot). Then the joint probability distribution table for the bits of the primes will be as follows.

$\downarrow q[i], p[i] \rightarrow$	UNKNOWN	KNOWN
UNKNOWN	$(1-\alpha)(1-\beta)$	$\alpha(1-\beta)$
KNOWN	$(1-\alpha)\beta$	$\alpha\beta$

As shown before, the growth of the search tree depends upon the knowledge of the bits in the primes. Hence, we can model the growth of the tree as a recursion on the level index i:

$$\begin{aligned} W_i &= (1-\alpha)(1-\beta)2W_{i-1} + \alpha(1-\beta)W_{i-1} + (1-\alpha)\beta W_{i-1} + \alpha\beta\gamma_i W_{i-1} \\ &= (2-\alpha-\beta+\alpha\beta\gamma_i)W_{i-1}. \end{aligned}$$

If we want to restrict W_i (that is the growth of the tree) as a polynomial of i (that is the number of level), we would like (roughly speaking) the value of $(2-\alpha-\beta+\alpha\beta\gamma_i)$ close to 1 on an average. Considering the average value γ (instead of γ_i at each level), we get, $2-\alpha-\beta+\alpha\beta\gamma \approx 1$ which implies $1-\alpha-\beta+\alpha\beta\gamma \approx 0$. If we assume that the same fraction of bits are known for p and q, then $\alpha = \beta$ and we get $1-2\alpha+\alpha^2\gamma \approx 0 \quad \Rightarrow \quad \alpha \approx \frac{1-\sqrt{1-\gamma}}{\gamma}$. If we assume [5, Conjecture 4.3], then $\gamma \approx 0.5$ and hence $\alpha \approx 2-\sqrt{2} \approx 0.5858$, as obtained in [5, Section 4.4]. One may note that our idea is simpler compared to the explanation in [5]. This simplification is achieved here by using average value for γ_i in the recurrence relation of W_i.

The most natural strategy is to first apply Algorithm 1 to retrieve the least significant half of any one of the primes and then apply the result of Boneh et. al. [2, Corollary 2.2] to factorize N. One may note that [5] utilizes their prime reconstruction algorithm to reconstruct the whole primes p, q whereas our idea is to use lattice based results after reconstructing just one half of any prime. This is more practical as it requires the knowledge of lesser number of random bits of the primes, namely, just about $0.5858 \times 0.5 \approx 0.3$ fraction of bits (from the LSB half) instead of 0.5858 fraction of the primes as explained in [5]. Moreover, factorization being the main objective, one need not reconstruct the primes completely, but just requires to obtain enough information that suffices for factoring the product N based on the existing efficient techniques. In this direction, let us first present the following result.

2.3 Known Prime Bits: Complementary Sets for p, q

Theorem 1. *Let $N = pq$, when p, q are primes of same bit size. Let $S = \{0, \ldots, \lceil l_N/4 \rceil\}$. Consider $U \subseteq S$ and $V = S \setminus U$. Assume that $p[i]$'s for $i \in U$ and $q[j]$'s for $j \in V$ are known. Then one can factor N in poly$(\log N)$ time.*

Proof. Let us apply Algorithm 1 in this case to retrieve the bits of the primes at each level. We shall use induction on the index of levels in this case.

For level 0, we know that $p[0] = 1$ and $q[0] = 1$. Hence, the width of the search tree is $W_0 = 1$ and we have a single correct partial (p_0, q_0). Let us suppose that we have possible pairs of partials (p_{i-1}, q_{i-1}) at level $i-1$, generated by Algorithm 1. At level i, two cases may arise. If $i \in U$ then we know $p[i], N, p_{i-1}, q_{i-1}$ which restricts the branching to a single branch and keeps the width of the tree fixed ($W_i = W_{i-1}$). Else one must have $i \in V$ ($V = S \setminus U$) and we know $q[i], N, p_{i-1}, q_{i-1}$. This restricts the branching to a single branch as well and keeps the width fixed ($W_i = W_{i-1}$). Hence, by induction on i, $W_i = W_{i-1}$ for $i = 0, \ldots, \lceil l_N/4 \rceil$. As $W_0 = 1$, this boils down to $W_i = 1$ for $i \leq \lceil l_N/4 \rceil$.

Thus we obtain a single correct partial pair p_i, q_i at level $i = \lceil l_N/4 \rceil$ using Algorithm 1 in $O(\log^3 N)$ time ($\lceil l_N/4 \rceil$ iterations and $O(\log^2 N)$ computing time for each iteration) and $O(l_N/2)$ space (actually we need space to store a single partial pair at the current level). This provides us with the least significant half of both the primes and using any one of those two, the lattice based method of [2, Corollary 2.2] completes the factorization of $N = pq$ in poly($\log N$) time. □

It is interesting to analyze the implications of this result in a few specific cases. An extreme case may be $U = S$, that is we know all the bits in the least significant half of a single prime p and do not know any such bits for q. In this scenario, one need not apply Algorithm 1 at all and the lattice based method in [2, Corollary 2.2] suffices for factorization. Second case is when $|U| = |S| - x$, i.e., missing x bits of p at random positions. In such a case, one can use a brute force search for these missing bits and apply lattice based factoring method [2, Corollary 2.2] for all of the 2^x possibilities, if x is small. However, for large x, e.g., $x \approx |U|$, i.e., around half of the random bits from the least significant halves of p as well as q are known, then the brute force strategy fails, and one must use Algorithm 1 before applying the lattice based method in [2, Corollary 2.2].

2.4 Known Prime Bits: Distributed at Random

Here we consider the case when random bits of p, q are available, lifting the constraint $V = S \setminus U$. That is, here we only consider U, V to be random subsets of S. For 512-bit primes, we observed that knowledge of randomly chosen half of the bits from least significant halves of p, q is sufficient to recover the complete least significant halves of p as well as q using Algorithm 1.

Now let us present a select few of our experimental results in Table 1. The first column represents the size of the RSA primes and the second column gives the fraction of bits known randomly from the least significant halves of the primes (call these α_p, β_q respectively). The value of t in the third column is the target level we need to run Algorithm 1 for, and is half the size of the primes. W_t is the final width of the search tree at the target level t. This denotes the number of possible partial solutions for p, q at the target bit level t, whereas the next column gives us the maximum width of the tree observed during the run of Algorithm 1. The last column depicts the average value of the shrink ratio γ, as we have defined earlier.

Table 1. Experimental results corresponding to Algorithm 1

Size $\|p\|, \|q\|$	Known α_p, β_q	Target t	Final W_t	$\max_{i=1}^{t} W_i$	Average γ
256, 256	0.5, 0.5	128	30	60	0.56
256, 256	0.5, 0.5	128	2816	5632	0.52
256, 256	0.47, 0.47	128	106	1508	0.54
256, 256	0.45, 0.45	128	6144	6144	0.49
512, 512	0.5, 0.5	256	352	928	0.53
512, 512	0.5, 0.5	256	8	256	0.55
512, 512	0.5, 0.5	256	716	3776	0.53
512, 512	0.5, 0.5	256	152	2240	0.59
512, 512	0.55, 0.45	256	37	268	0.51
512, 512	0.55, 0.45	256	64	334	0.51
512, 512	0.6, 0.4	256	1648	13528	0.55
512, 512	0.6, 0.4	256	704	5632	0.56
512, 512	0.7, 0.3	256	158	1344	0.53
512, 512	0.7, 0.3	256	47	4848	0.52
1024,1024	0.55, 0.55	512	1	352	0.53
1024,1024	0.53, 0.53	512	16	764	0.53
1024,1024	0.51, 0.51	512	138	15551	0.54
1024,1024	0.51, 0.5	512	17	4088	0.52

A few crucial observations can be made from the data presented in Table 1. We have run the experiments for different sizes of RSA keys, and though the theoretical requirement for the fraction of known bits (α, β) is 0.5858, we have obtained better results when $l_N \leq 2048$. For 512 bit N, the knowledge of just 0.45 fraction of random bits from the least significant halves of the primes proves to be sufficient for Algorithm 1 to retrieve the halves, whereas for 1024 and 2048 bit N, we require about 0.5 fraction of such bits. The main reason is that the growth of the search tree increases with increasing size of the target level t. As we have discussed before, the growth will be independent of the target if we know 0.5858 fraction of bits instead. One may also note that for 1024 bit N, we have obtained successful results when the fraction of bits known is not the same for the two primes. For such skewed cases, the average requirement of known bits stay the same, i.e, 0.5 fraction of the least significant halves. The examples for (0.7, 0.3) in such skewed cases provides interesting results compared to the result by Herrmann and May [6]. Knowing about 70% of the bits of one prime is sufficient for their method to factorize N, but the runtime is exponential in the number of blocks over which the bits are distributed. By knowing 35% of one prime (70% from the least significant half) and 15% of the other (30% of the least significant half), Algorithm 1 can produce significantly better results in the same direction.

Another important point to note from the results is the average value of the shrink ratio γ. It is conjectured in [5] that $\gamma = 0.5$. However, our experiments clearly show that the value of γ is more than 0.5 in *most* (17 out of 18) of the cases. A theoretical explanation of this anomaly may be of interest.

2.5 Known Prime Bits: Distributed in a Pattern

In addition to these results, some interesting cases appear when we consider the knowledge of the bits to be periodic in a systematic pattern, instead of being totally random. Suppose that the bits of the primes p, q are available in the following pattern: none of the bits is known over a stretch of U bits, only $q[i]$ is known for Q bits, only $p[i]$ is known for P bits and both $p[i], q[i]$ are known for K bits. This pattern of length $U + P + Q + K$ repeats over the total number of bits. In such a case, one may expect the growth of the tree to obey the following heuristic model – grows in doubles for U bits, stays the same for $Q+P$ length and shrinks thereafter (approximately by halves, considering $\gamma = 0.5$) for a stretch of K bits. If this model is followed strictly, one expects the growth of the tree by a factor of $2^U 2^{-K} = 2^{U-K}$ over each period of the pattern. The total number of occurrences of this pattern over the stretch of T bits is roughly $\frac{T}{U+Q+P+K}$. Hence the width of the tree at level T may be roughly estimated by $W_T \approx \left[2^{U-K}\right]^{\frac{T}{U+Q+P+K}} = 2^{\frac{T(U-K)}{U+Q+P+K}}$. A closer look reveals a slightly different observation. We have expected that the tree shrinks in half if both bits are known, which is based on the conjecture that $\gamma \approx 1/2$ on an average. But in practical scenario, this is not the case. So, the width W_T at level T, as estimated above, comes as an underestimate in most of the cases.

Let us consider a specific example for such a band-LSB case. The pattern followed is $[U = 5, Q = 3, P = 3, K = 5]$. Using the estimation formula above, one expects the final width of the tree at level 256 to be 1, as $U = K$. But in this case, the final width turns out to be 8 instead. The reason behind this is that the average value of γ in this experiment is 0.55 instead of 0.5.

It is natural for one to notice that the fraction of bits to be known in this band-LSB case is $(P+K)/(U+Q+P+K)$ for the prime p and $(Q+K)/(U+Q+P+K)$ for the prime q. If we choose $Q = P$ and $U = K$, then this fraction is 0.5. Thus, by knowing 50% of the bits from the least significant halves of the primes, that is, knowing just 0.25 fraction of bits in total, Algorithm 1 can factorize $N = pq$ in this case. One may note that the result by Herrmann and May [6] requires the knowledge of about 70% of the bits distributed over arbitrary number of small blocks of a single prime. Thus, in terms of total number of bits to be known (considering both the primes), our result is clearly better.

An extension of this idea may be applied in case of MSBs. Though we can retrieve information about the primes from random bits at the least significant side, we could not exploit similar information from the most significant part. But we could do better if bands of bits are known instead of isolated random bits. A novel idea for reconstructing primes based on such knowledge is presented in Section 4.

3 The LSB Case: Lattice Based Technique

Consider the scenario when a long run (length u) of $p[i], q[i]$ is not known, for $k < i \leq k + u$ say, starting at the $(k+1)$-th bit level. In such a case, Algorithm 1

will require large memory as the width of the tree will be at least 2^u at the u-th level. If u is large, say $u \geq 50$, then it will be hard to accommodate the number of options, which is greater than 2^{50}. We describe a lattice based method below to handle such situation.

For basics related to lattices and solution to modular equations using lattice techniques, one may refer to [4,7,8]. First we recall the following result from [7].

Lemma 1. *Let $g(x, y) \in \mathbb{Z}[x, y]$ be a polynomial which is the sum of ω many monomials. Suppose $g(x_1, y_1) \equiv 0 \bmod n$, where $|x_1| < X_1$ and $|y_1| < Y_1$. If $\| g(xX_1, yY_1) \|_2 < \frac{n}{\sqrt{\omega}}$, then $g(x_1, y_1) = 0$ holds over integers.*

We apply resultant techniques to solve for the roots of the bivariate polynomials. It may sometimes happen that the resultant between the two bivariate polynomials is zero. There is no way to avoid this and it is a common problem in bivariate Coppersmith method. Thus one cannot always find common roots using this method. Though our technique works in practice as noted from the experiments we perform, we formally state the following assumption, which proves to be crucial in Theorem 2.

Assumption 1. *Let $\{f_1, f_2\}$ be two polynomials in two variables sharing common roots of the form (x_1, y_1). Then it is possible to find the roots efficiently by calculating the resultant of $\{f_1, f_2\}$.*

Now we will state and prove the main result of this section.

Theorem 2. *Let $N = pq$ where p, q are of same bit size. Suppose τl_N many least significant bits (LSBs) of p, q are unknown but the subsequent ηl_N many LSBs of both p, q are known. Then, under Assumption 1, one can recover the τl_N many unknown LSBs of p, q in poly$(\log N)$ time, if $\tau < \frac{\eta}{2}$.*

Proof. Let p_0 correspond to the known ηl_N many bits of p and q_0 correspond to the known ηl_N bits of q. Let p_1 correspond to the unknown τl_N many bits of p and q_1 correspond to the unknown τl_N bits of q. Then we have $(2^{\tau l_N} p_0 + p_1)(2^{\tau l_N} q_0 + q_1) \equiv N \bmod (2^{(\tau+\eta) l_N})$. Let $T = 2^{(\tau+\eta) l_N}$. Hence we are interested to find the roots (p_1, q_1) of $f(x, y) = (2^{\tau l_N} p_0 + x)(2^{\tau l_N} q_0 + y) - N$ over $\mathbb{Z}_T$.

Let us take $X = 2^{\tau l_N}$ and $Y = 2^{\tau l_N}$. One may note that X, Y are the upper bounds of the roots (p_1, q_1) of $f(x, y)$, neglecting small constants. For a non negative integer m, we define two sets of polynomials

$g_{i,j}(x, y) = x^i f^j(x, y) T^{m-j}$, where $j = 0, \ldots, m,\ i = 0, \ldots, m - j$ and

$h_{i,j}(x, y) = y^i f^j(x, y) T^{m-j}$, where $j = 0, \ldots, m,\ i = 1, \ldots, m - j$.

Note that $g_{i,j}(p_1, q_1) \equiv 0 \bmod (T^m)$ and $h_{i,j}(p_1, q_1) \equiv 0 \bmod (T^m)$. We call $g_{i,j}$ the x-shift and $h_{i,j}$ the y-shift polynomials, as per their respective constructions following the idea of [8].

Next, we form a lattice L by taking the coefficient vectors of the shift polynomials $g_{i,j}(xX, yY)$ and $h_{i,j}(xX, yY)$ as basis. One can verify that the dimension of the lattice L is $\omega = (m + 1)^2$. The matrix containing the basis vectors of L is lower triangular and has diagonal entries of the form $X^{i+j} Y^j T^{m-j}$, for

$j = 0, \ldots, m$ and $i = 0, \ldots, m-j$, and $X^j Y^{i+j} T^{m-j}$ for $j = 0, \ldots, m$ and $i = 1, \ldots, m-j$, coming from $g_{i,j}$ and $h_{i,j}$ respectively. Thus, one can calculate

$$\det(L) = \left[\prod_{j=0}^{m}\prod_{i=0}^{m-j} X^{i+j}Y^{j}T^{m-j}\right]\left[\prod_{j=0}^{m}\prod_{i=1}^{m-j} X^{j}Y^{i+j}T^{m-j}\right] = X^{s_1}Y^{s_2}T^{s_3}$$

where $s_1 = \frac{1}{2}m^3 + m^2 + \frac{1}{2}m$, $s_2 = \frac{1}{2}m^3 + m^2 + \frac{1}{2}m$, and $s_3 = \frac{2}{3}m^3 + \frac{3}{2}m^2 + \frac{5}{6}m$.

To utilize resultant techniques and Assumption 1, we need two polynomials $f_1(x, y)$, $f_2(x, y)$ which share the roots (p_1, q_1) over integers. From Lemma 1, we know that one can find such $f_1(x, y), f_2(x, y)$ using LLL lattice reduction algorithm [9] over L when $det(L) < T^{m\omega}$, neglecting the small constants. Given $det(L)$ and ω as above, the condition becomes $X^{s_1}Y^{s_2}T^{s_3} < T^{m((m+1)^2)}$, i.e., $X^{s_1}Y^{s_2} < T^{s_0}$, where $s_0 = m((m+1)^2) - s_3 = \frac{1}{3}m^3 + \frac{1}{2}m^2 + \frac{1}{6}m$. Putting the values of the bounds $X = Y = 2^{\tau l_N}$, and neglecting $o(m^3)$ terms, we get $\frac{\tau}{2} + \frac{\tau}{2} < \frac{\tau+\eta}{3}$ and thus get the required bound for τ. Now, one can find the root (p_1, q_1) from f_1, f_2 under Assumption 1. The claimed time complexity of poly($\log N$) can be achieved as

- the time complexity of the LLL lattice reduction is poly($\log N$); and
- given a fixed lattice dimension of small size, we get constant degree polynomials and the complexity of resultant calculation is polynomial in the sum-of-degrees of the polynomials.

This completes the proof of Theorem 2. □

This lattice based technique complements Algorithm 1 by overcoming one of its limitations. As we discussed before, the search tree grows two-folds each time we do not have any information about the bits of the primes. Hence in a case where an initial chunk of LSBs is unknown for both the primes, one can not use Algorithm 1 for reconstruction as it would require huge storage space for the search tree. This lattice based technique provides a feasible solution in such a case. We present a few experimental results in Table 2 to illustrate the operation of this technique. All the experiments have been performed in SAGE 4.1 over Linux Ubuntu 8.04 on a Compaq machine with Dual CORE Intel(R) Pentium(R) D CPU 1.83 GHz, 2 GB RAM and 2 MB Cache.

Table 2. Experimental runs of the Lattice Based Technique with lattice dimension 64

# of Unknown bits (τl_N)	# of Known bits (ηl_N)	Time in Seconds		
		LLL Algorithm	Resultant Calculation	Root Extraction
40	90	36.66	25.67	< 1
50	110	47.31	35.20	< 1
60	135	69.23	47.14	< 1
70	155	73.15	58.04	< 1

The limitation of this technique is that it asks for double or more the number of missing bits for both the primes. If one misses 60 LSBs for the primes say, this method requires the next 120 or more bits of both the primes to be known to reconstruct all $60 + 120 = 180$ LSBs. In the practical scenario, the requirement of bits to be known is 135 (shown in Table 2), instead of 120, as we use limited lattice dimensions in the experiments. In all the cases mentioned above, we miss the first τl_N LSBs of the primes. If we miss the information of the bits of the prime in a contiguous block of size τl_N somewhere in the middle, after the i-th level say, then this method offers similar solution if we have $\eta > 2\tau + 2i/l_N$.

4 The MSB Case: Our Method and Its Analysis

In this section, we put forward a novel idea of reconstructing the most significant half of the primes p, q given the knowledge of some blocks of bits. To the best of our knowledge, this has not been studied in a disciplined manner in the existing literature. As before, $N = pq$, and the primes p, q are of the same bit size. For this section of MSB reconstruction, let us propose the following definition to make notations simpler.

Definition 2. *Let us define $X[i]$ to be the i-th most significant bit of X with $X[0]$ being the MSB. Also define X_i to be the partial approximation of X where X_i shares the most significant bits 0 through i with X. Let l_X denote the bit size of X, i.e, $l_X = \lceil \log_2 X \rceil$.*

4.1 The Reconstruction Idea

The idea for reconstructing the most significant halves of the primes is quite simple. We shall use the basic relation $N = pq$. If one of the primes, p say, is known, the other one is easy to find by $q = N/p$. Now, if a few MSBs of one of the primes, p say, is known, then we may obtain an approximation p' of p. This allows us to construct an approximation $q' = \lceil N/p' \rceil$ of the other prime q as well. Our idea is to use the known blocks of bits of the primes in a systematic order to repeat this approximation process until we recover half of one of the primes. A few obvious questions may be as follows.

- How accurate are the approximations?
- How probable is the success of the reconstruction process?
- How many bits of the primes do we need to know?

We answer these questions by describing the reconstruction algorithm in Section 4.2 and analyzing the same in Section 4.3. But first, let us present an outline of our idea.

Suppose that we have the knowledge of MSBs $\{0, \ldots, a\}$ of prime p. This allows us to construct an approximation p_a of p, and hence an approximation $q' = \lceil N/p_a \rceil$ of q. Lemma 2, discussed later in Section 4.3, tells us that q' matches q through MSBs $\{0, \ldots, a-t-1\}$, i.e, $q' = q_{a-t-1}$, with some probability

depending on t. Now, if one knows the MSBs $\{a-t,\ldots,2a\}$ of q, then a better approximation q_{2a} may be constructed using q_{a-t-1} and these known bits. Again, q_{2a} facilitates the construction of a new approximation $p' = \lceil N/q_{2a} \rceil$, which by Lemma 2, satisfies $p' = p_{2a-t-1}$ with some probability depending on t. With the knowledge of MSBs $\{2a-t,\ldots,3a\}$ of p, it is once again possible to construct a better approximation p_{3a} of p. This process of constructing approximations is recursed until one reconstructs the most significant half of one of the primes. A graphical representation of the reconstruction process is illustrated in Figure 4.

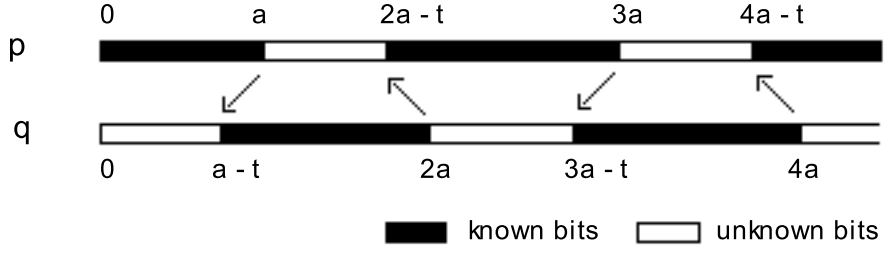

Fig. 4. The feedback mechanism in MSB reconstruction

4.2 The Reconstruction Algorithm

Let $S = \{0,\ldots,T\}$ denote the set of bit indices from the most significant halves of the primes. Let us assume that $k = \lfloor T/a \rfloor$ is odd in this case. Consider $U, V \subseteq S$ such that $U = \{0,\ldots,a\} \cup \{2a-t,\ldots,3a\} \cup \cdots \cup \{(k-1)a-t,\ldots,ka\}$, $V = \{a-t,\ldots,2a\} \cup \{3a-t,\ldots,4a\} \cup \cdots \cup \{ka-t,\ldots,T\}$. Also consider that $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$. Then, Algorithm 2 reconstructs T many contiguous most significant bits of the prime q.

The subroutine CORRECT used in Algorithm 2 (and Algorithm 4 later) takes as input a partial approximation Y of X and a set of contiguous known bits, $X[i]$ for $i \in \Sigma$, say. It outputs a better approximation Z of X by correcting the bits of the partial approximation Y using the knowledge of the known bits. Formally, the subroutine works as described is Algorithm 3.

In the case where $k = \lfloor T/a \rfloor$ is even, Algorithm 2 needs to be tweaked a little to work as expected. One may consider a slightly changed version of $U, V \subseteq S$ such that $U = \{0,\ldots,a\} \cup \{2a-t,\ldots,3a\} \cup \cdots \cup \{ka-t,\ldots,T\}$ and $V = \{a-t,\ldots,2a\} \cup \{3a-t,\ldots,4a\} \cup \cdots \cup \{(k-1)a-t,\ldots,ka\}$. As before, $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$. Then, Algorithm 4 reconstructs T many contiguous most significant bits of the prime p.

4.3 Analysis of the Reconstruction Algorithm

Algorithm 2 and Algorithm 4 follow the same basic idea of reconstruction as discussed in Section 4.1, and differs only in a minor issue regarding the practical implementation. We have stated both the algorithms in Section 4.2 for the sake

Input: N, T and $p[i], q[j]$ for all $i \in U$ and $j \in V$
Output: Contiguous T many MSBs of q

Initialize: $p_0 := 2^{l_p - 1}$, $q_0 := 2^{l_q - 1}$;
$p_a :=$ CORRECT(p_0, $p[j]$ for $j \in \{1, \ldots, a\} \subset U$);
$q_{a-t} := \lceil \frac{N}{p_a} \rceil$;
for *i from* 2 *to* $k-1$ *in steps of* 2 **do**
  $q_{ia} :=$ CORRECT($q_{(i-1)a-t}$, $q[j]$ for $j \in \{(i-1)a - t, \ldots, ia\} \subset V$);
  $p_{ia-t-1} := \lceil \frac{N}{q_{ia}} \rceil$;
  $p_{(i+1)a} :=$ CORRECT(p_{ia-t-1}, $p[j]$ for $j \in \{ia - t, \ldots, (i+1)a\} \subset U$);
  $q_{(i+1)a-t-1} := \lceil \frac{N}{p_{(i+1)a}} \rceil$;
end
$q_T :=$ CORRECT(q_{ka-t-1}, $q[j]$ for $j \in \{ka - t, \ldots, T\} \subset V$);
REPORT q_T;

Algorithm 2. The MSB reconstruction algorithm [k odd]

Input: Y and $X[i]$ for $i \in \Sigma$
Output: Z, the correction of Y

for *j from* 0 *to* l_X **do**
  if $j \in \Sigma$ **then** $Z[j] = X[j]$; `// Correct the j-th MSB if the bit X[j] is known`
  else $Z[j] = Y[j]$; `// Keep the j-th MSB of Y as X[j] is not known`
end
REPORT Z;

Algorithm 3. Subroutine CORRECT

of completeness. But in case of the analysis and the experimental results, we shall consider only one of them, Algorithm 2 say, without loss of generality.

Algorithm 2 requires the knowledge of at most $(T - ka + 1) + k(a + t + 1) \leq k(a+t) + (k+a) \leq T(1 + \frac{t}{a}) + (k+a)$ many bits of p and q to (probabilistically) reconstruct T contiguous MSBs of one prime. The runtime of Algorithm 2 is linear in terms of the number of known blocks, i.e, linear in terms of $k = \lfloor T/a \rfloor$. If we set the target $T = l_N/4$, then Algorithm 2 outputs the most significant half of one of the primes in $O(k)$ steps with some probability of success depending on a and t. In this context, we propose Theorem 3 to estimate the probability of success of Algorithm 2. Before that, let us introduce the following technical result (Lemma 2) which is necessary to prove Theorem 3.

Lemma 2. *If X and X' are two integers with same bit size and $|X - X'| < 2^H$, then the probability that X and X' share $l_X - H - t$ many MSBs for some $0 \leq t \leq H$ is at least $P_t = 1 - \frac{1}{2^t}$.*

Proof. We know that $|X - X'| < 2^H$, i.e, $X = X' + Y$ or $X' = X + Z$ where $0 \leq Y, Z < 2^H$, say. Let us consider the case $X = X' + Y$ first, and the other case will follow by symmetry between X and X'.

Input: N, T and $p[i], q[j]$ for all $i \in U$ and $j \in V$
Output: Contiguous T many MSBs of p

Initialize: $p_0 := 2^{l_p - 1}$, $q_0 := 2^{l_q - 1}$;
$p_a :=$ CORRECT(p_0, $p[j]$ for $j \in \{1, \ldots, a\} \subset U$);
for *i from* 1 *to* $k-3$ *in steps of* 2 **do**
  $q_{ia-t-1} := \lceil \frac{N}{p_{ia}} \rceil$;
  $q_{(i+1)a} :=$ CORRECT(q_{ia-t-1}, $q[j]$ for $j \in \{ia - t, \ldots, (i+1)a\} \subset V$);
  $p_{(i+1)a-t-1} := \lceil \frac{N}{q_{(i+1)a}} \rceil$;
  $p_{(i+2)a} :=$ CORRECT($p_{(i+1)a-t-1}$, $p[j]$ for $j \in \{(i+1)a - t, \ldots, (i+2)a\} \subset U$);
end
$q_{(k-1)a-t-1} := \lceil \frac{N}{p_{(k-1)a}} \rceil$;
$q_{ka} :=$ CORRECT($q_{(k-1)a-t-1}$, $q[j]$ for $j \in \{(k-1)a - t, \ldots, ka\} \subset V$);
$p_{ka-t-1} := \lceil \frac{N}{q_{ka}} \rceil$;
$p_T :=$ CORRECT(p_{ka-t-1}, $p[j]$ for $j \in \{ka - t, \ldots, T\} \subset U$);
REPORT p_T;

Algorithm 4. The MSB reconstruction algorithm [k even]

Let us split $X' = 2^H X_0 + X_1$. Then, clearly $X = 2^H X_0 + (X_1 + Y)$. The addition of $Y < 2^H$ affects the lower part X_1 directly and the carry from the sum $(X_1 + Y)$ affects the first half X_0 up to a certain level. Our goal is to find out the probability that the carry affects less than or equal to t bits of X_0 from the lower side. Let us assume that the probability of $(X_1 + Y)$ generating a carry bit is p_c. We also know that this carry bit will propagate through the lower bits of X_0 until it hits a 0, and we can assume any bit of X_0 to be 0 or 1 randomly with equal probabilities of 1/2 each. Then, the probability of the carry bit to propagate less than or equal to t bits of X_0 from the lower side is
$P[\text{carry propagation} \leq t]$
$= P[\text{no carry}] + \sum_{i=1}^{t} P[\text{carry}] P[\text{carry propagation} = i]$
$= P[\text{no carry}] + \sum_{i=1}^{t} P[\text{carry}] P[\text{first 0 occurs at } i\text{-th LSB of } X_0]$
$= (1 - p_c) + \sum_{i=1}^{t} p_c \frac{1}{2^i} = 1 - \frac{p_c}{2^t}$.

Now, one may expect the probability of carry generated from the sum $(X_1 + Y)$ to be $p_c \approx 1/2$. A careful statistical modelling of the difference Y will reveal a better estimate of p_c. As we do not assume any distribution of Y here, we consider the trivial bound $p_c \leq 1$. Thus the probability of X and X_0, and hence X and X', sharing $l_X - H - t$ many MSBs is $1 - \frac{p_c}{2^t} \geq 1 - \frac{1}{2^t}$. □

At this point, we can state and prove the main result of this section, the following theorem.

Theorem 3. *Let* $S = \{0, \ldots, T\}$ *and* $k = \lfloor T/a \rfloor$ *is odd. Suppose* $U, V \subseteq S$ *such that* $U = \{0, \ldots, a\} \cup \{2a - t, \ldots, 3a\} \cup \cdots \cup \{(k-1)a - t, \ldots, ka\}$, $V = \{a - t, \ldots, 2a\} \cup \{3a - t, \ldots, 4a\} \cup \cdots \cup \{ka - t, \ldots, T\}$, *where* $p[i]$*'s are known for* $i \in U$ *and* $q[j]$*'s are known for* $j \in V$*, as discussed before. Then, Algorithm 2*

reconstructs T many contiguous most significant bits of one of the primes correctly in $O(k)$ steps with probability at least $P_{a,t}(T) = \left(1 - \frac{1}{2^t}\right)^{\lfloor T/a \rfloor}$.

Proof. The success of Algorithm 2 relies on the correct construction of the approximations at various levels. The CORRECT function produces correct approximations with probability 1 given the known sets of bits U, V, as mentioned before. Hence, success probability of Algorithm 2 depends on the correctness of $\{q_{a-t-1}, p_{2a-t-1}, q_{3a-t-1}, \ldots, p_{(k-1)a-t-1}, q_{ka-t-1}\}$.

Let us first consider the case $p > q$. We know that in such a case, as p, q are of the same bit size, one must have $\sqrt{N/2} < q < \sqrt{N} < p < \sqrt{2N}$. Suppose that there exists an approximation p_{ha} of p, sharing the MSBs $\{0, \ldots, ha\}$ for some $1 \leq h \leq k$. In this case, $|p - p_{ha}| < 2^{l_p - ha}$. Using p_{ha}, one constructs an approximation $q' = \lceil N/p_{ha} \rceil$ of q. Then we have $|q - q'| \approx \left|\frac{N}{p} - \frac{N}{p_{ha}}\right| = \frac{N}{pp_{ha}}|p - p_{ha}| < |p - p_{ha}| < 2^{l_p - ha}$, as $p, p_{ha} > \sqrt{N}$. If $p_{ha} < \sqrt{N}$ from the initial approximation, we reassign $p_{ha} = \lceil \sqrt{N} \rceil$ as a better approximation to p. The case $p < q$ produces an approximation q' of q with $|q - q'| < 2|p - p_{ha}| < 2^{l_p - ha + 1}$.

Then, we know for sure that $|q - q'| < 2^{l_p - ha + 1}$. Thus, by Lemma 2, setting $H = l_p - ha + 1$, we get that q and q' share the first $l_p - (l_p - ha + 1) - t = ha - t - 1$ MSBs with probability at least $P_t = 1 - \frac{1}{2^t}$. In other words, the probability that q' correctly represents q_{ha-t-1} is at least $P_t = 1 - \frac{1}{2^t}$. The probability of correctness is the same in case of the approximations of p by p_{ga-t-1} for all $1 < g < k$.

Now, the k approximations of p, q at different bit levels, as mentioned above, can be considered independent. Hence, the probability of success of Algorithm 2 in constructing T many contiguous MSBs of q (or p in another case) is at least $P_{a,t} = P_t^k = \left(1 - \frac{1}{2^t}\right)^k = \left(1 - \frac{1}{2^t}\right)^{\lfloor T/a \rfloor}$. □

Once the most significant half of any one of the primes is known using Algorithm 2, one may use a lattice based method of to factorize $N = pq$. In this context, let us present the following result for factoring the RSA modulus N using Algorithm 2.

Corollary 1. *Let $S = \{0, \ldots, l_N/4\}$ and $k = \lfloor l_N/4a \rfloor$ is odd. Suppose $U, V \subseteq S$ such that $U = \{0, \ldots, a\} \cup \{2a - t, \ldots, 3a\} \cup \cdots \cup \{(k-1)a - t, \ldots, ka\}$, $V = \{a - t, \ldots, 2a\} \cup \{3a - t, \ldots, 4a\} \cup \cdots \cup \{ka - t, \ldots, l_N/4\}$, where $p[i]$'s are known for $i \in U$ and $q[j]$'s are known for $j \in V$, as discussed before. Then, one can factor N in poly(log N) time with probability at least* $P_{a,t} = \left(1 - \frac{1}{2^t}\right)^{\lfloor l_N/4a \rfloor}$.

Proof. By setting $T = l_N/4$ in Theorem 3 we obtain that Algorithm 2 is able to recover contiguous $l_N/4$ many MSBs of one of the primes p, q in $O(l_N/4)$ steps with probability at least $P_{a,t} = \left(1 - \frac{1}{2^t}\right)^{\lfloor l_N/4a \rfloor}$. Since one call of CORRECT costs $O(l_N)$, thus the total time complexity is $O(\log^2 N)$.

Once we get these $l_N/4$ MSBs, that is the complete most significant half, of one of the primes, one can use the existing lattice based method [4] by Coppersmith to factor $N = pq$ in poly(log N) time. □

4.4 Experimental Results for the Reconstruction Algorithm

We present some experimental results in Table 3 to support the claim in Theorem 3. The blocksize for known bits, i.e, a, and the approximation offset t are varied to obtain these results for $l_N = 1024$. The target size for reconstruction is $T = 256$ as the primes are 512 bits each. We have run the experiment 1000 times for each pair of fixed parameters a, t. The first value in each cell represents the experimental percentage of success in these cases and illustrate the practicality of our method. The second value in each cell is the theoretical probability of success obtained from Theorem 3.

Table 3. Percentage of success of Algorithm 2 with 512-bit p, q, i.e., $l_N = 1024$

a	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$
10	0, 0	2.5, 0.07	16.8, 3.55	41.5, 19.9	64.5, 45.2	82.1, 67.5	90.6, 82.2	95.0, 90.7	97.2, 95.2	-
20	1.8, 0.02	18.7, 3.17	44.5, 20.1	65.7, 46.1	*81.9, 68.3*	*90.6, 82.8*	*94.8, 91.0*	*97.5, 95.4*	98.5, 97.7	99.3, 97.6
40	15.5, 1.6	42.8, 17.8	66.7, 44.9	*81.8, 67.9*	*90.8, 82.7*	*95.2, 91.0*	*97.8, 95.4*	*98.6, 97.7*	*99.3, 98.8*	*99.9, 99.4*
60	29.1, 6.3	55.6, 31.6	*75.7, 58.6*	*86.6, 77.2*	*91.7, 88.1*	*95.3, 93.9*	*97.4, 96.9*	*98.9, 98.4*	*99.5, 99.2*	*99.9, 99.7*
80	41.9, 12.5	66.4, 42.2	*82.9, 67.0*	*91.0, 82.4*	*95.7, 90.9*	*98.3, 95.4*	*99.1, 97.7*	*99.4, 98.8*	*99.7, 99.4*	*100, 99.7*
100	50.6, 25.0	*74.4, 56.2*	*86.6, 76.6*	*93.7, 87.9*	*97.1, 93.8*	*98.8, 96.9*	*99.6, 98.4*	*99.8, 99.2*	*99.9, 99.6*	*100, 99.8*

One may note that our theoretical bounds on the probability (second value in each cell) is an underestimate compared to the experimental evidences (first value in each cell) in all the cases. This is because we have used the bound on the probability of carry loosely as $p_c \leq 1$ in Lemma 2, whereas a better estimate should have been $p_c \approx \frac{1}{2}$. As an example, let us check the case with $a = 40, t = 3$. Here, the theoretical bound on the probability with $p_c \leq 1$ is 44.9% whereas with $p_c = 0.5$, the same bound comes as 67.9%. The experimental evidence of 66.7% is quite clearly closer to the second one. But we could not correctly estimate the value of p_c and hence opted for a safe (conservative) margin.

The results in *italic font* are of special interest. In these cases one can factorize N in poly(log N)time, with probability greater than $\frac{1}{2}$ by knowing less than 70% of the bits of both the primes combined, that is, by knowing approximately just 35% of the bits of each prime p, q. Note that the result by Herrmann and May [6] requires about 70% of the bits of one of the primes in a similar case where the known bits are distributed over small blocks. Their result factorizes N in time exponential in the number of such blocks, whereas our method produces the same result in time polynomial in the number of blocks.

5 Conclusion

Our work discusses the factorization of RSA modulus N by reconstructing the primes from randomly known bits. The reconstruction method exploits the known bits to prune wrong branches of the search tree and reduces the total search space. We have revisited the work of Heninger and Shacham [5] in Crypto 2009 and provided a combinatorial model for the search where certain bits of the primes are known at random. This, combined with existing lattice based techniques, can factorize N given the knowledge of randomly chosen prime bits in the

least significant halves of the primes. We also explain a lattice based strategy to remedy one of the shortcomings of the reconstruction algorithm. Moreover, we study how N can be factored given the knowledge of some blocks of bits in the most significant halves of the primes. We propose an algorithm that recovers the most significant halves of one or both the primes exploiting the known bits. An obvious open question in this direction is to attack this problem when random MSBs (as in the case for LSBs) instead of certain blocks are available.

Acknowledgments. The authors are grateful to the reviewers for their invaluable comments and suggestions. The second and third authors would like to acknowledge the Council of Scientific and Industrial Research (CSIR) and the Department of Information Technology (DIT), India, for supporting their respective research. The authors are thankful to Amrita Saha of Jadavpur University for implementing the code for the LSB case in the preliminary phase of the work.

References

1. Boneh, D.: Twenty Years of Attacks on the RSA Cryptosystem. Notices of the AMS 46(2), 203–213 (1999)
2. Boneh, D., Durfee, G., Frankel, Y.: Exposing an RSA Private Key Given a Small Fraction of its Bits. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 25–34. Springer, Heidelberg (1998)
3. Cohen, H.: A Course in Computational Algebraic Number Theory. Springer, Heidelberg (1996)
4. Coppersmith, D.: Small Solutions to Polynomial Equations and Low Exponent Vulnerabilities. Journal of Cryptology 10(4), 223–260 (1997)
5. Heninger, N., Shacham, H.: Reconstructing RSA Private Keys from Random Key Bits. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009)
6. Herrmann, M., May, A.: Solving Linear Equations Modulo Divisors: On Factoring Given Any Bits. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 406–424. Springer, Heidelberg (2008)
7. Howgrave-Graham, N.: Finding Small Roots of Univariate Modular Equations Revisited. In: Darnell, M.J. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997)
8. Jochemsz, E., May, A.: A Strategy for Finding Roots of Multivariate Polynomials with new Applications in Attacking RSA Variants. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 267–282. Springer, Heidelberg (2006)
9. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring Polynomials with Rational Coefficients. Mathematische Annalen 261, 513–534 (1982)
10. May, A.: Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey. In: LLL+25 Conference in honour of the 25th birthday of the LLL algorithm (2007), `http://www.cits.rub.de/personen/may.html`
11. Rivest, R.L., Shamir, A.: Efficient Factoring based on Partial Information. In: Pichler, F. (ed.) EUROCRYPT 1985. LNCS, vol. 219, pp. 31–34. Springer, Heidelberg (1986)
12. Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of ACM 21(2), 158–164 (1978)