Publishing Upper Half of RSA Decryption Exponent

Subhamoy Maitra, Santanu Sarkar, and Sourav Sen Gupta

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

Abstract. In the perspective of RSA, given small encryption exponent e (e.g., $e = 2^{16} + 1$), the top half of the decryption exponent d can be narrowed down within a small search space. This fact has been previously exploited in RSA cryptanalysis. On the contrary, here we propose certain schemes to exploit this fact towards efficient RSA decryption.

Keywords: Cryptology, Decryption Exponent, Efficient Decryption, Public Key Cryptography, RSA.

1 Introduction

RSA cryptosystem, publicly proposed in 1978 and named after its inventors Ron Rivest, Adi Shamir and Len Adleman, is the most popular Public Key Cryptosystem till date. Let us first briefly describe the RSA scheme [11,13].

Cryptosystem 1 (RSA). *Let us define $N = pq$ where p and q are primes. By definition of the Euler totient function, $\phi(N) = (p-1)(q-1)$.*

- KeyGen: *Choose e co-prime to $\phi(N)$. Find d such that $ed \equiv 1 \bmod \phi(N)$.*
- KeyDist: *Publish public key $\langle N, e\rangle$ and keep private key $\langle N, d\rangle$ secret.*
- Encrypt: *For plaintext $M \in \mathbb{Z}_N$, ciphertext $C = M^e \bmod N$.*
- Decrypt: *For ciphertext C, plaintext $M = C^d \bmod N$.*

The efficiency of encryption and decryption in RSA depends upon the bit-sizes of e and d respectively, and further, both depend on the size of N too, as all the modular operations are done with respect to N. To improve the decryption efficiency of RSA, another variant of RSA was proposed that uses the Chinese Remainder Theorem (CRT). This is the most widely used variant of RSA in practice and is known as CRT-RSA [10,18].

Preliminaries. Before proceeding further, let us go through some preliminary discussions. For notational purpose, we denote the number of bits in an integer i by l_i; i.e., $l_i = \lceil \log_2 i \rceil$ when i is not a power of 2 and $l_i = \log_2 i + 1$, when i is a power of 2. By *Small* e, we mean $e = 2^{16} + 1$ or around that range, which is popularly used for fast RSA encryption.

Fact 1. *For* Small *e, the top half of d can be estimated efficiently.*

Proof. The RSA equation $ed = k\phi(N) + 1$ translates to $ed = k(N+1) - k(p+q) + 1$, where $l_k \approx l_e$ and $l_d \approx l_N$. In cases where e is *Small*, so is k, and hence can be found using a brute force search. Thus one can estimate d as follows.

$$d = \frac{k}{e}(N+1) + \frac{1}{e} - \frac{k}{e}(p+q) \approx \frac{k}{e}(N+1) + \frac{1}{e}$$

The error in this approximation is $\frac{k}{e}(p+q) < p+q$ as $1 < k < e$. Thus, considering that the primes p and q are of same bit-size, the error is $O(\sqrt{N})$, that is one gets an approximation with error-size less than or equal to $\max(l_p, l_q) \approx \frac{1}{2}l_N \approx \frac{1}{2}l_d$. If we write $d = d_0 + d_1$ with $d_0 = \lceil \frac{k}{e}(N+1) + \frac{1}{e} \rceil$, then $|d - d_0| < 2^{l_N/2}$, which implies that d_0 estimates the top half of d correctly and $d_1 \equiv d \pmod{2^{l_N/2}}$. Thus, for various values of k in the range $1 \leq k < e$, we get those many possibilities for the upper half of d, allowing for an efficient estimate. □

Our Motivation. The estimation of d stated in Fact 1 has been exploited in literature to propose partial key exposure attacks on RSA [2]. Our motivation though is to use this estimation in a constructive way. As one can estimate the half of the bits of d in most significant side anyway in cases where e is *Small*, there is no harm in choosing that top half on our own and to make it public. A few interesting questions come up in this direction.

- Can one choose the most significant half of the bits of d to make RSA decryption more efficient than in the case for general RSA?
- Can one choose the most significant half of the bits of d to *personalize* RSA in some way?
- Can one choose the least significant half of the bits of d in some way (no constraint on the most significant half) so that higher workload can be transferred to a server in case of a server-aided decryption?

Our Contribution. In this paper, we shall answer these questions one by one. First, in Section 2, we propose a scheme for RSA where one can choose around half of the bits of d in most significant side on his/her own, simply to make RSA decryption faster for *Small e*. It is important to note that our result does not compete with fast CRT-RSA decryption; this is only to highlight how simply the general RSA decryption can be made more efficient through this idea. Next, in Section 3, we answer the second question by proposing a *personalized* model of RSA by letting the user choose the most significant half of d on his own. We provide an answer to the third question in Section 4 and illustrate one of its potential applications in the form of a new RSA scheme for low end devices.

In modern cryptography, the issue of cryptographic implementation on low end devices is an emerging field of research. The importance of efficiency comes from the computational constraints in low end hand-held devices like smart cards, phones or PDAs. One knows that sometimes the low end devices (M, say) are connected to a server (S, say). In Section 4, we propose a scheme where S aids in the decryption process to decrease the workload of M without any significant loss of security. Note that there is quite a good amount of research in the field of Server-Aided RSA, initiated in [8] (one may refer to [9] for more recent trends and analysis in this area). However, the existing ideas use additional algorithms over basic RSA and the models are set on a more general framework. In this paper, we present a quite simple approach towards server-aided decryption that does not require any primitive other than RSA.

2 Efficient RSA for *Small e*

In this section, we propose a scheme for RSA where around half of the bits of d in most significant side can be *chosen* as per the will of the user. We consider e *Small* and hence it is logical to consider $l_d = l_N$. We write $d = d_0 + d_1$, where $d_1 = d \bmod 2^{\frac{1}{2}l_N + l_e}$. That is,

- d_0 is an l_N-bit integer where $\left(\frac{1}{2}l_N - l_e\right)$ many most significant bits of d_0 and d are same, and rest of the bits of d_0 in least significant side are zero.
- d_1 is an $\left(\frac{1}{2}l_N + l_e\right)$-bit integer where $\left(\frac{1}{2}l_N + l_e\right)$ many bits of d in least significant side constitute d_1.

We shall henceforth call d_0 the *top* half and d_1 the *bottom* half of d. According to Fact 1, the top portion d_0 of the decryption exponent d can be estimated efficiently as we have only a small number of options in case e is *Small*. The scheme we propose exploits this fact to make RSA decryption faster.

Key Generation. The user is allowed to choose his/her favorite *Small* encryption exponent e, and d_0 as described above. Once he/she decides upon the size of the primes p, q to be used, l_N is fixed. Next, the user fixes a *Small* e (e.g., $e = 2^{16} + 1$), and d_0 with $l_{d_0} = l_N$. Thereafter, the key generation follows Algorithm 1. This algorithm (and also Algorithm 2 presented in Section 4) is in a similar line to that of the key generation algorithms presented in [4,14].

Efficient-RSA. Once we have the power to choose the top half of the decryption exponent (d_0 shares the top half with d), one may choose it in such a fashion that makes the RSA decryption more efficient. A natural act is to choose the top half so that it has a low Hamming weight, which helps in faster decryption. In this line, we present our *Efficient-RSA* scheme as in Cryptosystem 2, and analyze the scheme thereafter.

Cryptosystem 2 (Efficient-RSA). *Choose a* Small *integer e as encryption exponent and choose l_p, l_q, the bit-sizes of the RSA primes. Choose the top half of the decryption exponent as $d_0 = 2^{l_p + l_q - 1}$.*

- KEYGEN: $(p, q, N, d) \leftarrow \text{KEYGENALGOMSB}(e, l_p, l_q, d_0)$.
- KEYDIST: *Publish public key $\langle N, e\rangle$ and keep private key $\langle N, d\rangle$ secret.*
- ENCRYPT: *For plaintext $M \in \mathbb{Z}_N$, ciphertext $C = M^e \bmod N$.*
- DECRYPT: *For ciphertext C, plaintext $M = C^d \bmod N$.*

```
    Input: Small encryption key $e$, Bitsize of primes $l_p, l_q$, and $d_0$ with $l_{d_0} = l_N$
    Output: RSA parameters $p, q, N, d$
 1  Pick a prime $p$ at random with bit-size $l_p$ such that $\gcd(p-1, e) = 1$;
 2  Pick a random number $d_{pad}$ of length $\frac{1}{2} l_{d_0}$ or less;
 3  Set $\tilde{d_0} \leftarrow d_0 + d_{pad}$;
 4  Pick a random number $k$ of length $l_e$ with $\gcd(k, e) = 1$;
 5  Set $x \leftarrow e - \left[(k(p-1))^{-1} \bmod e\right]$;
 6  Set $y \leftarrow \lceil \frac{1}{k(p-1)}(e\tilde{d_0} - 1) \rceil$;
 7  Set $z \leftarrow \lceil \frac{1}{e}(y - x) \rceil$;
 8  Set $w \leftarrow x + ze$;
 9  if $w + 1$ is prime and $l_w = l_q$ then
10      GOTO Step 12;
    end
11  GOTO Step 2;
12  Set $q \leftarrow w + 1$;
13  Set $N \leftarrow pq$;
14  Set $\tilde{d_1} \leftarrow -\frac{1}{e}((e\tilde{d_0} - 1) - k(p-1)w)$;
15  Set $d \leftarrow \tilde{d_0} + \tilde{d_1}$;
16  RETURN $p, q, N, d$;
```

Algorithm 1. The Key Generation Algorithm (KeyGenAlgoMSB)

2.1 Correctness of Cryptosystem 2

The correctness of the scheme depends on the correctness of Key Generation (Algorithm 1), as the other phases are similar to regular RSA. Note that

$$\begin{aligned} e\tilde{d_0} - 1 - k(p-1)w &\equiv -1 - k(p-1)x \\ &\equiv -1 + k(p-1)[k(p-1)]^{-1} \equiv -1 + 1 \equiv 0 \pmod{e}. \end{aligned}$$

Hence, $e\tilde{d_0} - 1 - k(p-1)w = -e\tilde{d_1}$, which implies $e(\tilde{d_0} + \tilde{d_1}) - k(p-1)w = 1$. That is, $ed - k(p-1)(q-1) = 1$, which is the well-known RSA equation. Thus, Algorithm 1 generates the keys as per RSA criteria.

Bitsize of Primes. We also need to verify that the bit-sizes of p and q are l_p and l_q respectively, as prescribed by the user. In Algorithm 1, note that p is already chosen to be of size l_p. Regarding q, note that we have chosen $l_{d_0} = l_p + l_q$ and $l_k = l_e$. By construction, $l_x \approx l_e$ and $l_y \approx l_e + l_{d_0} - l_k - l_p = l_q$. Thus we get $l_w = \max(l_x, l_{ze}) = l_{ze} \approx l_{y-x} = l_y \approx l_q$, as required.

Choice of d_0. Another important issue that requires verification is that the d_0 supplied by the user does actually share the top half with the decryption exponent d, and does not get changed during the miscellaneous operations performed in Algorithm 1. We prove the following result in this direction.

Theorem 1. *The output d generated by* $\textsc{KeyGenAlgoMSB}(e, l_p, l_q, d_0)$ *shares the top $\frac{1}{2}l_N - l_e$ bits with the input d_0.*

Proof. Note that we can write $d = d_0 + d_1$ with $d_1 = d_{pad} + \tilde{d_1}$ where we have chosen d_{pad} such that $l_{d_{pad}} < \frac{1}{2}l_{d_0} = \frac{1}{2}l_d$. Again, we have

$$\begin{aligned}|e\tilde{d_1}| = |e\tilde{d_0} - 1 - k(p-1)w| &= |e\tilde{d_0} - 1 - k(p-1)(x+ze)| \\ &= |e\tilde{d_0} - 1 - k(p-1)(y+\tilde{y})|,\end{aligned}$$

where $\tilde{y} = ze + x - y = \left\lceil \frac{1}{e}(y-x) \right\rceil e - (y-x) < e$. Thus, we obtain the following.

$$\begin{aligned}|e\tilde{d_1}| &\le |e\tilde{d_0} - 1 - k(p-1)y| + |k(p-1)\tilde{y}| \\ &< |e\tilde{d_0} - 1 - k(p-1)y| + |k(p-1)e| \\ &= |k(p-1)| \left| \frac{e\tilde{d_0} - 1}{k(p-1)} - y \right| + |k(p-1)e| \\ &< |k(p-1)| + |k(p-1)e| = (e+1)|k(p-1)|,\end{aligned}$$

and hence $l_{\tilde{d_1}} \le l_e + l_k + l_p - l_e = l_k + l_p = l_e + l_p \approx l_e + \frac{1}{2}l_d$, in cases where $l_p \approx l_q$. Combining these, we get $l_{d_1} = \max(l_{d_{pad}}, l_{\tilde{d_1}}) \le l_e + \frac{1}{2}l_d$. Thus, d_0 represents the $\frac{1}{2}l_N - l_e$ many most significant bits of d correctly. $\square$

2.2 Efficiency of Cryptosystem 2

We have already mentioned that for all practical applications, CRT-RSA is implemented as it is more efficient than RSA. Also, we accepted the fact that our implementation does not compete with CRT-RSA in terms of efficiency. Thus our explanation of efficiency here is as compared with standard RSA.

The encryption phase is the same as that of regular RSA and hence the efficiency is identical to that of a regular RSA scheme using *Small e*. The main advantage comes in case of decryption. As we have chosen $d_0 = 2^{l_N - 1}$ in our key generation algorithm and as $l_{d_1} \approx \frac{l_N}{2} + l_e$, we have the top half of d to be all 0's except for the 1 at the MSB. Also, in the lower half of length l_{d_1}, we have about $\frac{1}{2}l_{d_1}$ many 1's and rest 0's on an average. Now, we know that a 0 in d corresponds to just a squaring and a 1 corresponds to a squaring and a multiplication in our regular *square and multiply* algorithm used for modular multiplication in RSA. Thus, the number of computations in decryption phase will be as follows.

- Regular computation for the bottom half: $\frac{1}{2}l_{d_1}$ multiplications and l_{d_1} squares.
- Just squaring for the top half: $\frac{l_N}{2} - l_e - 1$ squares.
- Regular computation for the 1 at MSB: 1 multiplication and 1 square.

Assume that the cost of one modular squaring is equivalent to μ times the cost of one modular multiplication. Hence, total number of modular multiplications in the decryption phase is

$$\left(\frac{l_{d_1}}{2} + \mu l_{d_1}\right) + \mu\left(\frac{l_N}{2} - l_e - 1\right) + (1+\mu) = \frac{l_N}{4} + \mu l_N + \frac{l_e}{2} + 1 \approx \left(\mu + \frac{1}{4}\right) l_N + \frac{l_e}{2},$$

whereas the same in case of regular RSA (considering half of the bits of d are 1 on an average) is $(\mu + \frac{1}{2})l_d = (\mu + \frac{1}{2})l_N$, as $l_d = l_N$ in general for *Small* e. Thus, we obtain an advantage (in proportion of less number of operations) of

$$1 - \frac{(\mu + \frac{1}{4})l_N + \frac{l_e}{2}}{(\mu + \frac{1}{2})l_N} = \frac{1 - \frac{2l_e}{l_N}}{2(2\mu + 1)}.$$

Asymptotically, one can neglect $\frac{2l_e}{l_N}$ and hence we get the speed up of the order of $\frac{1}{2(2\mu+1)}$. When $\mu = 1$, we get an advantage of 16.67% in the decryption phase, and the advantage increases if $\mu < 1$ in practice. Considering a practical scenario with $l_N = 1024$ and $e = 2^{16} + 1$, the advantage is 16.11% considering $\mu = 1$.

Our result provides similar kind of improvements as that of [7, Section 2.1]. Moreover, the algorithm in [7] could not achieve the asymptotic improvement of 16.67% in practice, whereas our algorithm proposed here reaches that. Hence, in the sense of practical implementation, our algorithm betters that of [7].

Since all the exponentiation operations are modular, it is also important to see how the mod$N$ part in the calculation of $v^2 \bmod N$ or $uv \bmod N$ can be efficiently executed for $u, v \in \mathbb{Z}_N$. It has been pointed out by Lenstra [5] that the operations become efficient when N is of the form $N = 2^{l_N - 1} + t$ for some positive integer t which is significantly smaller than N. Around 30% improvement may be achieved for encryption and decryption with such 1024-bit RSA moduli. During the setup of RSA in this case, one of the primes p is chosen at random and the other one is *constructed* cleverly so that N gains its special form. Since our method chooses both primes p, q at random depending on the choice of d_0, our result does not consider any constraints on N, and hence improvement along the lines of [5] may not be immediately incorporated in our scheme.

2.3 Security of Cryptosystem 2

We have already discussed in Section 1 (Fact 1) that in case of RSA with *Small* e, the top half of the decryption exponent, that is d_0, can be obtained without much effort. Hence, choosing specific d_0 in our scheme does not leak out any extra information regarding the system. Thus, it is quite reasonable to claim that the security of our Efficient RSA cryptosystem is equivalent to the security of a regular RSA system having *Small* encryption exponent e.

Another observation is that we are constructing one of the primes (q) in the algorithm, based on the chosen d_0, d_{pad}, p, k and e. A natural question is whether this construction makes the prime q *special* instead of a random prime, as is expected in RSA. We claim that the prime q constructed in Algorithm 1 is a random prime of length l_q. The following arguments support our claim.

– One may notice that d_0 is chosen to be of a specific form, whereas d_{pad} is a random integer of length $\frac{1}{2}l_{d_0}$ or less. This makes the lower half of $\tilde{d_0}$ random, but the top half shares the same structure as that of d_0.

- Next, we choose p to be a random prime of length l_p and k to be random integer co-prime to e. Hence, the inverse $[k(p-1)]^{-1} \bmod e$ is random in the range $[1, e-1]$, and so is x.

- Let us now assume that $e\tilde{d_0}-1$ is of a specific structure. The reader may note that actually the lower half of this quantity is random, but the assumption of non-randomness just poses a stronger constraint on our argument. In this case, as $k(p-1)$ is totally random, we obtain y, and hence z to be random numbers as well.

- The argument above justifies the randomness of w, by construction, and hence the randomness of q.

One may also wonder whether p and q are *mutually independent* random primes, or do they possess any kind of interdependency due to the choices we made. The following arguments justify the mutual independence of p and q.

- Note that in Algorithm 1, we have the following two approximate relations: $q-1 = w \approx x + \frac{1}{e}(y-x) \cdot e = y$, and $y \approx \frac{1}{k(p-1)}(e\tilde{d_0}-1)$.

- Hence, we have another approximate relation $k(p-1)(q-1) \approx e\tilde{d_0}$, where p is a random prime of size $\frac{1}{2}l_N$, parameter k is random with $l_k = l_e$, and the bottom half of $e\tilde{d_0}$ is random (ed_{pad}) of size approximately $\frac{1}{2}l_N$.

- Now, notice that the relation $k(p-1)(q-1) \approx e\tilde{d_0}$, i.e., $(p-1)(q-1) \approx \frac{e}{k}\tilde{d_0}$, as discussed above, is analogous to the relation $pq = N$ where the top half of N is predetermined, and p is chosen at random. This setup is precisely the one proposed by Lenstra [5] to make RSA modular operations faster by fixing a special structure of N. In case of [5], the primes p and q are not related in any sense, and following a similar logic, the primes in our setup are mutually independent random primes as well.

2.4 Runtime of Setup in Cryptosystem 2

The runtime to set up the proposed RSA scheme is dominated by the runtime of Key Generation (Algorithm 1), which in turn depends on the probabilistic analysis of success of the same.

If we take a look at the **if** condition in Step 9 of Algorithm 1, then the probability of meeting the condition is heuristically of the order of $\frac{1}{\log N}$. This is due to the fact that w, of size $N^{0.5}$, is being constructed to be almost random and the distribution of primes of that size follows a density of $\log \sqrt{N}$, that is $O(\log N)$. Thus, the expected number of iterations of the algorithm is $O(\log N)$.

What we have discussed in this section gives us the power to choose the top half of d, that is d_0, on our own. One may use this idea to implement a *personalized* RSA system as described in the following section.

3 Personalized RSA Scheme

Here we explore the freedom of choosing d_0 in to obtain a *personalized* RSA implementation. Let us first talk about the motivation for this scheme.

Motivation. We are all acquainted with the idea of Domain Name System (DNS) in context of the Internet. According to Wikipedia, "The Domain Name System (DNS) is a hierarchical naming system for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participants." In this context, our motivation is to propose a personalized scheme for Public Key Cryptosystem which, in some sense, is similar to the structure of DNS. This scheme will associate, with the RSA keys of a participant, various information about the participant himself. Moreover, it will translate some identification information of a participant meaningful to humans into a binary string and embed this into the RSA keys of the participant. Now, the only question is how to achieve this task. We can describe this scheme in details as follows.

The Idea. Let us choose to use *Small* encryption exponent e for our scheme, $e = 2^{16} + 1$ say, as it is used most generally on a global frame. In such a case, as we have discussed earlier in Section 1 (Fact 1), one can easily estimate the top half of the decryption exponent d accurately. If that is so, why not let the top half of d be *published*, without affecting the security of RSA. Moreover, if one chooses the top half of d on his own, embeds personal (not private) information, and publishes it along with his or her public key, then we can implement a personalized notion of associating public keys with the users.

Personalized-RSA. Here the top half of the decryption exponent is *chosen* by the user to make the RSA personal. This applies only to the case with *Small* e. The user can fix *Small* e, the size of the primes p, q, and the personal top half d_0 of the decryption exponent d to be used. The RSA keys for our scheme are $\langle N, e, d_0 \rangle$ and $\langle N, d \rangle$, obtained from the output of KeyGenAlgoMSB, and the encryption and decryption are similar to the regular RSA. A formal description of our proposed scheme of is as in Cryptosystem 3.

Cryptosystem 3 (Personalized-RSA). *Choose a* Small *integer e as encryption exponent and choose l_p, l_q, the bit-sizes of the RSA primes. Choose a personal d_0 and embed user information (nothing secret) within d_0.*

- KeyGen: *$(p, q, N, d) \leftarrow$* KeyGenAlgoMSB*(e, l_p, l_q, d_0).*
- KeyDist: *Publish public key $\langle N, e, d_0 \rangle$ and keep private key $\langle N, d \rangle$ secret.*
- Encrypt: *For plaintext $M \in \mathbb{Z}_N$, ciphertext $C = M^e \bmod N$.*
- Decrypt: *For ciphertext C, plaintext $M = C^d \bmod N$.*

The correctness, key-sizes and runtime analysis of Cryptosystem 3 goes along the same line as Cryptosystem 2, and hence is being omitted to avoid duplicity.

Points to Note. There are no issues of guaranteed efficiency improvement in this case as the structure of d_0 may be arbitrary as per the choice of the user. We do not compromise on the security of the RSA system by publishing d_0 because in case with *Small* e, the top half d_0 can very easily be estimated anyway. While choosing a *personal* d_0, the user must keep in mind the following.

– It is not necessary to choose d_0 exactly of the length $l_p + l_q$, as this issue will be corrected in the key generation algorithm anyway.

– It is important to keep the embedded information shorter that $\frac{1}{2}(l_p + l_q)$, because the lower half of d will be modified by the KeyGenAlgoMSB. Suggested length of embedded information is $\min(l_p, l_q) - l_e$ or shorter.

We also like to clarify the fact that this personalized scheme does not offer any cryptographic identification or verification facilities for the user or the users respectively. As the encryption exponent e is *Small*, one can always obtain the top half d_0 of the decryption exponent and run KeyGenAlgoMSB using it. Thus, there is obviously a risk that Oscar can generate and use a RSA system where his d_0 is identical to the d_0 of Alice or Bob. But this is similar to faking the Domain Name to IP Address correspondence in case of DNS, and can not be prevented right away. And of course, for obvious reasons of privacy, one should not embed sensitive information into d_0, as this is being published in our scheme.

Potential Benefits. One may ask what benefits this Personalized-RSA scheme has to offer over the general RSA scheme used in practice. We would like to propose the following two ideas as potential benefits of Personalized-RSA.

– In case of RSA cryptosystem, the public key gets binded to the implementor using certificates issued by trusted authorities. In general, the public key is embedded within the certificate along with other authentication parameters. This application provides a good motivation to reduce the size of the RSA public key, and a lot of research has been undertaken in this direction. One may refer to [15] for an idea proposed by Vanstone and Zuccherato, which was later broken using the idea of Coppersmith [3]. In case of Personalized-RSA, the public key $\langle N, e, d_0 \rangle$ need not be completely embedded in the certificate as d_0 is based on some well-known public identity of the user, the email id or name, say. This reduces the size of the RSA public key and certificate.

– More importantly, one may note that the user creating the RSA framework must store the decryption key $\langle N, d \rangle$ as a secret parameter. In case of the proposed Personalized-RSA scheme, one just needs to store $\langle N, d_1 \rangle$, as the other part d_0 becomes a part of the public key in this case. This considerably reduces the storage cost for RSA secret key. Another idea for generating small RSA keys may be found at [12].

In the next section, we present an alternative application of our methods of choosing portions of d towards efficient RSA decryption in low end devices with the help of a server.

4 Server Aided Scheme: Choosing the Bottom Half of d

As mentioned earlier, there are lots of existing results in the sector of Server-Aided RSA (one may refer to papers, e.g., [8,9]), and we are not presenting any competing scheme in that aspect. The advantage of our simple approach is it uses only the RSA primitive and nothing else, unlike the existing schemes [8,9]. Consider a situation where Alice is using a hand-held device or a smart-card with low computing power. The term d_0 is not kept secret for *Small* e and hence some third party may share part of the decryption load.

The Scheme. Alice chooses *Small* e and fixes the lengths of the primes, l_p and l_q. She also chooses d_1 cleverly so that the weight of d_1 small, but it can prevent exhaustive search. The Hamming weight of d_1 should be considerably lower than that in a random case, so as to make decryption faster. Now, the participants Alice (low end device), Paulo (server) and a Bob (sender) behave as follows.

– KeyGen: Alice creates the keys using Algorithm 2 with input (e, l_p, l_q, d_1).
– KeyDist: Define $d_0 = d - d_1$. Alice publishes public key $\langle N, e\rangle$, gives Paulo the information $\langle N, d_0\rangle$, and keeps her decryption key $\langle N, d_1\rangle$ secret.
– Encryption: Bob encrypts plaintext $M \in \mathbb{Z}_N$ as $C = M^e \bmod N$.
– Server: Paulo computes $V = C^{d_0} \bmod N$, and sends (V, C) to Alice.
– Decryption: Alice receives (V, C) and computes $M = VC^{d_1} \bmod N$.

Let us present the key generation algorithm for the proposed scheme. Since the server will execute $V = C^{d_0} \bmod N$, we do not need any restriction on d_0, but we need to choose d_1 for efficiency. Once Alice fixes e and d_1, and decides upon the size of the primes p, q to be used, the key generation algorithm described in Algorithm 2 will provide the solution.

4.1 Correctness of Algorithm 2

The correctness of the proposed scheme relies on the correctness of the key generation algorithm (Algorithm 2). One may note that

$$ed = ew2^{l_q - l_e + 1} + ed_1 = (-ed_1 + 1 + k(p-1)(q-1)) + ed_1 = k(p-1)(q-1)$$

which represents the RSA equation $ed \equiv 1 \bmod \phi(N)$. This proves the correctness of Algorithm 2, and hence of the proposed scheme.

Bitsize of Primes. Note that we obtain $l_q = l_z = l_y - 1$ from Algorithm 2. Also, it is obvious that l_y is the size of $e \cdot 2^{l_q - l_e + 1}$ due to the modular operation while constructing y. Thus, $l_q = l_y - 1 = l_e + (l_q - l_e + 1) - 1 = l_q$, as expected.

Choice of d_1. Along similar lines of analysis performed in case of Algorithm 1, one may also verify the following result. Proof omitted to avoid duplicity.

Theorem 2. *The output d generated by* KeyGenAlgoLSB(e, l_p, l_q, d_1) *shares the bottom $\frac{1}{2}l_N - l_e$ bits with the input d_1.*

```
    Input: Encryption exponent $e$, Size of primes $l_p, l_q$, and $d_1$ with $l_{d_1} = \frac{1}{2}l_N$
    Output: RSA parameters $N, p, q, d$
 1  Choose random prime $p$ with $l_p = \frac{1}{2}l_N$, $\gcd(p-1, e) = 1$ and $\frac{1}{2}(p-1)$ odd;
 2  Choose a random integer $k$ with $l_k = l_e$ such that $\gcd(k, e, 2) = 1$;
 3  Set $x \leftarrow \left[\frac{1}{2}k(p-1)\right]^{-1} \bmod (e \cdot 2^{l_q - l_e + 1})$;
 4  Set $y \leftarrow [(ed_1 - 1) \cdot x] \bmod (e \cdot 2^{l_q - l_e + 1})$;
 5  Set $z \leftarrow \frac{1}{2}y + 1$;
 6  if $z$ is prime and $l_z = l_q$ then
 7      GOTO Step 9;
    end
 8  GOTO Step 2;
 9  Set $q \leftarrow z$;
10  Set $N = pq$;
11  Set $w \leftarrow \frac{1}{e \cdot 2^{l_q - l_e + 1}}(-ed_1 + 1 + k(p-1)(q-1))$;
12  Set $d \leftarrow w \cdot 2^{l_q - l_e + 1} + d_1$;
13  RETURN $N, p, q, d$;
```

Algorithm 2. The Key Generation Algorithm (KeyGenAlgoLSB)

4.2 Efficiency of the Protocol

The encryption phase is the same as that of regular RSA and hence the efficiency is identical to that of a regular RSA scheme using *Small e*. However, substantial advantage can be achieved in case of decryption. As we have already commented in Section 2.2, the decryption cost for regular RSA is $(\mu + \frac{1}{2})l_N$ many modular multiplications, assuming the cost of one modular square is equivalent to μ times the cost of one modular multiplication. So total number of bit operation will be $(\mu + \frac{1}{2})l_N^3$. In this case, we are not keen to reduce the load of the server, but of the hand-held device. Thus, let us check the decryption cost for Alice.

The computation performed by Alice during decryption will be $\frac{1}{2}l_N - l_e$ many modular squares (length of d_1), and w_1 many modular multiplications (number of 1's in d_1). Hence, total number of modular multiplications (including equivalent squaring) in decryption phase by Alice is $\mu\left(\frac{1}{2}l_N - l_e\right) + w_1$. Thus, Alice obtains an advantage (in proportion of less number of operations)

$$1 - \frac{\mu\left(\frac{1}{2} - \frac{l_e}{l_N}\right) + \frac{w_1}{l_N}}{\mu + \frac{1}{2}} = \frac{\mu + 1 + \frac{2\mu l_e}{l_N} - \frac{2w_1}{l_N}}{2\mu + 1}.$$

Considering a practical scenario with $l_N = 1024, e = 2^{16+1}$ and $w_1 = 40$, the advantage is 65.17% for $\mu = 1$. Asymptotically, one can neglect $\frac{l_e}{l_N}$ and $\frac{w_1}{l_N}$ and hence we get the speed up of the order of $\frac{\mu+1}{2\mu+1}$. When $\mu = 1$, we get an advantage of 66.67% in the decryption phase, and the advantage increases when

$\mu < 1$ (in practice, it is sometimes considered that squaring requires less effort than multiplication).

Along the line of CRT-RSA [10,18], Alice can calculate $C^{d_1} \bmod N$ using Chinese Remainder Theorem (CRT). She first calculates $C_p \equiv C^{d_1} \bmod p$ and $C_q \equiv C^{d_1} \bmod q$. From C_p and C_q, one can easily obtain $C^{d_1} \bmod N$ using CRT-RSA. Since $l_p = l_q = \frac{1}{2} l_N$, in this situation Alice needs to perform

$$2\left[\mu\left(\frac{1}{2}l_N - l_e\right) + w_1\right]\left(\frac{l_N}{2}\right)^2 = \left[\mu\left(\frac{1}{4} - \frac{l_e}{2l_N}\right) + \frac{w_1}{2l_N}\right] l_N^3$$

many bit operations. Considering a practical scenario with $l_N = 1024, e = 2^{16+1}$ and $w_1 = 40$, the advantage is 82.58% for $\mu = 1$. Neglecting $\frac{l_e}{l_N}, \frac{w_1}{l_N}$, one can achieve an advantage of

$$1 - \frac{\mu/4}{\mu + 1/2} = \frac{3\mu + 2}{4\mu + 2}.$$

When $\mu = 1$, asymptotic advantage of 83.33% can be obtained during decryption. In the next subsection we perform the security analysis of the proposed scheme and argue why the aforementioned choice of parameters is secure.

4.3 Security of the Protocol

Similar to the case of Cryptosystem 2, the prime q generated here is a random prime of size l_q, and does not seem to possess any special form. Notice that the randomness of q in Algorithm 1 was generated from a random choice of d_{pad} having greater information than the choice of d_1 in case of Algorithm 2. This apparently hints that the randomness of q in Algorithm 2 is lower than that in Algorithm 1. But a closer observation will reveal that the randomness of q in Algorithm 2 also depends on the randomness of x, which in turn, depends on the randomness of p and k, carrying similar information as d_{pad} in the earlier case. Thus the randomness of q is comparable in both algorithms. Moreover, the random choice of k and d_1 guarantees the independence of p and q in Algorithm 2, which is much desired in this kind of a setup. Also, as d is of order of N, a lot of known attacks [18,16,1,17] will not work in our case.

Now assume $d'_1 \equiv d \bmod (2^{\frac{1}{2}l_N})$ i.e., d'_1 represents the lower half of bits of d. One may verify that the $\frac{1}{2}l_N - l_e$ many least significant bits of d'_1 and d_1 are same. When e is *Small* and d_1 is known, then number of possible options of d'_1 will be very small. So we may assume that knowing d_1 is almost equivalent (with little bit of extra search) of knowing d'_1.

One must note that while choosing d_1 for *Small* e, there is a risk of brute force attack on this portion, and as the top half is known by default (Fact 1), this may make the system vulnerable. We need to refer the following results by Boneh et. al. [2, Theorems 3.1, 3.3] towards the security analysis of this proposal.

Fact 2. *Suppose $N \equiv 3 \bmod 4$ and $e < \frac{1}{8}N^{0.25}$. Then given $\frac{1}{4}\log_2 N$ many least significant bits of d, one can factor N in polynomial of $\log N$ and e.*

Fact 3. *Suppose $|p-q| > \frac{\sqrt{N}}{4}$ and $e < \frac{1}{8}N^{0.25}$. Then given $\frac{1}{4}\log_2 N$ many bits of d in the positions (from the least significant side) $\frac{1}{4}\log_2 N$ to $\frac{1}{2}\log_2 N$, one can factor N in polynomial of* $\log N$ *and e.*

Since in our case e is small, we should choose d_1 such a way so that it is computationally infeasible to find the lower half of d_1 (due to Fact 2). We should also choose d_1 such that it is computationally impossible to find upper half of d_1 (due to Fact 3). Thus one should choose d_1 with weight w_1 so that any half of the bit pattern of d_1 can not be searched exhaustively.

Let us illustrate the situation with a practical example for 1024-bit RSA. In case of our scheme, half of d_1 is of size $\frac{1}{2}(\frac{1024}{2} - 16) = 248$ and we know that the LSB must be a 1. Let us choose $w_1 \approx 40$ and assume that the 1's are distributed uniformly at random over the length of d_1. So, there are 19 possible places out of 247 in the lower half of d_1 for 1's, and the rest are 0's. Same is the case for the top half of d_1. A brute force search for these bit positions will result in a computational search complexity of $\binom{247}{19} \approx 2^{93}$. For a comparison, Number Field Sieve (NFS) [6] is the fastest known factorization algorithm that requires around 2^{86} time complexity to factor a 1024-bit RSA modulus. Hence, choosing $w_1 \approx 40$ in case of 1024 bit RSA with $e = 2^{16} + 1$ suffices for the security if the 1's are more or less uniformly distributed over the length of d_1.

In this direction, we would like to mention that the security of this scheme is comparatively weaker than the security of Efficient-RSA. If one knows t many LSBs of d, then t LSBs of the primes p, q are compromised as well. Knowing these t many LSBs is easier in this scheme as d_1 is constructed with low Hamming weight, whereas d_{pad} in the Efficient-RSA scheme was chosen at random.

4.4 Runtime of Setup

The runtime to set up the proposed Server Aided scheme is dominated by the runtime of key generation (Algorithm 2). The **if** condition in Step 6 of Algorithm 2 is satisfied with probability of the order of $\frac{1}{\log N}$. This is due to the fact that z, of size $N^{0.5}$, is being constructed to be almost random and the distribution of primes of that size follows a density of $\log \sqrt{N}$, that is $O(\log N)$. Thus, the expected number of iterations of Algorithm 2 is $O(\log N)$ during the setup.

5 Conclusion

In this paper, we have taken up a known but highly understated fact that half of the decryption exponent d can be obtained in case of RSA implementations with *Small* encryption exponent e. We have deviated from using this in cryptanalysis of RSA, and have tried to exploit this in a constructive fashion. In this direction, we proposed a couple of key generation algorithms, and illustrated their implications through a few proposed RSA schemes, as follows.

- Efficient-RSA: One can choose the upper half of the decryption exponent d, and obtain certain advantages in the decryption phase over natural RSA.

– Personalized-RSA: One can personalize the upper half of the decryption exponent d and publish it when e is *Small*, to create a DNS-like RSA convention.

– Server Aided Scheme: One can choose the lower half of d and publish the upper half of d (for *Small* e) so that a server can help decrease the computation cost during RSA decryption.

The uniqueness of our approach depends on the novelty of the motivation as well as the simplicity of the proposed schemes, which in most of the cases, use just a basic RSA primitive and nothing else.

Acknowledgments. The authors are grateful to the anonymous reviewers for their invaluable comments and suggestions. The third author would like to acknowledge the Department of Information Technology (DIT), India, for supporting his research at the Indian Statistical Institute.

References

1. Boneh, D., Durfee, G.: Cryptanalysis of RSA with Private Key d Less Than $N^{0.292}$. IEEE Transactions on Information Theory 46(4), 1339–1349 (2000)
2. Boneh, D., Durfee, G., Frankel, Y.: Exposing an RSA Private Key given a Small Fraction of its Bits, `http://crypto.stanford.edu/~dabo/abstracts/bits_of_d.html`
3. Coppersmith, D.: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. Journal of Cryptology 10(4), 233–260 (1997)
4. Galbraith, S., Heneghan, C., McKee, J.: Tunable Balancing RSA, `http://www.isg.rhul.ac.uk/~sdg/full-tunable-rsa.pdf`
5. Lenstra, A.: Generating RSA moduli with a predetermined portion. In: Ohta, K., Pei, D. (eds.) ASIACRYPT 1998. LNCS, vol. 1514, pp. 1–10. Springer, Heidelberg (1998)
6. Lenstra, A.K., Lenstra Jr., H.W.: The Development of the Number Field Sieve. Springer, Heidelberg (1993)
7. Maitra, S., Sarkar, S.: Efficient CRT-RSA Decryption for Small Encryption Exponents. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 26–40. Springer, Heidelberg (2010)
8. Matsumoto, T., Kato, K., Imai, H.: Speeding up secret computations with insecure auxiliary devices. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 497–506. Springer, Heidelberg (1990)
9. Nguyen, P.Q., Shparlinski, I.: On the Insecurity of a Server-Aided RSA Protocol. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 21–25. Springer, Heidelberg (2001)
10. Quisquater, J.-J., Couvreur, C.: Fast decipherment algorithm for RSA public-key cryptosystem. Electronic Letters 18, 905–907 (1982)
11. 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)
12. Sakai, R., Morii, M., Kasahara, M.: New Key Generation Algorithm for RSA Cryptosystem. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 77(1), 89–97 (1994)

13. Stinson, D.R.: Cryptography - Theory and Practice. 2nd Edition, Chapman & Hall/CRC (2002)
14. Sun, H.-M., Hinek, M.J., Wu, M.-E.: On the Design of Rebalanced RSA-CRT, `http://www.cacr.math.uwaterloo.ca/techreports/2005/cacr2005-35.pdf`
15. Vanstone, S.A., Zuccherato, R.J.: Short RSA Keys and Their Generation. Journal of Cryptology 8(2), 101–114 (1995)
16. Verheul, E., van Tilborg, H.: Cryptanalysis of less short RSA secret exponents. Applicable Algebra in Engineering, Communication and Computing 18, 425–435 (1997)
17. de Weger, B.: Cryptanalysis of RSA with small prime difference. Applicable Algebra in Engineering, Communication and Computing 13, 17–28 (2002)
18. Wiener, M.: Cryptanalysis of Short RSA Secret Exponents. IEEE Transactions on Information Theory 36(3), 553–558 (1990)