Correcting Errors in RSA Private Keys

Wilko Henecka, Alexander May[⋆], Alexander Meurer[⋆⋆]

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
Faculty of Mathematics
`wilko.henecka@rub.de`, `alex.may@rub.de`, `alexander.meurer@rub.de`

Abstract. Let $\mathsf{pk} = (N, e)$ be an RSA public key with corresponding secret key $\mathsf{sk} = (p, q, d, d_p, d_q, q_p^{-1})$. Assume that we obtain partial error-free information of sk, e.g., assume that we obtain half of the most significant bits of p. Then there are well-known algorithms to recover the full secret key. As opposed to these algorithms that allow for *correcting erasures* of the key sk, we present for the first time a heuristic probabilistic algorithm that is capable of *correcting errors* in sk provided that e is small. That is, on input of a full but error-prone secret key $\widetilde{\mathsf{sk}}$ we reconstruct the original sk by correcting the faults.

More precisely, consider an error rate of $\delta \in [0, \frac{1}{2})$, where we flip each bit in sk with probability δ resulting in an erroneous key $\widetilde{\mathsf{sk}}$. Our Las-Vegas type algorithm allows to recover sk from $\widetilde{\mathsf{sk}}$ in expected time polynomial in $\log N$ with success probability close to 1, provided that $\delta < 0.237$. We also obtain a polynomial time Las-Vegas factorization algorithm for recovering the factorization (p, q) from an erroneous version with error rate $\delta < 0.084$.

Keywords. RSA, error correction, statistical cryptanalysis

1 Introduction

RSA is the most widely deployed cryptosystem and has successfully withstood more than 30 years of cryptanalytic attacks [1]. An RSA modulus $N = pq$ is a product of two primes and the key-pair $e, d \in \mathbb{Z}^*_{\phi(N)}$ satisfies $ed = 1 \bmod \phi(N)$. Although theoretically, it would suffice to use (N, d) as the RSA private key it is recommended in PKCS#1 standard [10] to use the highly redundant tuple $(N, e, d, p, q, d_p, d_q, q_p^{-1})$ in order to also allow for a fast Chinese Remainder type decryption process. Here, the last three components of sk are defined as usual by $d_p = d \bmod p - 1$, $d_q = d \bmod q - 1$ and $q_p = q^{-1} \bmod p$.

[⋆] This research was supported by the German Research Foundation (DFG) as part of the project MA 2536/3-1 and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

[⋆⋆] This work was supported by the Ruhr-University Research School funded by Germanys Excellence Initiative [DFG GSC 98/1].

In the present work, we look at error-prone RSA keys, where we assume that the public information (N, e) is never affected by errors. Thus, we only look at erroneous tuples $\mathsf{sk} = (p, q, d, d_p, d_q, q_p^{-1})$. Our error-correction algorithm is motivated by side-channel attacks that are capable of extracting the complete private key but with some errors [5]. We assume that the errors are uniformly spread over the whole secret key, i.e., each secret key bit is flipped with some fixed error probability $\delta \in [0, \frac{1}{2})$. Notice that for $\delta = \frac{1}{2}$ we obtain a completely random string that does not provide any information about sk.

Theoretically, our attack framework is modeled by oracle-assisted attacks on RSA. Oracle-assisted attacks were first introduced by Rivest and Shamir [9] at Eurocrypt 1985. Rivest and Shamir used an oracle that allowed for querying bits of p in chosen positions. They showed that given such an oracle $\frac{3}{5} \log p$ queries are sufficient to factor N in polynomial time. This was later improved by Coppersmith [3] to only $\frac{1}{2} \log p$ queries. In 1992, Maurer [8] showed that for stronger oracles, which allow for any type of oracle queries with YES/NO answers, $\epsilon \log p$ queries are sufficient for any $\epsilon > 0$.

In this oracle-based line of research, the goal is to minimize both the power of the oracle and the number of queries to the oracle. At Crypto 2009, Heninger and Shacham [6] presented a polynomial time attack on RSA that works whenever a 0.27-fraction of the key bits of sk is given, provided that the given bits are uniformly spread over the whole secret key. So as opposed to the oracle used by Rivest, Shamir and Coppersmith, the attacker has no control about the positions in which he receives some of the bits but he knows the positions and on expectation the erasure intervals between two known bits are never too large.

Notice that all these aforementioned attacks require a limited number of *fault-free* information provided by the oracles, mostly in the form of secret key bits. Since side-channel attacks are practical instantiations of oracles, in most scenarios it is questionable to put a limit on the number of bits that one can obtain. Why should an attacker stop at some point to extract bits? Why should he not proceed until he has the full bit information? In a more realistic scenario an attacker is capable of extracting the full sk bit string but subject to some errors that were caused by the physical measurements of his side-channel attack. This is the *error-prone* scenario that we address in our paper. Hence our work might motivate to look for weaker forms of side-channel attacks that produce only erroneous data.

Our result and related work. We present the first attack running in expected polynomial time that recovers a secret key sk from a disturbed secret key $\widetilde{\mathsf{sk}}$, where every bit is flipped with a fixed error rate of $\delta < 0.237$. That is, we allow for *error correction* of an RSA secret key, provided that the public RSA exponent is small. We also give results where an attackers obtains an erroneous version for a subset of the entries of $sk = (p, q, d, d_p, d_q, q_p^{-1})$. E.g., we obtain a polynomial time attack for erroneous versions of (p, q, d) with error rates $\delta < 0.160$. Moreover, we obtain a polynomial time factorization algorithm that factors N given a faulty version of (p, q) with error rate $\delta < 0.084$. In this case, we do not need any restriction on the public exponent e.

Our work builds on the *erasure correction* algorithm of Heninger-Shacham [6] which allows for erasures of the secret key bits of sk with an erasure rate of $\delta < 0.73$. So as one might expect from coding theory, the correction of errors seems to be a much harder problem than the correction of erasures.

As our work builds on the Heninger-Shacham algorithm, let us briefly recall the idea of this construction. Heninger and Shacham recover the parameters p, q, d, d_p, d_q bit by bit in a 2-adic fashion by growing a search tree. In their algorithm, the information from q_p^{-1} is ignored. The nodes in depth k of the search tree correspond to partial solutions of $p \bmod 2^k, q \bmod 2^k, \ldots, d_q \bmod 2^k$.

Since in the erasure correction scenario, one has fragmentary but correct key material, one can easily prune partial solutions that do not coincide with the known secret key bits. This process will never discard the correct solution, since the correct solution will always fully agree with the incomplete key material. Thus, such an algorithm will always succeed to recover sk.

The only remaining problem is to bound the algorithm's running time. Intuitively, whenever one has sufficiently many key bits to falsify incorrect partial solutions, one will obtain a bounded number of false partial solutions per iteration and so the total number of nodes in the search tree will stay small. Heninger and Shacham showed that with high probability the total number of partial solutions is quadratic in $\log(N)$ whenever the erasure rate is smaller than 0.73, i.e., we know at least a 0.27-fraction of the key bits. In order to show this result, Heninger and Shacham had to make the heuristic assumption that once a key candidate differs from the correct key, the subsequent candidate key bits are distributed uniformly at random.

Clearly, the Heninger-Shacham comparison of key candidates with the given key material cannot naively been transferred to the error correction scenario. The reason is that a disagreement of a key candidate may originate from an incorrect key candidate *or* from faulty bits of the key material. Thus, in our construction we do no longer compare bit by bit but we compare larger blocks of bits. More precisely, we grow subtrees of depth t for each key candidate. This results in 2^t new candidates which we all compare with our faulty key material. If the bit agreement with our key material in these t bits is above some threshold parameter C we keep the candidate, otherwise we discard it.

Clearly, we obtain a trade-off for the choice t of the depth of our subtrees. On the one hand, t cannot be chosen too large since in each iteration wae grow our search tree by at least 2^t candidates. Thus, t must be bounded by $\mathcal{O}(\log \log N)$ in order to guarantee a polynomial size of the search tree.

On the other hand, depending on the error rate t has to be chosen sufficiently large to guarantee that the correct key candidate has large agreement with our key material $\widetilde{\mathsf{sk}}$ in each t successive bit positions, such that the correct candidate will never be discarded during the execution of our algorithm. Moreover, t has to be large enough such that the distribution corresponding to the correct candidate and the distribution derived from an incorrect candidate are separable by some threshold parameter C. If this property does not hold, we obtain too many faulty candidates for the next iteration.

We show that the above trade-off restrictions can be fulfilled whenever we have an error rate $\delta < 0.237 - \epsilon$ for some fixed $\epsilon > 0$. That is, if we choose t of size polynomial in $\log \log N$ and $\frac{1}{\epsilon}$, we are able to define a threshold parameter C such that the following holds.

1. With probability close to 1 the correct key candidate will never be discarded during the execution of the algorithm.
2. For fixed $\epsilon > 0$, our algorithm will consider no more than an expected total number of $\log^{\mathcal{O}(1)} N$ key candidates. E.g., our algorithm has expected running time polynomial in the bit-size of N.

We would like to point out that our proper choice of t and C assumes that we know a good upper bound for the error rate δ. In practical side-channel attacks where δ might be unknown to an attacker, one can apply an additional search step which successively increases the value of δ until a solution is found. Alternatively, we provide a way to compute an equate upper bound for δ during the initialization phase of the algorithm.

Our algorithm is a probabilistic algorithm of Las Vegas type, i.e., whenever it outputs a solution the output is the correct secret key sk. Our error correction algorithm is elementary. The main work that has to be done is to carefully choose the subtree depth t and the threshold parameter C, such that all trade-off restrictions hold. We achieve this goal by using a statistical analysis via Hoeffding bounds. Our analysis relies on a similar heuristic assumption as in [6], that is, as soon as a key candidate differs from the correct solution its subsequent bits are distributed uniformly at random.

Furthermore, we would like to stress that analogous to [6], our algorithm is restricted to the case of small public exponents e – except for the case of correcting erroneous factorizations (p, q). Small public exponent RSA appears to be the standard in practical applications [11].

We ran experiments to verify the predictions of our theoretical analysis and to validate the heuristic assumption. In practice, we achieved to correct error rates of up to $\delta = 0.2$ for 1024-bit RSA private keys with good success probabilities in a matter of seconds.

The paper is organized as follows. In Section 3, we briefly review the Heninger-Shacham algorithm. Section 4 introduces our new block-based threshold algorithm that grows subtrees of depth t. Section 5 is devoted to the theoretical analysis of the subtree depth t and the choice of the threshold parameter for pruning nodes that correspond to incorrect candidates. Experimental results are given in Section 6.

2 Notation and Mathematical Background

For an n-bit string $\mathbf{x} = (x_{n-1}, \ldots, x_0) \in \{0,1\}^n$ let $x[i] = x_i$ denote the i-th bit of $\mathbf{x}$ (where $x[0]$ is the least significant bit of $\mathbf{x}$) and let $\mathbf{x}[i..j] = (x_i, x_{i-1}, \ldots, x_j)$

for $i \geq j$. Throughout the paper we denote by $\ln(n)$ the natural logarithm of n to base e and we denote by $\log(n)$ the binary logarithm of n to base 2.

The main technical tool used in our analysis is Hoeffding's bound [7], which upper bounds the absolute error of sums of independent random variables from their mean value.

Theorem 1. *Let $X_1, \ldots, X_n$ be a sequence of independent Bernoulli trials with identical success probability $\mathbf{Pr}[X_i = 1] = p$ for all i. Define $X := \sum_{i=1}^{n} X_i$. Then for every $0 < \gamma < 1$ we have*

i) $\mathbf{Pr}[X \geq n(p+\gamma)] \leq e^{-2n\gamma^2}$,
ii) $\mathbf{Pr}[X \leq n(p-\gamma)] \leq e^{-2n\gamma^2}$.

A slightly more general version of Hoeffding's inequality allows for each random variable X_i an individual expectation $\mathbb{E}[X_i]$ as well as a wider range, i.e. $X_i \in [a, b]$ for $a, b \in \mathbb{R}$. We define $\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i]$ and the above statement transforms to

$$\mathbf{Pr}\left[X \gtrless \mathbb{E}[X] \pm n\gamma\right] \leq e^{-\frac{2n\gamma^2}{(b-a)^2}}. \tag{1}$$

3 The Heninger-Shacham Algorithm

Let (N, e) be an RSA public key with corresponding PKCS#1 standard secret key $\mathsf{sk} = (p, q, d, d_p, d_q, q_p^{-1})$, where

$$ed = 1 \bmod \phi(N),\, d_p = d \bmod p-1,\, d_q = d \bmod q-1 \text{ and } q_p^{-1} = q^{-1} \bmod p.$$

We will ignore the last parameter q_p^{-1} as it is not used in the attack. Let N be the product of two $\frac{n}{2}$-bit primes, i.e., all the secret key parameters except d can be represented by $\frac{n}{2}$ bits. The Heninger-Shacham algorithm recovers these parameters bit by bit starting from the least significant bit until bit $\frac{n}{2}-1$, where the factorization is revealed. Although by a result of Coppersmith [3] an amount of $\frac{n}{4}$ bits would suffice for factoring N in polynomial time, going up to bit $\frac{n}{2}-1$ instead does not significantly change the algorithm's analysis.

It is not hard to see that all parameters p, q, d, d_p, d_q alone reveal the factorization of N, see [4]. Thus, the secret key is a highly redundant representation of the prime factorization. This redundancy in turn implies that the following four RSA identities simultaneously hold

$$N = pq \tag{2}$$
$$ed = 1 + k\phi(N) \tag{3}$$
$$ed_p = 1 + k_p(p-1) \tag{4}$$
$$ed_q = 1 + k_q(q-1), \tag{5}$$

for some parameters k, k_p and k_q that we are able to compute for small public exponents e.

We have $0 < k < e\frac{d}{\phi(N)} < e$, so there are at most $e-1$ possible candidates for k. Therefore, we can brute-force search over all candidate values for k. Following an argument of Boneh, Durfee and Frankel [2], for each candidate value k', we define

$$d(k') = \left\lfloor \frac{1 + k'(N+1)}{e} \right\rfloor , \tag{6}$$

which differs for the right choice $k' = k$ from d by $\frac{k(p+q)}{e} < p + q$. Thus, for the right candidate choice of k the values of $d(k)$ and d agree roughly on half of their most significant bits.

In the *erasure correction* scenario, Heninger and Shacham simply compare each candidate $d(k')$ with the given fragmentary version of d in order to determine k uniquely with overwhelming probability.

We proceed similarly in the *error correction* szenario. Assume that we obtain some error-prone secret key

$$\widetilde{\mathsf{sk}} := (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d}_p, \tilde{d}_q),$$

which is derived from sk by flipping each bit individually with some fixed probability $\delta \in [0, \frac{1}{2})$. Intuitively, if δ is significantly below $\frac{1}{2}$, then among all $e - 1$ candidates $d(k'), k' = 1, \ldots, e-1$, the Hamming distance between the upper half most significant bits of $d(k')$ and $\tilde{d}$ should be minimal for the correct choice $k' = k$. In Appendix A, we show that this is true with overwhelming probability for the error rates δ that we allow.

This means that we can learn the unknown k in Eq. (3). Moreover, we can immediately correct almost half of the most significant bits of d. Notice that this information is not useful in the Heninger-Shacham algorithm as one stops to recover the secret key bits when reaching bit $\frac{n}{2} - 1$. However, we can use this information to compute a good approximation $\tilde{\delta}$ of the error rate δ. Therefore, we simply compute the normalized Hamming distance of $d(k)$ and $\tilde{d}$ by

$$\tilde{\delta} := \frac{2}{n} \sum_{i=n/2}^{n-1} \tilde{d}[i] \oplus d(k)[i]. \tag{7}$$

For n large enough and any fixed tolerance $\epsilon > 0$, we have $\delta \leq \tilde{\delta} + \epsilon$ with overwhelming probability. That is, in our asymptotic analysis it is reasonable to assume that the algorithm knows an upper bound of the error rate δ. For practical values of n, one can easily show that

$$\mathbf{Pr}[\delta < \tilde{\delta} + \epsilon] \geq \frac{3}{4}$$

where $\epsilon = 0.037$ for $n = 1024$, see App. B for more details.

Now that we are able to compute k, Heninger and Shacham show that this directly allows us to compute candidates for (k_p, k_q). If e is prime then there are only two candidate values. In general, for e with m different prime factors there exist up to 2^m candidates. So one has to run 2^m copies of the Heninger-Shacham algorithm in parallel. Since $m = \mathcal{O}(\log e)$ and since we consider small public exponent RSA only, this factor can be neglected. We denote this whole precomputation process by $(k, k_p, k_q) \leftarrow \mathsf{Init}(N, e)$.

Now let us start to reconstruct a secret key in a bitwise manner. Since p, q are odd primes, we have $p[0] = q[0] = 1$ and $2|p-1$ as well as $2|q-1$. Let $\tau(x)$ denote the largest exponent such that $2^{\tau(x)}$ divides x, i.e. $\tau(x) := \max\{k \in \mathbb{N} : 2^k | x\}$. From Eq. (4), we obtain

$$ed_p = 1 \bmod 2^{1+\tau(k_p)}.$$

Thus, we can immediately correct the least $1 + \tau(k_p)$ bits of d_p from the knowledge of e and k_p. Analogously, we can compute from Eq. (5) the $1 + \tau(k_q)$ least significant bits of d_q and from Eq. (3) the $2 + \tau(k)$ least significant bits of d.

Moreover, if we fix all bits $p[i - 1..0]$ then changing bit $p[i]$ will change bit $d_p[i + \tau(k_p)]$. For odd k_p this means that changing the i-th bit on the right hand side of Eq. (4) changes the corresponding i-th bit on the left hand side. Shifting by $\tau(k_p)$ on the right-hand side translates the change to position $i + \tau(k_p)$ on the left hand side.

Thus, Heninger and Shacham define for each bit index i a so-called *i-th bit slices*, which we denote by

$$\mathsf{Slice}(i) := (p[i], q[i], d[i + \tau(k)], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)]).$$

Let $\mathsf{Slice}(0) \leftarrow \mathsf{Mount}(e, k, k_p, k_q)$ be the computation of the initial first bit slice consisting of the steps described above, i.e., we set

$$\mathsf{Slice}(0) \leftarrow (1, 1, d[\tau(k)], d_p[\tau(k_p)], d_q[\tau(k_q)]),$$

where the last three entries can be easily computed once k, k_p and k_q are known. The running time of $\mathsf{Mount}(\cdot)$ can be neglected in our small public exponent RSA scenario.

Lifting solutions. Assume that we have computed a partial solution $\mathsf{sk}' = (p', q', d', d'_p, d'_q)$ up to $\mathsf{Slice}(i - 1)$. We would like to proceed by calculating all candidate solutions (p, q, d, d_p, d_q) for the subsequent $\mathsf{Slice}(i)$. Heninger and Shacham show that by applying a multivariate version of Hensel's Lemma to Eq. (2)-(5) one obtains the following identities

$$p[i] + q[i] = (N - p'q')[i] \bmod 2 \tag{8}$$
$$d[i + \tau(k)] + p[i] + q[i] = (k(N + 1) + 1 - k(p' + q') - ed')[i + \tau(k)] \bmod 2 \tag{9}$$
$$d_p[i + \tau(k_p)] + p[i] = (k_p(p' - 1) + 1 - ed'_p)[i + \tau(k_p)] \bmod 2 \tag{10}$$
$$d_q[i + \tau(k_q)] + q[i] = (k_q(q' - 1) + 1 - ed'_q)[i + \tau(k_q)] \bmod 2. \tag{11}$$

This means we have four linearly independent equations in the five unknowns $p[i], q[i], d[i+\tau(k)], d_p[i+\tau(k_p)], d_q[i+\tau(k_q)]$ of $\mathsf{Slice}(i)$. Therefore, each Hensel lift yields exactly two candidate solutions. We denote this lifting process by $\mathsf{Expand}(p', q', d', d'_p, d'_q)$.

In the *erasure correction* scenario, Heninger and Shacham use their knowledge of the correct secret key bits to prune incorrect candidates produced by the lifting process. The analysis in [6] mainly shows that the number of candidates is sufficiently upper bounded as long as enough secret key bits are available.

Notice that in our *error correction* scenario, such a simple pruning is not possible, since a disagreement of $\mathsf{Slice}(i)$ with the corresponding bits of sk might be due to errors in our faulty secret key.

4 Blockwise Threshold-Based Vector Correction

4.1 Generic Description

In this section, we present our new algorithm for error correction. We would like to point out that our construction is a generic, elementary algorithm for reconstructing arbitrary *unknown* tuples of bit vectors $\mathbf{x}$ given only a corrupted version $\tilde{\mathbf{x}}$ and some public information on $\mathbf{x}$, which does not directly allow for extracting $\mathbf{x}$. For example, $\mathbf{x}$ may be the prime factorization of some public N.

In coding theory language, our construction resembles a maximum likelihood approach. In each iteration, we keep those vectors that are locally closest to $\tilde{\mathbf{x}}$ in the Hamming distance. Hopefully, we are also able to discard many incorrect partial solutions due to our public information.

Let $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_m})$. In a nutshell, our algorithm tries to reconstruct $\mathbf{x}$ iteratively by calculating a block of t bits of each of $\mathbf{x_1}, \ldots, \mathbf{x_m}$ in each iteration. The algorithm proceeds in four phases, where the second and third phase are iterated until the candidates have the same bitlength as $\mathbf{x}$.

Initialization phase: Use the public information to compute some initial partial solution to $\mathbf{x}$. This initialization is optional and may result in the empty string as the only partial solution.

Expansion phase: Each partial solution is lifted for the next t most significant bits, i.e., we compute the next t bits of each of $\mathbf{x_1}, \ldots, \mathbf{x_m}$. Per partial solution this will result in up to 2^{mt} new partial solutions. By using our public information, we may hope to obtain considerably less than 2^{mt} candidates.

Pruning phase: For every new partial solution we count the number of matches of the mt expanded bits with the corresponding bits of $\tilde{\mathbf{x}}$. If this number is below some threshold parameter C then we discard the partial solution.

Finalization phase: We test with the help of our public information whether one of our candidate solutions is indeed equal to the desired $\mathbf{x}$.

Obviously, the choice of the blocksize t is crucial for our algorithm. Since the number of partial solutions in the expansion phase grows exponentially in t, we cannot allow for large parameters t. On the other hand, we cannot choose t too small, because we have to make sure that during the pruning phase the following two properties hold.

- The correct partial solution – the one that can be expanded to the desired $\mathbf{x}$ – is pruned only with small probability.
- Sufficiently many incorrect solutions are pruned such that the total number of candidates can be minimized.

4.2 Error Correction for RSA keys

Let us now specialize the generic description from the previous section to our RSA error correction scenario. We want to compute some *unknown* RSA secret key $\mathsf{sk} = (p, q, d, d_p, d_q)$ from an erroneous version $\widetilde{\mathsf{sk}} = (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d_p} \tilde{d_q})$ with the help of the public key (N, e). For describing our algorithm, we use the notion introduced in Sect. 3.

Algorithm Error-Correction

INPUT: (N, e), erroneus $\widetilde{\mathsf{sk}} = (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d_p}, \tilde{d_q})$ with error rate δ

```
Initialization phase:
  • $(k, k_p, k_q) \leftarrow \mathsf{Init}(N, e)$
  • $\mathsf{Slice}(0) \leftarrow \mathsf{Mount}(e, k, k_p, k_q)$
For $i = 1$ to $\left\lceil \frac{n/2-1}{t} \right\rceil$
    Expansion phase: For every candidate $(p', q', d', d'_p, d'_q)$
    with slices $0 \dots (i-1)t$ expand the candidate $t$ times with the
    $\mathsf{Expand}(\cdot)$ procedure of Heninger-Shacham. This results in
    $2^t$ new candidates which differ in the slices $(i-1)t+1, \dots, it$.

    Pruning phase: For every new candidate $(p', q', d', d'_p, d'_q)$
    count the number of bits in the expanded slices
    $(i-1)t+1, \dots, it$ that agree with the corresponding
    bits of $\widetilde{\mathsf{sk}}$. If this number is below some threshold parameter
    $C$, discard the solution.
Finalization phase: For every candidate $\mathsf{sk}' = (p', q', d', d'_p, d'_q)$
check all RSA identities (2)–(5). If all equations hold, output $\mathsf{sk}'$.
```

OUTPUT: $\mathsf{sk} = (p, q, d, d_p, d_q)$

Notice that during the *Expansion phase* for every partial solution we only obtain 2^t new candidates for the $5t$ new bits instead of the naive 2^{5t} candidates. This is due to the clever usage of our public information in the $\mathsf{Expand}(\cdot)$ procedure of Heninger and Shacham.

In the subsequent section, we will analyze the probability that our algorithm succeeds in computing the secret key sk. We will show that a choice of $t = \theta(\frac{\ln n}{\epsilon^2})$

will be sufficient for error rates $\delta < 0.237 - \epsilon$. The threshold parameter C will be chosen such that the correct partial solution will survive each pruning phase with probability close to 1 and such that we expect that the number of candidates per iteration is bounded by 2^{t+1}. For every fixed $\epsilon > 0$, this leads to an expected running time that is polynomial in n.

5 Choice of Parameters and Success / Runtime Analysis

We now give a detailed analysis for algorithm Error-Correction from the previous section. Afterwards, we show that this analysis easily generalizes to settings where an attacker obtains instead of a faulty version of all five parameters in sk only faulty versions of e.g. (p, q, d) or (p, q).

5.1 Full Analysis for the RSA Case

Remember that in algorithm Error-Correction, we count the number of matching bits between $5t$-bit blocks of $\widetilde{\mathsf{sk}}$ and every partial candidate solution. Let us define a random variable X_c for the number of matching bits between $\widetilde{\mathsf{sk}}$ and a correct partial solution.

The distribution of X_c is clearly the binomial distribution with parameters $5t$ and probability $(1-\delta)$, denoted by $X_c \sim \text{Bin}(5t, 1-\delta)$. That is, we have

$$\mathbf{Pr}[X_c = \gamma] = \binom{5t}{\gamma}(1-\delta)^{\gamma}\delta^{5t-\gamma} \tag{12}$$

for $\gamma = 0, \ldots, 5t$. The expected number of matches is thus $\mathbb{E}[X_c] = 5t(1-\delta)$.

Assume that we expand some incorrect partial solution (p', q', d', d'_p, d'_q) by $5t$ bits to 2^t new candidates. We let the random variable X_b represent the number of matching bits of $\widetilde{\mathsf{sk}}$ with the expanded $5t$ bits of these bad candidates.

In order to analyze the distribution of X_b, we make use of the following heuristic assumption which follows directly from the heuristic assumption of Heninger-Shacham [6] when applied to t-bit blocks.

Heuristic 2. *Every solution generated by applying the expansion phase to an incorrect partial solution is an ensemble of t randomly chosen bit slices.*

That is under Heuristic 2, every expansion of an incorrect candidate in Error-Correction results in an additional $5t$ uniformly random bits.

Heninger and Shacham verified the validity of this heuristic experimentally. Under Heuristic 2 we see that

$$\mathbf{Pr}[X_b = \gamma] = \binom{5t}{\gamma}2^{-5t}. \tag{13}$$

Now, we basically have to choose our threshold C such that the two distributions are sufficiently separated.

The remainder of this section is devoted to proof our main result.

Main Theorem 3. *Under Heuristic 2 for every fixed $\epsilon > 0$ the following holds. Let (N, e) be an RSA public key with n-bit N and fixed e. We choose*

$$t = \left\lceil \frac{\ln(n)}{10\epsilon^2} \right\rceil,\ \gamma_0 = \sqrt{(1+\tfrac{1}{t}) \cdot \tfrac{\ln(2)}{10}} \text{ and } C = 5t(\tfrac{1}{2} + \gamma_0).$$

Further, let $\widetilde{\mathsf{sk}} = (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d_p}, \tilde{d_q})$ be an RSA secret key with noise rate

$$\delta \leq \frac{1}{2} - \gamma_0 - \epsilon.$$

Then algorithm Error-Correction *corrects $\widetilde{\mathsf{sk}}$ in expected time $\mathcal{O}\left(n^{2+\frac{\ln(2)}{5\epsilon^2}}\right)$ with success probability at least $1 - \left(\frac{5\epsilon^2}{\ln(n)} + \frac{1}{n}\right)$.*

Remark. Notice that for sufficiently large n, t converges to infinity and thus γ_0 converges to $\sqrt{\frac{\ln(2)}{10}} \approx 0.263$. This means that Error-Correction asymptotically allows for error rates $\frac{1}{2} - \gamma_0 - \epsilon \approx 0.237 - \epsilon$ and succeeds with probability close to 1.

Proof. The proof of our main theorem is organized as follows. First, we upper bound the expected number of bad solutions that arise in each iteration of the algorithm. Second, we show that our correct solution survives all pruning steps with probability close to 1. Third, we upper bound the total number of partial solutions that arise during the execution of Error-Correction and conclude that Error-Correction runs in polynomial time.

Let the random variables Y_i represent the number of incorrect partial solutions that pass the threshold comparison in the pruning phase of the i-th iteration of Error-Correction. Further, let the random variable $Y = \sum_{i=1}^{\tau(n)} Y_i$ denote the total number of incorrect solutions examined by Error-Correction, where $\tau(n) := \left\lceil \frac{n/2-1}{t} \right\rceil$ denotes the total number of iterations.

Lemma 4. *The expected number of bad candidates that pass the i-th round's pruning phase is upper bounded by $\mathbb{E}[Y_i] < 2^{t+1}$.*

Proof. Define two random variables Z_g and Z_b as follows: Z_g denotes the number of bad candidates arising from the unique correct solution, Z_b counts the number of bad candidates generated from a single bad partial solution. It is not hard to see that

$$\mathbb{E}[Y_1] = \mathbb{E}[Z_g] \text{ and } \mathbb{E}[Y_2] = \mathbb{E}[Z_g] + \mathbb{E}[Z_b] \cdot \mathbb{E}[Y_1] = \mathbb{E}[Z_g] \cdot (1 + \mathbb{E}[Z_b]).$$

More generally, we obtain

$$\begin{aligned}\mathbb{E}[Y_i] = \mathbb{E}[Z_g] + \mathbb{E}[Z_b] \cdot \mathbb{E}[Y_{i-1}] &= \mathbb{E}[Z_g] + \mathbb{E}[Z_b] \cdot (\mathbb{E}[Z_g] + \mathbb{E}[Z_b] \cdot \mathbb{E}[Y_{i-2}]) \\ &= \ldots = \mathbb{E}[Z_g] \sum_{k=0}^{i-1} \mathbb{E}[Z_b]^k = \mathbb{E}[Z_g] \frac{1 - \mathbb{E}[Z_b]^i}{1 - \mathbb{E}[Z_b]}.\end{aligned}$$

Now, we aim at upper bounding $\mathbb{E}[Z_b] < 1$ in order to upper bound

$$\mathbb{E}[Y_i] = \mathbb{E}[Z_g]\frac{1-\mathbb{E}[Z_b]^i}{1-\mathbb{E}[Z_b]} < \frac{\mathbb{E}[Z_g]}{1-\mathbb{E}[Z_b]}. \tag{14}$$

Therefore, we define 2^t random variables Z_b^i for $i = 1, \ldots, 2^t$ such that

$$Z_b^i = \begin{cases} 1 & i\text{-th bad candidate passes} \\ 0 & \text{otherwise} \end{cases}.$$

Write $Z_b = \sum_{i=1}^{2^t} Z_b^i$. Since all the Z_b^i are identically distributed, we simplify this to $\mathbb{E}[Z_b] = 2^t\,\mathbb{E}[Z_b^i]$ and upper bound $\mathbb{E}[Z_b^i]$ for some fixed i. Note, that $Z_b^i = 1$ iff at least C bits match, i.e.,

$$\mathbb{E}[Z_b^i] = \mathbf{Pr}[Z_b^i = 1] = \mathbf{Pr}[X_b \geq C],$$

where $X_b \sim \text{Bin}(5t, \frac{1}{2})$ is defined as in (13). Applying Hoeffding's bound (Theorem 1) directly yields

$$\begin{aligned}\mathbf{Pr}[X_b \geq C] &= \mathbf{Pr}\left[X_b \geq 5t\left(\frac{1}{2}+\gamma_0\right)\right] \\ &\leq \exp(-10t\gamma_0^2) = 2^{-(1+\frac{1}{t})t} \leq 2^{-(t+1)}.\end{aligned}$$

This implies $\mathbb{E}[Z_b] \leq \frac{1}{2} < 1$ and we can simplify equation (14) to

$$\mathbb{E}[Y_i] < \frac{\mathbb{E}[Z_g]}{1-\mathbb{E}[Z_b]} < 2^{t+1},$$

since we clearly have $\mathbb{E}[Z_g] \leq 2^t - 1$. □

Lemma 5. Error-Correction *succeeds with probability at least* $1-\left(\frac{5\epsilon^2}{\ln(n)}+\frac{1}{n}\right)$.

Proof. The probability of pruning the correct solution at one single round is given by $\mathbf{Pr}[X_c < C]$, where $X_c \sim \text{Bin}(5t, 1-\delta)$ as defined in (12). Using $\frac{1}{2}+\gamma_0 \leq 1-\delta-\epsilon$ and applying Hoeffding's bound (Theorem 1) yields

$$\begin{aligned}\mathbf{Pr}[X_c < C] = \mathbf{Pr}\left[X_c < 5t\left(\frac{1}{2}+\gamma_0\right)\right] &\leq \mathbf{Pr}\left[X_c < 5t(1-\delta-\epsilon)\right] \\ &\leq \exp(-10t\epsilon^2) \leq \frac{1}{n}.\end{aligned}$$

Since algorithm Error-Correction runs $\tau(n) \leq \frac{n}{2t}+1$ rounds, the total success probability is given by

$$\begin{aligned}\mathbf{Pr}[\text{success}] = (1-\mathbf{Pr}[X_c < C])^{\tau(n)} &\geq \left(1-\frac{1}{n}\right)^{\tau(n)} \geq 1-\frac{\tau(n)}{n} \\ &\geq 1-\left(\frac{1}{2t}+\frac{1}{n}\right) = 1-\left(\frac{5\epsilon^2}{\ln(n)}+\frac{1}{n}\right).\end{aligned}$$

□

Lemma 6. Error-Correction *runs in expected time* $\mathcal{O}\left(n^{2+\frac{\ln(2)}{5\epsilon^2}}\right)$.

Proof. The total expected runtime T of Error-Correction is given by

$$T = T_{\mathsf{Init}} + \mathcal{O}(e) \cdot (T_{\mathsf{Mount}} + T_{\mathsf{main}})$$

where T_{Init}, T_{Mount} and T_{main} represent the runtime of the procedures Init, Mount and the main loop of Error-Correction, respectively. Recall that a factor of $\mathcal{O}(e)$ arises from the fact that $\mathsf{Init}(\cdot)$ possibly outputs up to e candidate tuples (k, k_p, k_q). Since we assume e to be fixed, we can neglect T_{Init} as well as T_{Mount} and obtain $T = \mathcal{O}(T_{\mathsf{main}})$.

In order to upper bound T_{main}, we upper bound the runtime needed by the expansion and pruning phase for one *single* partial solution:

– During the expanding phase, each partial solution implies the computation of $\sum_{i=0}^{t-1} 2^i < 2^t$ equation systems given by the equations (8)-(11). The right hand sides of equations (8)-(11) can be computed in time $\mathcal{O}(n)$ – when storing the results of the previous iteration. This yields a total computation time of $\mathcal{O}(n2^t)$ for the expanding phase.
– The pruning phase can be realized in time $\mathcal{O}(t)$ for each of the fresh 2^t partial solutions, summing up to $\mathcal{O}(t2^t)$.

We can upper bound $t \leq n$, which results in an overall runtime of $\mathcal{O}((n+t)\cdot 2^t) = \mathcal{O}(n2^t)$ per candidate.

An application of Lemma 4 yields an upper bound for the expected total number of partial solutions examined during the whole execution which is given by

$$\mathbb{E}[Y] = \sum_{i=1}^{\tau(n)} \mathbb{E}[Y_i] < \tau(n) \cdot 2^{t+1} \leq \left(\frac{n}{2t} + 1\right) \cdot 2^{t+1} = \mathcal{O}\left(n2^t\right).$$

Putting both together finally yields

$$T_{\mathsf{main}} = \mathcal{O}\left(n2^t \cdot n2^t\right) = \mathcal{O}\left(n^2 2^{2t}\right) = \mathcal{O}\left(n^{2+\frac{\ln(2)}{5\epsilon^2}}\right).$$

□

Combining Lemma 5 and 6 proves the Main Theorem. □

Although theoretically Lemma 6 gives us a polynomial running time for every fixed $\epsilon > 0$, our running time heavily depends on the parameter t and thus on ϵ. So one might expect that in practice one cannot achieve error rates close to the theoretical bound $\delta < 0.237$ since the running time already explodes for moderately small error terms ϵ.

However, we give in Appendix C a more refined analysis of the parameter t for moderately small ϵ. This analysis shows that our choice of t in Theorem 3 is quite conservative, since we insist on a success probability of Error Correction close to 1. We obtain more flexibility if we also allow for smaller success rates. This in turn leads to a smaller choice of t, which allows to easily correct error rates up to $\delta = 0.2$ in practice. We will use this refined analysis in the experimental section (Section 6).

5.2 Generalization

We now formulate a slightly generalized version of our Main Theorem 3. Therefore, we parametrize algorithm ERROR-CORRECTION such that it allows for a secret key with m components like in the generic description in Sect. 4.1.

So our RSA secret key $\mathsf{sk} = (p, q, d, d_p, d_q)$ resembles the parameter choice $m = 5$. We can apply the same analysis as in Section 5.1. The distributions of X_c and X_b are now given by $X_c \sim \text{Bin}(mt, 1-\delta)$ and $X_b \sim \text{Bin}(mt, \frac{1}{2})$.

Main Theorem 7. *Under Heuristic 2 for every fixed $\epsilon > 0$ the following holds. Let (N, e) be an RSA public key with n-bit N and fixed e. We choose*

$$t = \left\lceil \frac{\ln(n)}{2m\epsilon^2} \right\rceil, \ \gamma_0 = \sqrt{(1+\tfrac{1}{t}) \cdot \frac{\ln(2)}{2m}} \ \textit{and} \ C = mt(\tfrac{1}{2} + \gamma_0).$$

Further, let $\widetilde{\mathsf{sk}} = (\widetilde{\mathsf{sk}}_1, \ldots, \widetilde{\mathsf{sk}}_m)$ be a generic RSA secret key with noise rate

$$\delta \le \frac{1}{2} - \gamma_0 - \epsilon.$$

Then algorithm ERROR-CORRECTION *corrects $\widetilde{\mathsf{sk}}$ in expected time $\mathcal{O}\left(n^{2+\frac{\ln(2)}{m\epsilon^2}}\right)$ with success probability at least $1 - \left(\frac{m\epsilon^2}{\ln(n)} + \frac{1}{n}\right)$.*

As a consequence we obtain various results for scenarios where an attacker obtains an erroneous subset of the parameters in $\mathsf{sk} = (p, q, d, d_p, d_q)$. The resulting upper bounds for the error rates $\delta = \frac{1}{2} - \gamma_0$ are summarized in the following table. In the column "Equations" we indicate which of the Eqs. (8)-(11) are used.

Table 1. Parameters for varied RSA scenarios

sk	m	Equations	δ
(p, q)	2	(8)	0.084
(p, q, d)	3	(8),(9)	0.160
(p, q, d, d_p)	4	(8)-(10)	0.206
(p, q, d, d_q)	4	(8),(9),(11)	0.206
(p, q, d, d_p, d_q)	5	(8)-(11)	0.237

The case $\mathsf{sk} = (p, q, d_p)$ can also be handled by our algorithm by using Eqs. (8),(10) with parameter $m = 3$. The only problem is that we cannot derive k and therefore compute k_p as described in Sect. 3, since we do not have information of d. Instead, we simply run $e - 1$ copies of the algorithm in parallel for each possible choice of $1 \le k < e$.

6 Implementation and Experiments

We implemented our algorithm in Java and tested it on an Intel Xeon Quad-Core processor at 2.66 GHz with 8 GB of DDR2 SDRAM at 800 MHz. In all experiments we set the public exponent to $e = 2^{16} + 1$. For the case $\mathsf{sk} =$

(p, q, d, d_p, d_q) we ran a large number of experiments for a key size of 1024 bit and error rates $\delta \in [0.05, 0.2]$. We also carried out experiments for the scenarios $\mathsf{sk} = (p, q)$ and $\mathsf{sk} = (p, q, d)$ where we made experiments for different error rates up to the upper bounds presented in Table 1.

In each repetition, the RSA secret key was independently and randomly disturbed with error rate δ. For simplicity, we omitted the mounting phase, i.e., the calculation of k as well as k_q and k_p. Thereby, we avoided to choose the wrong assignment for k_q and k_p. Instead we just used the correct values for these parameters.

The choice of our tree depth t roughly followed the refined analysis in Appendix C, where we made some manual adjustments for very small error rates and for $\delta \geq 0.18$. The threshold parameter C was chosen as recommended in Theorem 3 with some rounding. All manual adjustments were made in order to obtain comparability of our experiments, i.e., we slightly tuned to achieve success probabilities in an interval between 20% and 50%. We point out that for small error rates it is easy to achieve much better success probabilities by a small increase of the parameter t.

For each experiment we generated 100 different RSA secret keys and disturbed each of these keys with 100 different error vectors resulting in a total sample size of 10.000 runs per error rate δ.

The tables below summarize our results. We computed the success probability by calculating the term $\mathbf{Pr}[X_c < C]$ as defined in Eq. (12) exactly for the given parameters (row "**Pr** theoretical"). The experimental results perfectly match the exact calculations. In the last row, we give the average running time of algorithm Error-Correction in order to reconstruct a *single key* successfully.

Table 2. Experimental results for $n = 1024$ and $\mathsf{sk} = (p, q, d, d_p, d_q)$

δ	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14	0.15	0.16	0.17	0.18	0.19	0.20
t	3	4	5	6	7	9	9	10	10	11	12	12	13	16	16	20
C	12	16	20	24	28	36	36	39	39	42	46	45	48	59	59	74
Pr theoretical	0.39	0.48	0.51	0.49	0.44	0.50	0.27	0.49	0.28	0.44	0.28	0.35	0.43	0.47	0.26	0.23
experimental	0.40	0.48	0.52	0.50	0.45	0.51	0.27	0.50	0.28	0.45	0.28	0.35	0.44	0.50	0.24	0.21
time	< 1s					...						< 1s	3.7s	23s	25s	3m

For error rates $\delta \leq 0.15$ we can easily achieve better success probabilities by using a slightly larger t, e.g., for $\delta = 0.15$ we experimentally achieved $\mathbf{Pr}[\text{success}] \approx 82\%$ with a modified choice $t = 15$ and $C = 56$.

For each run, we also recorded the total number of partial solutions examined by Error-Correction. The following boxplot diagram represents the statistics of the total number of candidates. The thick horizontal line marks the median, the gray boxes describe the region bounded by the lower quartile Q1 and the upper quartile Q3, i.e., half of the candidate numbers fall in this intervall. The dashed lines mark the sample minimum and maximum, respectively.

In our experiments for error rates $\delta \leq 0.15$, we always examined around 300 candidates on the average and the maximum number of candidates never exceeded 1000 candidates. We omit the box plot for error rates $\delta \geq 0.18$ since the number of candidates increases rapidly beyond this bound. This is where the exponential dependence of our running time $\mathcal{O}\left(n^{2+\frac{\ln(2)}{5\epsilon^2}}\right)$ on the parameter ϵ comes into play.

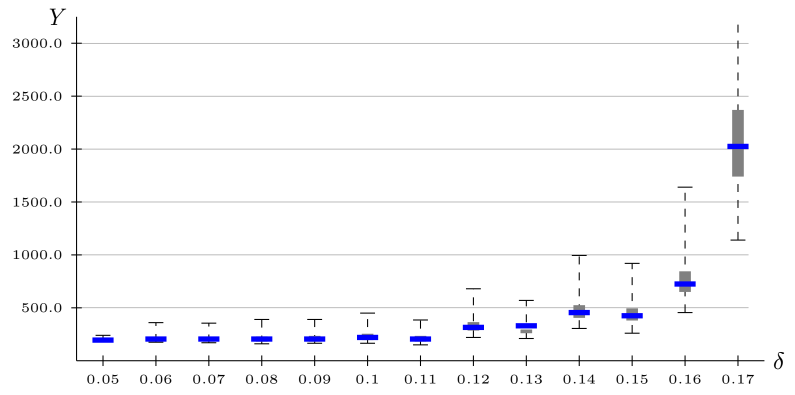

Fig. 1. Box plot diagram for 1024 bit key size and $\mathsf{sk} = (p, q, d, d_p, d_q)$

Table 3. Experimental results for $n = 1024$ and $\mathsf{sk} = (p, q, d)$

	δ	0.05	0.06	0.07	0.08	0.09	0.10	0.11	0.12	0.13	0.14
	t	3	5	7	9	11	13	16	20	26	29
	C	7	12	17	22	27	32	39	49	64	71
Pr	theoretical	0.24	0.34	0.38	0.36	0.30	0.23	0.33	0.26	0.21	0.17
	experimental	0.24	0.34	0.38	0.36	0.30	0.25	0.34	0.24	0.21	0.15
	time	< 1s			...			< 1s	1.7s	4.1s	32.2s	3m

Table 4. Experimental results for $n = 1024$ and $\mathsf{sk} = (p, q)$

	δ	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08
	t	4	7	7	11	15	20	24	28
	C	7	12	12	19	26	35	42	49
Pr	theoretical	0.70	0.83	0.56	0.61	0.58	0.45	0.34	0.24
	experimental	0.71	0.84	0.57	0.62	0.58	0.47	0.35	0.22
	time	< 1s		...			< 1s	2s	12.7s

Acknowledgement. The authors thank the anonymous CRYPTO reviewers for their comments, in particular for suggesting a way to approximate the error rate δ.

References

1. D. Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the American Mathematical Society (AMS)*, 46(2):203–213, 1999.
2. D. Boneh, G. Durfee, and Y. Frankel. An attack on rsa given a small fraction of the private key bits. In *ASIACRYPT '98: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, pages 25–34, London, UK, 1998. Springer-Verlag.
3. D. Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
4. J.-S. Coron and A. May. Deterministic polynomial-time equivalence of computing the rsa secret key and factoring. *J. Cryptology*, 20(1):39–50, 2007.
5. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In P. C. van Oorschot, editor, *USENIX Security Symposium*, pages 45–60. USENIX Association, 2008.
6. N. Heninger and H. Shacham. Reconstructing rsa private keys from random key bits. In S. Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.
7. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
8. U. M. Maurer. Factoring with an oracle. In *EUROCRYPT*, pages 429–436, 1992.
9. R. L. Rivest and A. Shamir. Efficient factoring based on partial information. In *EUROCRYPT*, pages 31–34, 1985.
10. RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard*, June 2002.
11. S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In A. Feldmann and L. Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, Nov. 2009.

A Mounting the attack

Recall that for generating the initial $\mathsf{Slice}(0)$ one has to determine the correct k in (3). We proposed in Sect. 3 to compute the $e-1$ candidates $d(k')$ for $0 < k' < e$ as defined in (6) and chose the k whose corresponding $d(k)$ has minimal Hamming distance to the error prone $\tilde{d}$.

We now give a formal justification for our claim that the Hamming distance between the error prone key $\tilde{d}$ and one of the candidates $d(k')$ is minimized for the correct $d(k)$ with probability close to 1. Therefore, we define random variables $X(k')$ counting the number of matching bits between $\tilde{d}$ and a fixed $d(k')$ on their $\alpha := \lfloor n/2 \rfloor - 2$ most significant bits. For every $k' \neq k$ let

$$D(k') := X(k) - X(k')$$

denote the gap of matching bits for the correct $d(k)$ and a fixed $d(k')$ in their window of α most significant bits. We aim to derive a lower bound for $\mathbf{Pr}[D(k') > 0]$ for arbitrary $k' \neq k$.

The main observation is that for the correct k and balanced p and q, we have $0 \leq d(k) < p + q < 3\sqrt{N}$. This implies that $d(k)$ agrees with the correct d on at least α most significant bits. On the contrary, for $k' \neq k$ one obtains that $d(k')$ and d agree on at most $\log(e)$ most significant bits. Notice that one can consider $D(k')$ as a sum of α random variables $D(k')_{n-i}$ where $i = 1, \dots, \alpha$, each taking values in $\{-1, 0, 1\}$ representing the following three cases.

1. $D(k')_{n-i} = 1$ if $d(k)[n-i]$ and $\tilde{d}[n-i]$ do match *but* $d(k')[n-i]$ and $\tilde{d}[n-i]$ do not match.
2. $D(k')_{n-i} = 0$ if both $d(k)[n-i]$ and $d(k')[n-i]$ match with $\tilde{d}[n-i]$.
3. $D(k')_{n-i} = -1$ if $d(k)[n-i]$ and $\tilde{d}[n-i]$ do not match *but* $d(k')[n-i]$ and $\tilde{d}[n-i]$ do match.

Assuming that in the case $k' \neq k$ every bit of $d(k')$ and $\tilde{d}$ except for the $(n - \log(e))^{\text{th}}$ most significant bits matches with probability $\frac{1}{2}$, we obtain

$$\mathbb{E}[D(k')_{n-i}] = (1-\delta)\frac{1}{2} - \delta\frac{1}{2} = \frac{1}{2}(1-2\delta)$$

for $i = \log(e) + 1, \dots, \alpha$. Summing over all i yields

$$\mathbb{E}[D(k')] \geq \frac{(\alpha - \log(e))(1-2\delta)}{2}.$$

An application of the generalized Hoeffding inequality from (1) yields

$$\mathbf{Pr}[D(k') > 0] = 1 - \mathbf{Pr}[D(k') \leq 0] \geq 1 - \exp\left(-\frac{(\alpha - \log(e))^2(1-2\delta)^2}{8\alpha}\right)$$

for arbitrary $k' \neq k$. Hence, we can lower bound the probability of the event that $D(k') > 0$ for *every* $k' \neq k$ by taking this expression to the $(e-2)^{\text{th}}$ power. For fixed e and $\delta \ll \frac{1}{2}$ we asymptotically achieve probability 1 since the exponent converges to $-\infty$. We calculated the probability for our experimental parameters $n = 1024$, $e = 2^{16} + 1\}$ and the theoretical upper bound $\delta = 0.237$. In this case the probability is very close to 1.

B Estimating the error rate

Recall the definition of $\tilde{\delta} := \frac{2}{n}\sum_{i=n/2}^{n-1} \tilde{d}[i] \oplus d(k)[i]$ from (7). We estimate the quality of $\tilde{\delta} + \epsilon$ as an upper bound for δ, when we allow for an arbitrary small buffer $\epsilon > 0$. This can easily be done by regarding $\tilde{\delta}$ as a sum of $\frac{n}{2}$ random variables

$$D[i] := \tilde{d}[i] \oplus d(k)[i].$$

Notice that $\mathbf{Pr}[D[i] = 1] = \delta$ since $d(k)$ coincides with the correct secret key d on its $\frac{n}{2}$ most significant bits. Applying Hoeffdings inequality yields

$$\mathbf{Pr}[\delta < \tilde{\delta} + \epsilon] = 1 - \mathbf{Pr}[\tilde{\delta} \leq \delta - \epsilon] \geq 1 - \exp\left(-2\epsilon^2\frac{n}{2}\right) = 1 - \exp(-n\epsilon^2),$$

i.e., for arbitrary fixed $\epsilon > 0$ and large enough n we can use $\tilde{\delta} + \epsilon$ as a reasonable upper bound for δ.

C Practical choice of t

We give a slightly refined analysis of the parameter t in order to obtain some flexibility in tuning the success probability p of Error-Correction. Therefore, we define a scaling parameter $\alpha := -5/\ln(p)$ and modify the choice of t to

$$t := \frac{\ln \alpha + \ln n - \ln \ln n + 2 \ln \epsilon}{10\epsilon^2}, \tag{15}$$

while keeping $C := 5t(\frac{1}{2} + \gamma_0)$ as proposed in Lemma 3. The following calculation which follows the proof of Lemma 5 derives a lower bound for the success probability depending on the additional parameter α.

$$\mathbf{Pr}[\text{success}] \approx (1 - \mathbf{Pr}[X_c < C])^{n/2t} \geq \left(1 - e^{-10t\epsilon^2}\right)^{n/2t} = \left(1 - \frac{\ln n}{\alpha \cdot n \cdot \epsilon^2}\right)^{n/2t}$$
$$\approx e^{-\frac{5 \ln n}{\alpha(\ln \alpha + \ln n - \ln \ln n + 2 \ln \epsilon)}} \xrightarrow{n \to \infty} e^{-5/\alpha} = p$$

The above approximation should be taken with some care for very small ϵ. Notice that our approximation gets tight for fixed ϵ and sufficiently large n. However, when we use it with realistic RSA values of $n \in [1024, .., 8192]$ the asymptotics of the Hoeffding bound do not yet apply for very small ϵ.

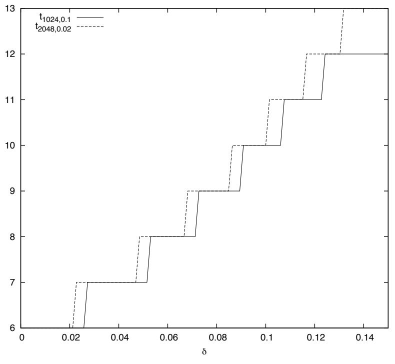

Fig. 2. Choice of $t_{n,p}$ for different n and p according to Eq. (15)

As one can see, the denominator in the exponent of e yields a non-negativity restriction $\ln n + \ln \alpha - \ln \ln n + 2 \ln \epsilon > 0$. This restriction simplifies to

$$\epsilon > \sqrt{\frac{\ln n}{n \cdot \alpha}},$$

e.g. for $n = 1024$ and $p = 0.1$ one obtains $\epsilon > 0.056$. Concerning our experiments, we modified our choice of t manually when $\delta \geq 0.18$, i.e., when ϵ fell below 0.06.