A Coding-Theoretic Approach to Recovering Noisy RSA Keys

Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn

Information Security Group, Royal Holloway, University of London

Abstract. Inspired by cold boot attacks, Heninger and Shacham (Crypto 2009) initiated the study of the problem of how to recover an RSA private key from a noisy version of that key. They gave an algorithm for the case where some bits of the private key are known with certainty. Their ideas were extended by Henecka, May and Meurer (Crypto 2010) to produce an algorithm that works when all the key bits are subject to error. In this paper, we bring a coding-theoretic viewpoint to bear on the problem of noisy RSA key recovery. This viewpoint allows us to cast the previous work as part of a more general framework. In turn, this enables us to explain why the previous algorithms do not solve the motivating cold boot problem, and to design a new algorithm that does (and more). In addition, we are able to use concepts and tools from coding theory – channel capacity, list decoding algorithms, and random coding techniques – to derive bounds on the performance of the previous and our new algorithm.

1 Introduction

Cold boot attacks [6, 7] are a class of attacks wherein memory remanence effects are exploited to extract data from a computer's memory. The idea is that modern computer memories retain data for periods of time after power is removed, so an attacker with physical access to a machine may be able to recover, for example, cryptographic key information. The time during which data is retained can be increased by cooling the memory chips. However, because the memory gradually degrades over time once power is removed, only a noisy version of the data may be recoverable. The question then naturally arises: given a noisy version of a cryptographic key, is it possible to reconstruct the original key?

This question was addressed for broad classes of cryptosystems, both symmetric and asymmetric, by Halderman *et al.* in [6, 7] and specifically for RSA private keys in [8, 9]. Similar problems arise in the context of side-channel analysis of cryptographic implementations, where noisy key information may leak through power consumption [11] or timing [2]. The question is also linked to the classical cryptanalysis problem of recovering an RSA private key when some bits of the key are known, for example the most or least significant bits, or contiguous bits spread over a number of blocks (see, for example, the surveys in [1, 12] and [10]).

X. Wang and K. Sako (Eds.): ASIACRYPT 2012, LNCS 7658, pp. 386–403, 2012.

Heninger and Shacham (HS) [9] considered the setting where a random fraction of the RSA private key bits is known with certainty. Their approach exploits the fact that the individual bits of an RSA private key of the form $\mathsf{sk} = (p, q, d, d_p, d_q)$ must satisfy certain algebraic relations. This enables the recovery of the private key in a bit-by-bit fashion, starting with the least significant bits, by growing a search tree. It is easy to prune the search tree to remove partial solutions which do not match with the known key bits. The resulting algorithm will always succeed in recovering the private key, since the pruning process will never remove a partial version of the correct solution. On the other hand, when only few bits are known, the search tree may grow very large, and the HS algorithm will blow up. It was proved in [9] that, under reasonable assumptions concerning randomness of incorrect solutions, the HS algorithm will efficiently recover an n-bit RSA private key in time $O(n^2)$ with probability $1-1/n^2$ when a random fraction of at least 0.27 of the private key bits are known with certainty. These theoretical results are well-matched by experiments reported in [9]. These experiments also confirm that the HS algorithm has good performance when the known fraction is as small as 0.24, and the analysis of [9] extends to cases where the RSA private key sk is of the form (p, q, d) or (p, q).

Henecka, May and Meurer (HMM) [8] took the ideas of [9] and developed them further to address the situation where no key bits are known with certainty. They consider the *symmetric* case where the two possible bit flips $0 \to 1$, $1 \to 0$ have equal probability δ. Their main idea was to consider t bit-slices at a time of possible solutions to the equations relating the bits of sk, instead of single bits at a time as in the HS algorithm. In the formulation where $\mathsf{sk} = (p, q, d, d_p, d_q)$, this yields 2^t candidate solutions on $5t$ new private key bits for each starting candidate at each stage of the algorithm. The HMM algorithm then computes the Hamming distance between the candidate solutions and the noisy key, keeping all candidates for which this metric is less than some carefully chosen threshold C. This replaces the procedure of looking for exact matches used in the HS algorithm. Of course, now the correct solution may fail this statistical test and be rejected; moreover the number of candidate solutions retained may explode if C is set too loosely. Nevertheless, it was shown in [8] that the HMM algorithm is efficient and has reasonable success in outputting the correct solution provided that $\delta < 0.237$. Again, the analysis depends on assumptions concerning the random behaviour of wrong solutions. To support the analysis, [8] reports the results of experiments for different noise levels and algorithmic parameters. For example, the algorithm can cope with $\delta = 0.20$.

In recent work independent of ours, Sarkar and Maitra [13] revisited the work of [8], applying the HMM algorithm to break Chinese Remainder implementations of RSA with low weight decryption exponents and giving *ad hoc* heuristics to improve the algorithm.

Limitations of Previous Work and Open Questions: Although inspired by cold boot attacks, it transpires that neither the HS algorithm nor the HMM algorithm actually solve the motivating cold boot problem. Let us see why.

One observation made in [6,7] is that for a given region of memory, the decay of memory bits is overwhelmingly either $0 \to 1$ or $1 \to 0$, while the decay direction in a given region can be inferred by comparing the number of 0s and 1s (since for an uncorrupted private key, we expect these to be roughly equal). Thus, in a $1 \to 0$ region, a 1 bit in the noisy version of the key is known (with high probability) to correspond to a 1 bit in the original key.

In the case of [9], the assumption is made that a certain fraction of the RSA private key bits – both 0s and 1s – is known with certainty. But, in the cold boot scenario, only 1 (or 0) bits are known, and not a mixture of both. Fortunately, the authors of [9] have informed us that their algorithm does still work when only 0 or only 1 bits are known, but this is not the case it was designed for, and, formally, the performance guarantees obtained in [9] do not apply in this case. Furthermore, in a real cold boot attack, bits are never known with *absolute* certainty, because even in a $1 \to 0$ region, say, bit flips in the reverse direction can occur. Halderman *et al.* report rates of 0.05% to 0.1% for this event. Such an event will completely derail the HS algorithm, as it will result in the correct solution being eliminated from the search tree. Based on an occurrence rate of 0.1%, this kind of fatal event can be expected to arise around 2.5 to 5 times in a real key recovery attack for 1024-bit RSA moduli with $\mathsf{sk} = (p, q, d, d_p, d_q)$. Thus, the HS algorithm really only applies to an "idealised" cold boot setting, where some bits are known for sure.

The HMM algorithm is designed to work for the symmetric case where the two possible bit flips have equal probability δ. Yet, in a cold boot attack, in a $1 \to 0$ region say, $\alpha := \Pr(0 \to 1)$ will be very small (though non-zero), while $\beta := \Pr(1 \to 0)$ may be relatively large, and perhaps even greater than 0.5 in a very degraded case. The use of Hamming distance as a metric for comparison and the setting of the threshold C are closely tied to the symmetric case, and it is not immediately clear how one can generalise the HMM approach to handle the type of errors occurring in real cold boot attacks. So it does not solve the cold boot problem for RSA keys.

Intriguing features of the work in [8,9] are the constants 0.27 and 0.237, which bound the fraction of known bits/noise rate the HS and HMM algorithms can handle. One can trace through the relevant analysis to see how these numbers emerge, but it would be more satisfying to have a deeper, unifying explanation. One might also wonder if these bounds are best possible or whether significant improvements might yet be forthcoming. Is there any ultimate limit to the noise level that these kinds of algorithms can deal with? And can we design an algorithm that works in the true cold boot setting, or for fully general noise models that might be expected to occur in other types of side channel attack?

Our contributions: We show how to recast the problem of noisy RSA key recovery as a problem in coding theory. That such a connection exists should be no surprise: after all, we are in a situation where bits are only known with certain probabilities and we wish to recover the true bits. However, this connection opens up the opportunity to apply to our problem the full gamut of sophisticated tools

that have been developed by coding theorists over the last 60 years. We sketch this connection and its main consequences next.

Recall that in the HMM algorithm, we generate from each solution so far a set of 2^t candidate solutions on $5t$ new bits. We now view the set of 2^t candidates as being a *code*, with one codeword s (representing bits of the true private key) being selected and transmitted over a noisy channel, resulting in a received word r (representing $5t$ bits of the noisy version of the key). In the HMM case, the noise is realised via bit flipping with probability δ. The HS algorithm can be seen as arising from the special case $t = 1$, where the noise now corresponds to erasing a fraction of key bits instead of flipping them. Alternatively, we can consider a generalisation of the HS algorithm which considers $5t$ bits at a time, generated just as in the HMM algorithm, and which then filters the resulting 2^t candidates based on matching with known key bits. Because filtering is based on exact matching, this algorithm has the same output as the original HS algorithm. This brings the two algorithms under a single umbrella.

In general, in coding theory, the way in which s is transformed into r depends on the *channel model*, which in its full generality defines the probabilities $\Pr(\mathsf{r}|\mathsf{s})$ over all possible pairs (s, r). In the case of [9], the assumption is that particular bits are known with certainty and others are not known at all, with the bits all being treated independently. The appropriate channel model is then an *erasure* channel, meaning that bits are independently either erased or transmitted correctly over the channel, with the receiver knowing the positions of the erasures. In the case of [8], the appropriate channel model is the binary symmetric channel with cross-over probability δ. It also emerges that the appropriate channel model for the true cold boot setting is a binary *non-symmetric* channel with cross-over probabilities (α, β). In general, the problem we are faced with is to decode r, with the aim being to reproduce s with high probability.

When couched in this language, it becomes obvious that the HS and HMM algorithms do not solve the original cold boot problem – simply put these algorithms use inappropriate channel models for that specific problem. We can also use this viewpoint to derive limits on the performance of *any* procedure for selecting which candidate solutions to keep in an HMM-style algorithm. To see why, we recall that the converse to Shannon's noisy-channel coding theorem [14] states that *no* combination of code and decoding procedure can jointly achieve arbitrarily reliable decoding when the code rate exceeds the (Shannon) capacity of the channel. Moreover, there are analogues of the converse of Shannon's theorem for so-called *list decoding* that essentially show that channel capacity is also the barrier to any efficient algorithm outputting lists of candidates, as the HS and HMM algorithms do.

When sk is of the form (p, q, d, d_p, d_q), for example, the code rate is fixed at $1/5$ (we have 2^t codewords and length $5t$). The channel capacity can be calculated as a function of the channel model and its parameters. For example, for the erasure channel with erasure probability ρ (meaning that a fraction $1 - \rho$ of the bits are known with certainty), the capacity is simply $1 - \rho$. Then we see that the limiting value is $\rho = 0.8$, meaning that the fraction of known bits must be

at least 0.2 to achieve arbitrarily reliable, efficient decoding. The analysis in [9] needs that fraction to be at least 0.27, though a fraction as low as 0.24 could be handled in practice. Thus a capacity analysis suggests that there should be room to improve the HS algorithm further, but capacity shows that it is impossible go below a fraction 0.2 of known bits with an efficient algorithm. See Section 3 for further details on list decoding and its application to the analysis of the HS and HMM algorithms.

Informed by our coding-theoretic viewpoint, we derive a new key recovery algorithm that works for any (memoryless) binary channel and therefore *is* applicable to the cold boot setting (and more). In essence, we modify the HMM algorithm to use a likelihood statistic in place of the Hamming metric when selecting from the candidate codewords. We keep the L codewords having the highest values of this likelihood statistic and reject the others. An important consequence of this algorithmic choice is that our algorithm has *deterministic* running time $O(L2^t n/t)$ and, when implemented using a stack, *deterministic* memory consumption $O(L+t)$. This stands in contrast to the running time and memory usage of the HS and HMM algorithms, which may blow up when the erasure/error rates are high. We note that private RSA keys are big enough that they may cross regions when stored in memory. We can handle this by changing the likelihood statistic used in our algorithm at the appropriate transition points, requiring only a simple modification to our approach. In the full version, we give an analysis of the success probability of our new algorithm, under different randomness hypotheses, using coding-theoretic tools. Essentially, we are able to show that, as $t \to \infty$, its success probability tends to 1 provided the code rate ($1/5$ when $\mathsf{sk} = (p, q, d, d_p, d_q)$) remains below the channel capacity. Moreover, from the converse to Shannon's theorem, we are unlikely to be able to improve this result if reliable key recovery is required.

We include the results of extensive experiments using our new algorithm. These demonstrate that our approach matches or outperforms the HS and HMM algorithms in the cases they are designed for, and achieves results close to the limits imposed by our capacity analysis more generally. For example, in the symmetric case with $\delta = 0.20$, we can achieve a 20% success rate in recovering keys for $t = 18$ and $L = 32$. This is comparable to the results of [8]. Furthermore, for the same t and L we achieve a 4% success rate for $\delta = 0.22$, whilst [8] does not report any experiments for an error rate this high. As another example, our algorithm can handle the idealised cold boot scenario by setting $\alpha = 0$ (in which case all the 1 bits in r are known with certainty, i.e. we are in a $1 \to 0$ region). Here, our capacity analysis puts a bound of 0.666 on β for reliable key recovery. Using our algorithm, we can recover keys for $\beta = 0.6$ with a 13% success rate using $t = 18$ and $L = 32$, whereas the HS algorithm can only reach $\beta = 0.52$ (and this under the assumption that the experimental results reported in [9] for a mixture of known 0 and 1 bits do translate to the same performance for the case where only 1 bits are known). In the same setting, we can even recover keys up to $\beta = 0.63$ with a non-zero success rate. We also have similar experimental

results for the 'true' cold boot setting where both α and β are non-zero, and for the situation where sk is of the form (p, q, d) or (p, q).

Paper Organisation: The remainder of this paper is organised as follows. In the next section, we give further background on the algorithms of [8,9]. In Section 3, we develop the connection with coding theory and explain how to use it to derive limits on the performance of noisy RSA key recovery algorithms. Section 4 describes our new maximum likelihood list decoding algorithm. Section 5 presents our experimental results. Finally, Section 6 contains some closing remarks and open problems.

2 The HS and HMM Algorithms

Let (N, e) be the RSA public key, where $N = pq$ is an n-bit RSA modulus, and p, q are balanced primes. As with [8, 9], we assume throughout that e is small, say $e = 3$ or $e = 2^{16} + 1$; for empirical justification of this assumption, see [15]. We start by assuming that private keys sk follow the PKCS#1 standard and so are of the form $(N, p, q, e, d, d_p, d_q, q_p^{-1})$, where d is the decryption key, $d_p = d \mod p - 1$, $d_q = d \mod q - 1$ and $q_p = q^{-1} \mod p$. However, neither the algorithms of [8, 9] nor ours make use of q_p^{-1}, so we henceforth omit this information. Furthermore, we assume N and e are publicly known, so we work only with the tuple $\mathsf{sk} = (p, q, d, d_p, d_q)$. We will also consider attacks where the private key contains less information – either $\mathsf{sk} = (p, q, d)$ or $\mathsf{sk} = (p, q)$.

Now assume we are given a degraded version of the key $\widetilde{\mathsf{sk}} = (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d}_p, \tilde{d}_q)$. We start with the four RSA equations:

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

where k, k_p and k_q are integers to be determined. A method for doing so is given in [9]: first it is shown that $0 < k < e$; then, since e is small, we may enumerate

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

for all $0 < k' < e$. We then find the k' such that $d(k')$ is "closest" to $\tilde{d}$ in the most significant half of the bits. Simple procedures for doing this are given in [8, 9]. In the more general setting where bit flips can occur in both directions and with different probabilities, we proceed as follows. First, we estimate parameters $\alpha = \Pr(0 \to 1)$ and $\beta = \Pr(1 \to 0)$ from known bits, e.g. from a noisy version of N that is adjacent in memory to the private key. Second, we compute for each k' an approximate log-likelihood using the expression

$$n_{01} \log \alpha + n_{00} \log(1 - \alpha) + n_{10} \log \beta + n_{11} \log(1 - \beta)$$

where n_{01} is the number of positions in the most significant half where a 0 appears in $d(k')$ and a 1 appears in $\tilde{d}$, etc. Finally, we select the k' that provides the highest log-likelihood.

At the end of this procedure, with high probability we will have $k' = k$ and we will have recovered the most significant half of the bits of d. Now we wish to find k_p and k_q. By manipulating the above equations we see that

$$k_p^2 - (k(N-1)+1)k_p - k \equiv 0 \mod e$$

If e is prime (as in the most common case $e = 2^{16}+1$) there will only be two solutions to this equation. One will be k_p and the other k_q. If e is not prime we will have to try all possible pairs of solutions in the remainder of the algorithm.

Now, for integers x, we define $\tau(x) := \max\{i \in \mathbb{N} : 2^i \mid x\}$. Then it is easy to see that $2^{\tau(k_p)+1}$ divides $k_p(p-1)$, $2^{\tau(k_q)+1}$ divides $k_q(q-1)$ and $2^{\tau(k)+2}$ divides $k\phi(N)$. These facts, along with relations (2) – (4), allow us to see that

$$\begin{aligned} d_p &\equiv e^{-1} \mod 2^{\tau(k_p)+1} \\ d_q &\equiv e^{-1} \mod 2^{\tau(k_q)+1} \\ d &\equiv e^{-1} \mod 2^{\tau(k)+2}. \end{aligned}$$

This allows us to correct the least significant bits of d, d_p and d_q. Furthermore we can calculate $\mathsf{slice}(0)$, where we define

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

with $x[i]$ denoting the i-th bit of the string x.

Now we are ready to explain the main idea behind the algorithm of [9]. Suppose we have a solution (p', q', d', d'_p, d'_q) from $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$. Then [9] uses a multivariate version of Hensel's Lemma to show that the bits involved in $\mathsf{slice}(i)$ must satisfy the following congruences:

$$\begin{aligned} p[i] + q[i] &= (N - p'q')[i] \mod 2 \\ d[i+\tau(k)] + p[i] + q[i] &= (k(N+1) + 1 - k(p'+q') - ed')[i+\tau(k)] \mod 2 \\ d_p[i+\tau(k_p)] + p[i] &= (k_p(p'-1) + 1 - ed'_p)[i+\tau(k_p)] \mod 2 \\ d_q[i+\tau(k_q)] + q[i] &= (k_q(q'-1) + 1 - ed'_q)[i+\tau(k_q)] \mod 2. \end{aligned}$$

Because we have 4 constraints on 5 unknowns, there are exactly 2 possible solutions for $\mathsf{slice}(i)$, rather than 32. This is then used in [9] as the basis of building a search tree for the unknown private key bits. At each node in the tree, representing a partial solution up to $\mathsf{slice}(i-1)$, at most two successor nodes are added by the above procedure. Moreover, since a random fraction of the bits is assumed to be known with certainty, the tree can be pruned of any partial solutions that are not consistent with these known bits. Clearly, if the fraction of known bits is large enough, then the tree will be highly pruned and the number of nodes in the tree will be small. The analysis of [9] shows that if the fraction of known bits is at least 0.27, then the tree's size remains close to linear in n, the

size of the RSA modulus, meaning that an efficient algorithm results. A similar algorithm and analysis can be given for the case where sk is of the form (p, q, d) or (p, q); in each case, there are exactly 2 possible solutions for each $\mathsf{slice}(i)$.

Instead of doing Hensel lifting bit-by-bit and pruning on each bit, the HMM algorithm performs t Hensel lifts for some parameter t, yielding, for each surviving candidate solution on $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$, a tree of depth t whose 2^t leaf nodes represent candidate solutions on slices $\mathsf{slice}(0)$ to $\mathsf{slice}(i+t-1)$, involving $5t$ new bits (in $\mathsf{slice}(i)$ to $\mathsf{slice}(i+t-1)$). A solution is kept for the next iteration if the Hamming distance between the $5t$ new bits and the corresponding vector of noisy bits is less than some threshold C. Clearly the HS algorithm could also be modified in this way, lifting t times and then doing pruning based on matching known key bits. Alternatively, one can view the HS algorithm as being the special case $t = 1$ of the HMM algorithm (with a different pruning procedure). The HMM algorithm can also be adapted to work with sk of the form (p, q, d) or (p, q). Henecka *et al.* [8] showed how to select C and t so as to guarantee that their algorithm is efficient and produces the correct solution with a reasonable success rate. In particular, they were able to show that this is the case provided the probability of a bit flip δ is at most 0.237.

At each stage in the HMM algorithm, candidate solutions on t new slices are constructed. Then roughly $n/2t$ iterations or stages of the algorithm are needed, since all the quantities being recovered contain at most $n/2$ bits. As pointed out in [8], only half this number of stages is required since once we have the least significant half of the bits of the private key, the entire private key can be recovered using a result of Coppersmith [3]. At their conclusion, the HS and HMM algorithms outputs lists of candidate solutions rather than a single solution. But it is easy to verify the correctness of each candidate by using a trial encryption and decryption, say. Thus the success rate of the algorithms is defined to be the probability that the correct solution is on the output list. We adopt the same measure of success in the remainder of the paper.

3 The Coding-Theoretic Viewpoint

In this section, we develop our coding-theoretic viewpoint on the HS and HMM algorithms, using it to derive limits on the performance of these and similar algorithms. In particular, we will explain how channel capacity plays a crucial role in setting these limits.

We begin by defining the parameter m. We set $m = 5$ when $\mathsf{sk} = (p, q, d, d_p, d_q)$, $m = 3$ when $\mathsf{sk} = (p, q, d)$, and $m = 2$ when $\mathsf{sk} = (p, q)$. Consider a stage of the HMM algorithm, commencing with M partial solutions that have survived the previous stage's pruning step. The HMM algorithm produces a total of $M2^t$ candidate solutions on mt bits, prior to pruning. We label these $\mathsf{s}_1, \ldots, \mathsf{s}_{M2^t}$, let $\mathcal{C}$ denote the set of all $M2^t$ candidates, and use r to denote the corresponding vector of mt noisy bits in sk.

Now we think of $\mathcal{C}$ as being a code. This code has rate $R \geq 1/m$, but its other standard parameters such as its minimum distance are unknown (and

immaterial to our analysis). The problem of recovering the correct candidate s_j given r is clearly just the problem of decoding this code. Now both the HS and HMM algorithms have pruning steps that output lists of candidates for the correct solution, with the list size being dynamic in both cases and depending on the number of candidates surviving the relevant filtering process (based either on exact matches for the HS algorithm or on Hamming distance for the HMM algorithm). In this sense, the HS and HMM algorithms are performing types of *list decoding*, an alternative to the usual unique decoding of codes that was originally proposed by Elias [4].

To complete the picture, we need to discuss what error and channel models are used in [8,9], and what models are appropriate to the cold boot setting. As noted in the introduction, [9] assumes that some bits of r are known exactly, while no information at all is known about the other bits. This corresponds to an *erasure* model for errors, and an *erasure* channel. Usually, this is defined in terms of a parameter ρ representing the fraction of erasures. So $1-\rho$ represents the fraction of known bits, a parameter denoted δ in [9]. On the other hand, [8] assumes that all bits of r are obtained from the correct s_j by independent bit flipping with probability δ. In standard coding terminology, we have a (memoryless) binary symmetric channel with crossover probability δ. From the experimental data reported in [6, 7], an appropriate model for the cold boot setting would be a binary non-symmetric channel with crossover probabilities (α, β), with α being small and β being significantly larger in a $1 \to 0$ region (and vice-versa in a $0 \to 1$ region). In an idealised cold boot case, we could assume $\alpha = 0$, meaning that a $0 \to 1$ bit flip can never occur, so that all 1 bits in r are known with certainty. This is better known as a Z-channel in the coding-theoretic literature.

This viewpoint highlights the exact differences between the settings considered in [8, 9] and the cold boot setting. It also reveals that, while the HS algorithm can be applied for the Z-channel seen in the idealised cold boot setting, there is no guarantee that the performance proven for it in [9] for the erasure channel will transfer to the Z-channel. Moreover, one might hope for substantial improvements to the HS algorithm if one could somehow take into account the (partial) information known about 0 bits as well as the exact information known about 1 bits.

3.1 The Link to Channel Capacity

We can use this coding viewpoint to derive limits on the performance of *any* procedure for selecting which candidate solutions to keep in the HS and HMM algorithms. To see why, we recall that the converse to Shannon's noisy-channel coding theorem [14] states that *no* combination of code and decoding procedure can jointly achieve arbitrarily reliable decoding when the code rate exceeds the capacity of the channel. Our code rate is at least $1/m$ where $m = 2, 3$ or 5 and the channel capacity can be calculated as a function of the channel model and its parameters.

Two *caveats* must be made here. Firstly, capacity only puts limits on *reliable* decoding, and even decoding with low success probability is of interest in

cryptanalysis. Secondly, Shannon's result applies only to decoding algorithms that output a single codeword s, while both the HS and HMM algorithms are permitted to output many candidates at each stage, with the final output list only being required to contain the correct private key. Perhaps such list-outputting algorithms can surpass the bounds imposed by Shannon's theorem? Indeed, the HS algorithm is guaranteed to output the correct key provided the algorithm terminates. Similarly, the threshold C in the HMM algorithm can always be set to a value that ensures that every candidate passes the test and is kept for the next stage, thus guaranteeing that the algorithm is always successful. However, neither of these variants would be *efficient* and in fact there are analogues of the converse of Shannon's noisy-channel coding theorem that essentially show that capacity is the barrier for efficient list decoding too.

For the binary symmetric channel, it is shown in [5, Theorem 3.4] that if $\mathcal{C}$ is *any* code of length n and rate $1 - H_2(\delta) + \epsilon$ for some $\epsilon > 0$, then some word r is such that the Hamming sphere of radius δn around r contains at least $2^{\epsilon n/2}$ codewords. Here $H_2(\cdot)$ is the binary entropy function:

$$H_2(x) = -x \log_2(x) - (1 - x) \log_2(1 - x)$$

and $1 - H_2(\delta)$ is just the capacity of the channel. The proof also shows that, over a random choice of r, the average number of codewords in a sphere of radius δn around r is $2^{\epsilon n/2}$. Since the expected number of errors in r is δn, we expect the correct codeword to be in this sphere, along with $2^{\epsilon n/2}$ other codewords. This implies that, if the rate of the code exceeds the channel capacity $1 - H_2(\delta)$ by a constant amount ϵ, then $\mathcal{C}$ cannot be list decoded using a polynomial-sized list, either in the worst case or on average, as $n \to \infty$.

An analogous result can be proved for the erasure channel, based on a similarly simple counting argument as was used in the proof of [5, Theorem 3.4]: if ρ is the erasure probability and $\mathcal{C}$ is any code of rate $1 - \rho + \epsilon$ (i.e. ϵ above the erasure channel's capacity), then it can be shown that on average there will be $2^{\epsilon n}$ codewords that differ from r in its erasure positions, assuming r contains ρn erasure symbols. Hence reliable list decoding for $\mathcal{C}$ cannot be achieved using a polynomial-sized list.

In the next sub-section, we will examine in more detail the implications of these results on list decoding for the HS and HMM algorithms.

3.2 Implications of the Capacity Analysis

The Binary Symmetric Channel and the HMM Algorithm. If the HMM algorithm is to have reasonable success probability in recovering the key, then at each stage, it must set the threshold C in such a way that all words $\mathsf{s}_i \in \mathcal{C}$ with $d_H(\mathsf{s}_i, \mathsf{r}) \approx \delta mt$ are accepted by the algorithm. This is because δmt is the expected number of errors occurring in r, and if the threshold is set below this value, then the correct codeword is highly likely to be rejected by the algorithm. (In fact, the HMM algorithm sets C to be slightly higher than this, which makes good sense given that there is an even chance of there being more than δmt

errors.) Recall that we have rate $R \geq 1/m$. Now suppose δ is such that $R = 1 - H_2(\delta) + \epsilon$ for some $\epsilon > 0$, i.e. δ is chosen so that that the code rate is just above capacity. Then the argument above shows that there will be on average at least $2^{\epsilon mt/2}$ codewords on the output list at each stage. Thus, as soon as δ is such that R exceeds capacity by a constant amount ϵ, then there must be a blow-up in the algorithm's output size at each stage, and the algorithm will be inefficient asymptotically.

We write $C_{\mathrm{BSC}}(\delta) = 1 - H_2(\delta)$ for the capacity of the binary symmetric channel. Table 1 shows that $C_{\mathrm{BSC}}(\delta) = 0.2$ when $\delta = 0.243$. Thus what our capacity analysis shows is that the best error rate one could hope to deal with in the HMM algorithm when $m = 5$ is $\delta = 0.243$. Notice that this value is rather close to, but slightly higher than, the corresponding value of 0.237 arising from the analysis in [8]. The same is true for the other entries in this table. This means that significantly improving the theoretical performance of the HMM algorithm (or indeed any HMM-style algorithm) whilst keeping the algorithm efficient will not be possible. The experimental work in [8] gives results up to a maximum δ of 0.20; compared to the capacity bound of 0.243, it appears that there is some room for practical improvement in the symmetric case.

The Erasure Channel and the HS Algorithm. As noted above, for the erasure channel, the capacity is $1 - \rho$, where ρ is the fraction of bits erased by the channel. Note that the list output by the HS algorithm is independent of whether pruning is done after each lift or in one pass at the end (but obviously doing so on a lift-by-lift basis is more efficient in terms of the total number of candidates examined). Then considering the HS algorithm in its entirety (i.e. over $n/2$ Hensel lifts), we see that it acts as nothing more than a list decoder for the erasure channel, with the code $\mathcal{C}$ being the set of all $2^{n/2}$ words on $mn/2$ bits generated by doing $n/2$ Hensel lifts without any pruning, and the received word r being the noisy version of the entire private key sk.

Then our analysis above applies to show that the HS algorithm will produce an exponentially large output list, and will therefore be inefficient, when the rate (which in this case is exactly $1/m$) exceeds the capacity $1 - \rho$. For $m = 5$, we have rate 0.2 and so our analysis shows that the HS algorithm will produce an exponentially large output list whenever ρ exceeds 0.8. Now [9] reports good results (in the sense of having a reasonable running time) for ρ as high as 0.76 (corresponding to Heninger and Shacham's parameter δ being equal to 0.24),

Table 1. Private key-type, equivalent rate R, and maximum crossover probability δ allowing reliable key recovery, symmetric channel case

sk	R	δ
(p, q, d, d_p, d_q)	1/5	0.243
(p, q, d)	1/3	0.174
(p, q)	1/2	0.110

Table 2. Private key-type, equivalent rate R, and maximum error probability ρ allowing reliable key recovery, Z-channel case

sk	R	β
(p, q, d, d_p, d_q)	1/5	0.666
(p, q, d)	1/3	0.486
(p, q)	1/2	0.304

leaving a gap between the experimental performance and the theoretical bound. Similar remarks apply for the cases $m = 2, 3$: for $m = 2$, the HS algorithm should be successful for $\rho = 0.43$ ($\delta = 0.57$), while the bound from capacity is 0.50; for $m = 3$, we have $\rho = 0.58$ ($\delta = 0.42$) and the capacity bound is 0.67. Hence, further improvements for $m = 2, 3$ are not ruled out by the capacity analysis.

The Z-channel. We may also apply the above capacity analysis to the idealised cold boot setting, where the crossover probabilities are of the form $(0, \beta)$. Here we have a Z-channel, whose capacity can be written as:

$$C_{\mathrm{Z}}(\beta) = \log_2(1 + (1 - \beta)\beta^{\frac{\beta}{1-\beta}}).$$

Solving the equation $C_{\mathrm{Z}}(\beta) = R$ for $R = 1/5, 1/3, 1/2$ gives us the entries in Table 2. We point out the large gap between these figures and what we would expect to obtain both theoretically and experimentally if we were to directly apply the HS algorithm to the idealised cold boot setting. For example, when $m = 5$, the analysis of [9] suggests that key recovery should be successful provided that β does not exceed 0.46 (the value of $\delta = 0.27$ translates into a β value of 0.46 using the formula $\delta = (1 - \beta)/2$ given in [9]), whereas the capacity analysis suggests a maximum β value of 0.666. This illustrates that the HS algorithm is not well-matched to the Z-channel. Our new algorithm will close this gap substantially.

The True Cold Boot Setting. For the true cold boot setting, we must consider the general case of a memoryless, binary channel with crossover probabilities (α, β). We can calculate the capacity $C(\alpha, \beta)$ of this channel and obtain the regions for which $C(\alpha, \beta) > R$ for $R = 1/5, 1/3, 1/2$. The results are shown in Figure 1. Notice that these plots include as special cases the data from Tables 1 and 2. If we set $\alpha = 0.001$, say, we see that the maximum achievable β is quite close to that in the idealised cold boot setting. Note also that the plots are symmetric about the lines $y = x$ and $y = 1 - x$, reflecting the fact that capacity is preserved under the transformations $(\alpha, \beta) \to (\beta, \alpha)$ and $(\alpha, \beta) \to (1 - \alpha, 1 - \beta)$.

However, we must caution that capacity-based bounds for list decoding for the general binary non-symmetric channel (including the Z-channel) are not known in the coding-theoretic literature. Strictly speaking, then, our capacity analysis for this case does not bound the performance of key recovery algorithms that are allowed to output many key candidates, but only the limited class of algorithms that output a *single* key candidate. This said, our capacity analysis sets a target for our new algorithm, which follows.

4 The New Algorithm and Its Analysis

In this section, we give our new algorithm for noisy RSA key recovery that works for any memoryless, binary channel, as characterised by the cross-over probabilities (α, β). Our algorithm has the same basic structure as the HMM

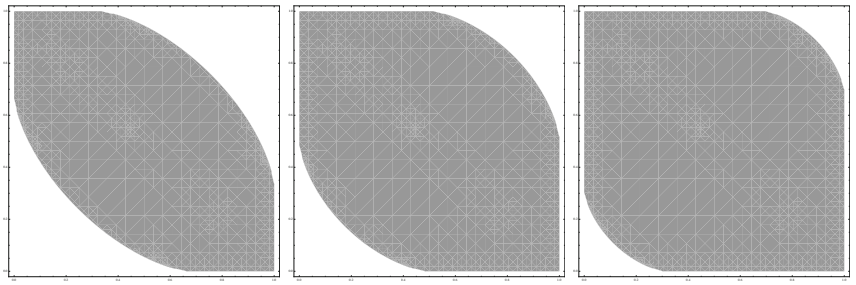

Fig. 1. Plots showing achievable (α, β) pairs for private keys containing 5, 3 and 2 components, respectively. The vertical axis is β, the horizontal axis is α. The shaded area in each case represents the unachievable region.

algorithm but uses a different procedure to decide which candidate solutions to retain and which to reject. Specifically, we use a likelihood measure in place of Hamming distance.

Recall that we label the $M2^t$ candidate solutions on mt bits arising at some stage in the HMM algorithm $\mathsf{s}_1, \ldots, \mathsf{s}_{M2^t}$ and let us name the corresponding vector of mt noisy bits in the RSA private key r. Then the Maximum Likelihood (ML) estimate for the correct candidate solution is simply:

$$\arg \max_{1 \le i \le M2^t} \Pr(\mathsf{s}_i | \mathsf{r}).$$

that is, the choice of i that maximises the conditional probability $\Pr(\mathsf{s}_i|\mathsf{r})$. Using Bayes' theorem, this can be rewritten as:

$$\arg \max_{1 \le i \le M2^t} \frac{\Pr(\mathsf{r}|\mathsf{s}_i)\Pr(\mathsf{s}_i)}{\Pr(\mathsf{r})}.$$

Here, $\Pr(\mathsf{r})$ is a constant for a given set of bits r. Let us make the further mild assumption that $\Pr(\mathsf{s}_i)$ is also a constant, independent of i. Then the ML estimate is obtained from

$$\arg \max_{1 \le i \le M2^t} (\Pr(\mathsf{r}|\mathsf{s}_i)) = \arg \max_{1 \le i \le M2^t} \left((1-\alpha)^{n_{00}^i} \alpha^{n_{01}^i} (1-\beta)^{n_{11}^i} \beta^{n_{10}^i} \right)$$

where $\alpha = \Pr(0 \to 1)$ and $\beta = \Pr(1 \to 0)$ are the crossover probabilities, n_{00}^i denotes the number of positions where s_i and r both have 0 bits, n_{01}^i denotes the number of positions where s_i has a 0 and r has a 1, and so on.

Equivalently, we may maximise the log of these probabilities, and so we seek:

$$\begin{aligned} &\arg \max_{1 \le i \le M2^t} (\log \Pr(\mathsf{r}|\mathsf{s}_i)) \\ &\quad = \arg \max_{1 \le i \le M2^t} \left(n_{00}^i \log(1-\alpha) + n_{01}^i \log \alpha + n_{11}^i \log(1-\beta) + n_{10}^i \log \beta \right) \end{aligned}$$

which provides us with a simpler form for computational purposes.

Algorithm 1. Pseudo-code for the maximum likelihood list decoding algorithm for reconstructing RSA private keys.

```
list ← slice(0);
for stage = 1 to n/2t do
    Replace each entry in list with a set of 2^t candidate solutions obtained by
    Hensel lifting;
    Calculate the log-likelihood log Pr(r|s_i) for each entry s_i on list;
    Keep the L entries in list having the highest log-likelihoods and delete the
    remainder;
Output list;
```

Then our proposed algorithm is simply this: select at each stage from the candidates generated by Hensel lifting those L candidates s_i which produce the highest values of the log-likelihood $\log \Pr(\mathsf{r}|\mathsf{s}_i)$ as in the equation above. These candidates are then passed to the next stage. So at each stage except the first we will generate a total of $L2^t$ candidates and keep the best L. We may then test each entry in the final list by trial encryption and decryption to recover a single candidate for the private key. Pseudo-code for this algorithm is shown in Algorithm 1. Note that here we assume there are $n/2t$ stages; this number can be halved as in the HS and HMM algorithms.

Our algorithm has fixed running time $O(L2^t)$ for each of the $n/2t$ stages, and fixed memory consumption $O(L2^t)$. This is a consequence of choosing to keep the L best candidates at each stage in place of all candidates surpassing some threshold as in the HMM algorithm. The memory consumption can be reduced to $O(L+t)$ by using a depth-first approach to generating and filtering the candidates. The main overhead is then the Hensel lifting to generate candidate solutions; the subsequent computation of log-likelihoods for each candidate is relatively cheap. Notice that if $\alpha = 0$ (as in the Z-channel for an idealised cold boot setting), then any instance of a $0 \to 1$ bit flip is very heavily penalised by the log-likelihood statistic – it adds a $-\infty$ term to $\log \Pr(\mathsf{r}|\mathsf{s}_i)$. In practice, for $\alpha = 0$, we just reject any solution containing a $0 \to 1$ transition. For the erasure channel, we reject any candidate solution that does not match r in the known bits.

A special case of our algorithm arises when $L = 1$ and corresponds to just keeping the single ML candidate at each stage. This algorithm then corresponds to Maximum Likelihood (ML) decoding. However, at a given stage, it is likely that the correct solution will be rejected because a wrong solution happens to have the highest likelihood. This is especially so in view of how similar some candidates will be to the correct solution. Therefore, ML decoding is likely to go awry at some stage of the algorithm.

4.1 Remarks on the Asymptotic Analysis of Our Algorithm

In the full version, we give two analyses of our algorithm, using tools from coding theory to assist us. The first analysis uses a strong randomness assumption, that

Table 3. Success probabilities for the symmetric case ($(\alpha, \beta) = (\delta, \delta)$). Experiments with $\delta \leq 0.16$ are based on 500 trials. Capacity bound on δ is 0.243.

δ	0.08	0.10	0.12	0.14	0.16	0.18	0.19	0.2	0.21	0.22
t	6	8	10	12	16	18	18	18	18	18
L	4	4	8	32	32	32	32	32	32	64
Success rate	1	0.921	0.932	0.963	0.84	0.60	0.38	0.20	0.08	0.04
Time per trial (ms)	113	98	474	4323	85662	395069	399451	380139	377342	722341

the $L2^t$ candidates s_i generated at each stage of Algorithm 1 are independent and uniformly random mt-bit vectors. It shows that, asymptotically, our algorithm will be successful in recovering the RSA private key provided $1/m$ is less than the capacity of the memoryless, binary channel with crossover probabilities (α, β). In fact, this result follows as a simple application of Shannon's noisy-channel coding theorem [14], which states that, asymptotically, the use of random codes in combination with Maximum Likelihood (ML) decoding achieves arbitrarily small decoding error probability, provided that the code rate stays below the capacity of the channel. Unfortunately, it is easy to see that our strong randomness assumption is in fact *not* true for the codes $\mathcal{C}$ generated in our algorithm, because of the iterative nature of the Hensel lifting. The second analysis proves a similar result for the symmetric case under weaker randomness assumptions for which we have good experimental evidence. Details can be found in the full version.

5 Experimental Results

For our experiments, we used a multi-threaded implementation based on Java code kindly supplied by the authors of [8]. We ran our code on an 8x virtual CPU hosted on a 2x Intel Xeon X5650, clocked at 2.67 GHz (IBM BladeCenter HS22V). Except where noted below, our experiments were run for 100 trials using a randomly-generated RSA key for each trial. Except where noted, our results refer to private keys of the form $\mathsf{sk} = (p, q, d, d_p, d_q)$ and are all for 1024-bit RSA moduli.

We have conducted extensive experiments for the symmetric case considered in [8]. Our results are shown in Table 3. For small values of δ, we achieve a success rate of 1 or very close to 1 using only moderate amounts of computation. By contrast the HMM algorithm does not achieve such high success rate for small δ. This cannot be solved by increasing t in the HMM algorithm because this leads to a blow-up in running time. For larger δ, the success rate of our algorithm is comparable to that of [8] for similar values of t. We were able to obtain a non-zero success rate for $\delta = 0.22$, while [8] only reached $\delta = 0.20$. The bound from capacity is 0.243.

For the idealised cold boot setting where $\alpha = 0$, our experimental results are shown in Table 4. Recall that the HS algorithm can also be applied to this case. Translating the fraction of known bits $(1 - \rho)$ to the idealised cold boot setting,

Table 4. Success probabilities for the idealised cold boot case ($\alpha = 0$). Capacity bound on β is 0.666.

ρ	0.1	0.2	0.3	0.4	0.46	0.5	0.55	0.6	0.62	0.63
t	6	6	8	12	16	18	18	18	18	18
L	4	4	8	8	8	16	16	16	64	64
Success rate	1	1	1	0.98	0.87	0.81	0.43	0.13	0.07	0.03
Time per trial (ms)	69	88	147	1518	22349	292834	282235	290254	692532	683421

Table 5. Success probabilities for the true cold-boot case with $\alpha = 0.001$. Capacity bound on β is 0.658.

β	0.1	0.2	0.3	0.4	0.5	0.55	0.6	0.61
t	6	6	8	12	16	18	18	18
L	4	4	8	8	16	32	64	64
Success rate	1	1	0.97	0.97	0.66	0.31	0.09	0.04
Time per trial (ms)	80	80	273	4268	42732	384262	740244	735169

and assuming the HS algorithm works just as well when only 1 bits are known (instead of a mixture of 0 and 1 bits), the maximum value of β that could be handled by the HS algorithm theoretically would be 0.46 (though results reported in [9] would allow β as high as 0.52). Our algorithm still has a reasonable success rate for β as high as 0.6 and non-zero success rate even for $\beta = 0.63$, beating the HS algorithm by some margin. Our capacity analysis for this case suggests that the maximum value of β will be 0.666. Thus our algorithm is operating within 5% of capacity here.

Table 6. Success probabilities for the true cold-boot case with $\alpha = 0.001$ and $\mathsf{sk} = (p, q, d)$. Capacity bound on β is 0.479.

β	0.1	0.15	0.20	0.25	0.30	0.35	0.40	0.43
t	6	10	14	16	18	18	18	18
L	4	16	16	16	16	16	32	64
Success rate	0.99	0.99	0.98	0.96	0.63	0.55	0.12	0.04
Time per trial (ms)	46	371	4441	19906	117502	108523	165418	301457

Table 7. Success probabilities for the true cold-boot case with $\alpha = 0.001$ and $\mathsf{sk} = (p, q)$. Capacity bound on β is 0.298.

β	0.05	0.1	0.15	0.20	0.26
t	10	12	16	18	18
L	8	8	16	32	64
Success rate	0.95	0.83	0.68	0.29	0.06
Time per trial (ms)	404	904	9492	87273	217214

We present experimental results for the true cold boot setting in Table 5. Given $\alpha = 0.001$, it follows from our asymptotic analysis that the theoretical maximum value of β which can be handled by our algorithms is 0.658. Our algorithm still has a non-zero success rate for β as high as 0.61. We reiterate that this true cold boot setting is not handled by any of the algorithms previously reported in the literature.

Furthermore, for private keys of the form $\mathsf{sk} = (p, q, d)$ and $\mathsf{sk} = (p, q)$, our algorithm performs very well in the true cold boot setting. For $\mathsf{sk} = (p, q, d)$, the maximum value of β suggested by our capacity analysis is 0.479. With $\beta = 0.4$, $t = 20$ and $L = 16$ our success rate is 0.12 and we have non-zero success rate even with $\beta = 0.43$. Similarly, when $\mathsf{sk} = (p, q)$ our capacity analysis shows that the maximum β is 0.298. When $\beta = 0.2$, $t = 18$ and $L = 16$ we still have a success rate of 0.29, but we can even recover keys with non-zero success rate for β as high as 0.26. Tables 6 and 7 show our results for these cases.

In the full version, we report further results for the erasure channel that improve on the results of [9] and nearly close the gap to our capacity bound. For example, when $m = 5$, we can achieve reliable key recovery up to an erasure rate of 0.79 for this channel, where the bound from capacity is 0.80. By contrast, the best result reported in [9] is for erasure rate 0.76. These and other improvements are obtained using an optimised 'C' implementation of a depth-first search.

6 Conclusions

We have introduced an coding-theoretic viewpoint to the problem of recovering an RSA private key from a noisy version of the key. This provides new insights on the HS and HMM algorithms and leads to a new algorithm which is efficiently implementable and enjoys good performance at high error rates. In particular, ours is the first algorithm that works for the true cold boot case, where both $\Pr(0 \to 1)$ and $\Pr(1 \to 0)$ are non-zero. Our algorithm is amenable to asymptotic analysis, and our experimental results indicate that this analysis provides a good guide to what is actually achievable with reasonable computing resources. Open problems include:

1. Developing a rigorous asymptotic analysis of our algorithm in the general case. However, in view of the state-of-the-art in list decoding, this seems to be hard to obtain.
2. Generalising our approach to the situation where soft information is available about the private key bits, for example reliability estimates of the bits. In general, and by analogy with the situation in the coding theory literature, one would expect to achieve better performance by exploiting such information.

Acknowledgements. The first and third authors were supported by EPSRC Leadership Fellowship, EP/H005455/1. The second author was supported by the Lilian Voudouri Foundation. We thank Mihir Bellare and the referees of Crypto 2012 for thought-provoking comments on an earlier version of this paper.

References

1. Boneh, D.: Twenty years of attacks on the RSA cryptosystem. Notices of the American Mathematical Society 46(2), 203–313 (1999)
2. Brumley, D., Boneh, D.: Remote timing attacks are practical. Computer Networks 48(5), 701–716 (2005)
3. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. Cryptology 10(4), 233–260 (1997)
4. Elias, P.: List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics. MIT (1957)
5. Guruswami, V.: Algorithmic results in list decoding. Foundations and Trends in Theoretical Computer Science 2(2) (2006)
6. Alex Halderman, J., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold boot attacks on encryption keys. In: van Oorschot, P.C. (ed.) USENIX Security Symposium, pp. 45–60. USENIX Association (2008)
7. Alex Halderman, J., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: cold-boot attacks on encryption keys. Commun. ACM 52(5), 91–98 (2009)
8. Henecka, W., May, A., Meurer, A.: Correcting Errors in RSA Private Keys. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 351–369. Springer, Heidelberg (2010)
9. Heninger, N., Shacham, H.: Reconstructing RSA Private Keys from Random Key Bits. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009)
10. Herrmann, M., May, A.: Solving Linear Equations Modulo Divisors: On Factoring Given Any Bits. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 406–424. Springer, Heidelberg (2008)
11. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
12. May, A.: Using LLL-reduction for solving RSA and factorization problems: A survey. In: Nguyen, P. (ed.) Proceedings of LLL+25, p. 3 (June 2007)
13. Sarkar, S., Maitra, S.: More on correcting errors in RSA private keys: Breaking CRT-RSA with low weight decryption exponents. Cryptology ePrint Archive, Report 2012/106 (2012); To appear at CHES (2012)
14. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal 27, 379–423, 623–656 (1948)
15. Yilek, S., Rescorla, E., Shacham, H., Enright, B., Savage, S.: When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In: Feldmann, A., Mathy, L. (eds.) Internet Measurement Conference, pp. 15–27. ACM (2009)