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 algorithms and our new algorithm.

1 Introduction

1.1 Cold Boot Attacks

Cold boot attacks [10, 11] 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 [10, 11] and specifically for RSA private keys in [12, 13]. Similar problems arise in the context of side-channel analysis of cryptographic implementations, where noisy key information may leak through power consumption [16] or timing [4]. 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 [2, 18] and [14]).

Heninger and Shacham (HS) [13] 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 [13] 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 [13]. 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 [13] extends to cases where the RSA private key sk is of the form (p, q, d) or (p, q).

Henecka, May and Meurer (HMM) [12] took the ideas of [13] and developed them further to address the situation where no RSA private 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 [12] 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, [12] reports the results of experiments for different noise levels and algorithmic parameters. For example, the algorithm can cope with $\delta = 0.20$, having an experimental success rate of 21% and a running time of 3 minutes at this noise level.

1.2 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 [10, 11] 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 [13], 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 [13] 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 [13] 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 $\mathsf{sk} = (p, q, d, d_p, d_q)$. Naturally, one could correct for this by re-running the HS algorithm many times with a small subset of bits being flipped in sk each time. However, this would be highly inefficient, as well as inelegant. 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, while the HMM algorithm may be appropriate for RSA private key recovery in some side-channel settings (such as power analysis attacks), it does not solve the cold boot problem for RSA keys.

Intriguing features of the work in [12, 13] 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?

1.3 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, with many details to come in the main body of the paper.

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 [13], 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 [12], 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 [21] 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. There is a subtle technicality here: the converse to Shannon's theorem 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. The correct key can then be determined, for example using a trial encryption and decryption. It is then plausible that such list-outputting algorithms might surpass the bounds imposed by the converse to Shannon's theorem. However, this is not the case: 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 [13] 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 to go below a fraction 0.2 of known bits with an efficient algorithm. Similar remarks apply to the HMM algorithm: here the relevant cross-over probability δ at which a capacity of 1/5 is reached is $\delta = 0.243$, which is in remarkable agreement with the maximum value of 0.237 for δ arising in the analysis of [12], but still a little above the figure of 0.20 achieved experimentally in [12]. Again, the capacity analysis indicates that going above a noise level of 0.243 is likely to be very difficult, if not impossible, for an efficient HMM-style 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 contrast to the HS and HMM algorithms. 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; from a coding perspective, our algorithm uses maximum likelihood list decoding, where L is the size of the list. 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.

Also using coding-theoretic tools, we are able to give an analysis of the success probability of our new algorithm. We 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. For the symmetric case, we are able to give a *rigorous* proof of our claim under assumptions comparable to those used in [12, 13]. Our analysis takes the standard proof of the random coding bound for list decoding [8, 9] and modifies it to cope with the non-random nature of our code. For the general case, our analysis is very simple but non-rigorous: it assumes the code behaves like a random code and follows from a direct application of Shannon's noisy-channel coding theorem. We note that it seems unlikely that a rigorous proof for the full case will be easy to obtain: such a proof would likely yield a Shannon-style random coding bound for list decoding on non-symmetric channels, and such bounds are not known, despite list decoding having been the subject of many decades of study in the information theory community.

As a complement to our theoretical analysis, and as validation of it, 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 19% success rate in recovering keys for $t = 18$ and $L = 32$. This is comparable to the results of [12]. Furthermore, for the same t and L we achieve a 3% success rate for $\delta = 0.22$, whilst [12] 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 14% 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 [13] 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). Finally, we report results showing that a stack-based, depth-first version of the HS algorithm can achieve key recovery at higher noise rates than reported in [13] for key sizes of practical interest. For example, we can reach the lower bound set by capacity, where the known fraction of the private RSA key bits is 0.20. By comparison, the results reported in [13] only reach a known fraction of 0.24. We also examine the application of our stack-based algorithm for the erasure channel to the problem of recovering an RSA private key when multiple, contiguous blocks of private key bits are known.

1.4 Further Related Work

In recent work that is independent of ours, Sarkar and Maitra [20] revisited the work of [12], 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. Also in recent independent work, Kunihiro *et al.* [17] generalised the work of [12, 13] to consider a combined erasure and symmetric error setting, where each bit of the private key is either erased or flipped.

Key recovery for various types of symmetric key was considered in [1, 15, 22].

1.5 Paper Organisation

The remainder of this paper is organised as follows. In the next section, we give further background on the algorithms of [12,13]. 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 and gives our analysis of its performance. Section 5 presents our experimental results. Finally, Section 6 contains some closing remarks and open problems.

2 The HS and HMM Algorithms

In this section, we describe the work of [12, 13], thereby making this paper self-contained. As far as possible, we maintain the notation of these earlier papers.

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 [12,13], we assume throughout that e is small, say $e = 3$ or $e = 2^{16}+1$; for empirical justification of this assumption, see [23]. 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 [12, 13] 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 [13]: 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 [12,13]. 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 [13]. Suppose we have a solution (p', q', d', d'_p, d'_q) from $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$. Then [13] 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 [13] 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 [13] 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.* [12] 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.

Some additional remarks on these algorithms follow. Firstly, the HMM algorithm is iterative: at each stage in the 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 [12], 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 [6]. Secondly, the analysis in [12] is based on the heuristic that every candidate solution on bit slices i to $i + t - 1$ "is an ensemble of t randomly chosen bit slices" when the starting solution on bit slices 0 to $i - 1$ is an incorrect solution. Equivalently, it is assumed that each of the 2^t vectors of $5t$ bits $\mathsf{s} = (\mathsf{slice}(i), \ldots, \mathsf{slice}(i + t - 1))$ representing all candidate solutions generated from a single incorrect partial solution is uniformly distributed. This seems to be well-supported by experimental evidence [12,13]. Note also that, in the analysis of [12], these 2^t vectors do not need to be independent of one another, though independence of the bits of any given candidate is needed at one point in the analysis of [12] (in order to be able to apply Hoeffding's bound). Thirdly, 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 [7].

To complete the picture, we need to discuss what error and channel models are used in [12,13], and what models are appropriate to the cold boot setting. As noted in the introduction, [13] 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 [13]. On the other hand, [12] 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 [10,11], 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 [12,13] 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 [13] 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 [21] 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 [9, Theorem 3.4] that if $\mathcal{C}$ is *any* code of length $n=mt$ 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 [9, 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 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 $C_{\text{BSC}}(\delta) = 1 - H_2(\delta)$ for the capacity of the binary symmetric channel. Table 1 shows that $C_{\text{BSC}}(\delta) = 0.2$ 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 [12]. 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 [12] 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 [13] 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, the fraction of known bits in the private key, being equal to 0.24), leaving a gap between the experimental performance and the theoretical bound. We will close this gap significantly with our new implementation of the HS algorithm. 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

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

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

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 [13] 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 via the formula $\delta = (1-\beta)/2$ given in [13]), 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: an algorithm to recover the key should incorporate information from both 1 bits and possibly incorrect 0 bits, while the HS algorithm exploits only information obtained from the 1 bits known with certainty. Our new algorithm will substantially close the gap between the HS algorithm and the capacity analysis.

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 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:

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

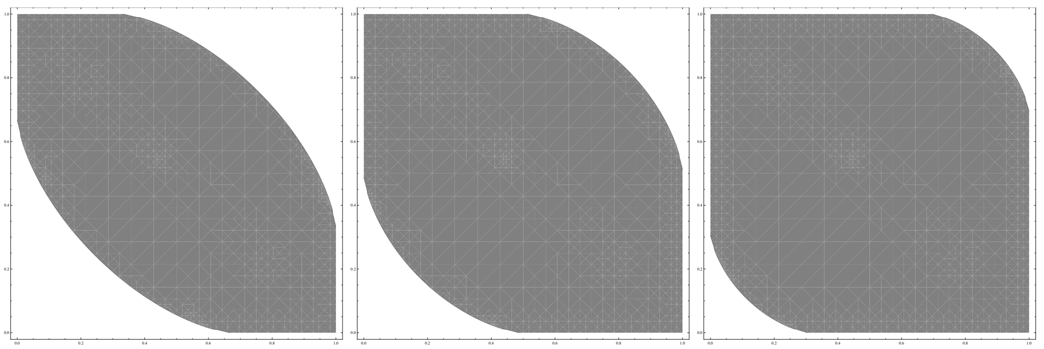

Fig. 1. Plots showing achievable (α, β) pairs for private keys containing 5, 3 and 2 components, respectively. The vertical axis is β, the horizontal axis is α. The shaded area in each case represents the unachievable region.

Algorithm 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 $\log \Pr(\mathsf{r}|\mathsf{s}_i)$ for each entry $\mathsf{s}_i$ on list;
    Keep the $L$ entries in list having the highest log-likelihoods and delete the remainder;
Output list;
```

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

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

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.

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. In practice, we do not generate all $L2^t$ candidates and then filter. Instead, we generate the candidates using a depth-first search approach, implemented using a stack. The stack is initialised with the L starting candidates at each stage. We then filter each candidate as it is produced, adding it to the list only if its log-likelihood is superior to the worst entry currently on the list (expunging the worst entry if this is the case). This radically reduces the memory consumption of the algorithm for large t, from $O(L2^t)$ down to $O(L+t)$. 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 $0 \rightarrow 1$ 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 $0 \rightarrow 1$ 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.

This approach is broadly comparable to that taken in [12]: whereas Henecka *et al.* use a threshold to accept or reject solutions and thus also keep lists of "likely" partial solutions, we avoid having to set a threshold and instead simply keep a list of the best L solutions according to our likelihood statistic. We reiterate that the approach of [12] is formulated only in a case equivalent to that of a binary symmetric channel, where the crossover probabilities are equal, while our modified algorithm can handle this case, the Z-channel case implicit in [13], and a realistic model of the cold boot setting where bit flips in both directions are possible but may have substantially different probabilities.

4.1 Asymptotic Analysis of Our Algorithm

We next give two analyses of our algorithm, using tools from coding theory to assist us. The first analysis uses a strong randomness assumption, that 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). 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.

First Analysis: The first analysis is very simple and is based on the following:

Strong randomness assumption: The $L2^t$ candidates s_i generated at each stage of Algorithm 1 are independent and uniformly random mt-bit vectors.

Our first analysis then proceeds as follows. As before we consider the set $\mathcal{C} = \{\mathsf{s}_i : 1 \leq i \leq L2^t\}$ of candidate solutions at a given stage as a code of length mt and rate $(t + \log_2 L)/mt$. One codeword s_j (corresponding to the correct private key bits) is selected and sent over a memoryless, binary channel with crossover probabilities (α, β) to produce the received vector r. The problem of recovering the correct sent codeword, given r, is the problem of decoding this code. Under our strong randomness assumption, the set of $L2^t$ candidates at each stage is a random code. Now Shannon's noisy-channel coding theorem [21] states that, as $mt \rightarrow \infty$, 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[1]. For fixed L and m, for our code, this holds provided $1/m$ is strictly less than the capacity as $t \to \infty$. Our algorithm does not quite implement ML decoding at each stage, but it always includes the ML candidate in its output list at each stage, since it selects the L most likely candidates. Hence, at each stage, it will be successful in including the correct candidate s_j in its output list provided that ML decoding is also successful. Finally the arbitrarily small error probability per stage guaranteed by Shannon's theorem translates into a success rate at each stage that can be made arbitrarily close to 1 as $t \to \infty$. Since the algorithm runs over $n/2t$ stages, by setting t appropriately as a function of n, we can achieve an overall success rate that is also arbitrarily close to 1.

Summarising, the above analysis 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). This concludes the first analysis.

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 2^t candidates arising from one starting point are arranged in a tree of depth t, with adjacent leaves in the tree agreeing in the first $m(t-1)$ bits, groups of 4 leaves agreeing in the first $m(t-2)$ bits, and so on. Nevertheless, this strong assumption allows a very simple analysis of our algorithm, and the bounds that it yields are well-supported by our experiments to follow.

Second Analysis: Now we give a rigorous analysis of our algorithm under reasonable assumptions in the symmetric case (where $\alpha = \beta = \delta$). We show that we can achieve reliable decoding when the rate is close to the capacity bound $1 - H_2(\delta)$ imposed by the binary symmetric channel.

At each stage the algorithm generates a forest of L trees, $T_0, \ldots, T_{L-1}$, with each tree T_i being of depth t and having 2^t leaves.The internal nodes in each tree represent partial candidate solutions generated during the Hensel lifting and each leaf represents a candidate solution on mt bits.

Now consider 2 leaves in some T_i that have a common ancestor node at depth ℓ (and no common ancestor at any depth $\ell' > \ell$). These 2 leaves are equal in their first $m\ell$ bits, with the next m bits being generated as two distinct solutions to a system of $m-1$ linear equations. On the other hand, the last $m(t-\ell-1)$ bits of two such candidates are generated starting from distinct solutions on m bits at level $\ell+1$ of the tree. Informally, the closer together in a tree two leaves are, the more correlated their bits are. We now make the following assumptions about the candidate solution bits at the leaves of these trees:

Weak randomness assumptions: At any stage in the algorithm:

- The bits of all candidate solutions are uniformly distributed over $\{0,1\}$.
- For some $k \geq 1$, leaves in the same tree are independent on the last $m(t-\ell-k)$ bits, provided the leaves have no common ancestor at depth greater than ℓ.
- The bits at leaves in distinct trees are independent of each other across all mt bits.

In the above assumptions, $k \geq 1$ is a parameter which determines the number of fully independent bits in leaves of the tree. We have done experiments which show that the assumption certainly does *not* hold for $k = 1$ for all RSA moduli; our experiments also indicate that $k = 5$ usually suffices to make the bits approximately independent. The question of whether it is possible *prove* independence (or some approximation to it) for sufficiently large k is open.

To bound the success rate of our algorithm under these assumptions, we adapt a standard result on list decoding of random codes, see for example [8, 9]. Let $\mathcal{C}$ denote the code of $L2^t$ candidate solutions on mt bits at a given stage and let r denote the noisy RSA private key bits. In the symmetric case, the log likelihood expression for $\log \Pr(\mathsf{r}|\mathsf{s}_i)$ simplifies to

$$(n^i_{00} + n^i_{11}) \log(1-\delta) + (n^i_{01} + n^i_{10}) \log \delta$$

[1] The usual proof of Shannon's theorem does not employ ML decoding but instead uses a less powerful decoding algorithm based on *jointly typical* sequences. However ML decoding will do at least as well as any alternative procedure.

where now $n^i_{01} + n^i_{10} = d_H(\mathsf{r}, \mathsf{s}_i)$ and $n^i_{00} + n^i_{11} = mt - d_H(\mathsf{r}, \mathsf{s}_i)$. Thus maximising likelihood is equivalent to minimising Hamming distance and we may assume that our algorithm outputs the L codewords from $\mathcal{C}$ that are closest to r in the Hamming metric.

Then a sufficient condition for our algorithm to be successful on average for this stage is that

$$\Pr(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt \ \forall \mathsf{s} \in S)$$

should be small (in a sense to be made precise) over random choices of r and subset S of $\mathcal{C}$ with $|S| = L + 1$. For, if this is the case, then, given that r will have on average δmt bits in error compared to the correct codeword s_j, this bound on probability dictates that there is a small chance that L wrong codewords will be close enough to r that they are all preferred over s_j in the list output by the algorithm. Hence the algorithm is highly likely to output the correct s_j in its list when this probability is small.

Such a bound can be established for our code $\mathcal{C}$ under our weak randomness assumptions:

Theorem 1. *Let $c > 0$ be a constant, and write $R = 1/m$. Suppose δ is such that*

$$R \leq 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - (c + \frac{1}{L})$$

Then with notation as above, and under our weak randomness assumptions, we have:

$$\Pr(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt \ \forall \mathsf{s} \in S) \leq L^{L+1}2^{-mt/L} + \frac{L+1}{2^{\lfloor ct \rfloor - k+1}}.$$

In particular, for fixed $c > 0$, k and L, the above probability tends to 0 as $t \to \infty$.

The theorem merits some interpretation. The standard bound on list decoding for random codes over the symmetric channel [8, 9] states that, provided $R \leq 1 - H_2(\delta) - 1/L$, then a rate R random code of length n is such that the probability

$$\Pr(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta n \ \forall \mathsf{s} \in S)$$

tends to zero as $n \to \infty$. Here, $1 - H_2(\delta)$ is just the classical Shannon capacity of the channel. One interprets this result as saying that list decoding for a list of length L will be successful provided the rate is not too close to capacity. Our theorem shows that the same is true for our code $\mathcal{C}$, up to a small defect induced by c. Since $c > 0$ is arbitrary, we obtain, qualitatively, a bound of the same strength in the asymptotic setting, i.e. as $t \to \infty$. We stress that, as with most analyses based on random codes in the coding theory literature, one should not expect the theorem to be effective for small values of t.

The bound in the theorem applies to a single stage of our algorithm. It extends in the obvious way to all $n/2t$ stages, providing that $n/2t$ does not grow too fast as $t \to \infty$, showing that our complete algorithm has success probability arbitrarily close to 1 as $t \to \infty$.

Finally, we note that the argument we use in the proof of Theorem 1 is specific to the symmetric case. We do not know how to generalise it to the case of general memoryless binary channels. Indeed, based on our analysis of the literature, the state-of-the-art appears to be that no random coding bounds are known for list decoding for asymmetric channels.

Proof. (of Theorem 1) For $S \subset \mathcal{C}$ with $|S| = L + 1$, we say that S is *bad* if any 2 words in S correspond to leaves from the same tree having a common ancestor at depth greater than or equal to $\ell_c := \lfloor ct \rfloor - k$. Otherwise, we say that S is *good*. Note that if S is bad, then some pair of words in S will have their first $m\ell_c$ bits in common, while if S is good, then all pairs of words in S will have (at least) their last $m(t - \ell_c - k)$ bits independent and uniformly random.

Now, over a uniformly random choice of $S' \subset \mathcal{C}$ of size 2, we have

$$\Pr(S' \text{ bad}) \leq \frac{2^{t-\ell_c} - 1}{L2^t} \leq \frac{1}{L2^{l_c}}$$

since there are $L2^t$ leaves in total and, given a choice of a first leaf, there are $2^{t-\ell_c} - 1$ choices of other leaves below the first leaf's ancestor node at depth ℓ_c. Applying a union bound over pairs of words from S of size $L+1$, we get:

$$\Pr(S \text{ bad}) \leq \binom{L+1}{2} \cdot \frac{1}{L2^{\ell_c}} \leq \frac{L+1}{2^{\ell_c+1}}.$$

Notice that, for our choice of ℓ_c, $\Pr(S \text{ bad})$ becomes arbitrarily small as $t \to \infty$ for fixed L.

Let $E(\mathsf{r}, S)$ denote the event that $d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt$ for all $\mathsf{s} \in S$. Now, to bound the probability of $E(\mathsf{r}, S)$ over random choices of r, S, we condition on S being good or bad:

$$\begin{aligned}\Pr(E(\mathsf{r}, S)) &= \Pr(E(\mathsf{r}, S) | S \text{ good}) \cdot \Pr(S \text{ good}) + \Pr(E(\mathsf{r}, S) | S \text{ bad}) \cdot \Pr(S \text{ bad}) \\ &\leq \Pr(E(\mathsf{r}, S) | S \text{ good}) + \Pr(S \text{ bad}).\end{aligned}$$

Here, we already have a bound for the second term which tends to 0 for fixed L as $t \to \infty$. We proceed to bound the first term. Now for fixed r and good S, we know that all words in S are uniformly and independently distributed on their last $m(t - \ell_c - k)$ bits. Moreover, r is randomly distributed. We will focus the remainder of the proof on these last $m(t-\ell_c-k)$ bits. The condition $d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt$ is certainly satisfied if it holds already on the last $m(t - \ell_c - k)$ bits. Hence, when S is good, for each $\mathsf{s} \in S$, we have:

$$\Pr(d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt) \leq \frac{\mathrm{Vol}(\delta mt, m(t - \ell_c - k))}{2^{m(t-\ell_c-k)}}$$

where $\mathrm{Vol}(a, b)$ denotes the volume of a Hamming ball of radius a in b-dimensional Hamming space. Approximating ℓ_c by $ct - k$, and using the standard volume bound

$$\mathrm{Vol}(\delta b, b) \leq 2^{bH_2(\delta)}$$

we obtain, after some simplification:

$$\Pr(d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt) \leq \frac{2^{(mt(1-c))H_2(\delta/(1-c))}}{2^{mt(1-c)}}.$$

Now we use a union bound, noting that there are 2^{mt} choices for r, at most $(L2^t)^{L+1}$ choices for S of size $L+1$, and everything is independent on the last $m(t - \ell_c - k)$ bits, to obtain:

$$\begin{aligned}\Pr(E(\mathsf{r}, S) | S \text{ good}) &\leq 2^{mt} \cdot (L2^t)^{L+1} \cdot \left(\frac{2^{(mt(1-c))H_2(\delta/(1-c))}}{2^{mt(1-c)}}\right)^{L+1} \\ &= L^{L+1} (2^{mt})^{1+(1/m)(L+1)+(1-c)(L+1)H_2(\delta/(1-c))-(L+1)(1-c)}.\end{aligned}$$

Now we select parameters so that the exponent of 2^{tm} here simplifies. This exponent is

$$\begin{aligned}&1 + \frac{1}{m}(L+1) + (1-c)(L+1)H_2(\frac{\delta}{1-c}) - (L+1)(1-c) &(5)\\ &= 1 + (L+1)\left[\frac{1}{m} + (1-c)H_2(\frac{\delta}{1-c}) - (1-c)\right]. &(6)\end{aligned}$$

Suppose we set δ such that

$$\frac{1}{m} = 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - (c + \frac{1}{L})$$

(such a value δ will always exist provided c is sufficiently small and $m = 2, 3$ or 5). After some manipulation, we find that for this carefully chosen value of δ, the exponent of 2^{mt} in (6) simplifies to $-1/L$, and hence, for this value of δ, we have:

$$\Pr(E(\mathsf{r}, S) | S \text{ good}) \leq L^{L+1}(2^{mt})^{-1/L}.$$

δ	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.19	0.2	0.21	0.22
t	6	6	8	10	12	16	18	18	18	18	18
L	2	4	4	8	32	32	32	32	32	32	64
Success rate	0.993	0.982	0.991	0.985	0.978	0.82	0.61	0.4	0.19	0.06	0.03
Time per trial (ms)	46	95	172	211	1852	35062	161512	159132	160325	159215	325193

Table 3. Success probabilities for the symmetric case ($(\alpha, \beta) = (\delta, \delta)$). Experiments with $\delta \leq 0.14$ are based on 500 trials. Capacity bound on δ is 0.243.

Moreover, for sufficiently small c, the above bound also holds for every δ such that

$$\frac{1}{m} \leq 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - (c + \frac{1}{L}). \tag{7}$$

This follows easily from noting that $1 - (1-c)H_2\left(\frac{\delta}{1-c}\right)$ is a decreasing function of δ for $0 \leq \delta \leq (1-c)/2$.

Putting everything together, we have that, provided δ satisfies (7),

$$\Pr(E(\mathsf{r}, S)) \leq L^{L+1}2^{-mt/L} + \frac{L+1}{2^{\lfloor ct \rfloor - k + 1}}$$

which concludes the proof.

5 Experimental Results

For our experiments using our algorithm, we used a multi-threaded 'C' implementation and Shoup's NTL library for large integer arithmetic. The stack-based, depth-first approach to generating trees of candidates greatly reduced the amount of memory consumed by the algorithm. Because of certain thread-safety issues in NTL, we had to introduce some artificial delays into our code, making all of our running times higher than they could otherwise have been. 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. Also 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.

5.1 The Symmetric and Cold Boot Channels

We have conducted extensive experiments using our algorithm for the symmetric case considered in [12]. 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 [12] for similar values of t. We were able to obtain a non-zero success rate for $\delta = 0.22$, while [12] 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, 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 [13] 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.

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	0.99	0.95	0.94	0.82	0.80	0.40	0.14	0.05	0.01
Time per trial (ms)	45	90	129	1135	9323	154562	155328	156515	288531	282957

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

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	0.99	0.99	0.98	0.61	0.33	0.11	0.04
Time per trial (ms)	71	73	281	2385	22316	171356	333614	339198

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

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.

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 0.11 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 0.31, 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.

5.2 The Erasure Channel

We have also experimented with a stack-based, depth-first approach to implementing the algorithm of [13] for the erasure channel. Our main computational results are shown in Table 8. In this table, ρ is the (random) fraction of bits of the private key that are erased. In each trial, we aborted the trial if the total number of Hensel lifts performed exceeded a *panic size* of 10 million. The figures in brackets are averages for the non-aborted trials; the main figures include aborted trials. This panic size parameter is akin to the panic width parameter of [13], but reflects our use of a depth-first rather than a breadth-first algorithm. The success rate reports the proportion of trials for which the set panic size of 10 million was not exceeded. Note that we attain a success rate of 0.4 even at the limit of $\rho = 0.8$ set by our capacity analysis. This success rate does not contradict our capacity analysis, which we recall puts a limit on *efficient* key recovery. The benefits of using a stack-based approach are evident from Table 8: the running time is less than 1s per trial even for error rates close to the capacity (indeed, for low error rates it was enough to search only the first path through the tree to recover the correct solution), and the memory consumption is low since the maximum stack size stays small. Our results when $\mathsf{sk} = (p, q, d)$ are shown in Table 9, also for a panic size of 10 million. The capacity bound for this case is 0.67. Our results for the case where $\mathsf{sk} = (p, q)$ are shown in Table 10, also for a panic size of 10 million. The upper bound derived from our capacity analysis in this case is 0.5. In all three cases, we reach capacity and thereby significantly improve on the maximum error rates reported in [13].

We have also conducted experiments to explore the effect of panic size on success rate. One would expect that as the panic size is increased, less runs are aborted, and the success rate increases. Figure 2 shows the success rate as a function of error rate (ρ) for panic sizes of 10,000, 100,000, 1 million and 10 million and for the three different key settings ($m = 5, 3, 2$); each data point in the graphs in this figure was generated using 100 trials for random 1024-bit bit RSA moduli. The expected behaviour is seen: the curve shifts to the right as the panic size is increased, and, in line with our capacity analysis, there is a marked drop-off in success rate once capacity is reached (but capacity is attained and even exceeded with non-zero success rate).

The box plots in Figure 3 show the distribution of the total number of keys examined by our stack-based algorithm with increasing ρ. The data were generated using 1024-bit bit RSA moduli and 100 trials per ρ value. Each box plot depicts the smallest observation, the lower quartile where 25% of the data is less than this value, the median where 50% of the data is greater than this value, the upper quartile where 25% of the date is greater than this value, and the largest observation. We see that the total number of keys examined remains reasonable even as ρ approaches capacity. This is in contrast to the HS algorithm, where an explosion in the number of key candidates that need to be considered is experienced as capacity is approached.

5.3 Recovering RSA Keys with Block-wise Partial Knowledge of p and q

Herrmann and May [14], in work continuing that in [3,5,19], used lattice-based techniques to show that an RSA modulus N can be factored given some small number of contiguous blocks of bits in one of the primes p dividing N. Their running time is polynomial only when the number of blocks is $O(\log \log N)$, and the algorithm needs 70% of the bits of p to be known in total across the blocks.

A related scenario not addressed previously in the literature is where the adversary is given some number of contiguous blocks of bits from both factors p and q of N. These might be obtained from a side-channel analysis, for example. It is readily seen that this scenario is a special case of the erasure channel, and so one might expect that the techniques of [13] and this paper could be readily applied. Moreover, it would seem reasonable that, provided roughly 50% of the bits of p and q are known, the remainder could be recovered, while below this level, a capacity analysis would indicate that key recovery should fail. We explore this scenario and intuition in the remainder of this section.

For simplicity, we assume the known and unknown bits of p and q are arranged in alternating blocks, beginning with a known block in each of p and q in the least significant bit positions. We let κ denote the length of all the known blocks and λ the length of all the unknown blocks. The error rate is then approximately $\lambda/(\kappa+\lambda)$. The behaviour of the HS algorithm can then be predicted as follows: when traversing blocks of unknown bits, we expect the number of surviving candidates to double at each new bit considered (since there is no information by which to reject candidates); on the other hand, when traversing blocks of known bits, we expect the number of surviving candidates to roughly half for each new bit considered (since each candidate on bits $\mathsf{slice}(i-1)$ will produce 2 new candidates on bits $\mathsf{slice}(i)$, but these must agree with the 2 known bits of p and q in position i). Thus we expect to see repeated explosions and collapses in the number of candidate solutions as the algorithm progresses. Moreover, if $\kappa < \lambda$ (and the approximate error rate is greater than 0.5), we would expect to see a steady upward trend in the number of surviving candidate solutions, since the number of doublings will outweigh the number of halvings. Conversely, when $\kappa > \lambda$ (and the approximate error rate is less than 0.5), we would expect the number of surviving candidate solutions to stay small. This matches with what a simple capacity analysis would predict. Since our stack-based version of the HS algorithm traverses the same tree as the original HS algorithm but in a different order, we would expect it to have roughly the same behaviour.

We have run our stack-based version of the HS algorithm for various values of κ and λ, and Tables 11 and 12 show our results. Table 11 concerns the case where $\kappa = \lambda$, so that the error rate ρ is close to capacity. It can be seen that our algorithm performs well even here. However for certain values of κ (for example $\kappa = 8, 16$), the running time and number of keys considered increases substantially relative to neighbouring values. We do not have an explanation for this phenomenon.

For larger values of κ, our algorithm does not perform well. This seems to be because of the large number of candidate solutions that need to be considered as the algorithm traverses regions of unknown bits; however, this does not explain the large increase in running time at $\kappa = 26$. Table 12 concerns the case where κ is fixed (to 16) and λ increases gradually, so that the erasure rate approaches and then exceeds 0.5. It can be seen that our algorithm efficiently recovers the RSA private key up to the bound imposed by capacity, but is still successful even for the pair $(\kappa, \lambda) = (16, 17)$.

6 Conclusions

We have introduced a 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 $\Pr(0 \rightarrow 1)$ and $\Pr(1 \rightarrow 0)$ 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. Martin R. Albrecht and Carlos Cid. Cold boot key recovery by solving polynomial systems with noise. In Javier Lopez and Gene Tsudik, editors, *ACNS*, volume 6715 of *Lecture Notes in Computer Science*, pages 57–72, 2011.
2. Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society*, 46(2):203–313, 1999.
3. Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In *ASIACRYPT*, pages 25–34, 1998.
4. David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
5. Don Coppersmith. Finding a small root of a univariate modular equation. In *EUROCRYPT*, pages 155–165, 1996.
6. Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
7. Peter Elias. List decoding for noisy channels. Technical Report 335, Research Laboratory of Electronics, MIT, 1957.
8. Peter Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37(1):5–12, 1991.
9. Venkatesan Guruswami. Algorithmic results in list decoding. *Foundations and Trends in Theoretical Computer Science*, 2(2), 2006.
10. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 45–60. USENIX Association, 2008.

11. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, 2009.
12. Wilko Henecka, Alexander May, and Alexander Meurer. Correcting errors in RSA private keys. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 351–369. Springer, 2010.
13. Nadia Heninger and Hovav Shacham. Reconstructing RSA private keys from random key bits. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.
14. Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In Josef Pieprzyk, editor, *ASIACRYPT*, volume 5350 of *Lecture Notes in Computer Science*, pages 406–424. Springer, 2008.
15. Abdel Alim Kamal and Amr M. Youssef. Applications of SAT solvers to AES key recovery from decayed key schedule images. *IACR Cryptology ePrint Archive*, 2010:324, 2010.
16. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.
17. Noboru Kunihiro, Naoyuki Shinohara, and Tetsuya Izu. Recovering RSA secret keys from noisy key bits with erasures and errors. In *PKC*, page to appear, 2013.
18. Alexander May. Using LLL-reduction for solving RSA and factorization problems: A survey. In Phong Nguyen, editor, *Proceedings of LLL+25*, page 3, June 2007.
19. Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In *EUROCRYPT*, pages 31–34, 1985.
20. Santanu Sarkar and Subhamoy Maitra. Side channel attack to actual cryptanalysis: Breaking CRT-RSA with low weight decryption exponents. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2012.
21. Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.
22. Alex Tsow. An improved recovery algorithm for decayed AES key schedule images. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 215–230. Springer, 2009.
23. Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Internet Measurement Conference*, pages 15–27. ACM, 2009.

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	1	0.99	0.97	0.95	0.60	0.58	0.11	0.05
Time per trial (ms)	62	402	3056	12135	62537	62139	121358	120985

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	0.05	0.1	0.15	0.20	0.26
t	10	12	16	18	18
L	8	8	16	32	64
Success rate	0.96	0.80	0.65	0.31	0.08
Time per trial (ms)	318	629	6451	61213	131600

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.

ρ	0.4	0.5	0.6	0.7	0.75	0.76	0.77	0.78	0.79	0.8
Success rate	1	1	1	1	1	1	1	0.98	0.77	0.4
Keys examined	527	553	627	971	5791	13106	167762	458685(263835)	3014051(923938)	7404887(2875484)
Hensel lifts	520	536	593	910	5707	13008	167634	458558(263959)	3002894(912849)	7131894(2829735)
Time per trial (s)	0.00234	0.00236	0.00259	0.00409	0.0251	0.0673	0.783	2.14(1.18)	13.8(4.18)	34.4(13.1)

Table 8. Experimental results for the erasure channel with our stack-based algorithm. Capacity bound on ρ is 0.8. See text for further explanation.

ρ	0.4	0.5	0.6	0.62	0.64	0.65	0.66	0.67
Success rate	1	1	1	1	0.93	0.86	0.74	0.34
Keys examined	643	864	9303	34755	1069737(382659)	2178775(904091)	3857730(1668825)	7702819(2973115)
Hensel lifts	604	801	9211	34511	1055410(382162)	2169500(894768)	3830567(1662998)	7593812(2922975)
Time per trial (s)	0.00172	0.00189	0.00251	0.0294	3.45(1.21)	7.14(2.94)	12.6(5.44)	25.9(9.79)

Table 9. Experimental results for the erasure channel with our stack-based algorithm, $\mathsf{sk} = (p, q, d)$. Capacity bound on ρ is 0.67.

ρ	0.4	0.45	0.46	0.47	0.48	0.49	0.5
Success rate	1	1	1	1	0.97	0.89	0.69
Keys examined	2279	22643	54573	212477	759372(473579)	2240527(1278423)	4041475(1437785)
Hensel lifts	2234	22453	54431	209793	757598(471750)	2236743(1277240)	4020695(1334341)
Time per trial (s)	0.00373	0.0299	0.0972	0.377	1.31(0.832)	3.97(2.25)	7.23(2.38)

Table 10. Experimental results for the erasure channel with our stack-based algorithm, $\mathsf{sk} = (p, q)$. Capacity bound on ρ is 0.5.

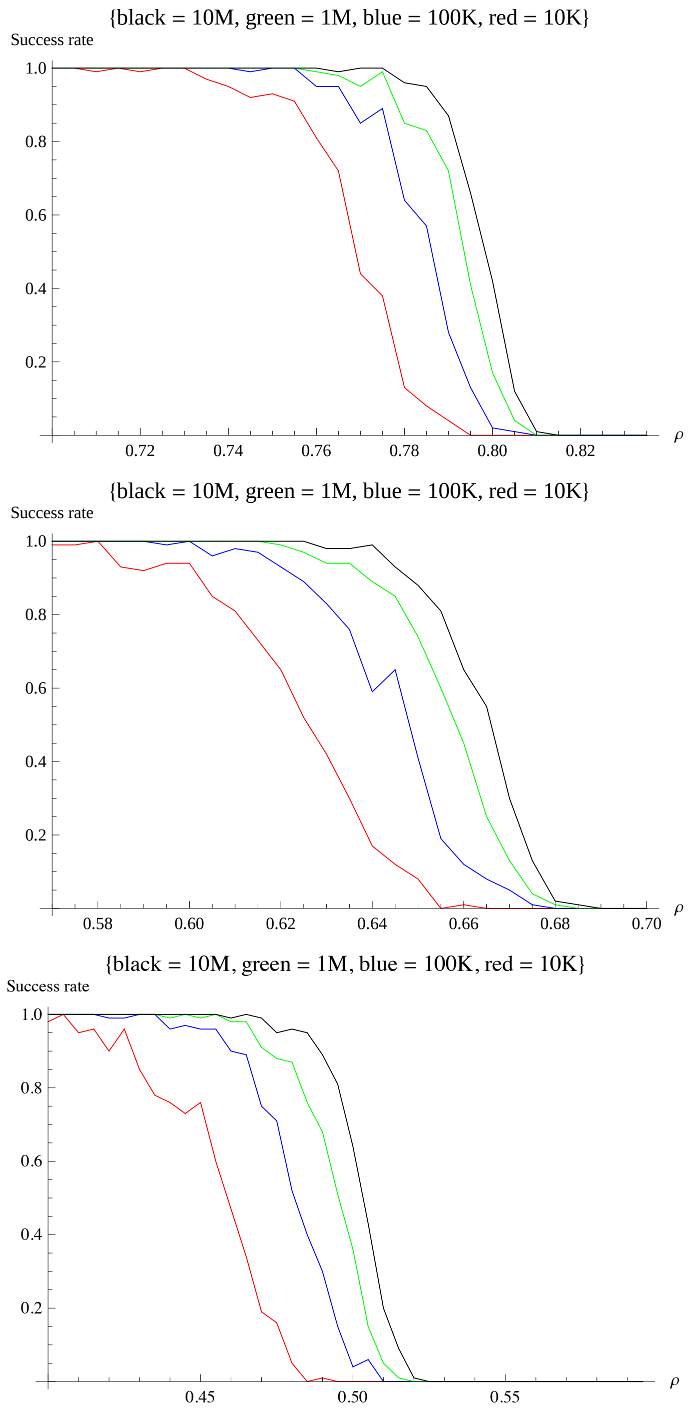

Fig. 2. Graphs showing achievable error rates based on different panic sizes for private keys containing 5, 3 and 2 components, respectively. In each graph, the vertical axis represents success rate and the horizontal axis the erasure rate ρ.

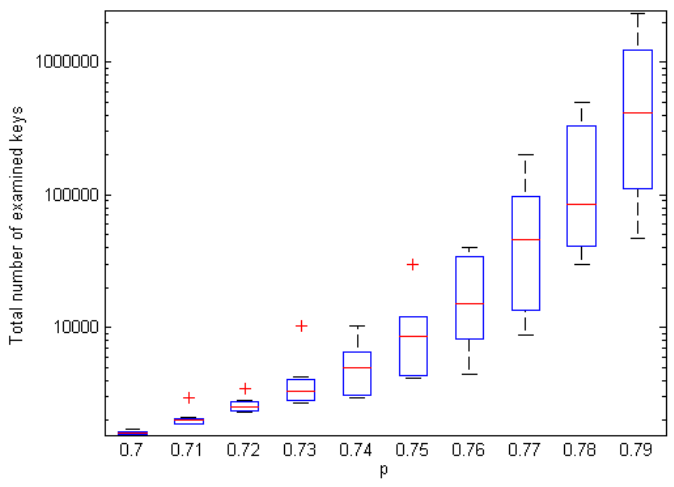

Fig. 3. Box plots for the total number of keys examined for varying ρ.

κ	λ	Total unknown bits	Max stack size	Keys Examined	Time (s)
2	2	510	135	31740	0.0571
4	4	508	137	44369	0.0782
6	6	508	47	15948	0.0285
8	8	504	138	172403	0.3
10	10	504	30	7942	0.014
12	12	496	48	16887	0.0292
14	14	492	61	59174	0.105
16	16	496	140	9200234	12.1
18	18	502	79	404272	0.711
20	20	484	81	1004018	1.78
22	22	496	39	25207	0.0441
24	24	496	47	134339	0.237
26	26	478	100	11521189	152

Table 11. Experimental results for the block-wise erasure channel with $\kappa = \lambda$.

κ	λ	Total unknown bits	Max stack size	Keys Examined	Time (s)
16	2	112	1	512	0.000907
16	4	192	1	512	0.000905
16	6	272	1	512	0.000902
16	8	336	1	512	0.000914
16	10	384	21	832	0.00141
16	12	432	39	2075	0.00345
16	14	464	108	225767	0.381
16	16	496	140	9200234	12.1
16	17	512	16	869974	1.67

Table 12. Experimental results for the block-wise erasure channel with $\kappa = 16$ and increasing λ.