RSA private key reconstruction from random bits using SAT solvers

Constantinos Patsakis

Distributed Systems Group, Department of Computer Science, Trinity College, Dublin, Ireland
WWW home page: `https://www.scss.tcd.ie/~patsakik/`
email: `patsakik@scss.tcd.ie`

Abstract. SAT solvers are being used more and more in Cryptanalysis, with mixed results regarding their efficiency, depending on the structure of the algorithm they are applied. However, when it comes to integer factorization, or more specially the RSA problem, SAT solvers prove to be at least inefficient. The running times are too long to be compared with any well known integer factorization algorithm, even when it comes to small RSA moduli numbers.
The recent work on cold boot attacks has sparkled again the interest on partial key exposure attacks and in RSA key reconstruction. In our work, contrary to the lattice-based approach that most of these works use, we use SAT solvers. For the special case where the public exponent e is equal to three, we provide a more efficient modeling of RSA as an instance of a satisfiability problem, and manage to reconstruct the private key, given a part of the key, even for public keys of 1024 bits in few seconds.

Keywords: SAT solvers, RSA, partial key exposure, factoring, public-key cryptography

1 Introduction

1.1 Background

The past decades, cryptanalysis has followed the path of Cryptography, evolving from art to science. Its scope is to analyze cryptographic primitives, allowing the cryptanalyst to:

- can find weaknesses that expose the key or part of it,
- expose part of the encrypted content,
- relations to other content,
- expose the used key,
- forge authenticated content.

Obviously, its history starts more over the same period with cryptography, however, its methods differ, as the attacker has to try several ways to find a “good” entry point, that will allow him to "break" the cipher. Therefore, we have an endless loop, with the introduction of advanced encryption algorithms, for which more sophisticated attacks are designed and so on.

Satisfiability problem is in the core of Computer Science, as there are links with very basic problems in Algorithms and Complexity Theory. It is quite easy to set the problem: "For a given logic formula C, decide whether C is satisfiable", which in other words means, either find values (True/False) for the logical variables that so that C evaluates to True, or prove that there is no such combination. The problem, contrary to its statement, is one of the hardest to solve, actually, the Satisfiability problem is the first one to be proved as NP-complete Cook [4], Levin [24].

SAT solvers are programs that try to solve SAT problems, usually given in their conjunctive normal form (CNF). One of the first steps towards their efficient implementations was the introduction of the DPLL algorithm Davis and Putnam [7]. Afterwards, the introduction of SAT planning Selman and Kautz [37], WalkSAT algorithm Selman and Kautz [37], phase transition Gent and Walsh [15], as well the Discrete Lagrangian Method (DLM) Wah et al. [41] gave the necessary theoretical background to develop the modern SAT solvers, which have very good performance.

Therefore, SAT solvers started having many applications in a wide range of areas like model checking, automated theorem proving, software verification, planning in AI, drug design and testing, test pattern generation, scheduling and protocol design. Recently, there is a trend towards using SAT solvers in cryptanalysis, mainly in block ciphers and hash functions with mixed results, however, the attempt seems quite promising as it gives a new tool to study the underlying structures of cryptographic primitives, possibly revealing several unknown vulnerabilities.

1.2 Attack scenario

Most of the cryptographic primitives that are used in modern applications, have been proved immune to many theoretical attacks. However, the attacker in many attack scenarios, not only knows which cryptographic primitives are being used, but has access to the implementation as well, whether this is a device or the code used, giving him extra weapons to its arsenal.

Let's assume that Alice is encrypting her partition, in order to further protect her data in an event of a physical security breach. However, for as long as her PC is working, the partition is mounted and the key is loaded in RAM. The motivation behind the cold boot attacks in Halderman et al. [17] was to recover this key from RAM. When the computer is shutting down, RAM is not deleted, it is corrupted. Gradually, depending on the manufacturer and the medium, either 0s are turning to 1s or 1s are turning to 0s. If the attacker freezes the RAM, he is able to delay this corruption. Therefore, after rebooting the machine, the memory dump will contain a corrupted version of its previous state. The quicker and colder RAM is, the less corrupted this version will be. Thus, the attacker can find many bits of the used key.

In other attack scenarios, the attacker can guess parts of the bits of the decryption key judging from the power consumption of the processor (power analysis), from the time it takes to perform encryption/decryption (time analysis) Brumley and Boneh [3]. Of course, there are even more powerful attacks, like the direct memory access from FireWire devices that are plugged to exploit the features of this port.

Under these assumptions, it is realistic to assume that an attacker may have access to part of the decryption key. Therefore, it is very important that the following questions must be addressed to:

Question 1. How much information about the decryption key is leaked by a specific implementation?

Question 2. How much information about the decryption key the attacker has to know for a specific algorithm, in order to break it in reasonable time?

Question 3. Can we construct algorithms that are immune to such attacks? The idea here is that the attacker even if he as access to part of the decryption key, has to resolve to brute force attack for the rest of the bits.

This work focuses on the second question. More precisely, we address to the following:

Question 4. How many random bits of the private key d and p, q of RSA are needed in order to recover the private key?

1.3 Contributions of this paper

The main contributions of this work can be summarized as follows:

- A new, more efficient CNF model for RSA.
  Currently, only two CNF converters exist for integer factorization. Both of them do not manage to provide any applicable results/solid solution to integer factorization. The provided modeling might focus on a specific instance of RSA, however, it outperforms current implementations, using simple replacements, which decrease drastically the complexity of the problem.
- A logical cryptanalysis of RSA with partial key exposure.
  The illustrated attack in this area.
- A novel powerful use of SAT solvers in cryptanalysis.
  So far, there are no significant results in the use of SAT solvers in public key cryptanalysis.
- New attacks with fewer requirements.
  The launched attacks, even if the improvement in key size is small, manage to extract the private key, with fewer bit of keys disclosed in reasonable time.

1.4 Plan of the paper

In this context, the next section provides the necessary background for the applications of SAT solvers in cryptanalysis, partial key exposure and cold boot attacks. In the third section we provide a more efficient modeling of RSA for SAT solvers, which adds several new equations, that can help solving the problem more efficiently, when part of the private key is known. The next section focuses on experimental results. It is illustrated that SAT solvers are a powerful tool for public key cryptanalysis, not yet adequately explored. Even if the problem is very hard, containing several thousands of binary variables and equations, partial exposure of the private key can lead to full key exposure with the use of SAT solvers, in reasonable time. Finally, we conclude discussing the results, the contributions and the drawbacks of this new attack, as well as some remarks and ideas for future work.

2 Related work

2.1 SAT solvers in cryptanalysis

Massacci was the first one to talk about logical cryptanalysis in Massacci [26]. His next step was to provide the logical model of DES and attack it with a SAT solver Massacci and Marraro [27]. In his point of view, we can describe all the encryption algorithms as logical problems, which can be converted to the according system of CNFs. The created system will then be passed to a SAT solver, which will try to solve it. However, the generated SAT problem in the case of DES was very hard to be solved, therefore, SAT solvers couldn't break it. Nonetheless,, several interesting results on simplified versions of DES are given.

In the following years, there was an increasing interest in SAT problems, as it managed to solve many industrial problems. This resulted to the development of several very good SAT solvers like: Chaff, Glucose, Minisat, Picosat , SAT4J, SATzilla to name a few implementations.

This development, showed the potential use of SAT solvers as a tool for cryptanalysis, as at that moment, not only the tools where more efficient, but several attempts were made to make SAT solvers more cryptanalytic-friendly Soos et al. [40]. Soon, several tools were developed using SAT solvers to cryptanalyze cryptographic primitives like, CryptoMiniSat Soos [38], a SAT solver for cryptographic problems, or Grain of SALT Soos [39], a toolkit to analyze stream ciphers both developed by M. Soos.

Having these tools, soon several cryptanalytic attacks using SAT solvers emerged having different impact on each algorithm they were applied, like Eibach et al. [10], Golle and Wagner [16], Mironov and Zhang [28], Morawiecki and Srebrny [30], Erickson et al. [11], Homsirikamol et al. [20], De et al. [8], Courtois et al. [6], Mohamed et al. [29], Kamal and Youssef [21].

SAT solvers in cryptanalysis, in once sense, start where Gröebner bases stop. When a non-linear system of equations is too difficult to be solved algebraically, SAT solvers come to the rescue. Hence, if SAT solvers offer a very quick attack, there must be an underlying problem in the algebraic system that the cipher creates, which Gröebner bases cannot trace. After all, SAT solvers, are a "smart" brute force attack, they evaluate all the possible combinations of values of the variables, subject to several constraints, trying at each step to eliminate variables and possible combinations of variable values.

The only references to our knowledge on results regarding applications of SAT solvers to asymmetric ciphers are Fiorini et al. [14], Dylkeyt et al. [9] and Faizullin et al. [13]. In the first case, the focus is on forging RSA signature, by proper encoding of the modular root finding as a SAT problem. In the second case the researchers try to approximate the number of CNFs that are generated by the conversion of an integer factorization problem to its analogous SAT problem. In the last case, the authors reformulate the CNF, reducing the SAT problem to a minimization problem. The resulting problem can have a more elegant representation, with fewer variables and disjunctions. Moreover, for the case of 512 bit integer factorization, the authors can determine bits in specific positions with very high probability.

2.2 Partial key exposure and cold boot attacks

Quite ironically, one of the first partial key exposures attacks in RSA was made two of its creators, namely Rivest and Shamir, showing that a RSA moduli number n can be factored in polynomial time, given $2/3$ of the LSBs of one of its primes Rivest and Shamir [34].

In 1996 Coppersmith presented one of the most significant theorems in algebraic cryptanalysis, which is named after him, proving that with proper use of the LLL algorithm, one can recover in polynomial time, the roots of a polynomial modulo n, even when the factorization of n is not known Coppersmith [5]. One of the corollaries of this theorem is that if n is a RSA moduli, then it can be factored in polynomial time, given $1/2$ of the MSBs of one of its primes.

Two years later, using Coppersmith's theorem, it was proved that if n is a RSA moduli, then it can be factored in polynomial time, given $1/2$ of the LSBs of one of its primes, or $\frac{\log n}{4}$ bits of d Boneh et al. [2]. In the same work, the researchers provided many results on how many bits of d need to be known to factor the RSA modulus n, when $e < \sqrt{n}$. These results were later improved in 2003, by Blömer and May for $e < N^{0.725}$ in Blömer and May [1].

In Ernst et al. [12], the authors showed that if d has N bits and e has αN bits, if $0 < \delta < \frac{1}{2} < \alpha < 1$, then given $(1-\delta)N$ MSBs of d, n can be factored in polynomial time if:

$$-\delta \leq \frac{1}{3} + \frac{1}{3}\alpha - \frac{1}{3}\sqrt{4\alpha^2 + 2\alpha - 2} - \epsilon$$

In 2008, the researchers focused on cold boot attacks, showing that several disk encryption systems as BitLocker, FileVault, dm-crypt, and TrueCrypt, store information in RAM that can be restored with no special hardware, even after shutdown. The stored keys in the RAM, even if they are partially corrupted, can be used in order to recover the original key Halderman et al. [17]. Following this work, in Heninger and Shacham [19], the authors showed that it is possible to recover the RSA private key given a random fraction of its bits. More precisely:

– 27% of the bits of $p,\ q,\ d,\ d_p$, and d_q,
– 42% of the bits of $p,\ q$, and d,
– 57% of the bits of p and q.

In another approach of the same scenario, Henecka et al. [18] studied the reconstruction of the private key when all the bits are known, with some probability of error.

Improving the results of Heninger and Shacham [19] in Maitra et al. [25], the researchers use lattices, given only knowledge of random bits in the LSB halves of the primes, or blocks of bits in the MSB halves of the primes.

Sarkar tried to reconstruct the RSA private key when there is a pattern in the unknown corrupted version of d, so most of d is considered known and some contiguous blocks are unknown Sarkar [36]. The complexity of the proposed algorithm is polynomial in the length of the key, but exponential in the number of unknown blocks of the key.

In Santanu et al. [35], the researchers reconstruct the private key using partial information with error that they have for the MSBs of the secret parameters and going down to the first bits.

Recently, two independent works Paterson et al. [31] and Kunihiro et al. [22] use a code theoretic approach to recover the private key of RSA, given a "noisy" version of the key. This means that the attacker has access to the private key, but not all of the bits are correct, some of them contain errors. However, in the latter work, the authors not only deal with "noisy" keys, but with with erasures as well, which means that there are "gaps" in the given private key. Therefore in order to recover it, one has to fill the gaps with the proper values and detect and correct the erroneous bits as well.

3 Modeling RSA for SAT solvers

In many partial key exposure attacks of RSA, the main tools that are being used are lattice based. They mainly focus on trying to formulate modular equations that have relatively small solutions, so that Coppersmith univariate or bivariate theorem can be applied. The motivation for this work was that this way some information that is provided could be "lost", in the sense of not being used. However, the bits of n are related to the bits of p and q, as n is their product. Translating this fact to mathematical formulas will give us a very hard system of equations, that cannot be solved algebraicaly for RSA numbers of the length of 1024 bits that are used in applications. But what happens if we know the values of some of the variables? How much is this system simplified?

The questions above are the core of this work. So, to launch the attack firstly, we will provide the system of equations linking the bits of the private with the bits of the public key of RSA. Then we shall replace the parts that are known to us and try to solve the problem. Since we are working with bits, so the values are 0 and 1, we may formulate the problem as a satisfiability problem and use SAT solvers to recover the bit keys.

Even though integer factorization is a subject widely studied, there are few studies linking it with SAT solvers, the general impression is because it is tested and has poor performance, just like Genetic Algorithms. In both cases, a brute force approach is adopted, trying to cover the key space with "random walks". Therefore, they are expected to find solutions in around $\sqrt{n}$ steps.

Currently, there are two implementations which convert the problem of integer factorization to its SAT instance, one by Purdom and Sabry, written in Haskel Purdom and Sabry [33] and one by Yuen and Bebel, written in Python Yuen and Bebel [42]. For our experiments, we based our attacks on the output of the latter. In both implementations, the approach is quite straight forward, take two numbers, p and q in their binary representation and calculate their product according to a multiplier, which is based on full adders and half adders. The result is a representation of each bit of the product with binary operations, that can be easily transformed to its respective system of CNFs.

For general integer factorization problems this approach is very correct, however, when addressing to RSA there are several facts that can simplify our system of CNFs. In the aforementioned implementations the approach was that if n is N bits, then p can have at most $N-1$ bits (n is even) and q can have $\lceil N/2 \rceil$ bits. Since in the RSA case p and q are of the same length, the upper $N-1-\lceil N/2 \rceil$ bits are set to zero, decreasing significantly the number of unknown variables. Moreover, since both p and q are odd, their last bits are 1, decreasing the number of variables by 2.

Since in most implementations of RSA, the value of e is standardized taking the values of $3,\ 17$ and $2^{16}+1$, we focused on the most easy case, of $e = 3$. The reason for this selection is that in this approach we can achieve further simplification. From $ed \equiv 1 \ mod \ \phi(n)$ we have that:

$$3d \equiv 1 \ mod \ \phi(n)$$

$$3d = 1 + k\phi(n)$$

but we have that $0 < k < e = 3$. It can be easily proved that in this case k is always equal to 2. Even if this simplification seems rather restrictive, k in most of the cases, when e is small, should be considered known, as it can be easily found Heninger and Shacham [19]. Therefore, in our case, we have that:

$$3d = 1 + 2\left(n - (p+q) + 1\right)$$

Moreover, we can approximate d with $\bar{d}$, where:

$$\bar{d} = \left\lfloor \frac{2n+1}{3} \right\rfloor$$

The approximation is quite good as:

$$|\bar{d} - d| \leq k\frac{p+q}{e} \leq 2\frac{2\sqrt{n}}{3} \leq \frac{4}{3}\sqrt{n}$$

which means that almost half of the upper bits of d are known, by calculating $\bar{d}$.

Taking into consideration the above, we take the equation:

$$3d = 2n - 2(p + q) + 3$$

convert it to the respective system of CNFs and replace the values of the upper half bit of d with the correct ones. This means that we add N equations on the system, no new variables as d solely depends on p and q, and some extra information from the approximation of $\bar{d}$. Therefore, the resulting system after these additions is much easier to be solved by a common SAT solver, compared to the original one, as it is more aware of the algebraic structure and of the constraints that exist.

The selection of e to be equal to 3, might seem rather restrictive and beyond current standards, nevertheless, there are public keys still in use that contain it Lenstra et al. [23]. Moreover, we have to note that even if in the decryption we need only d, going according to PKCS #1, using p and q, we significantly improve the speed of the decryption process. Thus, in implementations, p and q are stored and used, so the assumption that the attacker may have access to part of them is valid.

4 Experimental results

4.1 Setting up the environment

The implementation is based on ToughSAT code Yuen and Bebel [42] to make the first equations and then with the proper Python script added the proposed CNFs. This script is freely available in Patsakis [32]. For solving the SAT problem, the miniSAT solver was used on a machine with Intel® Core™ i7-2600 CPU at 3.40GHz processor with 16GB of RAM, running on 64 bit Ubuntu GNU/Linux kernel 3.2.0-29. Several prior experiments showed that miniSAT was more efficient in solving this problem, compared to other solvers, like clasp and CryptoMiniSAT. The parameters that were used for miniSAT were "asymm" and "rnd-init" to enable asymmetric branching, for shrinking clauses and randomize the initial activity value respectively.

In each experiment, a key of the appropriate size is created using the Python's default random library. The generated RSA modulo number is then parsed to TaughSAT to generate the appropriate CNFs and output the relative DIMACS file. In this file we append the aforementioned equations for the upper and lower bits, as well as the estimation of the MSB half of d. The generated key is then censored, exposing exactly the requested fraction of the bits of the private key components (p, q) or (p, q, d), creating the appropriate CNFs to be appended to the DIMACS file.

4.2 The results

Table 1 summarizes the space and time requirements for generating the systems of CNFs. As it is illustrated there, generating the systems of CNFs, was quite time consuming, most of the times, it takes longer than to solve the actual problem. Therefore, in order to reduce the time needed for the tests, we reused keys with different known bits each time. Hence, for each bit size we created 100 keys and for each key we repeated the same experiment 10 times, selecting a random set of bits known each time for the components and for each of the percentages of known bits. In order to stop possible bottlenecks, we created a panic limit of 500 secs, so that if the SAT solver takes more time to solve the system, then the process gets killed. Totally, 10*100*8*6=48,000 tests were made for the measurements of this work.

The experiments were made for key sizes from 128 bits up to 1024 bits. 1536 bit keys could be generated, however, the system of CNFs is so big (1.5 GB) that couldn't be solved by miniSAT. For the case of 2048 bits, the system of CNFs could not be created by the PC.

In Heninger and Shacham [19] the percentages of random known bits for the case of primes is 57%, and if we add d in the game, the percentage is 42. Therefore, our experiments were targeted in same region and below. So for the case of primes we started from 59% and reached 53%. For the case of the triplet p, q and d the experiments involve percentages from 44% down to 38%. Further decrease in the amount of known bits, in most of the tests that were made, resulted to raising the panic alert too many times, hence these results are not included in the illustrated results.

RSA key size	Space needed	Time to create
128 bits	8.5 MB	1.3 secs
256 bits	36 MB	5.5 secs
384 bits	84 MB	13.9 secs
512 bits	150 MB	23.5 secs
768 bits	355 MB	80 secs
1024 bits	650 MB	2 mins 52 secs
1536 bits	1.5 GB	10 mins 14 secs

Table 1: Time and space requirements for creating systems of CNFs.

The results of these tests are illustrated in figures 1 and 2 and tables 2 and 3. The times that are shown refer only to the time it takes the SAT solver to solve the problem. The TaughSAT implementation, as mentioned above, is made in Python, which in terms of performance, as any other scripting language is not very efficient. Moreover, the continuous recursive calls of several functions stall the process of creating the system of CNFs. Nevertheless, it provides a nice, clean and easy to use modeling of the factorization problem to be wrapped by other scripts. Since this implementation cannot be cached to save processing time, the time for creating the system of CNFs is not considered in the experiments.

The numeric results of these experiments clearly show that SAT solvers can provide the necessary framework for partial key exposure attacks on RSA. For

	% known	Average	Max	Min	StDev	Panic limit	Upper Outliers≈	New average
128	53	0.24	0.32	0.21	0.02	0%	0%	0.24
	55	0.24	0.37	0.20	0.03	0%	0%	0.24
	57	0.24	0.36	0.19	0.02	0%	0%	0.24
	59	0.24	0.40	0.20	0.03	0%	0%	0.24
256	53	1.33	4.49	0.94	0.48	0%	1%	1.29
	55	1.32	3.90	0.93	0.55	0%	0%	1.32
	57	1.27	5.45	0.94	0.52	0%	1%	1.22
	59	1.23	4.36	0.96	0.38	0%	1%	1.20
384	53	8.38	65.69	2.55	17.78	0%	1%	3.18
	55	4.55	13.31	2.28	3.7	0%	0%	4.55
	57	2.78	4.35	2.24	0.65	0%	0%	2.78
	59	2.53	2.96	2.26	0.22	0%	0%	2.53
512	53	8.68	180.97	4.42	17.79	0%	2%	6.68
	55	6.16	13.14	4.37	1.76	0%	0%	6.16
	57	6.87	38.94	4.31	4.64	0%	2%	6.87
	59	8.72	214.98	4.34	20.96	0%	1%	6.64
768	53	29.21	450.81	11.58	57.55	1%	6%	17.12
	55	29.16	490.42	10.97	59.56	1%	5%	18.35
	57	18.10	53.84	11.17	7.71	1%	0%	18.10
	59	22.99	257.36	11.05	30.66	1%	5%	17.08
1024	53	43.38	267.21	20.96	36.91	3%	2%	38.75
	55	50.72	379.28	21.28	59.05	5%	4%	39.85
	57	41.35	362.92	19.51	39.01	0%	2%	36.26
	59	46.42	463.72	22.61	54.81	5%	3%	38.19

Table 2: Partial information for p and q.

Upper outliers are the values above 3 times the average value. Panic limit is set to 500 secs.

	% known	Average	Max	Min	StDev	Panic limit	Upper Outliers	New average
128	38	0.25	0.50	0.21	0.04	0%	0%	0.25
	40	0.24	0.42	0.20	0.03	0%	0%	0.24
	42	0.25	0.95	0.20	0.07	0%	1%	0.24
	44	0.24	0.54	0.20	0.04	0%	0%	0.24
256	38	1.42	12.55	0.95	1.22	0%	2%	1.28
	40	1.29	3.76	0.93	0.41	0%	0%	1.29
	42	1.40	19.72	0.97	1.86	0%	1%	1.22
	44	1.27	3.31	0.95	0.38	0%	0%	1.27
384	38	4.67	12.38	2.75	2.64	0%	1%	4.12
	40	3.79	5.49	2.7	0.9	0%	0%	3.79
	42	2.81	3.59	2.31	0.39	0%	0%	2.81
	44	2.74	3.7	2.41	0.39	0%	0%	2.74
512	38	6.88	29.28	4.37	3.79	0%	2%	6.49
	40	8.68	99.30	4.32	10.93	0%	4%	6.89
	42	6.51	33.28	4.18	4.06	0%	2%	5.97
	44	7.84	138.24	4.34	13.68	0%	3%	6.10
768	38	26.71	432.53	11.62	48.46	5%	3%	19.34
	40	24.54	298.20	10.96	36.30	0%	4%	18.48
	42	24.13	376.65	10.94	42.87	2%	3%	17.62
	44	21.56	145.45	11.14	20.09	1%	5%	17.51
1024	38	60.41	370.17	19.93	70.22	2%	9%	40.37
	40	57.91	442.33	20.48	79.77	2%	7%	37.51
	42	46.52	318.06	21.44	47.97	0%	6%	35.46
	44	42.60	432.64	21.05	44.84	1%	3%	36.20

Table 3: Partial information for p, q and d.

Upper outliers are the values above 3 times the average value. Panic limit is set to 500 secs.

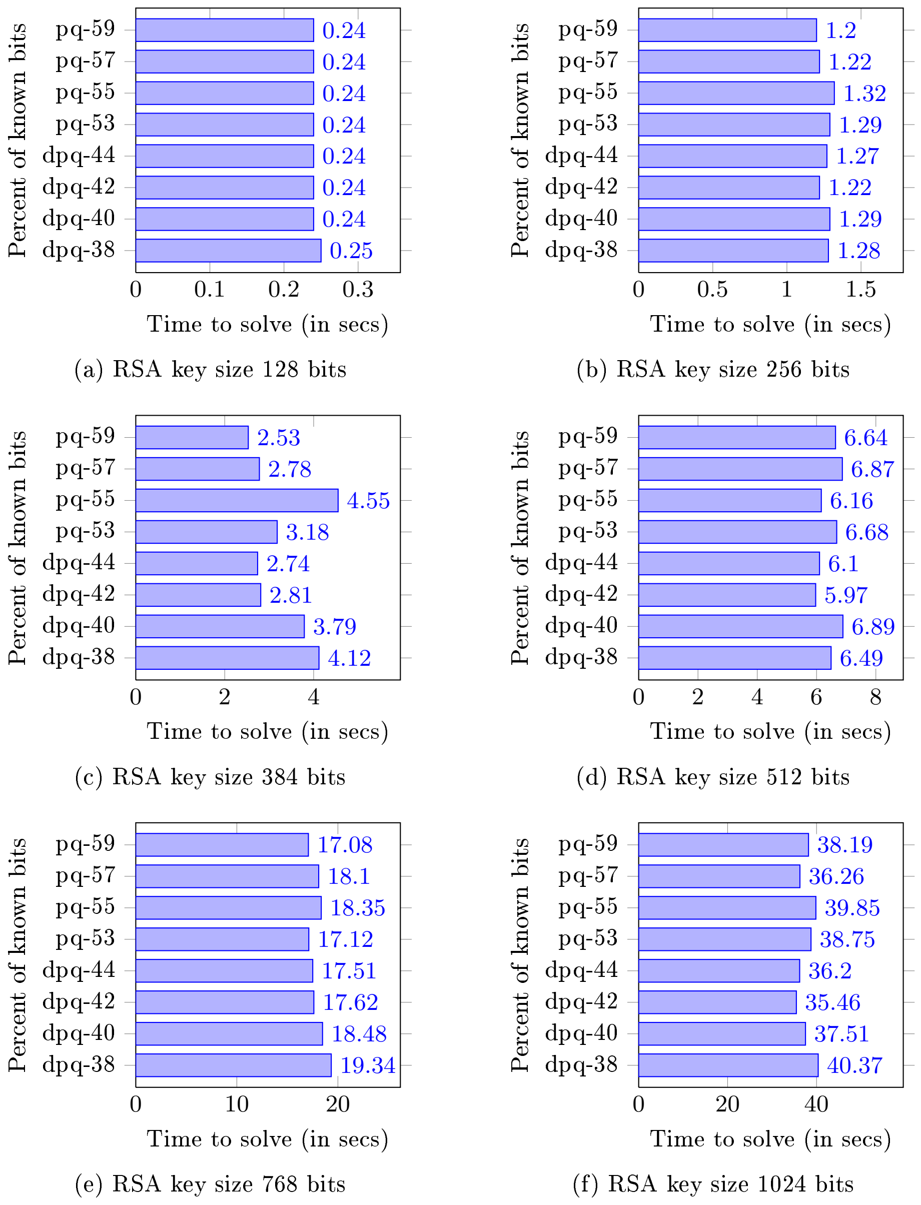

Fig. 1: Recovering private key of RSA having partial information about the private key.

The illustrated time, is the average time without the outliers and the experiments that reached the panic limit.

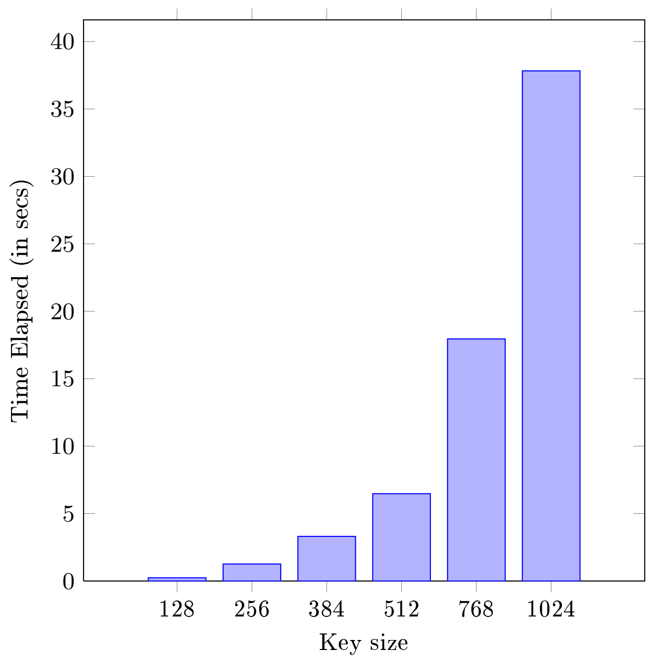

Fig. 2: Summary of the results for all bit sizes.
The illustrated time, is the average time for every bitsize without the outliers and the experiments that reached the panic limit, combining all the experiments of partial information about p, q and d.

the case of 1024 bits, with only 38% of the bits of p, q and d known, 40secs to find the key is a very good timing for practical applications. As expected, for RSA keys of 1024 bits, we meet the most measurements which either reach the panic limit, or belong to the upper outliers.

4.3 Discussion

Surprisingly, after discarding the outliers, most of the means values are very close independently of the key size, showing in one sense that these problems are moreover of the same difficulty. As expected, in most cases, the more the information is disclosed, the less time is needed to find the key. However, the difference is not so big in many cases.

From Figure 2, it is quite clear that 50% percent increase in the size of the integers to be factored, results to doubling the expected time. Therefore, we can infer that the attack behaves exponentially to the key length, however, for key-sizes up to 1024 bits, the average elapsed time is acceptable for practical attacks.

The time needed to generate the systems of equations as well as the space needed to store them, is multiplied by a factor round 3 and 4 with every 50% increase of the key size, showing again an exponential nature.

It is worth to note here, that for several cases of the experiments which reached the panic limit, we stored the DIMACS file to process it later without the time limit. Many of these systems were solved in less than 30mins, while there where others, needing more than 1 hour to be solved. This fact shows that even if the problem is very demanding in many cases it can be solved.

5 Conclusions

The main aim of this research was to show that SAT solvers can be a useful tool in cryptanalyst's arsenal for the case of RSA, by providing a set of proof of concept attacks, therefore, the implementations were not fully optimized. Nonetheless, the experimental results not only show that such attacks are possible, something that wasn't studied so far, but they are very powerful as well.

5.1 Limitations

Even if the proposed model, provides real-world attacks within reasonable time frame, there are three main drawbacks, which mainly stem from the implementation of the attack and not from its nature:

Time: In the attack, each time we have to construct the CNFs from the begging and then parse them to the SAT solver. For normal keys, eg 1024 bits, this cat take some minutes. However, this part could be cached, and the appropriate values would be changed on each use.

Space: The proposal is quite "greedy" in terms of space, as the CNFs that are produced for the equations are very lengthy. For example for the case of 1024 bit keys to a system of equations around 700MB.

Memory: In order to create the CNFs for 1024 bit keys, 4GB of RAM are not enough.

Obviously, the attack can be extended for other e as well, since as discussed before, the value of k can be considered most of the times as known, or to be selected from a small set, as in most practical applications the public exponent is very low. The latter could result to an unsatisfiable problem if the value of k correctly selected, however, this has not yet been tested.

Comparing the time to the results of Heninger and Shacham [19], the proposed reconstruction is not so efficient. Nevertheless, this is the first such approach for SAT solvers in this field and the time is at least within the reasonable bounds of several seconds. As previously discussed, the main delay is not the actual attack, but the conversion of the problem to its SAT instance. The current implementation does not support caching. This means that every time that we want to launch the attack, we have to generate the problem from scratch and not change the values in specific positions, drastically increasing the performance of the procedure. Moreover, the creation of the DIMACS file was made using Python, which is a scripting language, hence, the use of C or another language could further speedup the process and decrease the memory needs enabling us to test longer keys.

Finally, we have to notice that due to the nature of SAT solvers, they cannot stop when for example they have successfully recovered a part of the LSBs or MSBs of the key in order to use Coppersmith's theorem and recover the whole key, something that will of course boost their efficiency, but they have to find the values of all the rest variables. On this light, their timings are rather fast and show that further improvements can be achieved.

5.2 Future work

Based on this work, the main things that have to be studied in the future are the following:

- Extend the results for other public exponents e.
- Improve the current code for
  - faster generation of the systems of equations,
  - better CNF conversion, so that the systems are smaller.
- Try to decrease even more the number of the random known bits.
- SAT solvers
  - Probably fine tuning the parameters of miniSAT could result to faster attacks.
  - Test other SAT solvers. Even though other SAT solvers, which generally outperform miniSAT were tested, miniSAT proved to be faster. However testing with other solvers and parameters could significantly improve the performance of the illustrated attacks.

Bibliography

[1] Johannes Blömer and Alexander May. New partial key exposure attacks on rsa. In *CRYPTO*, pages 27–43, 2003.

[2] 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.

[3] D. Brumley and D. Boneh. Remote timing attacks are practical. In *Proceedings of the 12th conference on USENIX Security Symposium-Volume 12*, pages 1–1. USENIX Association, 2003.

[4] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[5] Don Coppersmith. Finding a small root of a univariate modular equation. In *EUROCRYPT*, pages 155–165, 1996.

[6] Nicolas T. Courtois, Gregory V. Bard, and David Wagner. Fast software encryption. chapter Algebraic and Slide Attacks on KeeLoq, pages 97–115. Springer-Verlag, Berlin, Heidelberg, 2008.

[7] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960. ISSN 0004-5411.

[8] Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using sat solvers. In *Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, SAT'07, pages 377–382, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] V. I. Dylkeyt, R. T. Faizullin, and I. G. Khnykin. Reducing the problem of asymmetric ciphers cryptanalysis to solving satisfiability problems. In *Proceedings of the XIII All-Russian Conference Mathematical Methods in Pattern Recognition*, pages 249–251. MAKC press, 2007.

[10] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking bivium using sat solvers. In *Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, SAT'08, pages 63–76, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] Jeremy Erickson, Jintai Ding, and Chris Christensen. Algebraic cryptanalysis of sms4: gröebner basis attack and sat attack compared. In *Proceedings of the 12th international conference on Information security and cryptology*, ICISC'09, pages 73–86, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] Matthias Ernst, Ellen Jochemsz, Alexander May, and Benne de Weger. Partial key exposure attacks on rsa up to full size exponents. In *EUROCRYPT*, pages 371–386, 2005.

[13] R. T. Faizullin, I. G. Khnykin, and V. I. Dylkeyt. The sat solving method as applied to cryptographic analysis of asymmetric ciphers. *The Computing Research Repository*, abs/0907.1755, 2009.

[14] C. Fiorini, E. Martinelli, and F. Massacci. How to fake an rsa signature by encoding modular root finding as a sat problem. *Discrete Applied Mathematics*, 130(2):101–127, 2003.

[15] Ian P. Gent and Toby Walsh. The sat phase transition. pages 105–109. John Wiley & Sons, 1994.

[16] Philippe Golle and David Wagner. Cryptanalysis of a cognitive authentication scheme (extended abstract). In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 66–70, Washington, DC, USA, 2007. IEEE Computer Society.

[17] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Cal, Ariel J. Feldman, and Edward W. Felten. Least we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, 2008.

[18] Wilko Henecka, Alexander May, and Alexander Meurer. Correcting errors in rsa private keys. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 351–369, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] Nadia Heninger and Hovav Shacham. Reconstructing rsa private keys from random key bits. In *In CRYPTO*, pages 1–17, 2009.

[20] Ekawat Homsirikamol, Pawel Morawiecki, Marcin Rogawski, and Marian Srebrny. Security margin evaluation of sha-3 contest finalists through sat-based attacks. In *Computer Information Systems and Industrial Management*, volume 7564 of *Lecture Notes in Computer Science*, pages 56–67. Springer Berlin Heidelberg, 2012.

[21] A.A. Kamal and A.M. Youssef. Applications of sat solvers to aes key recovery from decayed key schedule images. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pages 216 –220, july 2010.

[22] Noboru Kunihiro, Naoyuki Shinohara, and Tetsuya Izu. Recovering rsa secret keys from noisy key bits with erasures and errors. 2012. `http://eprint.iacr.org/`.

[23] A.K. Lenstra, J.P. Hughes, M. Augier, J.W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, whit is right. *IACR eprint archive*, 64, 2012.

[24] L.A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[25] Subhamoy Maitra, Santanu Sarkar, and Sourav Sen Gupta. Factoring rsa modulus using prime reconstruction from random known bits. In *Proceedings of the Third international conference on Cryptology in Africa*, AFRICACRYPT'10, pages 82–99, Berlin, Heidelberg, 2010. Springer-Verlag.

[26] Fabio Massacci. Using walk-sat and rel-sat for cryptographic key search. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 290–295, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[27] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a sat problem. *J. Autom. Reason.*, 24(1-2):165–203, February 2000. ISSN 0168-7433.

[28] Ilya Mironov and Lintao Zhang. Applications of sat solvers to cryptanalysis of hash functions. In *Proceedings of the 9th international conference on Theory and Applications of Satisfiability Testing*, SAT'06, pages 102–115, Berlin, Heidelberg, 2006. Springer-Verlag.

[29] M.S.E. Mohamed, S. Bulygin, and J. Buchmann. Using sat solving to improve differential fault analysis of trivium. pages 62–71. Springer, 2011.

[30] Pawel Morawiecki and Marian Srebrny. A sat-based preimage analysis of reduced keccak hash functions. `http://eprint.iacr.org/2010/285`, 2010. pawelm@wsh-kielce.edu.pl 14742 received 13 May 2010.

[31] K. Paterson, A. Polychroniadou, and D. Sibborn. A coding-theoretic approach to recovering noisy rsa keys. *Advances in Cryptology–ASIACRYPT 2012*, pages 386–403, 2012.

[32] Constantinos Patsakis. Rsat. `https://sourceforge.net/projects/rsat`, January 18, 2013.

[33] Paul Purdom and Amr Sabry. Cnf generator for factoring problems. `http://www.cs.indiana.edu/cgi-pub/sabry/cnf.html`, November 23, 2007.

[34] Ronald L. Rivest and Adi Shamir. Efficient factoring based on partial information. In *EUROCRYPT*, pages 31–34, 1985.

[35] Sarkar Santanu, Gupta Sourav Sen, and Maitra Subhamoy. Reconstruction and Error Correction of RSA Secret Parameters from the MSB Side. In *WCC 2011 - Workshop on coding and cryptography*, pages 7–16, Paris, France, April 2011. URL `http://hal.inria.fr/inria-00607242`.

[36] Santanu Sarkar. Partial key exposure: Generalized framework to attack rsa. In *Progress in Cryptology - INDOCRYPT 2011*, volume 7107 of *Lecture Notes in Computer Science*, pages 76–92. Springer Berlin / Heidelberg, 2011.

[37] Bart Selman and Henry Kautz. Domain-independent extensions to gsat: solving large structured satisfiability problems. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1*, IJCAI'93, pages 290–295. Morgan Kaufmann Publishers Inc., 1993.

[38] Mate Soos. Cryptominisat - a sat solver for cryptographic problems. `http://planete.inrialpes.fr/~soos/CryptoMiniSat2/index.php`, 2009.

[39] Mate Soos. Grain of Salt — an Automated Way to Test Stream Ciphers through SAT Solvers. In *Tools'10: Proceedings of the Workshop on Tools for Cryptanalysis 2010*, pages 1–2, RHUL, 2010.

[40] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 244–257, Berlin, Heidelberg, 2009. Springer-Verlag.

[41] Benjamin W. Wah, Yi Shang, and Zhe Wu. Discrete lagrangian method for optimizing the design of multiplierless qmf filter banks. In *IEEE Transactions on Circuits and Systems, Part II*, pages 529–538. IEEE, 1997.

[42] Henry Yuen and Joseph Bebel. Toughsat. `http://toughsat.appspot.com`, July 18, 2011.