

# A Hybrid SAT and Lattice Reduction Approach for Integer Factorization

Yameen Ajani<sup>1</sup>, Curtis Bright<sup>1</sup>

<sup>1</sup>University of Windsor, Windsor, Ontario, Canada

## Abstract

The difficulty of factoring large integers into primes is the basis for cryptosystems such as RSA. Due to the immense popularity of RSA there have been many proposed attacks on the factorization problem such as side-channel attacks where some bits of the prime factors are available. When enough bits of the prime factors are known, two methods that are effective at solving the factorization problem are satisfiability (SAT) solvers and Coppersmith's method. The SAT approach reduces the factorization problem to a Boolean satisfiability problem, while Coppersmith's approach uses lattice basis reduction. Both methods have their advantages, but they also have their limitations: Coppersmith's method does not apply when the known bit positions are randomized, while SAT-based methods can take advantage of known bits in arbitrary locations but have no knowledge of the algebraic structure exploited by Coppersmith's method. This work is the first to explore the potential of using a hybrid SAT and computer algebra approach to efficiently solve random leaked-bit factorization problems. Specifically, Coppersmith's method is invoked by a SAT solver to determine whether a partial bit assignment can be extended to a complete assignment. Our preliminary results demonstrate that this augmentation improves the efficiency of the solver by orders of magnitude.

## Keywords

Factoring, SAT, Lattice Basis Reduction, Cryptography, RSA, Coppersmith's Method

## 1. Introduction

Integer factorization is a fundamental problem in mathematics and computer science with wide-ranging applications in cryptography, coding theory, and number theory. The difficulty of factoring large integers is the basis for cryptosystems such as RSA [1], which relies on the fact that it is hard to factor the product of two large prime numbers. As a result, integer factorization has been the subject of intense research for decades. Many algorithms have been developed to tackle this problem, some of which rely on additional information that may be leaked through side-channel attacks.

Side-channel attacks are a class of attacks that aim to exploit information that is unintentionally leaked by a computer system or a device during its normal operation. Cold boot attacks are a type of side-channel attack exploiting the information remaining in the random-access memory (RAM) of a computer system even after it has been powered off and then back on


---


*8th International Workshop on Satisfiability Checking and Symbolic Computation, July 28, 2023, Tromsø, Norway, Collocated with ISSAC 2023*

✉ [ajaniy@uwindsor.ca](mailto:ajaniy@uwindsor.ca) (Y. Ajani); [cbright@uwindsor.ca](mailto:cbright@uwindsor.ca) (C. Bright)

🌐 <http://www.curtisbright.com/> (C. Bright)

🆔 0000-0002-0462-625X (C. Bright)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

again. Halderman et al. [2] demonstrated that this remanence effect makes possible practical and nondestructive attacks that recover some bits of secret keys stored in a computer’s memory.

In 2013, Patsakis [3] demonstrated that information obtained through cold boot attacks could be utilized to reconstruct RSA private keys with partial key exposure via the usage of Boolean satisfiability (SAT) solvers. The cold boot attack retrieves some bits of the two primes  $p$  and  $q$  and the decryption exponent used in RSA. With this information, he created SAT instances that when solved would determine the bits of the factors  $p$  and  $q$ .

A separate approach to the factorization problem, when partial information about the factors is known, was proposed by Coppersmith [4]. Coppersmith’s method uses lattice basis reduction to factor integers in polynomial time when enough bits of one of the factors is known and the unknown bits are consecutive.

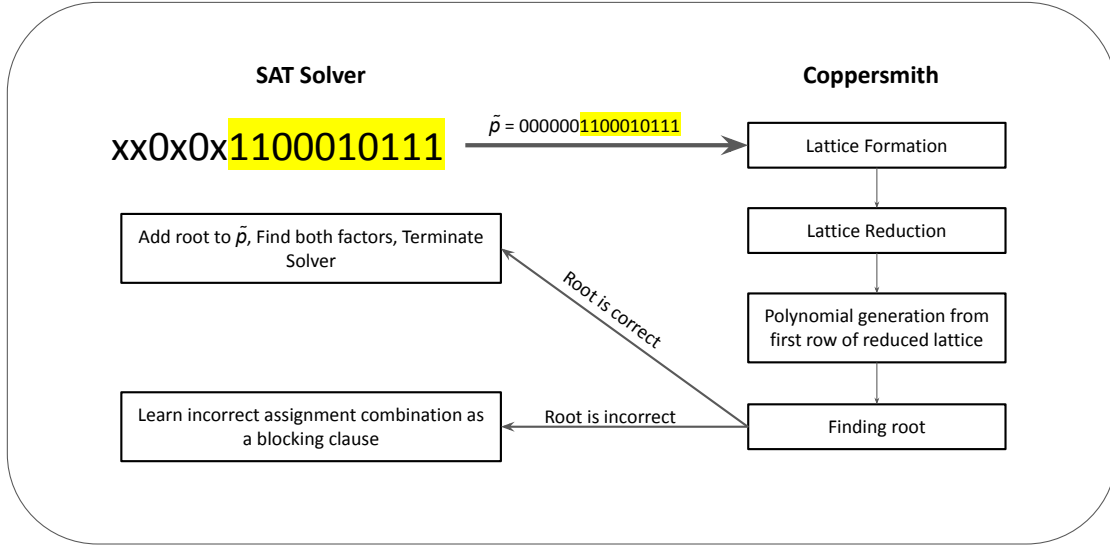
**The SC<sup>2</sup> Project.** Combining SAT with computer algebra systems (CAS) was proposed in 2015 by E. Ábrahám [5, 6] and independently by Zulkoski et al. [7]. Soon afterwards, the “SC<sup>2</sup> project” [8] started with the aim of facilitating connections between the satisfiability checking and symbolic computation communities. Many successful applications have arisen as a result of this connection: in particular, see M. England’s summary [9] for an overview of progress up to 2021, and Bright et al.’s summary [10] of MathCheck, a system that queries a CAS from inside a SAT solver. These works show the power of using SAT solvers in combination with symbolic computation and inspire our SAT+CAS approach for the factorization problem. More precisely, we explore the potential of a hybrid approach that combines SAT solvers with Coppersmith’s method—the first attempt to investigate the effectiveness of such a hybrid approach.

## 2. Proposed Method

We focus on the RSA factoring problem in this work. Hence, we consider two primes,  $p$  and  $q$ , of the same bitlength. The task is to factor the semiprime  $N = p \cdot q$ . We assume that a certain percentage of bits of both the primes is known. This is a strong assumption, but in practice, a cold boot attack may leak this extra information [2]. However, we do not presume that the attack has the ability to control which bits are known and suppose the known bits are distributed uniformly at random.

Our approach combines a SAT solver with Coppersmith’s method for integer factorization. Coppersmith’s algorithm is an approach that uses lattice basis reduction for finding small solutions to polynomials modulo an integer in polynomial time [4, 11]. In particular, it can factor  $N$  when at least 50% of the most or least significant bits (MSB/LSBs) of  $p$  are known. For example,  $p$  can be written as  $p = \tilde{p} + x_0$  where  $\tilde{p}$  is an integer that has at least 50% of the same MSBs as  $p$  and  $x_0$  is an integer that encodes the unknown low bits of  $p$ . As an example (using decimal digits instead of binary digits for simplicity), if  $p = 2837$  and  $\tilde{p} = 2830$  then  $x_0 = 7$ .

Coppersmith’s method constructs a lattice where every vector in the lattice corresponds to a polynomial having  $x_0$  as a root modulo  $N$ , the number to factor. If the vector is short enough then  $x_0$  will also be a root of its associated polynomial *over the integers*, not just modulo  $N$ . Since the integer roots of a polynomial can be computed in polynomial time [12] this reduces the problem of finding  $x_0$  to the problem of finding a short vector in Coppersmith’s lattice. This is accomplished with a lattice basis reduction algorithm such as Lenstra–Lenstra–Lovász’s LLL



**Figure 1:** A diagram of our SAT+CAS method for the factorization problem. Coppersmith’s method is invoked whenever at least 60% of the low bits of  $p$  are assigned. If the low bits of  $p$  were set correctly then Coppersmith’s method will reveal its high bits and the solver terminates. If the low bits were set incorrectly, then Coppersmith’s method will fail and a “blocking clause” is learned telling the solver to backtrack and try a new assignment of bits.

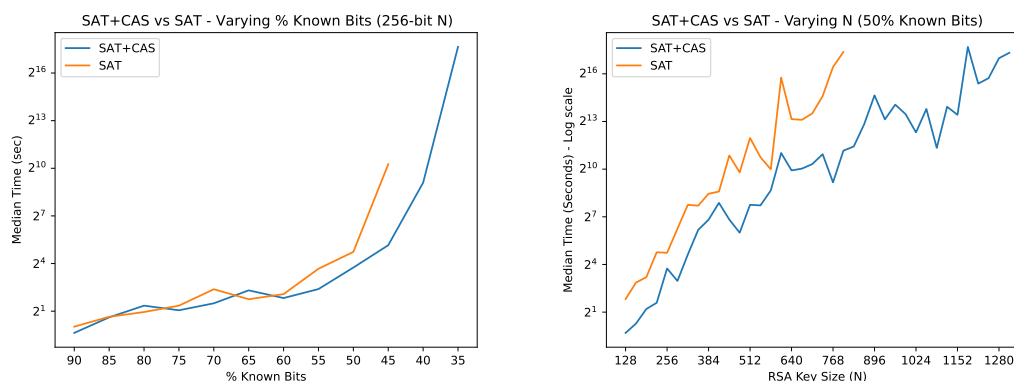
algorithm [13].

Because Coppersmith’s method requires the unknown bits of the prime  $p$  to be consecutive, it cannot directly be used in the case when the known bits of  $p$  are randomly distributed. Thus, our method starts with a SAT encoding of the factorization problem [14] allowing the leaked bits to be given to the solver as unit clauses. However, the SAT solver alone will not exploit the algebraic properties used by Coppersmith’s method. Thus, in order to achieve the best of both worlds we call Coppersmith’s method from within the SAT solver whenever the solver’s current partial assignment has assigned values to the top 60% of the bits of  $p$ . Even though Coppersmith’s method works when 50% LSBs of  $p$  are known, we call Coppersmith when 60% LSBs are known to reduce the number of calls to Coppersmith. Figure 1 visually depicts how the technique works.

### 3. Implementation and Results

The implementation uses a programmatic version of MapleSAT [15] developed for the Math-Check project [16]. The conjunctive normal form (CNF) instances are generated using the *CNF Generator for Factoring Problems* by P. Purdom and A. Sabry [17]. The version of Coppersmith’s algorithm used is a custom implementation in C++ using the GMP [18], MPFR [19], fplll [20], and FLINT [21] libraries.

Ten randomly generated instances encoding the factorization problem of a 256-bit semiprime  $N$  were generated and each instance was tested with known bit percentages ranging from 30–90%. The hardest instances show an orders-of-magnitude decrease in the running time



**Figure 2:** The left plot shows a comparison of the running time of the SAT and SAT+CAS approaches on a 256-bit factorization problem using a varying percentage of known bits. The right plot compares the running time for different sizes of  $N$  and 50% known bits. Both plots show times on a logarithmic scale.

when the hybrid SAT+CAS approach is compared with the SAT solver alone. For instance, with 45% known bits the median runtime of the SAT solver by itself across the ten random instances was 1220.8 seconds, while the median runtime of the SAT+CAS solver was 35.6 seconds. In these instances Coppersmith was called a median of 745 times with a mean running time of 2 milliseconds in each execution. Plots of the running times we observed are shown in Figure 2. The plot on the right shows how the running time varies with different bit sizes of  $N$  when 50% of bits of both the primes are randomly set. For example, a 1024-bit  $N$  can be factored in a median of about 85 minutes. This is significantly faster than the SAT approach and brute-force guessing (even using Coppersmith to speed up the guessing process), given that the lower-half of  $p$  will contain around 128 unknown bits.

## References

- [1] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Communications of the ACM* 21 (1978) 120–126. doi:10.1145/359340.359342.
- [2] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, E. W. Felten, Lest we remember: Cold-boot attacks on encryption keys, *Commun. ACM* 52 (2009) 91–98. doi:10.1145/1506409.1506429.
- [3] C. Patsakis, RSA private key reconstruction from random bits using SAT solvers, *IACR Cryptol. ePrint Arch.* 2013 (2013) 26.
- [4] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerabilities, *J. Cryptology* 10 (1997) 233–260. doi:10.1007/s001459900030.
- [5] E. Ábrahám, Building bridges between symbolic computation and satisfiability checking, in: K. Yokoyama, S. Linton, D. Robertz (Eds.), *Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2015, Bath, United Kingdom, July 6–9, 2015*, ACM, 2015, pp. 1–6. doi:10.1145/2755996.2756636.

- [6] E. Abraham, Symbolic computation techniques in satisfiability checking, in: 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, 2016, pp. 3–10. doi:10.1109/synasc.2016.014.
- [7] E. Zulkoski, V. Ganesh, K. Czarnecki, MathCheck: A math assistant via a combination of computer algebra systems and SAT solvers, in: Automated Deduction - CADE-25, Springer International Publishing, 2015, pp. 607–622. doi:10.1007/978-3-319-21401-6\_41.
- [8] E. Ábrahám, J. Abbott, B. Becker, A. M. Bigatti, M. Brain, B. Buchberger, A. Cimatti, J. H. Davenport, M. England, P. Fontaine, S. Forrest, A. Griggio, D. Kroening, W. M. Seiler, T. Sturm, Satisfiability checking and symbolic computation, *ACM Communications in Computer Algebra* 50 (2017) 145–147. doi:10.1145/3055282.3055285.
- [9] M. England, SC-Square: Overview to 2021, in: C. Bright, J. Davenport (Eds.), Proceedings of the 6th SC-Square Workshop, 2022, pp. 1–6. URL: <https://ceur-ws.org/Vol-3273/invited1.pdf>.
- [10] C. Bright, I. Kotsireas, V. Ganesh, When satisfiability solving meets symbolic computation, *Communications of the ACM* 65 (2022) 64–72. doi:10.1145/3500921.
- [11] N. Howgrave-Graham, Finding small roots of univariate modular equations revisited, in: IMA Conference on Cryptography and Coding, 1997, pp. 131–142. doi:10.1007/BFb0024458.
- [12] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, 2013. doi:10.1017/cbo9781139856065.
- [13] A. K. Lenstra, H. W. Lenstra, L. Lovász, Factoring polynomials with rational coefficients, *Mathematische Annalen* 261 (1982) 515–534. doi:10.1007/bf01457454.
- [14] D. E. Knuth, *The art of computer programming, Volume 4, Fascicle 6: Satisfiability*, Addison-Wesley Professional, 2015. URL: <https://dl.acm.org/doi/abs/10.5555/2898950>.
- [15] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, Learning rate based branching heuristic for SAT solvers, in: Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5–8, 2016, Proceedings, 2016, pp. 123–140. doi:10.1007/978-3-319-40970-2\_9.
- [16] C. Bright, V. Ganesh, A. Heinle, I. Kotsireas, S. Nejati, K. Czarnecki, MATHCHECK2: A SAT+CAS verifier for combinatorial conjectures, in: *Computer Algebra in Scientific Computing*, Springer International Publishing, 2016, pp. 117–133. doi:10.1007/978-3-319-45641-6\_9.
- [17] P. Purdom, A. Sabry, Cnf generator for factoring problems, 2003. <https://cgi.luddy.indiana.edu/~sabry/cnf.html>.
- [18] T. Granlund, the GMP development team, GNU MP: The GNU Multiple Precision Arithmetic Library, 5.0.5 ed., 2012. <http://gmplib.org/>.
- [19] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélicier, P. Zimmermann, MPFR: A multiple-precision binary floating-point library with correct rounding, *ACM Trans. Math. Softw.* 33 (2007) 13–es. doi:10.1145/1236463.1236468.
- [20] The fplll development team, fplll, a lattice reduction library, Version: 5.4.4, 2023. Available at <https://github.com/fplll/fplll>.
- [21] W. Hart, F. Johansson, S. Pancratz, FLINT: Fast Library for Number Theory, 2013. Version 2.9.0, <https://flintlib.org>.