M.Sc. Thesis Defence

# A Hybrid SAT and Lattice Reduction Approach for Integer Factorization

Yameen Ajani

| | | |
|---|---|---|
| **Internal Reader** | - | Dr. Ahmad Biniaz |
| **External Reader** | - | Dr. Ilya Shapiro |
| **Supervisor** | - | Dr. Curtis Bright |
| **Chair** | - | Dr. Jessica Chen |

# Publications

- Yameen Ajani and Curtis Bright. 2024. SAT and Lattice Reduction for Integer Factorization. In Proceedings of International Symposium on Symbolic and Algebraic Computation 2024 (ISSAC 2024). ACM, New York, NY, USA, 9 pages - **Accepted, To be published**

- Ajani, Y. and Bright, C. (2023). A hybrid SAT and lattice reduction approach for integer factorization. Proceedings of the 8th SC-Square Workshop co-located with the 48th International Symposium on Symbolic and Algebraic Computation, SC-Square@ISSAC 2023, Tromsø, Norway, July 28, 2023, volume 3455 of CEUR Workshop Proceedings, pages 39–43. CEUR-WS.org.

# Why this problem?

- Factoring is a problem that has been around for a long time.
- The difficulty of factoring large numbers increases as the numbers get larger. The Number Field Sieve doesn't run in polynomial time.
- This makes it a challenging problem to solve even with the most powerful computers.
- This complexity of the factoring problem is the basis for many cryptographic algorithms like RSA used in practical applications.

# What is SAT?

- SAT or Boolean Satisfiability Problem provides us with the answer whether a given propositional logic statement has any combination of variable assignments such that the statement evaluates to TRUE.
- SAT is an NP-Hard Problem.
- Even though NP-hard problems are considered impractical to be solved by current computer systems, SAT solvers have proven to solve very large SAT instances quite efficiently.
- SAT solvers require that the logical expression to be solved be in the Conjunctive Normal Form (CNF).

# What is CNF?

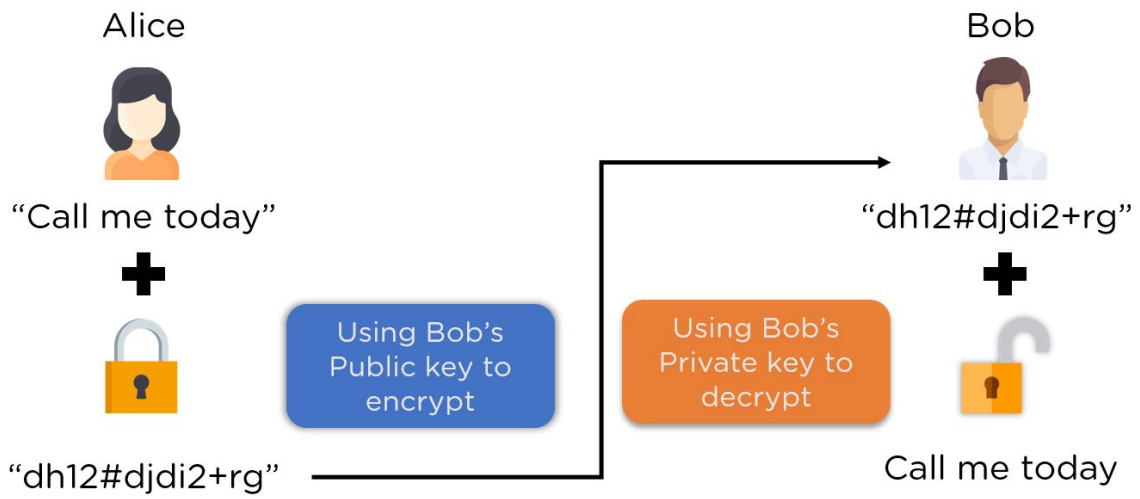
- A ***literal*** is a single variable or a negated variable. For example, $a$ and $\neg a$.
- A ***clause*** is a disjunction of literals (literals connected by OR). Example of a clause is $(p \vee q)$.
- A statement is in CNF if it is a **conjunction of clauses** (clauses connected by AND). For instance –

  $(p \vee q) \wedge (\neg r \vee s)$ is in CNF;

  $(\neg a \vee (b \wedge c))$ is not in CNF.

# What is RSA?

- RSA is an asymmetric cryptography (public & private key) technique developed by Rivest, Shamir, and Adleman.
- Reliability of RSA is dependent upon the difficulty to factorize large integers.




**Notations -**

| | |
|---|---|
| *p, q* | Two large primes |
| *N* *(N = p*q)* | Semi-prime |
| *e* | Public exponent |
| *d* | Private exponent |

# RSA Key Reconstruction

- A groundbreaking work in the field of RSA key reconstruction was presented by Heninger and Shacham in 2009 [1].
- Discusses a method for reconstructing the RSA private key using random known bits of the key.
- The proposed technique is a **smart brute-force method**.
- Based on what important about the private components is known, the authors have given the requirement for known bits to successfully factor the semi-prime in polynomial time -
  - 57% of the bits of $p$ and $q$
  - 42% of the bits of $p$, $q$ and $d$

[1] - Heninger et al. Reconstructing RSA Private Keys from Random Key Bits. CRYPTO 2009.

# 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 that exploit the information that remains in the random-access memory (RAM) of a computer system even after it has been powered off and then back on again.
- Halderman et al. [2], demonstrated that this DRAM remanence effect makes possible practical, nondestructive attacks that recover some bits of secret keys stored in a computer's memory.

[2] - Halderman et al. Lest we remember: Cold boot attacks on encryption keys. Proceedings of USENIX Security. 2008

# Side-Channel Attacks & SAT

- In 2013, Patsakis [3] demonstrated that information obtained through cold boot attacks could be utilized to reconstruct the RSA private key with partial key exposure by providing it to SAT solvers.
- The cold boot attack can be used to retrieve some bits of the two primes ($p$ and $q$) as well as the private exponent ($d$) used in RSA.
- With this information, he created CNF instances that were then given to the SAT solver to find an assignment that satisfies all the given constraints.
- He tested different key sizes as well as varying percentages of known bits.

[3] - C. Patsakis. RSA private key reconstruction from random bits using SAT solvers. IACR Cryptol. ePrint Arch. 2013

# The SAT+CAS Revolution

- Proposed by Ábráham [4] and Zulkoski et al. [5]
- Brings together Satisfiability Solving and Symbolic Computation domains.
- Proven to be useful in a lot of domains like digital circuit design, combinatorics, etc.
- For instance, Bright et al. [6, 7] used a SAT+CAS implementation to resolve combinatorial conjectures.
- Initiatives like the SC-Square project are focused on promoting research in this area.

[6] - C. Bright et al. The SAT+CAS method for combinatorial search with applications to best matrices. Ann Math Artif Intell 87. 2019
[7] - C. Bright et al. When Satisfiability Solving Meets Symbolic Computation. Communications of the ACM. 2022.

# Conflict Driven Clause Learning (CDCL)

- A modern and highly efficient SAT solver technique.
- Enhances the basic DPLL (Davis-Putnam-Logemann-Loveland) algorithm [8].
- Key features: learning from conflicts, non-chronological backtracking, and variable selection heuristics.
- Solvers using CDCL - MapleSAT, CaDiCaL, etc.

[8] - Davis et al. A Machine Program for Theorem-Proving. Commun. ACM, 5(7):394–397.

# Basic Concepts of CDCL

- **Conflict**: A situation where no assignment can satisfy the current set of clauses.
- **Clause Learning**: The process of recording conflicts to prevent re-exploration of the same conflicting states.
- **Non-Chronological Backtracking**: Backtracks to the most recent decision point that is relevant to the conflict.
- **Heuristics**: Techniques used to select the next variable to assign, such as VSIDS and Learning Rate Based branching heuristics.

# What is Coppersmith's Algorithm?

- Coppersmith's algorithm is a mathematical method that uses lattice basis reduction for finding small solutions to polynomials modulo an integer in **polynomial time** [9,10].
- This method requires at least 50% consecutive high or low bits of one of the primes to be known.
- $p = F(x) = \tilde{p} + x$

  $\tilde{p}$ is an integer that has, **at least**, the same 50% MSBs as $p$.

  $p = F(x) = x + (2^{-m} \bmod N) \cdot \tilde{p}$

  $\tilde{p}$ is an integer that has the same "*m*" LSBs as $p$.

[9] - D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities, J. Crypt. 10. 1997
[10] - N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. IMA Cryptography and Coding, Springer. 1997

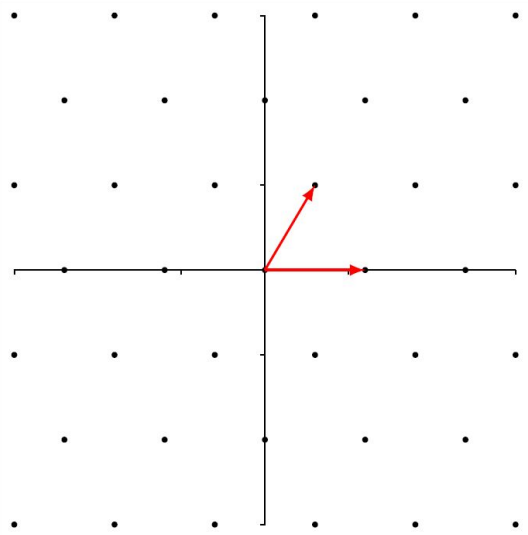
# What is Coppersmith's Algorithm?

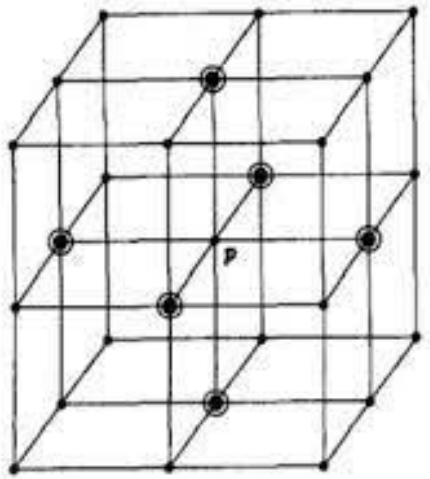
Let's say we know 60% MSBs of $p$ like
$p = 100111xxxx.$
In this case, $\tilde{p} = \mathbf{1001110000}$ and we can write
$p = F(x) = \tilde{p} + x = 624 + x$

On the other hand, if we know 60% of the LSBs of $p$ such that
$p = xxxx110001.$
Now, $\tilde{p} = \mathbf{110001}$, $m = 6$ and we have
$p = F(x) = x + (2^{-m} \mod N) \cdot \tilde{p} = x + (2^{-6} \mod N) \cdot 49$

# What is a Lattice?

- A lattice is a collection of points in space that are arranged in a regular pattern, like the points on a grid.
- The points in a lattice can be described using vectors, which are basically just arrows that point from the origin point (0, 0, ..., 0) to each of the lattice points.

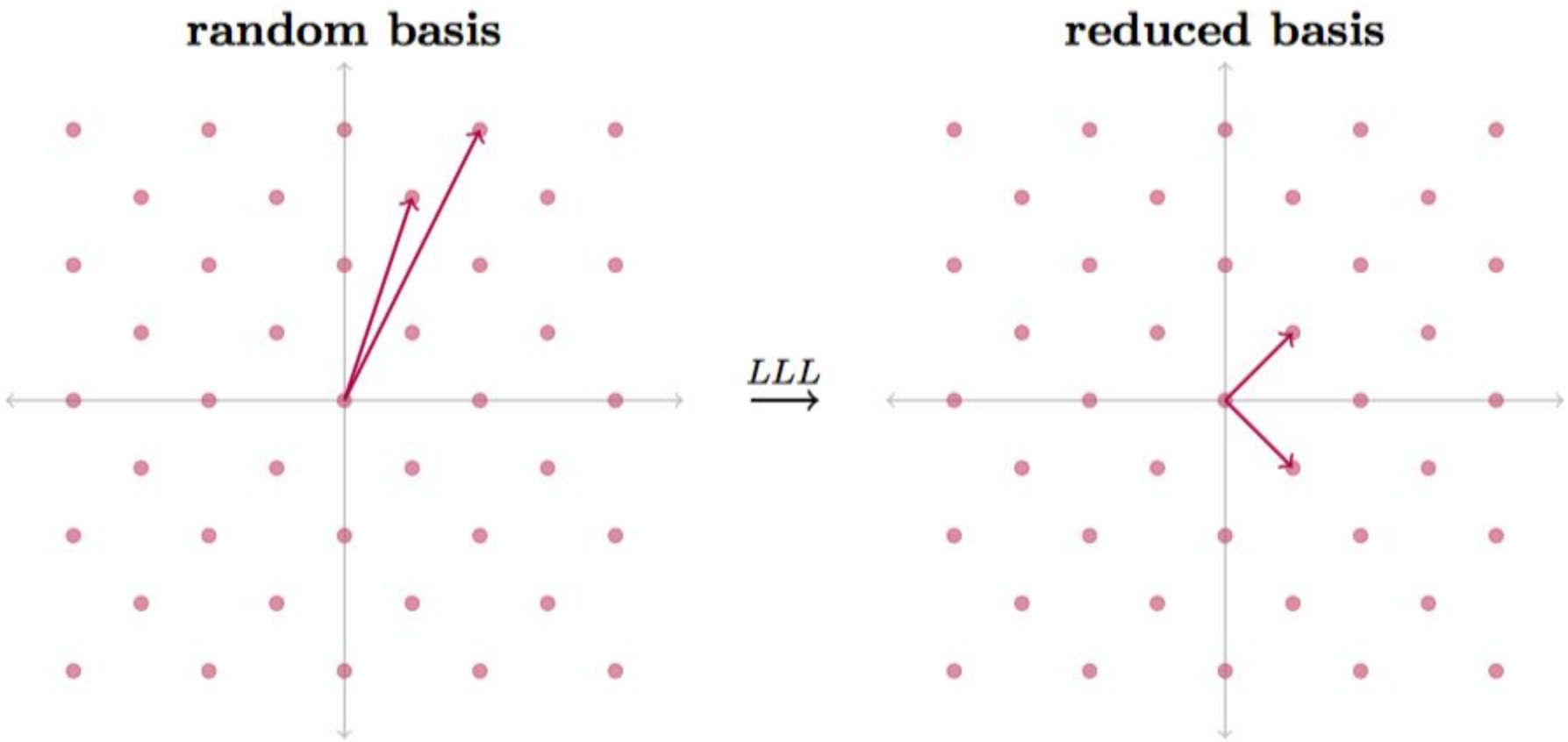




# Lattice Basis Reduction & LLL Algorithm

- Lattice basis reduction is a mathematical technique used to simplify a set of basis vectors for a lattice.
- The reduction results in a new set of shorter and more orthogonal vectors that preserve the structure of the original lattice.
- The LLL algorithm is a technique utilized for the purpose of lattice basis reduction.

# Lattice Basis Reduction & LLL Algorithm



# Coppersmith's Algorithm - Intuition

Coppersmith's algorithm consists of 4 key steps -

1. Lattice Formation
2. Lattice Reduction
3. Formation of polynomial from the first row of the reduced basis
4. Solving the polynomial to find the root

We will understand each step with an example where the lattice dimension is 5. We have

$$N = 16803551 \qquad p = F(x) = \tilde{p} + x = 2830 + x$$

# Coppersmith's Algorithm - Intuition

1. **Lattice Formation**
   - Construct a lattice using the coefficients of the polynomial equations that have a small root ($x_0$) modulo $p^2$.
   - $N^2$, $NF(x)$, $F(x)^2$, $xF(x)^2$ and $x^2F(x)^2$ are the polynomials having $x_0$ as a root modulo $p^2$.
   - A polynomial $a_0 + a_1x + a_2x^2 + a_3x^3$ is represented by the lattice vector $(a_0, Xa_1, X^2a_2, X^3a_3)$.
   - $X$ is an upper bound on the size of $x_0$. In our example, we consider $X = 10$.

# Coppersmith's Algorithm - Intuition

1. **Lattice Formation**
   - Following that we get

$$\begin{pmatrix} 282359326209601 & 0 & 0 & 0 & 0 \\ 47554049330 & 168035510 & 0 & 0 & 0 \\ 8008900 & 56600 & 100 & 0 & 0 \\ 0 & 80089000 & 566000 & 1000 & 0 \\ 0 & 0 & 800890000 & 5660000 & 10000 \end{pmatrix}$$

# Coppersmith's Algorithm - Intuition

2. **Lattice Reduction**
   - We use LLL reduction on the lattice to get a reduced basis.
   - After LLL reduction we get the following reduced lattice:

$$\begin{pmatrix} -100492 & 390240 & 253800 & -495000 & -530000 \\ 262843 & -832520 & 995200 & -580000 & 130000 \\ 1252230 & -976550 & -809100 & 114000 & -880000 \\ 530012 & 559190 & -1023000 & -2513000 & 1840000 \\ 3459355 & 3259580 & 2455200 & 2751000 & 670000 \end{pmatrix}$$

# Coppersmith's Algorithm - Intuition

3. **Polynomial Formation**
   - We use the first vector in the reduced basis to form a polynomial.
   - To form the polynomial, we divide each entry in the first row of the reduced basis by the appropriate power of $X$.
   - For our example, the polynomial we get is

$$-53x^4 - 495x^3 + 2538x^2 + 39024x - 100492$$

# Coppersmith's Algorithm - Intuition

4. **Solving the Polynomial**
   - Since the reduced basis is short enough, we can solve the polynomial over the integers to get $x_0$.

$$-53x^4 - 495x^3 + 2538x^2 + 39024x - 100492 = 0 \longrightarrow x_0 = 7$$

# Coppersmith's Algorithm - Example

$$N = 16803551 \qquad p = F(x) = \tilde{p} + x = 2830 + x$$

$$\begin{pmatrix} 282359326209601 & 0 & 0 & 0 & 0 \\ 47554049330 & 168035510 & 0 & 0 & 0 \\ 8008900 & 56600 & 100 & 0 & 0 \\ 0 & 80089000 & 566000 & 1000 & 0 \\ 0 & 0 & 800890000 & 5660000 & 10000 \end{pmatrix}$$

**LLL Reduction**

$$\begin{pmatrix} -100492 & 390240 & 253800 & -495000 & -530000 \\ 262843 & -832520 & 995200 & -580000 & 130000 \\ 1252230 & -976550 & -809100 & 114000 & -880000 \\ 530012 & 559190 & -1023000 & -2513000 & 1840000 \\ 3459355 & 3259580 & 2455200 & 2751000 & 670000 \end{pmatrix}$$

**Convert first row to polynomial**

$$-53x^4 - 495x^3 + 2538x^2 + 39024x - 100492$$

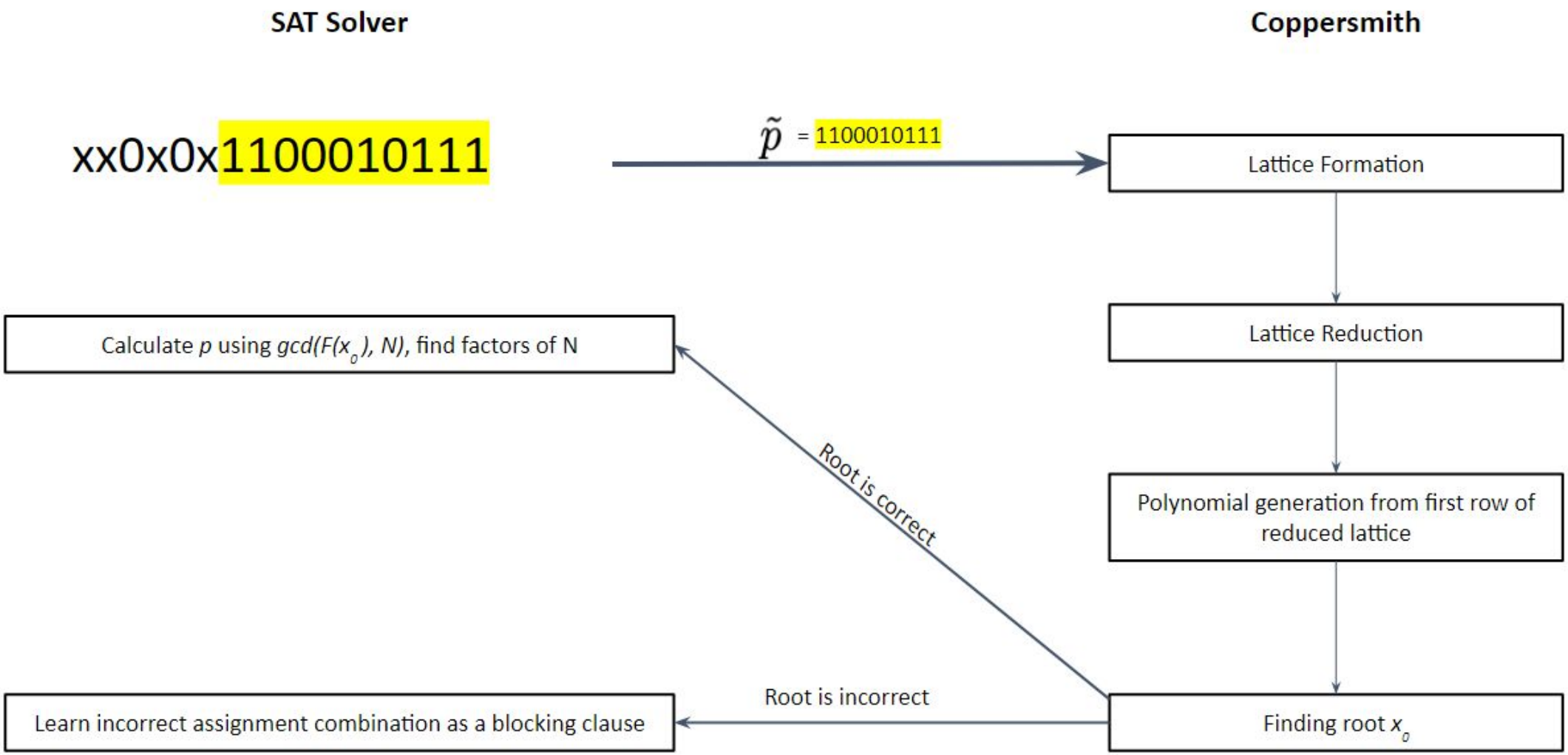
**Solve**

$$x_0 = 7$$

# Proposed Solution

- First attempt to develop a hybrid approach with SAT and Coppersmith's algorithm.
- Certain percent of random bits of the primes are known.
- The SAT solver tries different assignments of bits to satisfy the given constraints.
- Whenever the solver reaches a stage when 60% of bits (LSBs or MSBs) of the first prime is set, Coppersmith's algorithm is invoked.
- Coppersmith tries to extend this partial assignment to a complete assignment.
- If the partial assignment is found to be incorrect, a clause is learnt to prevent the same partial assignment again.

# Proposed Solution



# Blocking Clause

- A blocking clause is a special type of clause added to the SAT solver when a certain partial assignment leads to a failure.
- Encodes that a specific combination of low bits used in a computation (e.g., Coppersmith's method) was incorrect.
- It prevents the solver from revisiting the same erroneous assignment by ensuring at least one of the bits changes.

**Example -**

$$p = ???10011$$

$$\neg p_4 \vee p_3 \vee p_2 \vee \neg p_1 \vee \neg p_0$$

$p_i$ is the $i^{th}$ bit of $p$ ( $p_0$ is the LSB).

# Implementation

- CNF instance generated using "**CNF Generator for Factoring Problems"** by Paul Purdom and Amr Sabry.
- The solvers used are **MapleSAT** and **CaDiCaL**.
- Coppersmith's algorithm used is a **custom implementation** in C++ using GMP, FPLLL and FLINT libraries.

# The Encoding

- Utilized the "**CNF Generator for Factoring Problems"** by Paul Purdom and Amr Sabry.
- Chose the **N-bit** adder type and the **Karatsuba** multiplier type.
- The encoder is written in Haskell.

Original Encoding (**Unbalanced**):

- Represents *p* with *2k-1* variables and *q* with *k* variables.
- Extra high bits in *p* are set to 0 using unit clauses.

Modified Encoding (**Balanced**):

- Adjusted to represent both *p* and *q* with *k* bits each.

# The Encoding - Simple Optimizations

- $p$ and $q$ must be odd:
  - Fix low bits $p_0$ and $q_0$ to 1 with unit clauses.

**Unit Clause** - A clause that contains exactly one literal.

- Both $p$ and $q$ are of bitlength $k$:
  - Fix high bits $p_{k-1}$ and $q_{k-1}$ to 1.

- Random known bits are also added as unit clauses.

# Encoding Extra Information

- Purpose is to reduce the number of unknown variables by inferring additional details.
- Based on number theory principles.
- Certain conditions need to be satisfied for this inference to work.
- The same idea is also used as a custom branching heuristic (user-selectable option).

# Encoding RSA Private Exponent

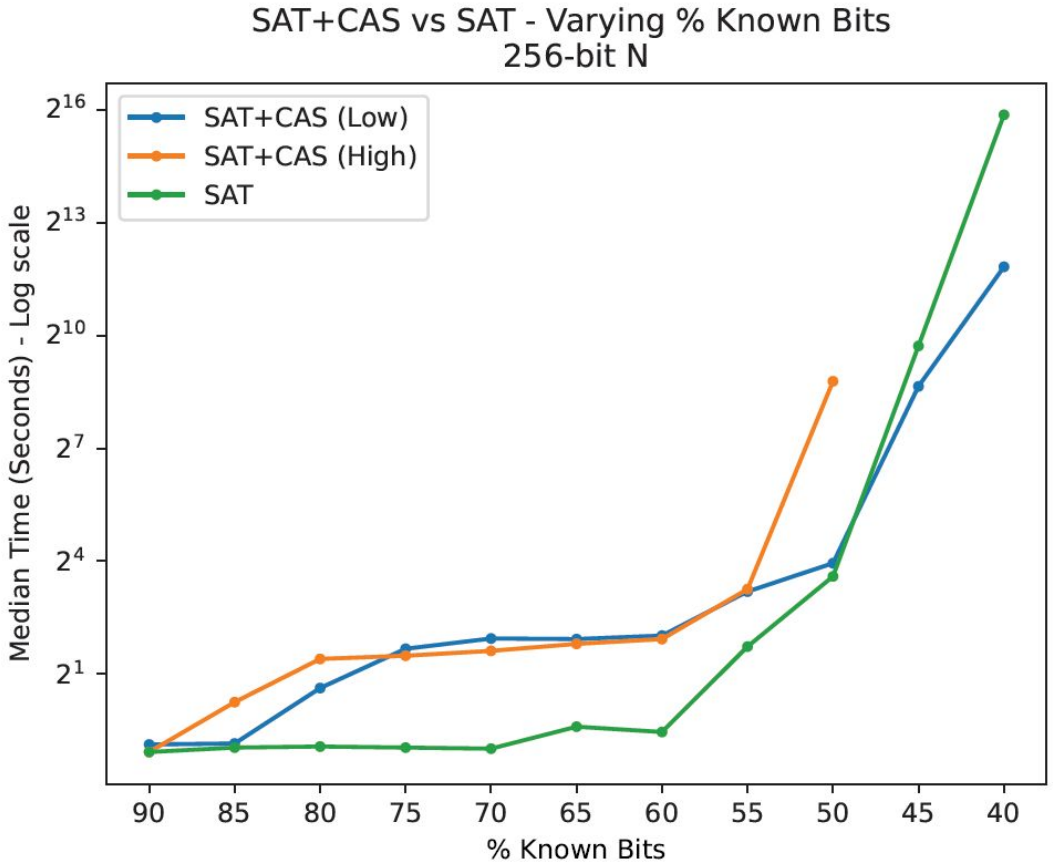
- A more constrained case of the factorization problem.
- Factoring an RSA modulus *N* with a public exponent of *e=3* (both *p-1* and *q-1* are not divisible by 3).
- We infer and encode some high bits of the private exponent *d*.
- We also encode the same percentage of randomly known bits of *d* as that of *p* and *q*.

# Experimental Setup

- Ran 15 randomly generated instances of each test case.
- We ran the following test scenarios to assess the performance of the solution in each case -
  - Calling Coppersmith using High Bits vs Low Bits
  - Known Bits of Primes Only
  - Incorporating the Private Exponent
  - Effect of Branching Heuristic
  - Effect of Different Encodings
  - Effect of Changing Lattice Size
  - Effect of Known Bits in One Prime Only
  - Comparison with Other Works

# Results

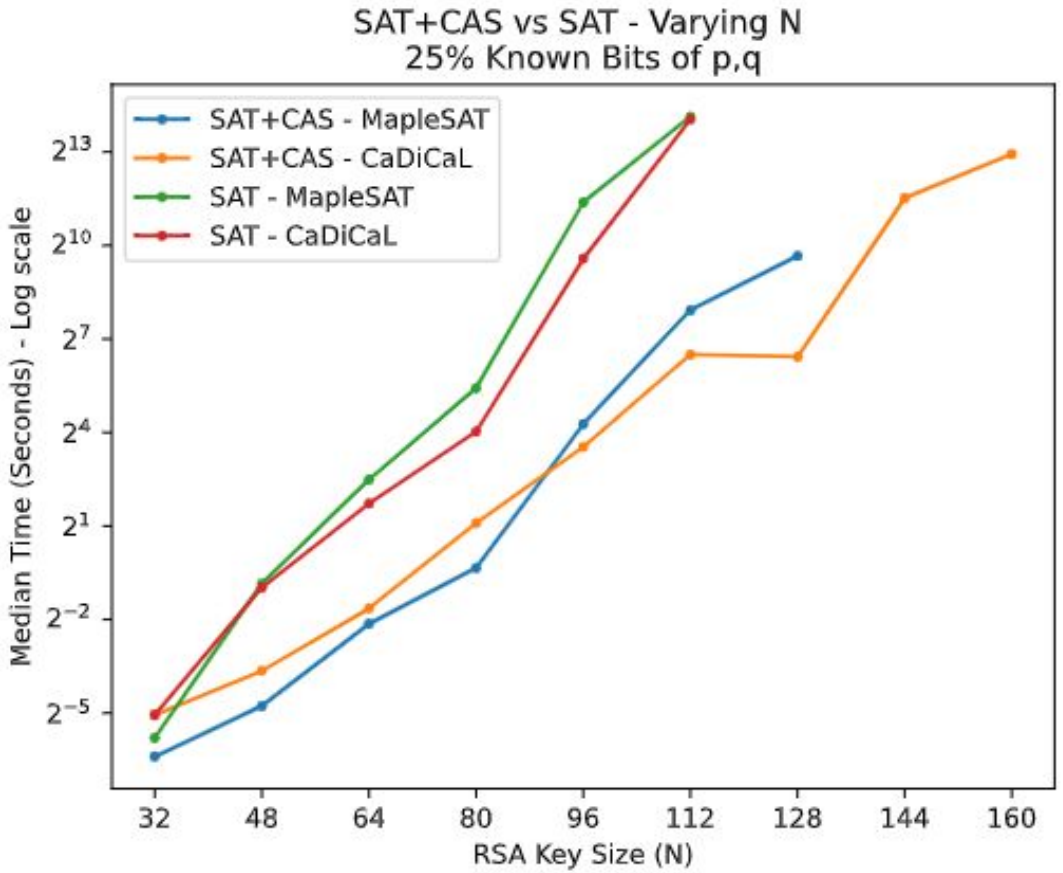
Calling Coppersmith using High Bits vs Low Bits

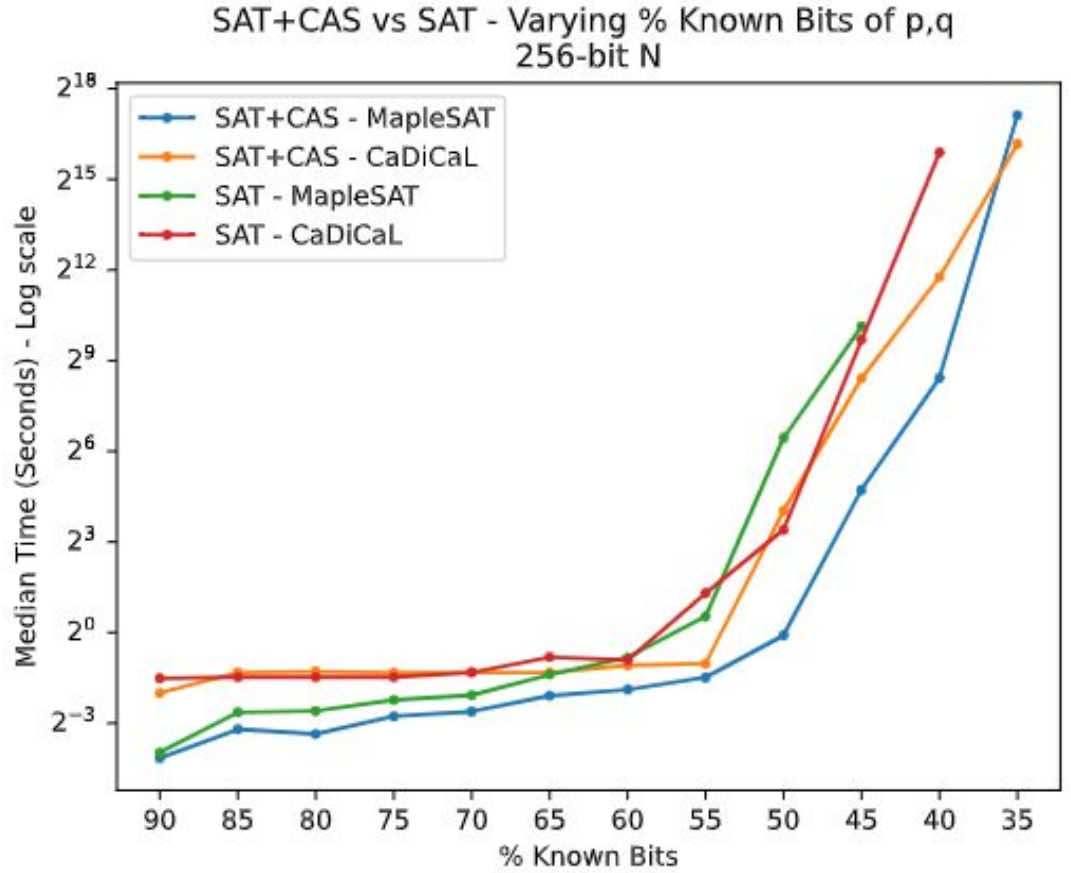



**Timeout:** 2 days

# Results

Known Bits of Primes Only



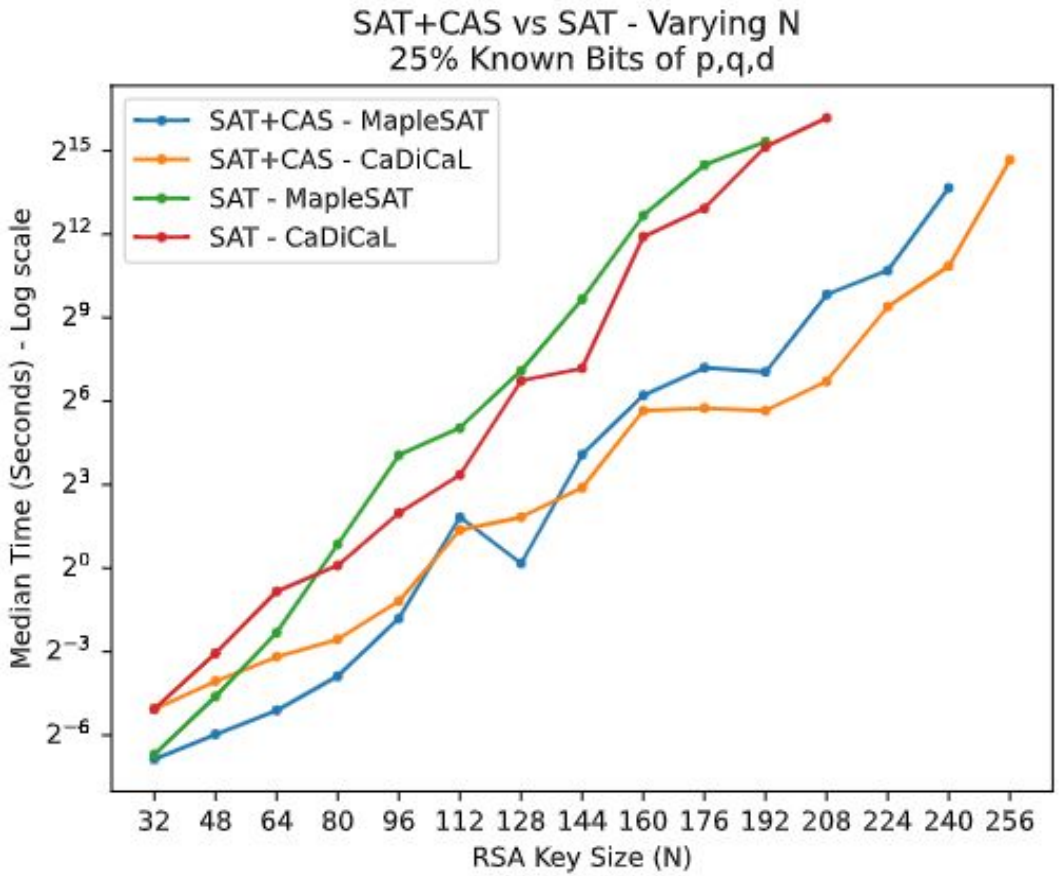

**Timeout:** 2 days

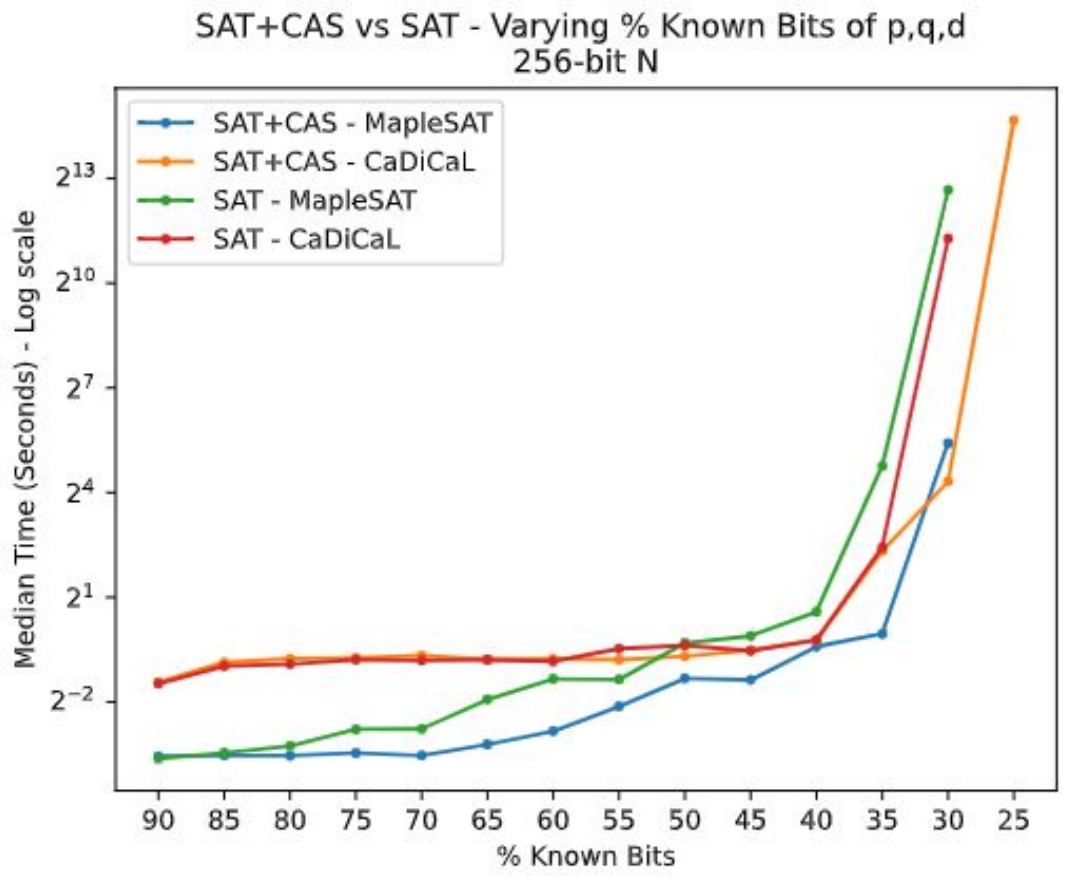


University of Windsor

# Results

Incorporating the Private Exponent

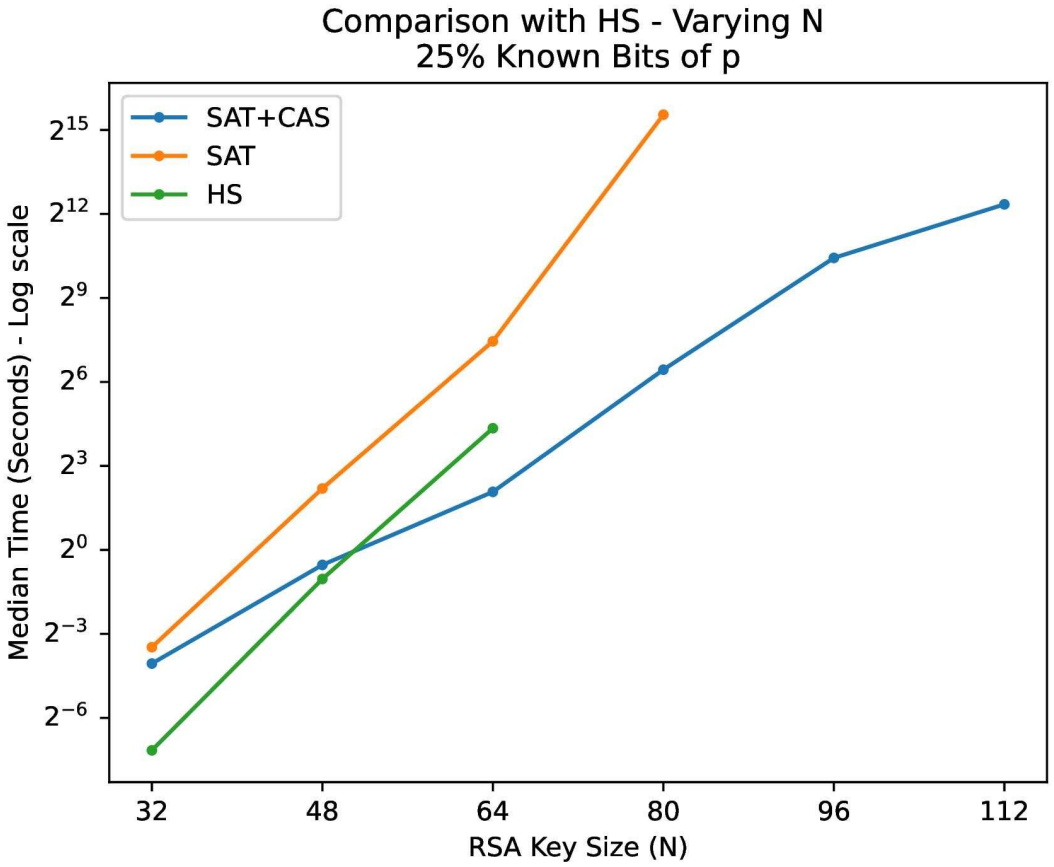


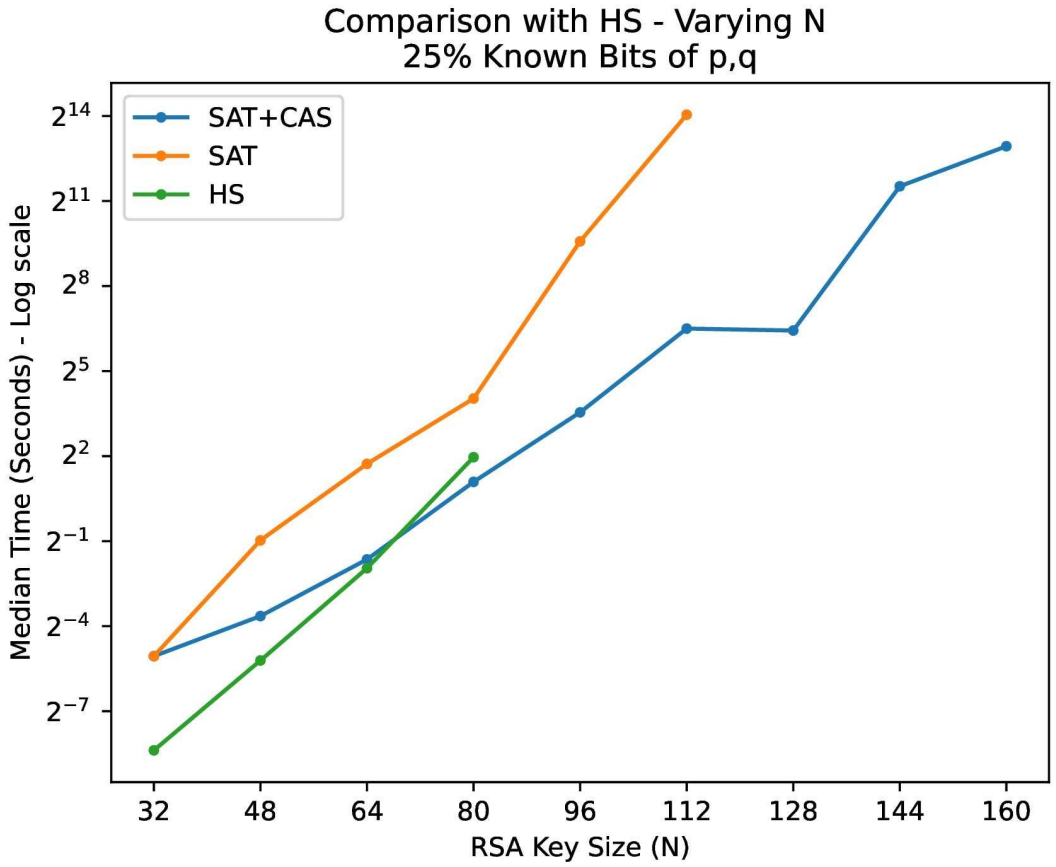




**Timeout:** 2 days

# Results

## Comparison with Other Works - Heninger Shacham

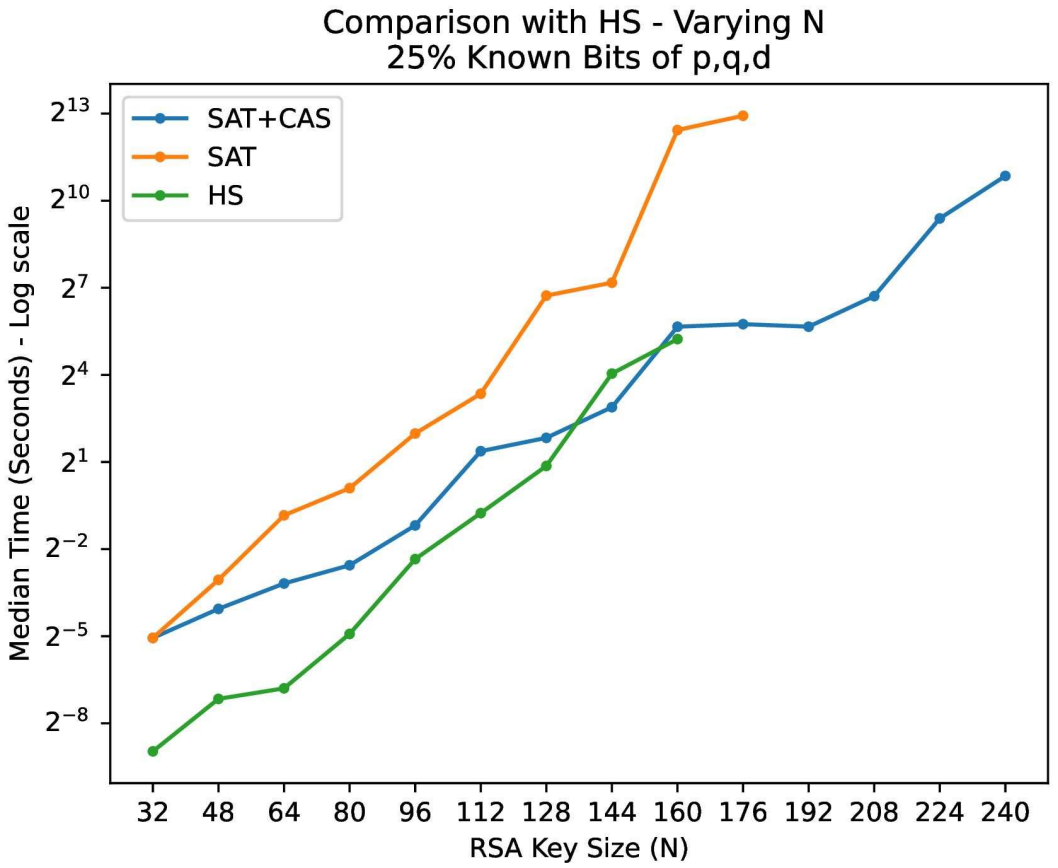



**Timeout:** 1 day ; **Memory:** 4 GB

# Results

## Comparison with Other Works - Heninger Shacham

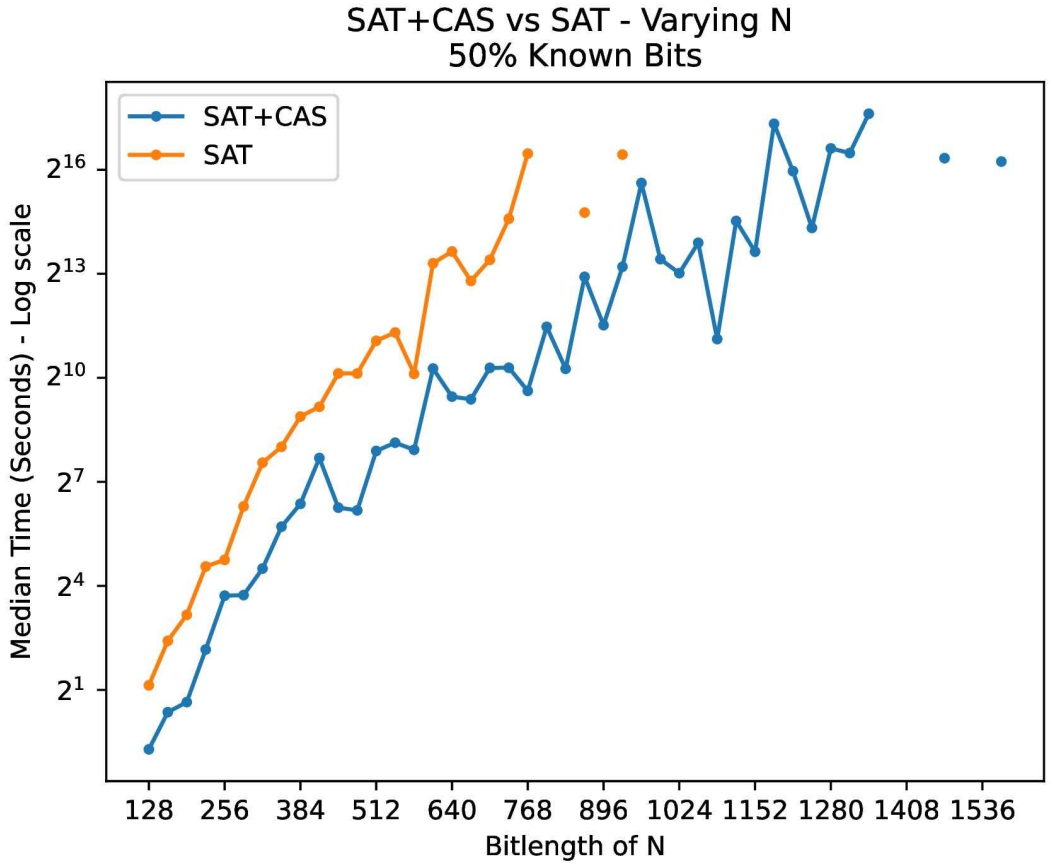



**Timeout:** 1 day

**Memory:** 4 GB

# Results

Comparison with Other Works - Numerical Methods



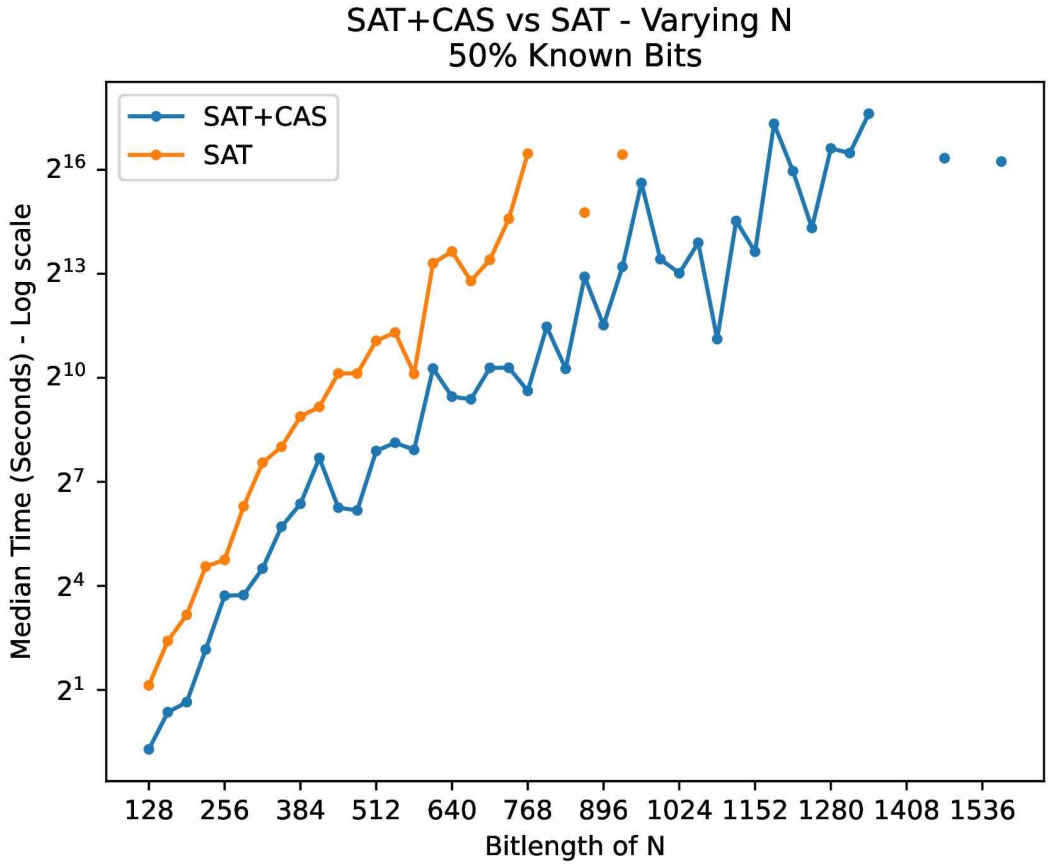

**SAT+CAS vs. Brute-Force Approach:**

- SAT+CAS solver factors 512-bit *N* with 50% leaked bits in a median of 237 seconds.
- Brute-force approach needs to determine values for approx. 64 unknown bits in the lower half of *p* before applying Coppersmith.

**Timeout:** 3 days

# Results

Comparison with Other Works - Numerical Methods




**SAT+CAS vs. Number Field Sieve (NFS):**

- SAT+CAS significantly faster for factoring a 512-bit *N* with 50% leaked bits.
- NFS takes approx. 2770 CPU hours on Amazon's EC2 for the same task.

**Timeout:** 3 days

# Conclusion

- Performance of off-the-shelf SAT solvers is greatly improved by incorporating Coppersmith's method.

- The SAT+CAS approach is able to reduce the runtime by up to 99.8% compared to a pure SAT approach.

- The SAT+CAS method is practically feasible and relevant considering the method is able to factor a 1024-bit RSA key with 50% known bits of the primes.

# Thank You

# References


1. J. A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Appelbaum, and E. Felten. Lest we remember: Cold boot attacks on encryption keys. In P. Van Oorschot, editor, Proceedings of USENIX Security 2008, pages 45–60. USENIX, July 2008.
2. Patsakis, Constantinos. RSA private key reconstruction from random bits using SAT solvers. IACR Cryptol. ePrint Arch. 2013 (2013): 26.
3. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities, J. Crypt. 10 (1997), no. 4, 233–260.
4. N. Howgrave-Graham, Finding small roots of univariate modular equations revisited, IMA Cryptography and Coding (M. Darnell, ed.), LNCS, vol. 1355, Springer, 1997, pp. 131–142.
5. Bright, C., Đoković, D.Ž., Kotsireas, I. et al. The SAT+CAS method for combinatorial search with applications to best matrices. Ann Math Artif Intell 87, 321–342 (2019).

# HOW RSA WORKS? (1/2)

- *p and q are large primes*
- $n = pq$
- $\varphi(n) = lcm\ (p - 1, q - 1)$
  - $\varphi(n)$ represents Euler's totient for $n$
- *e is a random prime between 1 and* $\varphi(n)$. Because the public key is shared openly, it's not so important for *e* to be a random number. In practice, *e* is generally set at **65,537**, because when much larger numbers are chosen randomly, it makes encryption much less efficient

# HOW RSA WORKS? (2/2)

- **Public key is (*e, n*)**
- ***c* = $m^e$ *mod n***
- *d* =1/*e mod* φ(*n*)
- **Private key is (*d, n*)**
- ***m* = $c^d$ *mod n***

# Coppersmith's Algorithm - Intuition

- $p = x_0 + (2^{-m} \bmod N) \cdot \tilde{p}$

  $x_0$ is an integer with the same value as the unknown high bits of $p$ .

- We form a lattice where every vector in the lattice corresponds to a polynomial having $x_0$ as a root modulo $p^h$.
  The polynomials are $N^h, N^{h-1}f(x), N^{h-2}f(x)^2, \ldots, Nf(x)^{h-1}, f(x)^h, xf(x)^h, \ldots, x^h f(x)^h$
  where $2h+1$ is the dimension of the lattice.
- If the coefficients of the polynomial are small enough, then $x_0$ will also be a root over the integers, not just *mod* $p^h$.
- To make the coefficients smaller, we use the LLL algorithm which is a lattice reduction technique.

# Encoding Extra Information

- Purpose is to reduce the number of unknown variables by inferring additional details.
- Based on number theory principles.
- To infer bit $p_i$ the following conditions need to satisfied -
  1. Bit *i* in the second prime *q* must be known i.e. $q_i$
  2. Bits *0* to *i-1* must be known in both primes *p* and *q*.
- Then we can you the congruence to derive $p_i$

$$\boldsymbol{p_i + q_i \equiv (N - p'q')[i] \pmod 2}$$

$$p' = p_{i-1} \ldots p_0$$

$$q' = q_{i-1} \ldots q_0$$

*(N - p'q')[i] is the $i^{th}$ bit of N - p'q'*

# Encoding Extra Information - Example

- **Given:**

  $p$ = 1x0xxx011

  $q$ = xx1x01011

  $N$ = 10000111011000001

- Derive $p_3$ since $q_3$ and bits ($p_0$ …. $p_2$), ($q_0$ …. $q_2$) are known:

  $p' = (011)_2 = (3)_{10}$

  $q' = (011)_2 = (3)_{10}$

  $p'q' = (1001)_2 = (9)_{10}$

  $N - p'q' = (10000111011011000)_2$

- Congruence

  $p_3 + q_3 \equiv (N - p'q')[3] \pmod 2$

  $p_3 + 1 \equiv 1 \pmod 2$

- **$p_3$ must be 0**

# Encoding RSA Private Exponent

- A more constrained case of the factorization problem - Factoring an RSA modulus *N* with a public exponent of *e=3* (both *p-1* and *q-1* are not divisible by 3).
- From the basics of RSA we can derive

$$3d + 2(p + q) = 2N + 3$$

- We can approximate *d* using

$$\tilde{d} = \lfloor 2N/3 + 1 \rfloor$$

- We also encode the same percentage of randomly known bits of *d* as that of *p* and *q*.

# Branching Heuristics

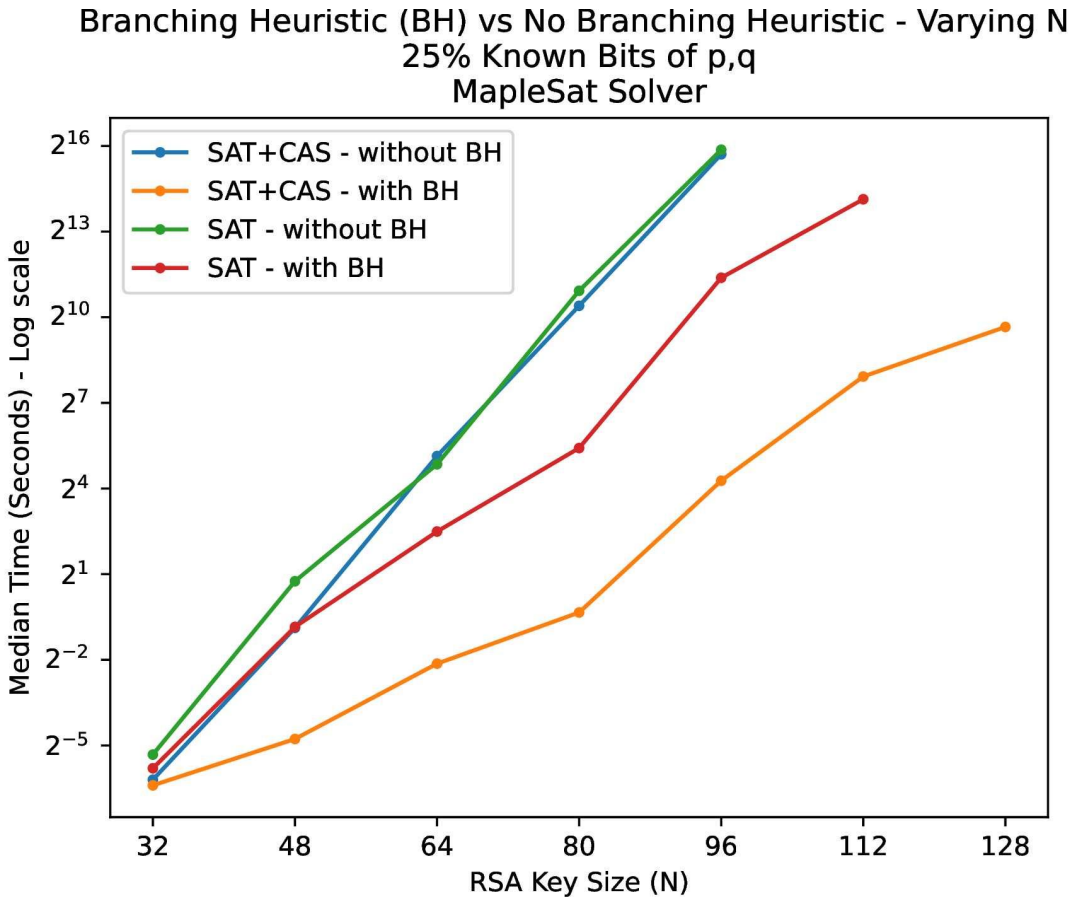
- **Default Branching Strategies:**
  - MapleSAT and CaDiCaL both utilize sophisticated heuristics.
  - Guide the selection of variables for branching during the search for a satisfying assignment.
- **Custom Branching Heuristic:**
  - Influenced by same constraints from “Encoding Extra Information” slides.
  - Directs solver towards potential solutions by prioritizing variables for Coppersmith's method.
  - User-selectable option

# Branching Heuristics

- **MapleSAT Variable Activity Heuristic:**
  - Assign high activity values to variables corresponding to low bits of the first prime.
  - Prioritizes branching on these variables.
  - Facilitates earlier application of Coppersmith's method.
  - Optional, user selectable.

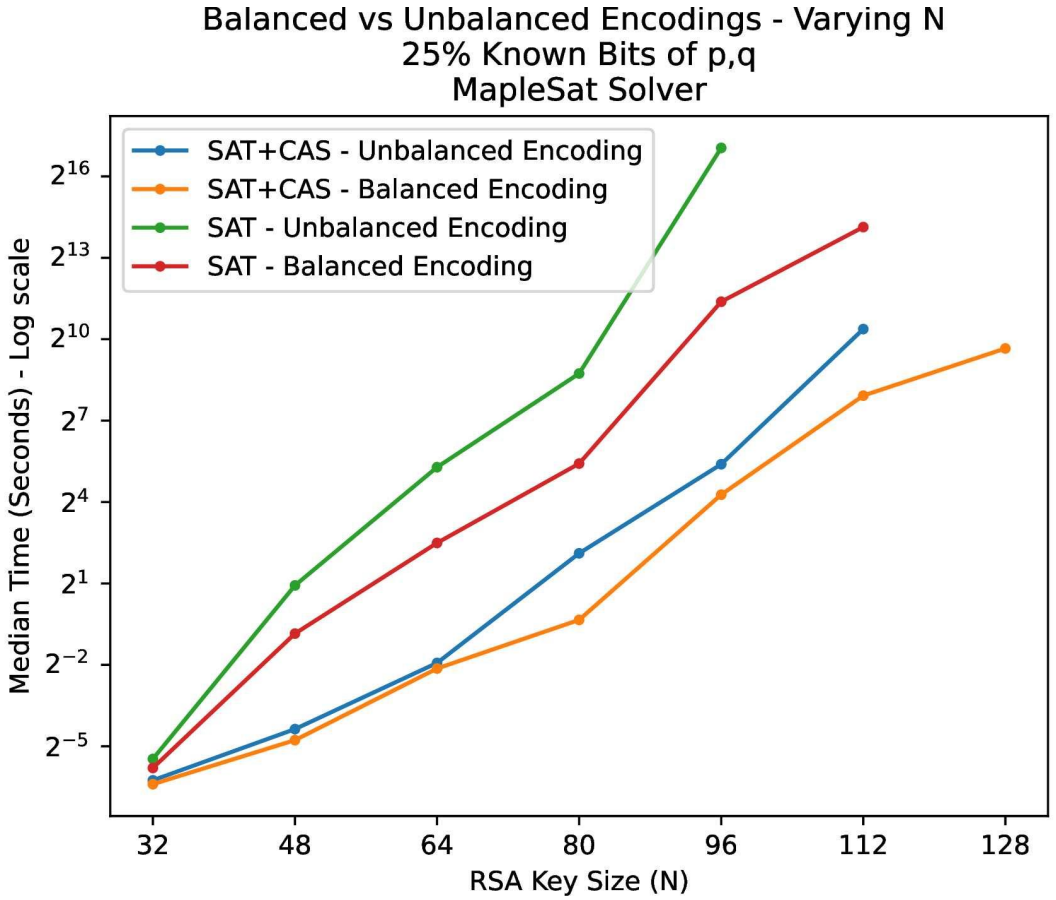
# Results

## Effect of Branching Heuristic



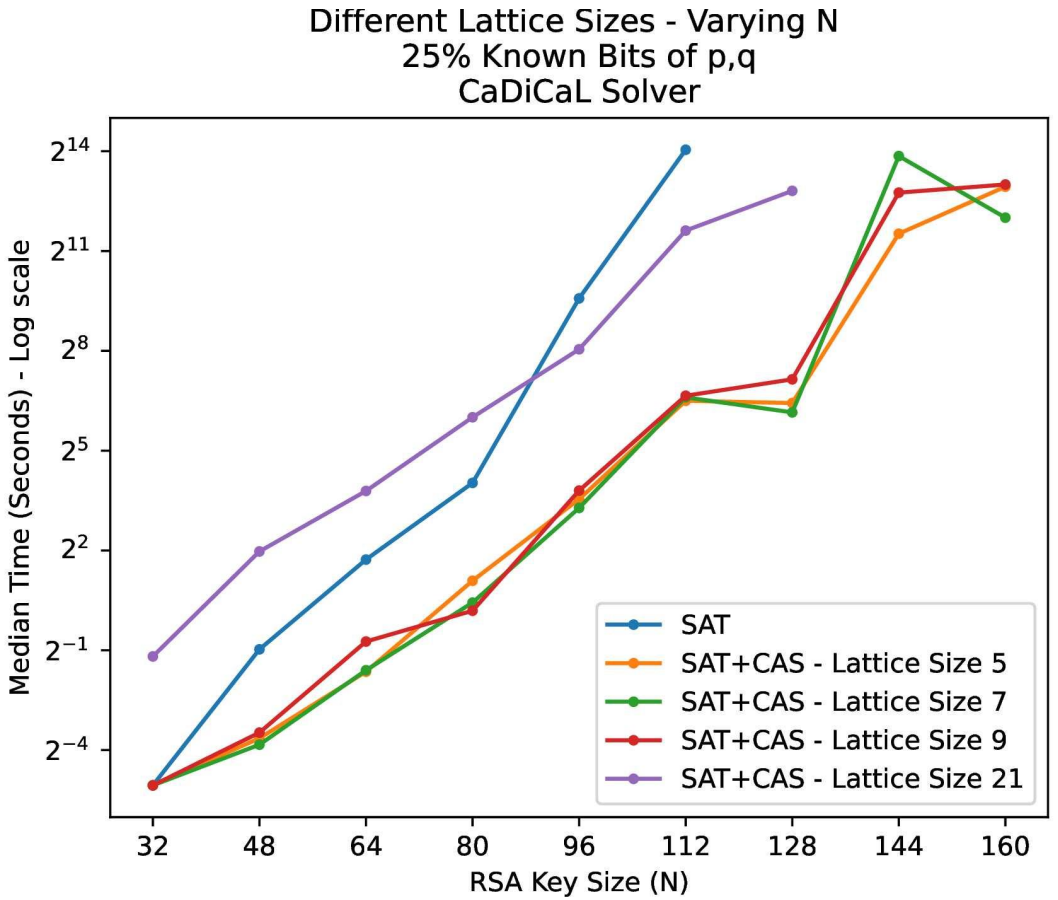
# Results

## Effect of Different Encodings



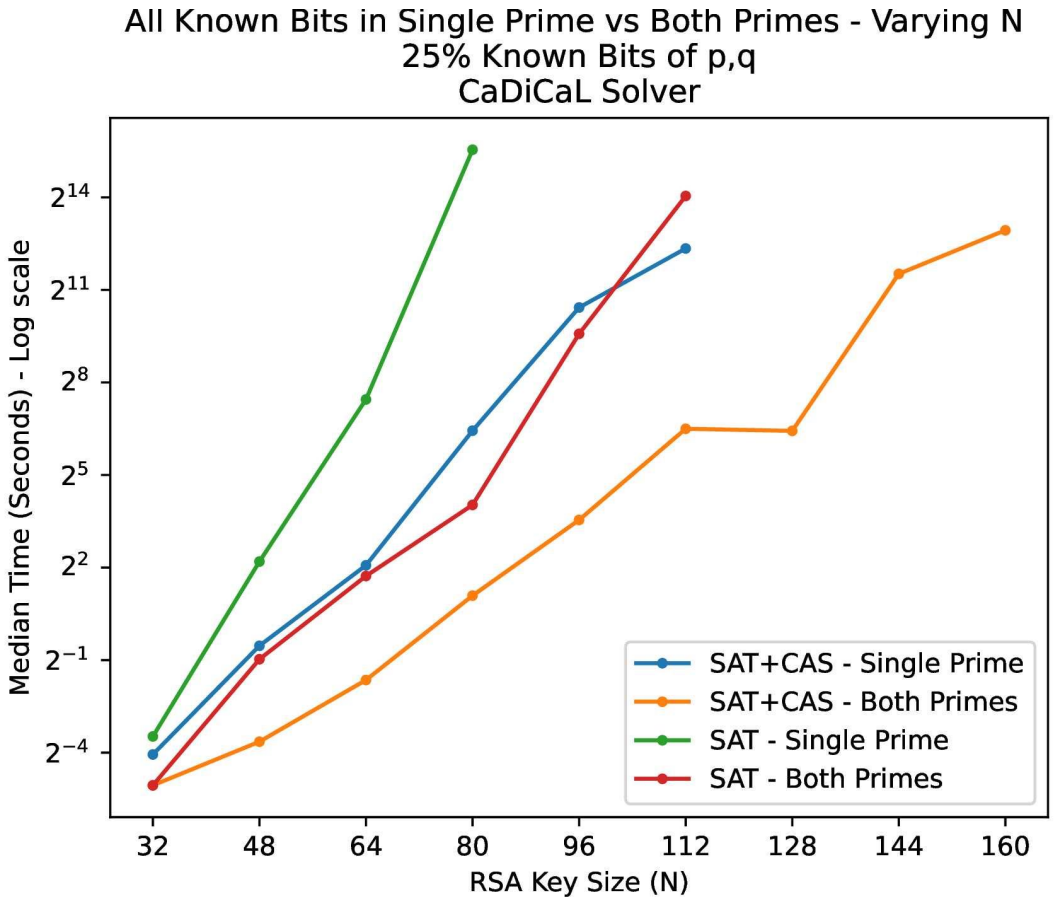
# Results

## Effect of Changing Lattice Size



# Results

## Effect of Known Bits in One Prime Only



# Coppersmith's Algorithm - Example

$$N = 16803551 \qquad p = F(x) = \tilde{p} + x = 2830 + x$$

$$\begin{pmatrix} 16803551 & 0 & 0 & 0 \\ 2830 & 10 & 0 & 0 \\ 0 & 28300 & 100 & 0 \\ 0 & 0 & 283000 & 1000 \end{pmatrix}$$

**LLL Reduction**

$$\begin{pmatrix} 105 & -1200 & 800 & 1000 \\ 336 & -1040 & 1500 & -1000 \\ 1094 & 1580 & 600 & 1000 \\ -1400 & 530 & 2100 & 0 \end{pmatrix}$$

**Convert first row to polynomial**

$$x^3 + 8x^2 - 120x + 105$$

**Solve**

$$x_0 = 7$$