# Comparative Analysis of SMT Solvers for Differential Cryptanalysis of SHA-2

Marcel Barlik[1], Martin Brain[1]

[1]City St. George's University of London, Northampton Square, London, EC1V 0HB, United Kingdom

**Abstract**

This work presents an experimental investigation on the performance differences among various SMT solvers in generating differential cryptanalysis collisions for the SHA-2 family of cryptographic hash functions, a widely adopted hash function, critical for maintaining data integrity and security of protocols like TLS. The research involved examining different parameters with these solvers, and their effects on the overall solving performance. These findings provide both a methodological baseline and actionable insights regarding solver effectiveness tailored towards helping shape future research in automated cryptanalysis.

**Keywords**

Satisfiability Modulo Theory, Differential Cryptanalysis, Cryptographic Hash Function, SHA-2, Theory of BitVectors

## 1. Introduction

A hash function is a deterministic algorithm that takes an arbitrary length input string, known as the *message*, and maps it to a fixed-size output, known as the *hash value*, *digest* or *checksum*. Hash functions are a key building block in modern cryptography used for authentication, digital signatures and data integrity. This work will focus on Secure Hashing Algorithm 2 (SHA-2), a set of cryptographic hash functions published in 2001 by the National Institute of Standards and Technology of the United States [1]. It is widely used in many cryptographic protocols and applications including TLS [2], HTTPS [3], DNSSEC [4] and SSH [5] protocols, PGP [6], package authentication and cryptocurrencies such as Bitcoin [7] and Ethereum [8].

Hash functions must be efficient to compute, but hard to reverse. Specifically, it must be computationally infeasible to derive a message that matches a chosen digest (a *pre-image*) and it must be infeasible to find a pair of messages which hash to the same digest (a *collision*). As described by [9] in 4.1, SHA-256 has a collision resistance of $2^{128}$, meaning it is computationally infeasible. This mathematical strength is why SHA-2 continues to be trusted for critical security applications, despite being developed over two decades ago. Discovery of a full collision on SHA-2 algorithms would have a critical security impact on billions of users around the globe.

One approach to finding collisions, or showing the impracticality of finding them is *Differential Cryptanalysis*. First published by [10] (but known in closed organisations earlier, as described by [11]) this technique reasons about the differences (referred to as the *differential*) between the same variables in two instances of the algorithm. If it is possible to find small differences between the input messages which result in small differences in the digest, then it is possible to find collisions. Designers attempt to ensure that hashes follow the avalanche effect principle for all inputs – a minor change in the input propagates a huge output change.

A recent trend in differential cryptanalysis is to use automated reasoning tools, such as Satisfiability Modulo Theory (SMT) solvers, to reason about the differentials. These works are cryptographically sophisticated but tend to use SMT solvers in a relatively simple way. In this paper we investigate the following questions:

*SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK*
marcelbarlik@gmail.com (M. Barlik); martin.brain@citystgeorges.ac.uk (M. Brain)
https://github.com/Supermarcel10 (M. Barlik)
0000-0003-4216-7151 (M. Brain)

**RQ1** How does SMT solver choice influence SHA-256 performance[1]?

**RQ2** Which SMT solver parameters affect SHA-256 performance most?

**RQ3** How do differential cryptanalysis encodings impact SHA-256 collisions?

### 1.1. Contributions

In Section 2, we describe SHA-256 and previous work on differential cryptanalysis using automated reasoning tools. Building on this foundation, Section 3 presents a series of experiments and methodologies for investigating the research questions given above. Finally, Section 4 analyses the results of our experiments.

## 2. Background

We will use $\ggg$ to denote right roll by a constant, $\oplus$ to denote Exclusive Or (XOR) and $+$ represents modulo addition (respectively `rotate_right`, `bvxor` and `bvadd` in SMT-LIB).

### 2.1. SHA-2

The SHA-2 family, designed by the National Security Agency (NSA) in 1995, standardised by NIST in 2001, comprises of six primary hash functions: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. These algorithms employ the Merkle–Damgård construction with a Davies–Meyer compression function, differing primarily in digest size (224 to 512 bits), initial values, and round counts (64 for SHA-256, 80 for SHA-512) [1].

The SHA-2 algorithm processes input through a series of deterministic transformations governed by its Merkle–Damgård structure.

**Message Processing** Input messages undergo a pre-processing step to conform to the 512-bit block for SHA-256 (1024-bit block for SHA-512 respectively). This step is necessary only when converting an input message to a digest. However, this step is irrelevant for automated reasoning tools like SMTs, which directly use arbitrary message blocks to reason about values.

**Function Definition** Each SHA-2 hash function uses its own set of constants for functions. SHA-256 functions are defined as:

$$\gamma_0(w) = (w \ggg 7) \oplus (w \ggg 18) \oplus (w \ggg 3)$$
$$\gamma_1(w) = (w \ggg 17) \oplus (w \ggg 19) \oplus (w \ggg 10)$$

$$\Sigma_0(a) = (a \ggg 2) \oplus (a \ggg 13) \oplus (a \ggg 22)$$
$$\Sigma_1(e) = (e \ggg 6) \oplus (e \ggg 11) \oplus (e \ggg 25)$$

$$Maj(a, b, c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$$
$$Ch(e, f, g) = (e \wedge f) \oplus (\neg e \wedge g)$$

**Message Expansion** The pre-processed message makes up words $w_i$, for $0 \leq i \leq 15$. Remaining words, $16 \leq i \leq 63$, are expanded using $w_i = w_{i-16} + \gamma_0(w_{i-15}) + w_{i-7} + \gamma_1(w_{i-2})$.

[1] Better performance refers to reduced solving time and improved round solvability.

**Constants and Initialisation** Each hash function part of the SHA-2 family uses different *H-constants*, also known as the *initialisation vector (IV)*. These serve as the initial digest values when starting the compression function. SHA-256 uses fractional parts of square roots from the first 8 primes, whereas SHA-512 uses cube roots. The round constants, also known as *K-constants*, are derived from fractional parts of cube roots for the first 64 primes (SHA-256) or first 80 primes (SHA-512). These aim to break symmetry in message scheduling during modular addition of the compression function.

**Compression Function** Each H-constant updates one of eight 32-bit registers $a$–$h$, often referred as the *working variables*, which are part of the 256-bit starting state. The compression function employs 64 rounds (80 for SHA-512), where for each round, the following are executed:

$$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_i + w_i$$
$$T_2 = \Sigma_0(a) + Maj(a, b, c)$$

$$h = g; g = f; f = e; e = d + T_1$$
$$d = c; c = b; b = a; a = T_1 + T_2$$

What is often referred to as a round-reduced model, is simply a compression with less *iterations*, also knowns as *steps* or simply *rounds*, than standard. For round-reduced models, a lot of K-constants and expanded messages are irrelevant for obtaining the digest.

**Finalisation** After processing all blocks, the final digest concatenates the eight registers' contents. Truncated variants trim the output size. For SHA-256, this produces a 64-character hexadecimal value.

## 2.2. Collisions

A cryptographic collision occurs when distinct inputs $M \neq M'$ yield identical digests $H(M) = H(M')$. As described in [9], due to SHA-2's collision resistance, a true brute-force collision requires $O(2^{n/2})$ operations for $n$-bit hashes. Reasoning and analytical attacks, such as [12], exploit weaknesses with the use of encodings and heuristics, to achieve practical breaks at reduced-rounds.

Collisions can be divided into three categories based on the IV used:

- **Free-Start Collisions (FS):** Attacker controls both message blocks and IVs. A collision occurs when either the message or IV are unique. demonstrated a weakness in the compression function, showing round structure vulnerabilities.
- **Semi-Free-Start Collisions (SFS):** Attacker chooses a fixed IV for both messages, while controlling the message blocks. A middle-ground between FS and STD, demonstrating significant structural weaknesses.
- **Classical/Standard Collisions (STD):** Attacker controls only the message blocks. The NIST provided IV is used. Completely breaks collision resistance properties, deeming the hash function cryptographically insecure.

Standard collision is always unsatisfiable (UNSAT) for the first 8 rounds. This is likely due to choice of constants, creating infeasible conditions for a collision.

### 2.2.1. Current State-of-the-Art

Table 1 gives the best cryptanalysis results publicly available for each kind of collision. Although many researchers have attempted finding vulnerabilities since the big crash of MD4 and similarly structured hash functions. [18] The pace of progression in this domain has been very slow.

**Table 1**
Historical SHA-256 and SHA-512 collisions from 2008 to present, including this paper's SHA-256 results for reference.

| Hash Function | CT | Rounds | Time | Memory | References |
|---|---|---|---|---|---|
| SHA-256 | | 18 | practical | practical | This paper |
| | | 31 | $2^{49.8}$ | $2^{48}$ | [12] |
| | STD | 28 | practical | - | [13] |
| | | 31 | $2^{65.5}$ | - | [14] |
| | | 24 | $2^{22.5}$ | - | [15] |
| | FS | 39 | practical | - | [12] |
| | SFS | 38 | $2^{19.2}$ | - | [16] |
| | SFS | 38 | $2^{37}$ | - | [14] |
| SHA-512 | STD | 31 | $2^{115.6}$ | $2^{77.3}$ | [12] |
| | STD | 24 | $2^{22.5}$ | - | [15] |
| | FS | 39 | practical | - | [13] |
| | SFS | 38 | $2^{40.5}$ | - | [17] |

A SAT + CAS approach was used in [16]. Their claim was that the SAT + CAS solver is capable of better performance as opposed to just a SAT approach. During their research, another work [12] utilised very different encodings with a SAT solver, beating them and becoming the current record.

A direct attack targetting OpenWRT's use of SHA-256 was described in [19]. OpenWRT used only 12 characters out of the total output digest, where full collisions are more probable due to the smaller search space. The researcher was able to find a full collision on those 12 characters using true brute-force approaches, with GPU compute on HashCat [20]. Since this is not a reasoning tool, this is likely near the practical limit for current GPU compute, and falls into the same category of NP-Hard as SAT solving.

There have been attempts to make a GPU accelerated SAT solver [21], but all attempts were outperformed by standard CPU alternatives.

A wider variety of of cryptanalysis tools, including SAT, SMT, MILP and CP were applied to multiple ciphers, permutations and hash functions in [22]. Their winner categorising strategy is described as: "The best solver for each cipher is the one with the highest number of wins. The winner of our competition (for every formalism) is the solver that performs best for the highest number of ciphers (more than 20, each from round 2 to 6)." Their claim is that "In the SMT solvers category, Z3 and MathSAT are always inferior to Yices2, which is thus clearly the best SMT solver in our testing".

No research has previously defined the baseline of what SMT solvers are capable for SHA-2 collisions. It is likely that researchers utilising reasoning tools, like SAT or SMT solvers, were only getting a few rounds, taking this as an implied hard limit without the use of sophisticated encodings. This creates the basis for RQ1 and RQ2, which aims to fill this knowledge gap.

## 3. Method

We have written a custom benchmark generator which creates SMT-LIB files modelling a pair of SHA-256 or SHA-224 instances running a configurable number of rounds. The initialisation values can be configured for standard, free-start or semi-free start.

All satisfiable instances were verified using our SHA-2 implementation, which is known-good against standard known values.

To investigate RQ1 we aimed to use an exhaustive list of currently available SMT solvers. These include: Z3 version 4.13.4 [23], cvc5 version 1.2.1 [24], Yices2 version 2.6.5 [25], Bitwuzla version 0.7.0 [26], Boolector version 3.2.3 [27], Colibri2 version 0.4-dirty [28], MathSAT version 5.6.11 [29].

Only the most promising SMT solver, Bitwuzla, was benchmarked with different parameters related to theory of BitVectors, rewriting, solver backends.

## 3.1. Encodings

**Simplifying Compression Functions** The non-linear $Ch$ and $Maj$ functions, described in Section 2.1 can be simplified, while preserving the logical output, by replacing XOR operations with OR:

$$Ch(e, f, g) = (e \wedge f) \vee (\neg e \wedge g)$$
$$Maj(a, b, c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$$

**Alternative Bitwise Addition** Encodings of `bvadd` in solvers tend to be optimised for propagation and space [30]. This does not allow for easy reasoning about the differential between pairs of adders. To improve this, a bitwise carry-lookahead multi-operand adder has been generated in SMT-LIB. The adder computed generate ($g$) and propagate ($p$) signals, for two BitVector operand inputs $a$ and $b$, where $i \leq$ number of BV input bits:

$$g_0 = a \wedge b$$
$$p_0 = a \oplus b$$

$$g_{i+1} = g_i \vee (p_i \wedge (g_i \ll 2^i))$$
$$p_{i+1} = p_i \wedge (p_i \ll 2^i)$$

$$\text{bitadd-2}(a, b) = p_0 \oplus (g_{length} \ll 1)$$
$$\text{bitadd-}n(x_1, \ldots, x_n) = \text{bitadd-2}(\bigoplus_{k=1}^{n} x_k \ll \bigvee_{1 \leq i < j \leq n} (x_i \wedge x_j))$$

This hierarchical structure, in theory, enables $O(\log n)$ carry propagation, though practical SMT contraints necessitated sequential left-to-right chaining for multi-operand cases.

**Differential Encoding** In order to move to a differential cryptanalysis reasoning model, differentials have been defined between corresponding pairs of computation values in the two SHA-2 instances. These encodings include Delta Subtraction (DSub) $\Delta_- = x - x'$, and Delta Exclusive Or (DXOR) $\Delta_\oplus = x \oplus x'$.

Both encodings were systematically applied to message block differences during expansion, working variables ($a$–$h$) in the compression function, round specific constants ($K$), and final digests. Base assertions on absolute values have been converted to assert on differential values, but otherwise no additional assertions have been used. All the underlying absolute values were still present and calculated for each variable to ensure compliance with the SHA-2 definition.

## 3.2. Reproducibility

All code was written in Rust (rustc version 1.85.1) [31], compiled with the provided LLVM backend, and linked with mold (version 2.37.1) [32]. The code can be accessed, with further documentation: https://github.com/Supermarcel10/CryptographicAnalysisOfSha2. The default parameters part of our software documentation have been used, unless otherwise specified.

A dedicated X86_64 machine was set up, with a fresh installation of NixOS 25.05 (Warbler), utilising the Linux Realtime 6.6.77-rt50 kernel. For hardware, the machine consisted of an AMD Ryzen 9 5900X processor, paired with 4x32GiB DDR4 memory sticks. The BIOS was configured with `CPU Clock 37.00`, `CPU Clock Control 100.000 MHz`, `DDR4-3600 18-22-22-52-64-1.35V`, `CPU VCore 0.818V`, `CPU VCore Loadline Calibration LOW`, `CSM Support ENABLED`. This runner can be rebuilt at any time, using the NixOS configuration: https://github.com/Supermarcel10/NixOSConfig/blob/f1d26ec/devices/E01/configuration.nix.

# 4. Results

This section presents highlights of our experimental results. Outputs deemed invalid by our known-good SHA-2 implementation, or resulting in an error code, have been excluded from plots. One example of this is Colibi2, which failed to meet SMTLIB constraints. For an exhaustive set of results, please refer to Appendix B. All raw results are available in the codebase, described in Section 3.2.

Two different visualisation layouts are used to address the research questions.

**Comparison Graph** These provide a side-by-side comparison of each solver's performance as lines on a 2D plot. The rounds are displayed on the X-axis and $\log 2$ time on the Y-axis. Results that time out are set to the maximum possible time. The line that goes the most to the right (more rounds), while staying low (less time) is the best. Additionally, the consistency and linearity of the line provides insight about the reasoning of the problem. This graph is used to answer RQ1 in Subsection 4.1.

**Baseline Graph** The baseline graph was designed to compare between different runs of the same solver with parameters and encodings. It is a 2D line graph with the baseline in the middle. Any missing or invalid data is skipped from plotting. Deviation data is then calculated from the baseline based on time difference, and represented as a percentage.

If no plot exists on baseline, but a valid deviation is plotted, $+\infty$ will be used as the deviation value. This can be interpreted as "This was infinitely faster than the nothing achieved (on baseline) within timeout". If a baseline exists, but no deviation for that round is present, the plot will be skipped.

A rough run-to-run variance is shown with a grey colour near the middle. The green area with $-\%$ implies "this took $x\%$ less time, compared to baseline". Similarly, the red area with $+\%$ implies "this took $x\%$ more time, compared to baseline".

The graph scales with the results, and tops out at 100% in both directions. When a result is plotted on the edge of the graph, the deviation is $\geq 100\%$ (i.e. twice as long or half as long). This graph is used to visualise results for RQ2 and RQ3 in Subsection 4.1.

## 4.1. Choice of Solver

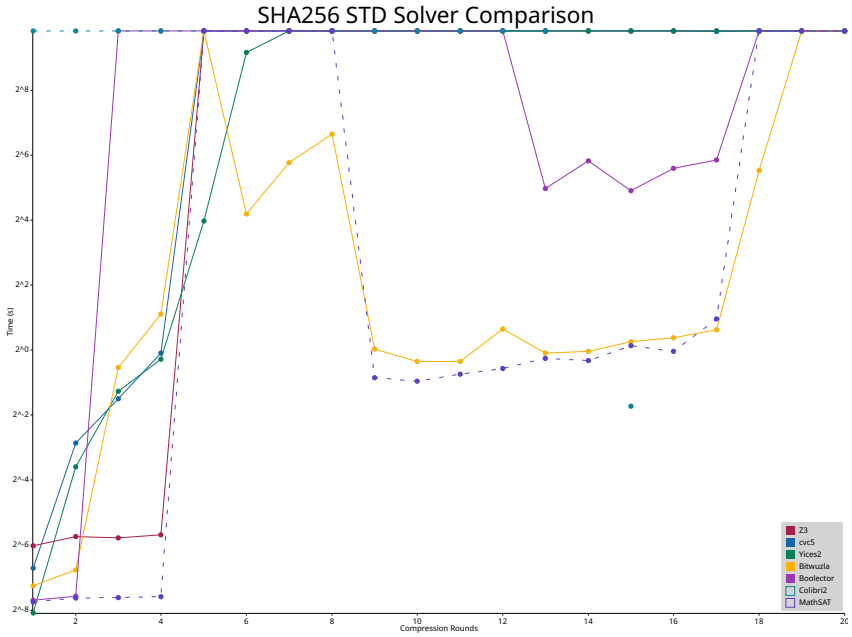


**Figure 1:** Comparison graph representing SHA-256 standard collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha256 --collision-type std --continue-on-fail true`.

Figure 1 helps answer RQ1 and showcases interesting aspects of SMT solver choice. No solver could find a collision for 5 rounds, although subsequent rounds terminated with their respective outcomes. We believe this could be misleading; the common approach to using automated reasoning tools of "increasing the rounds until you get a timeout" could cause incorrect conclusions.

Bitwuzla was the most consistent SMT solver, being the only one capable of proving non-existence of a collision for 7 and 8 rounds. This could suggests that Bitwuzla may have stronger capabilities for UNSAT formulas as opposed to its competitors.

As most solvers struggled with higher-round collisions, Bitwuzla and MathSAT were capable of pushing through and delivering 18 and 17 rounds respectively. Bitwuzla is the definitive winner, being both more consistent and able to deliver the most rounds within the timeout period. This means Bitwuzla is the baseline SMT solver to beat.

With a 10 hour timeout, no collisions were found for 19 rounds using Bitwuzla.

## 4.2. Choice of Parameters

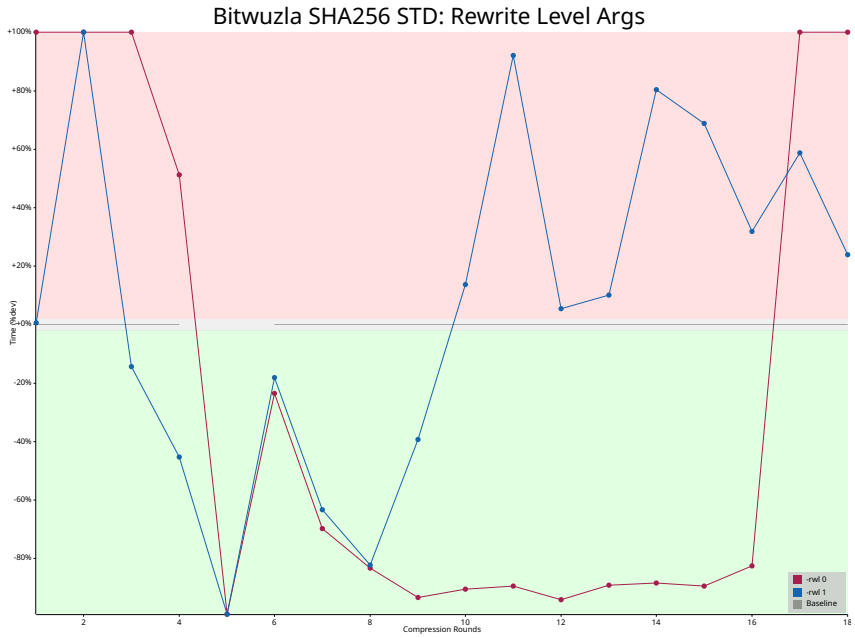


**Figure 2:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to rewrite level, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

To answer RQ2, we tried an exhaustive list of parameters available on the most promising SMT solver, Bitwuzla. Figures 2 and 3 signified that adjusting Bitwuzla's parameters can impact performance both positively and negatively. Despite this, collision performance did not improve.

Setting the rewrite level to 0 or disabling `variable-subst` provided a positive performance uplift for short-running collisions but hindered long-running ones.

The Kissat SAT solver backend outperformed the baseline (CaDiCaL) in short-running rounds, but may perform worse in longer-running collisions, as implied by the upwards trend near the right edge of Figure 4.

Enabling Bitwuzla's propagation-based local search engine (`--bv-solver prop`) negated performance gains, highlighting the performance advantage likely stems from an optimized bit-blasting strategy compared to other solvers.

Enabling multithreading via CryptoMiniSat improved overall performance, though increasing threads beyond four (the lowest tested) led to diminishing returns. This is likely caused by practical trade-offs related to SMT parallelism and SMT solver memory characteristics.

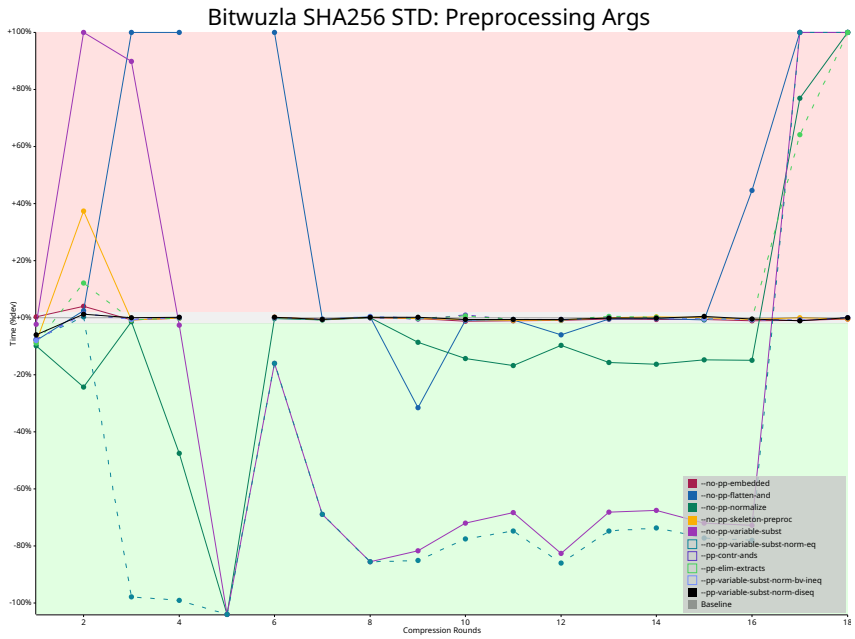


**Figure 3:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to preprocessing, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

### 4.3. Choice of Encodings

To answer RQ3, all combinations of differential cryptanalysis encodings presented in Section 3.1 have been benchmarked. These can be seen in Figure 5 and Figure 6.

While there were no improvements in the last round computable, this data gives a clearer direction for future work.

The delta subtract encoding, did not allow the solver to reason more freely as expected, and performed within margin of error. It did help influence the solvability of round 5, where previously no result was found within the timeout period.

The delta XOR encoding proved more meaningful, especially in short-running collisions. It is unclear if delta XOR provides benefits for long-running collisions, but seems more promising than delta subtract.

As for the alternative bitwise add, it performed worse for solving time. Despite this, improved linearity for some output cases has been observed. Listings 2 and 1 showcases less altered bits, This implies the SMT solver has taken a different direction, reasoning more about reducing these differenes.

$Maj$ and $Ch$ simplification had minor effects, but did not paint a clear enough picutre for a conclusive answer.

In combination with more powerful differential cryptanalysis encodings, it is possible the presented encodings could be help in other aspects.

## 5. Conclusion

The graphs answer RQ1: Bitwuzla [26] stands out as the most promising SMT solver among those tested. While MathSAT [29] performed closely, it fell short in terms of round solvability.

It is plausible that an alternative set of parameters could improve other solvers' performance, such as MathSAT, leaving RQ2 partially unanswered. It is conclusive that default parameters for Bitwuzla are the most stable for round solvability and solving time.

In order to answer RQ3, this work analysed basic encodings to provide insights. Differential cryptanalysis by subtraction simply does not work for SHA-256.

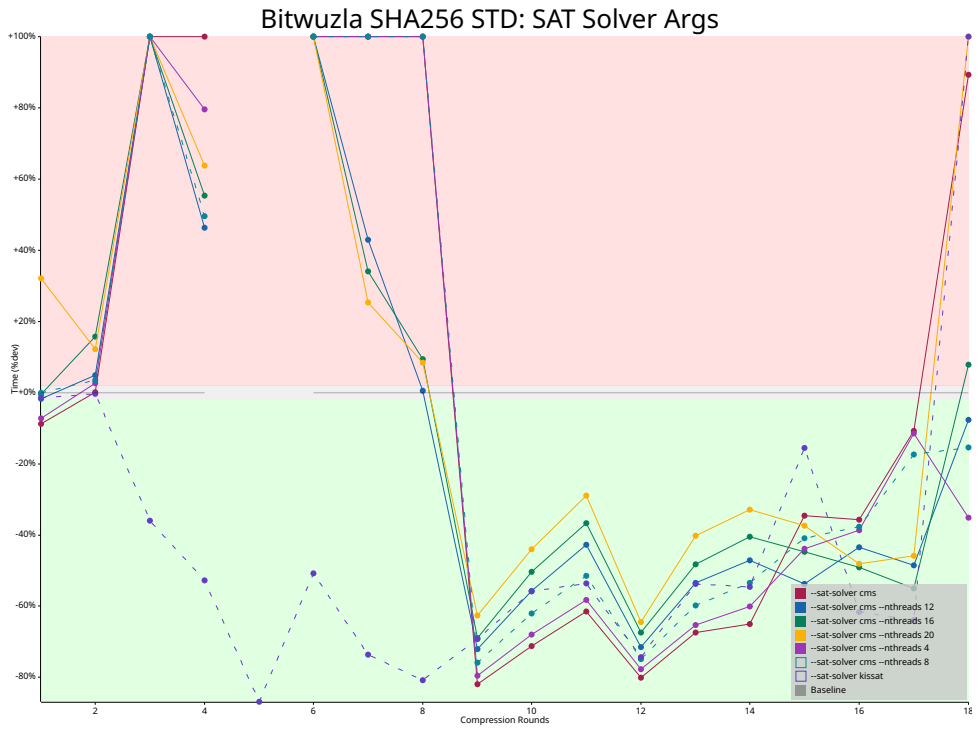


**Figure 4:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to SAT solver backend, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

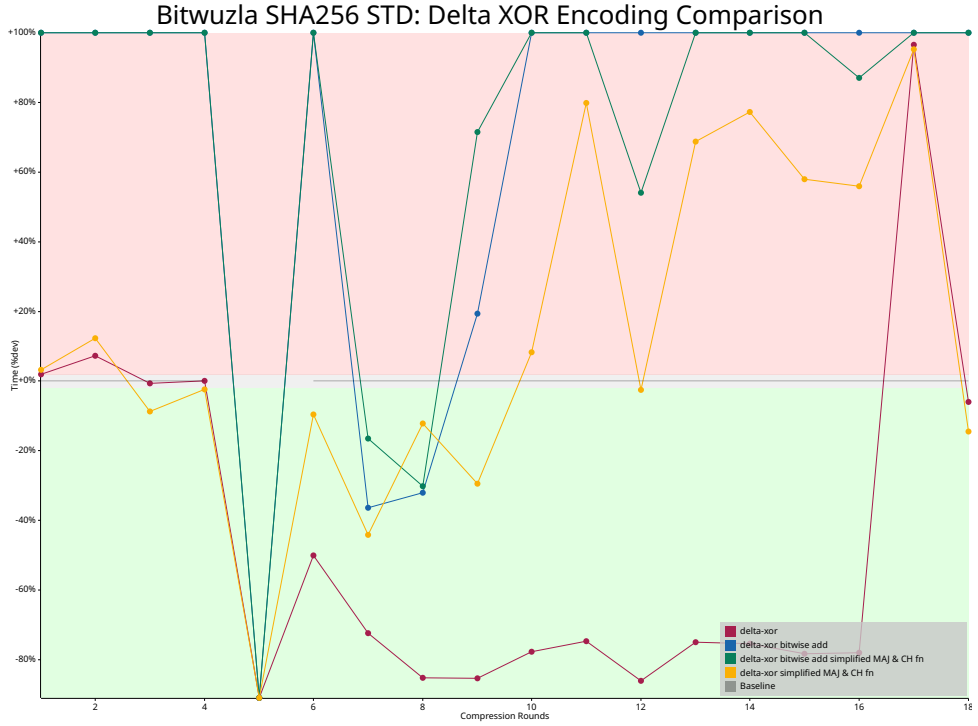


**Figure 5:** Baseline graph showcasing Bitwuzla SHA-256 standard collision from 1 to 20 rounds using $\Delta_{\oplus}$ encoding. Where each colour line represents a different encoding variant. Results ran with parameters `--round-range 1..19 --stop-tolerance 0 --continue-on-fail true --encoding-type dxor:<maj/ch simpl>:<a.add>`

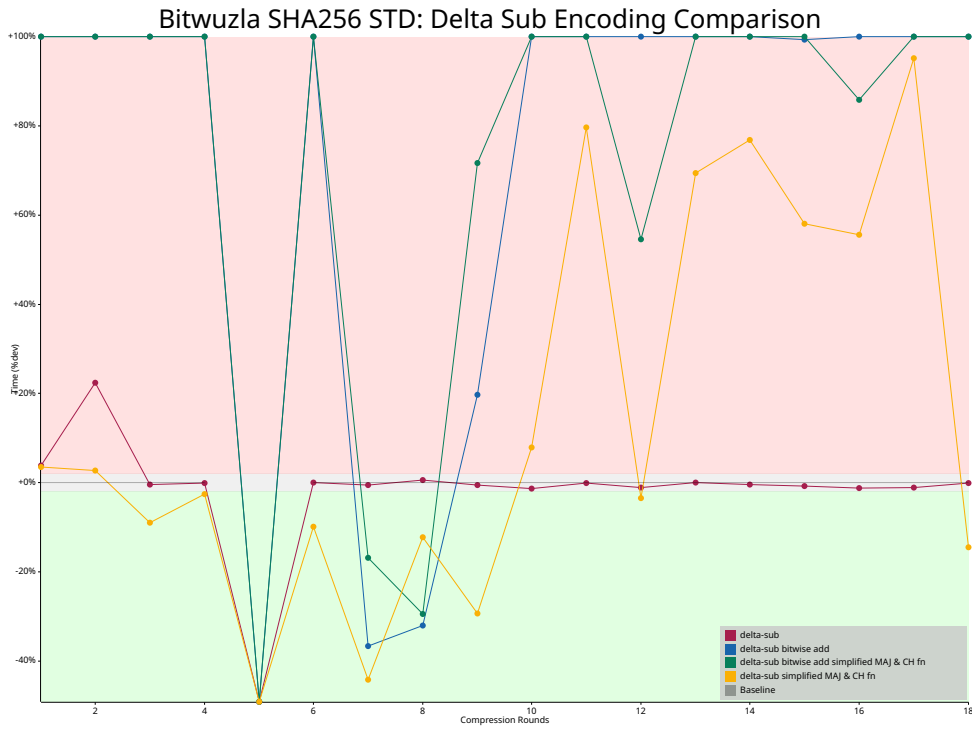


**Figure 6:** Baseline graph showcasing Bitwuzla SHA-256 standard collision from 1 to 20 rounds using $\Delta_{-}$ encoding. Where each colour line represents a different encoding variant. Results ran with parameters `--round-range 1..19 --stop-tolerance 0 --continue-on-fail true --encoding-type dsub:<maj/ch simpl>:<a.add>`

Generating a brute-force approach modelling the SHA-2 standard proved ineffective, but performed significantly better than previous literature would imply. By identifying the most promising SMT solver, this work established a recommended SMT solver choice and a baseline claim to beat for future research.

[22]'s claim that MathSAT is always inferior to Yices2 holds untrue for a larger number of compression rounds, as seen by our contradicting results.

## 6. Future Work

**Encoding Based Work** Encodings described in [12] or [16], or a combination of both could be built in SMT-LIB. Running these encodings on Bitwuzla, would allow comparable results against SAT or SAT + CAS approaches of these works.

**SMT Parameter Exploration Work** Further exploration into parameter performance effects could be assessed. Other promising SMT solvers, such as MathSAT, could outperform Bitwuzla with the right choice of parameters.

**Hardware Based Work** Hardware choice has not been directly benchmarked towards solving performance. A performance comparison of different processors would paint a clearer picture and recommendation on choice towards better solving performance. For example, an AMD Ryzen 7 9800X3D [33], may benefit solving performance due to the nature of solving being dependent on memory locality.

## Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

## References


[1] National Institute of Standards and Technology (NIST), Secure Hash Standard (SHS), Federal Information Processing Standards Publication (FIPS PUB) 180-4, 2015. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf, specifies hash algorithms including SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. Effective March 6, 2012, with updates in August 2015.

[2] E. Rescorla, RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3, Technical Report, IETF, 2018. URL: https://www.rfc-editor.org/rfc/rfc8446.

[3] E. Rescorla, HTTP Over TLS, Technical Report, IETF, 2000.

[4] D. Eastlake, C. Kaufman, RFC 2065: Domain Name System Security Extensions, Technical Report, IETF, 1997. URL: https://datatracker.ietf.org/doc/html/rfc2065.

[5] T. Ylonen, C. Lonvick, RFC 4251: The Secure Shell (SSH) Protocol Architecture, Technical Report, IETF, 2006. URL: https://datatracker.ietf.org/doc/html/rfc4251.

[6] E. P. Wouters, D. Huigens, J. Winter, Y. Niibe, RFC 9580: OpenPGP Message Format, Technical Report, IETF, 2024. URL: https://www.rfc-editor.org/info/rfc9580.

[7] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[8] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, 2018. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[9] Q. Dang, Recommendation for Applications Using Approved Hash Algorithms, Technical Report NIST SP 800-107 Rev. 1, National Institute of Standards and Technology (NIST), Gaithersburg, MD, 2012. URL: https://csrc.nist.gov/pubs/sp/800/107/r1/final. doi:10.6028/NIST.SP.800-107r1.

[10] E. Biham, A. Shamir, Differential cryptanalysis of des-like cryptosystems, 1991. URL: https://link.springer.com/article/10.1007/BF00630563.

[11] D. Nugent, The story and math of differential cryptanalysis, 2023. URL: https://evervault.com/blog/Story-and-Math-of-Differential-Cryptanalysis, describes the history of Differential Cryptanalysis including the in-depth analysis of Whitfield Diffie and Martin Hellman related to NSA's secretly hand-picked numbers in 1970s.

[12] Y. Li, F. Liu, G. Wang, New records in collision attacks on SHA-2, Cryptology ePrint Archive, Paper 2024/349, 2024. URL: https://eprint.iacr.org/2024/349.

[13] C. Dobraunig, M. Eichlseder, F. Mendel, Analysis of SHA-512/224 and SHA-512/256, Cryptology ePrint Archive, Paper 2016/374, 2016. URL: https://eprint.iacr.org/2016/374. doi:10.1007/978-3-662-48800-3_25.

[14] F. Mendel, T. Nad, M. Schläffer, Improving local collisions: New attacks on reduced sha-256, in: T. Johansson, P. Q. Nguyen (Eds.), Advances in Cryptology – EUROCRYPT 2013, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 262–278. URL: https://link.springer.com/chapter/10.1007/978-3-642-38348-9_16.

[15] S. K. Sanadhya, P. Sarkar, New collision attacks against up to 24-step sha-2, in: D. R. Chowdhury, V. Rijmen, A. Das (Eds.), Progress in Cryptology - INDOCRYPT 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 91–103. URL: https://link.springer.com/chapter/10.1007/978-3-540-89754-5_8.

[16] N. Alamgir, S. Nejati, C. Bright, Sha-256 collision attack with programmatic sat, 2024. URL: https://arxiv.org/abs/2406.20072. arXiv:2406.20072.

[17] M. Eichlseder, F. Mendel, M. Schläffer, Branching heuristics in differential collision search with applications to SHA-512, Cryptology ePrint Archive, Paper 2014/302, 2014. URL: https://eprint.iacr.org/2014/302. doi:10.1007/978-3-662-46706-0_24.

[18] X. Wang, D. Feng, X. Lai, H. Yu, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD, Cryptology ePrint Archive, Paper 2004/199, 2004. URL: https://eprint.iacr.org/2004/199.

[19] RyotaK, Flatt Security Inc., Compromising OpenWrt supply chain via truncated SHA-256 collision and command injection, 2024. URL: https://flatt.tech/research/posts/compromising-openwrt-supply-chain-sha256-collision/, disclosed vulnerabilities in Open-

Wrt's Attended SysUpgrade (ASU) service, combining command injection and truncated hash collisions (CVE-2024-54143).

[20] Hashcat Developers, hashcat, 2025. URL: https://github.com/hashcat/hashcat.

[21] W. A. Osama M., B. A., Certified sat solving with gpu accelerated inprocessing, 2024. URL: https://doi.org/10.1007/s10703-023-00432-z.

[22] E. Bellini, A. D. Piccoli, M. Formenti, D. Gerault, P. Huynh, S. Pelizzola, S. Polese, A. Visconti, Differential cryptanalysis with SAT, SMT, MILP, and CP: a detailed comparison for bit-oriented primitives, Cryptology ePrint Archive, Paper 2024/105, 2024. URL: https://eprint.iacr.org/2024/105. doi:10.1007/978-981-99-7563-1_13.

[23] L. de Moura, N. Bjørner, Z3: An efficient smt solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 337–340. URL: https://link.springer.com/chapter/10.1007/978-3-540-78800-3_24.

[24] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength smt solver, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer International Publishing, Cham, 2022, pp. 415–442. URL: https://link.springer.com/chapter/10.1007/978-3-030-99524-9_24.

[25] B. Dutertre, Yices 2.2, in: A. Biere, R. Bloem (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2014, pp. 737–744. URL: https://link.springer.com/chapter/10.1007/978-3-319-08867-9_49.

[26] A. Niemetz, M. Preiner, Bitwuzla, in: C. Enea, A. Lal (Eds.), Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II, volume 13965 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 3–17. URL: https://doi.org/10.1007/978-3-031-37703-7_1. doi:10.1007/978-3-031-37703-7_1.

[27] R. Brummayer, A. Biere, Boolector: An efficient smt solver for bit-vectors and arrays, in: S. Kowalewski, A. Philippou (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 174–177. URL: https://link.springer.com/chapter/10.1007/978-3-642-00768-2_16.

[28] Frama C, Colibri2, N/D. URL: https://colibri.frama-c.com/.

[29] A. Cimatti, A. Griggio, B. J. Schaafsma, R. Sebastiani, The mathsat5 smt solver, in: N. Piterman, S. A. Smolka (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 93–107. URL: https://link.springer.com/chapter/10.1007/978-3-642-36742-7_7.

[30] M. Brain, L. Hadarean, D. Kroening, R. Martins, Automatic generation of propagation complete sat encodings, in: B. Jobstmann, K. R. M. Leino (Eds.), Verification, Model Checking, and Abstract Interpretation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 536–556. URL: https://link.springer.com/chapter/10.1007/978-3-662-49122-5_26.

[31] The Rust Project Developers, The rust programming language, 2025. URL: https://www.rust-lang.org, rust C Version 1.85.1.

[32] R. Ueyama, mold: A modern linker, 2025. URL: https://github.com/rui314/mold, version 2.37.1.

[33] Advanced Micro Devices, Inc., Amd ryzen 7 9800x3d, 2025. URL: https://www.amd.com/en/products/processors/desktops/ryzen/9000-series/amd-ryzen-7-9800x3d.html.

# A. Listings

Both of these outputs showcase the differential graphs with notation as [12]. The $\Delta$ and $_i$ notation has been escaped due to being invalid UTF-8 in LaTeX.

Listing 1: 14 round collision output obtained by running `sha2-collision benchmark --solver bitwuzla --hash-function sha256 --collision-type std --round-range 14..15 -R true -E bruteforce::true`

```
14 rounds; SHA256 STD collision; Bitwuzla; SMT solver PID: 57943
File: smt/SHA256_STD_14_ALTADD.smt2
CV   : 6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
CV'  : 6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
M    : ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   ffffffff ffffffff fffffe9 00000000 00000000
M'   : 7fffffff ddf3fdbf 7c8b10a7 de0fffbf a1ec023f 9dec01bf ffffffff 7fffffff 7fffffff ffffffff ffffffff
   ffffffff ffffffff fffffe9 00000000 00000000
Hash : 1121e8fd ad9d9f9f 5e16068c 8acbfb6b 9cde4233 a73a2f5f dc9ced0a d8f47aa2 (Valid? true)
Hash': 1121e8fd ad9d9f9f 5e16068c 8acbfb6b 9cde4233 a73a2f5f dc9ced0a d8f47aa2 (Valid? true)
```

| i | A | E | W |
|---|---|---|---|
| 0 | ================================ | ================================ | u================================ |
| 1 | u================================ | u================================ | ==u===u=====uu========u==u====== |
| 2 | ================================ | =nn==========n========u========= | u=====uu=uuu=u==uuu=uuuu=u=uu=== |
| 3 | ================================ | u================================ | ==u====uuuuu=============u====== |
| 4 | ================================ | n================================ | =u=uuuu====u==uuuuuuuu=uuu====== |
| 5 | ================================ | u================================ | =uu===u====u==uuuuuuuuu==u====== |
| 6 | ================================ | ================================ | ================================ |
| 7 | ================================ | ================================ | u================================ |
| 8 | ================================ | ================================ | u================================ |
| 9 | ================================ | ================================ | ================================ |
| 10 | ================================ | ================================ | ================================ |
| 11 | ================================ | ================================ | ================================ |
| 12 | ================================ | ================================ | ================================ |
| 13 | ================================ | ================================ | ================================ |

Listing 2: 14 round collision output obtained by running `sha2-collision benchmark --solver bitwuzla --hash-function sha256 --collision-type std --round-range 14..15 -R true -E bruteforce::`

```
14 rounds; SHA256 STD collision; Bitwuzla; SMT solver PID: 58100
File: smt/SHA256_STD_14.smt2
CV    : 6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
CV'   : 6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
M     : 00000000 00269eb0 073a5f45 870a253d 10cf61c3 340c932a 252046ec a8e31d41 3bf4e7e6 ba76205e 9c4e594a
    38c84784 c504f3aa f3ea62bc 00000000 00000000
M'    : 00900018 a97b43d8 bc209ac8 bb2001f0 a79d7d46 fe43c1a8 38304343 860d9489 7b9b7537 d7879422 20a912e5
    227a74d1 6e52b216 6efc2748 00000000 00000000
Hash  : 2d09ea67 bb67ae85 3c6ef372 a54ff53a 8550d704 9f05689c b00572e3 a5670f5a (Valid? true)
Hash' : 2d09ea67 bb67ae85 3c6ef372 a54ff53a 8550d704 9f05689c b00572e3 a5670f5a (Valid? true)
```

| i | A | E | W |
|---|---|---|---|
| 0 | ================================ | ================================ | ========n==n==============nn=== |
| 1 | ========n==n==============n=u=== | =======nu==n==============nn=== | n=n=n==n=n=nnu=nun=uuu=n=nu=n=== |
| 2 | nu===nuu==nn=n=nnn=n=n=u=un===== | ==nuuuu==nn=u=n=n==u=nnn==nnn=== | n=nnn=uu===uu=u=nu===u=un===nu=u |
| 3 | u=n====nn=nu=nunnn===nnunu=n==u= | ==u=unun===u===nnu==nunuu==nnu== | ==nnnu====n=u=u===u==u==nn==uu=u |
| 4 | unn=nu=u==uu=u===uuu==nu=uu==n=u | ===nnn=uuu==u===n==un======un=un | n=nu=nnn=u=n==u====nnn==u====n=u |
| 5 | n=u=uu=nnn===n=======nn========= | ==u=n=====n=u===nn====nn==n=u=== | nn==n=n==n==uunn=n=u==u=n=====u= |
| 6 | u===nu=n===u==u==n===u==u==u=n== | nuun==nn=n===n=n======u===n=nnnn | ===nnu=u===n=========u=nu=u=uunn |
| 7 | ================================ | u=u===u===n=u===u=u=n==n=n=nnn=n | ==u=unn=uuu=nnu=n===u==unu==n=== |
| 8 | ================================ | ====n=uunuuuu==un===nnn=n=n==n=u | =n=======uu=nunnu==n==u=uu=n===n |
| 9 | ================================ | =n=n===nnn===n=======nn====n==== | =nu=un=nnuuu===nn=un=n===unuuu== |
| 10 | ================================ | u====n=n==u=nnn==n===u==u==u=n== | u=nuuu==nun==uun=u==u=nun=n=unun |
| 11 | ================================ | ================================ | ===uu=n=u=nn==n===nn==uu=n=n=u=n |
| 12 | ================================ | ================================ | u=n=n=nu=n=n=un==u=====uu=unun== |
| 13 | ================================ | ================================ | u==unn=u===n=nu==u===n=nunuu=u== |

# B. Full Results

Expanding on the graphs mentioned in Section 4, an additional graph is used.

**Detailed Graph** Unlike other graphs, only a single benchmark run is plotted in this graph. It utilises a pair of 2D coordinate systems, where the first line graph relates to time taken in log 2, and the second to memory usage in Mibibytes (MiB). Since SMT solving prefers more locality in memory hierarchy, this graph can help understand if the performance degradation is due to hardware limitations, such as saturating all CPU L3 cache. This graph is used to visualise data in Appendix B.

## B.1. Graph Results

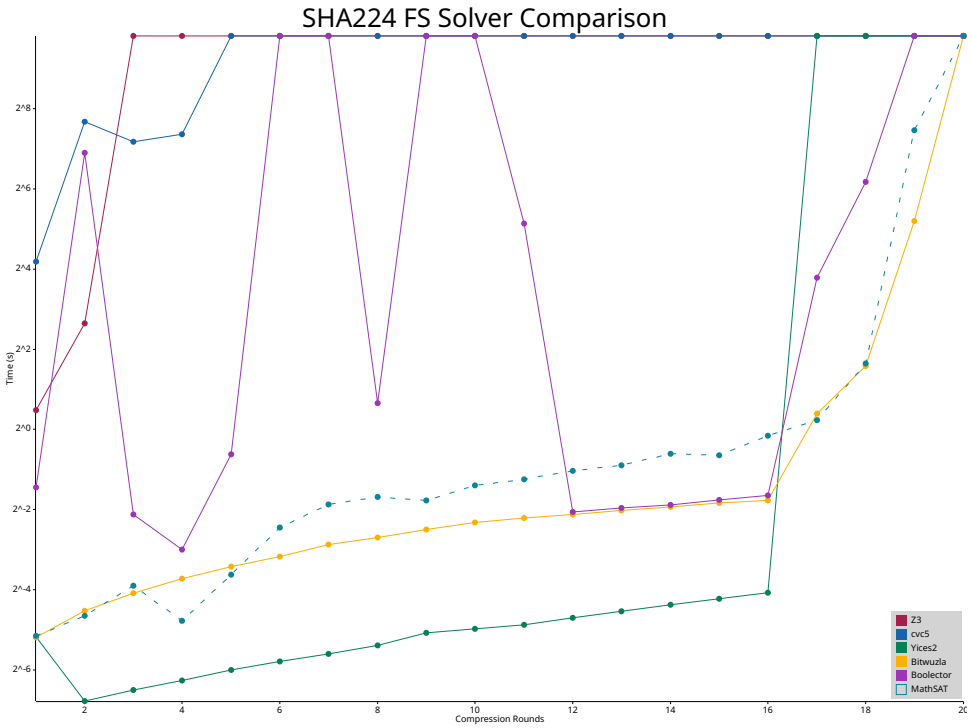


**Figure 7:** Comparison graph representing SHA-224 free-start collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha224 --collision-type fs --continue-on-fail true`.

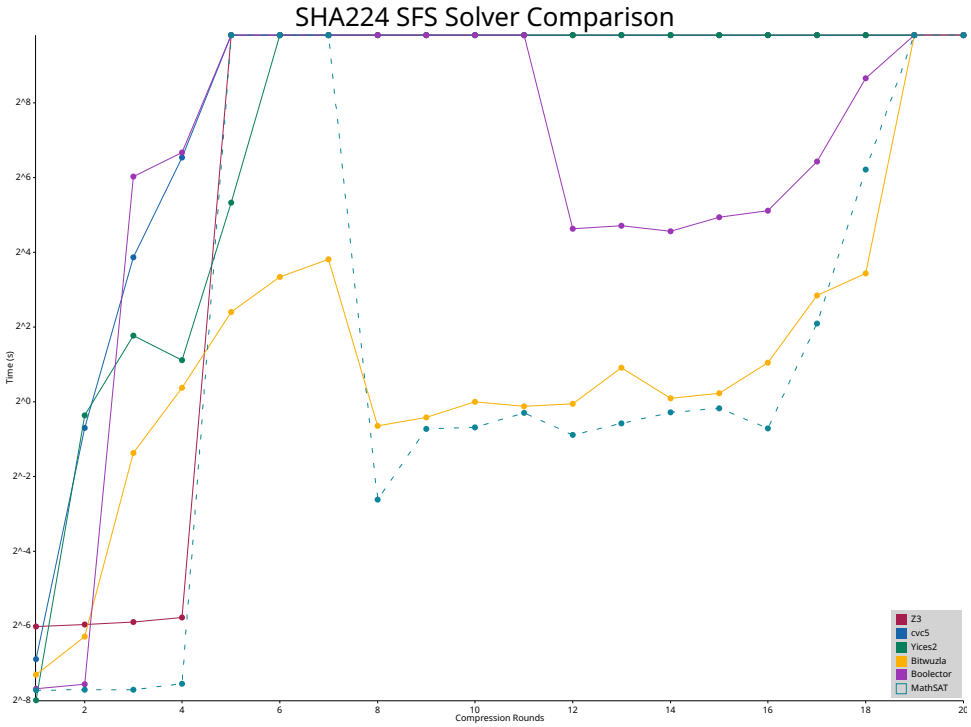


**Figure 8:** Comparison graph representing SHA-224 semi-free-start collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha224 --collision-type sfs --continue-on-fail true`.

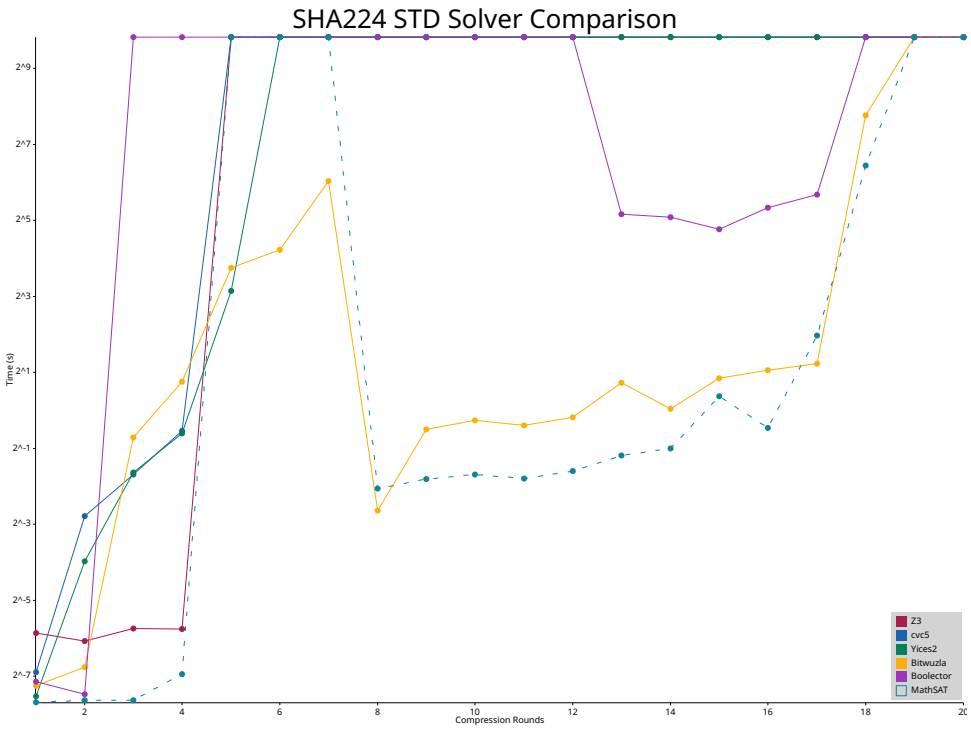


**Figure 9:** Comparison graph representing SHA-224 standard collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha224 --collision-type std --continue-on-fail true`.

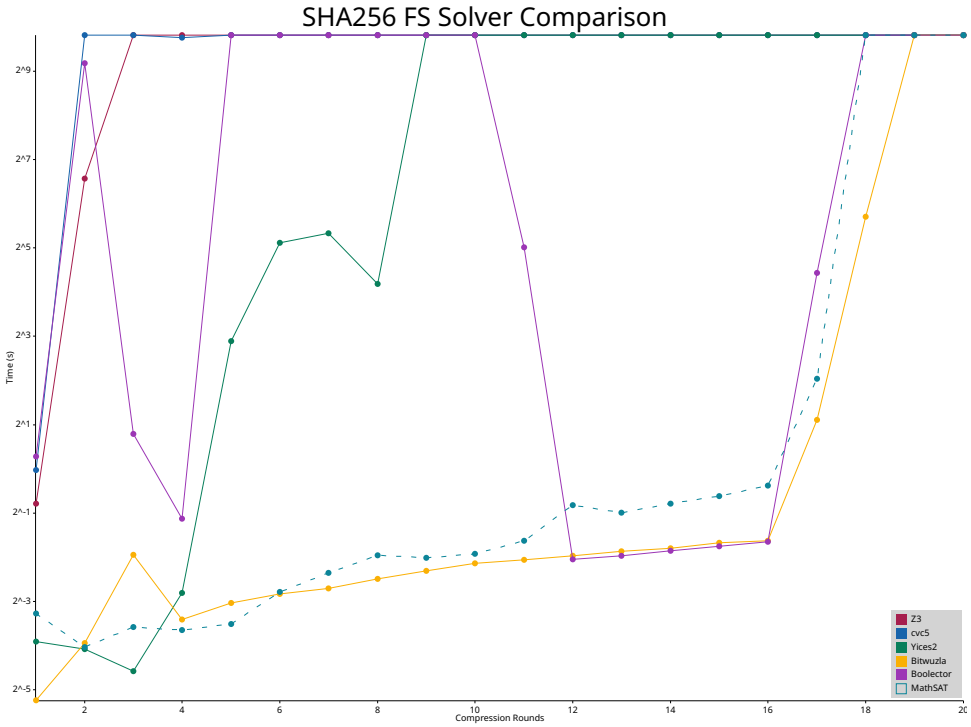


**Figure 10:** Comparison graph representing SHA-256 free-start collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha256 --collision-type fs --continue-on-fail true`.

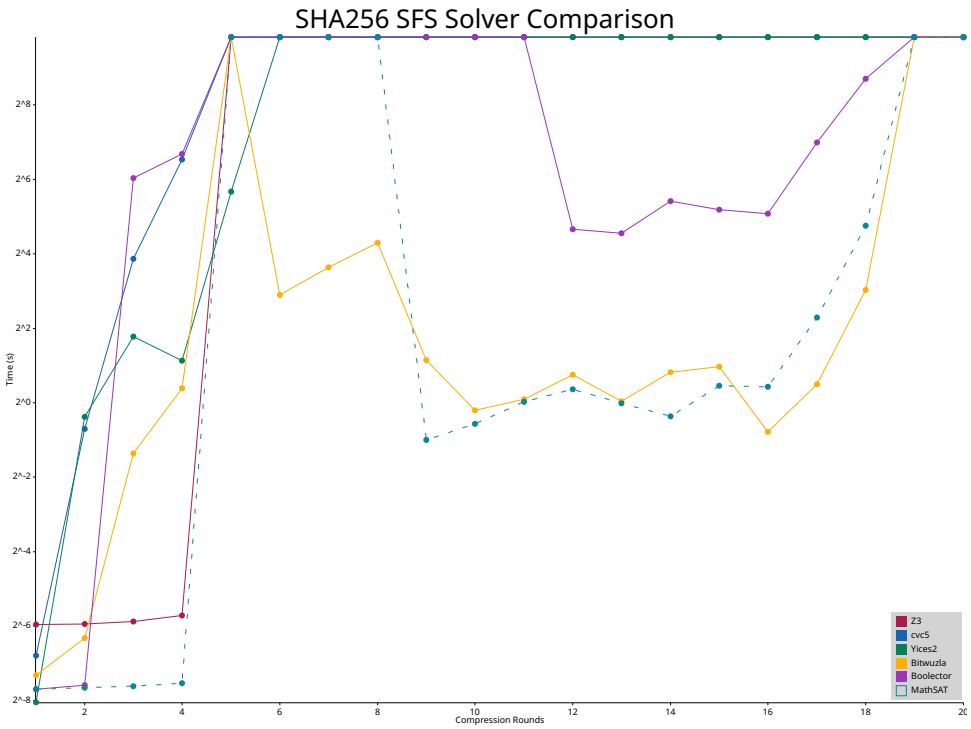


**Figure 11:** Comparison graph representing SHA-256 semi-free-start collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha256 --collision-type sfs --continue-on-fail true`.

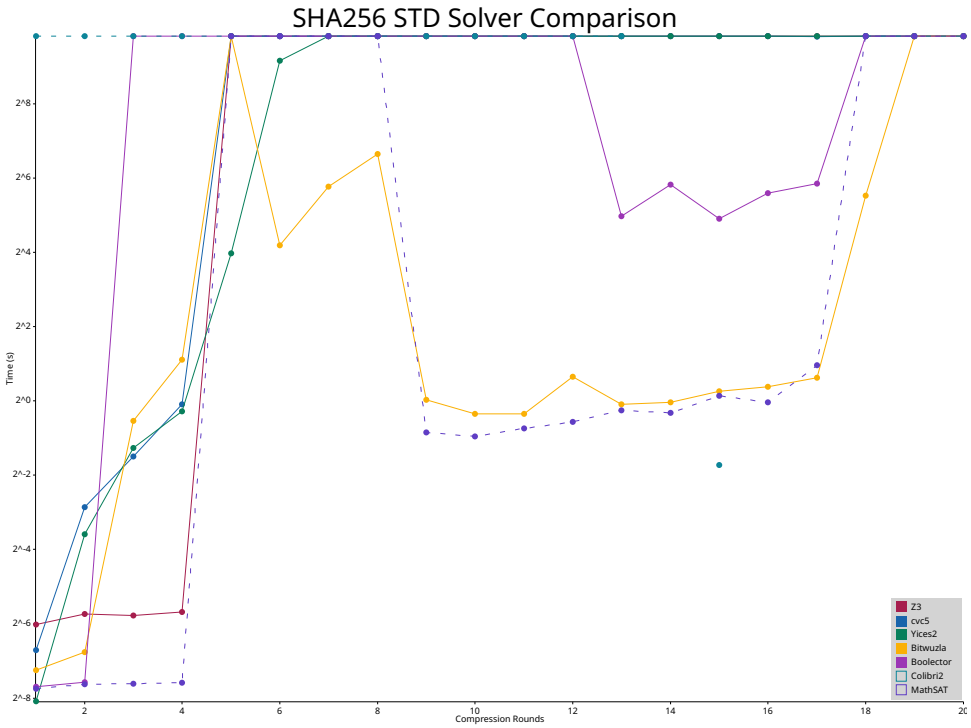


**Figure 12:** Comparison graph representing SHA-256 standard collisions from 1 to 20 rounds using brute-force, where each colour line represents a separate solver. Results ran with parameters `--round-range 1..21 --hash-function sha256 --collision-type std --continue-on-fail true`.

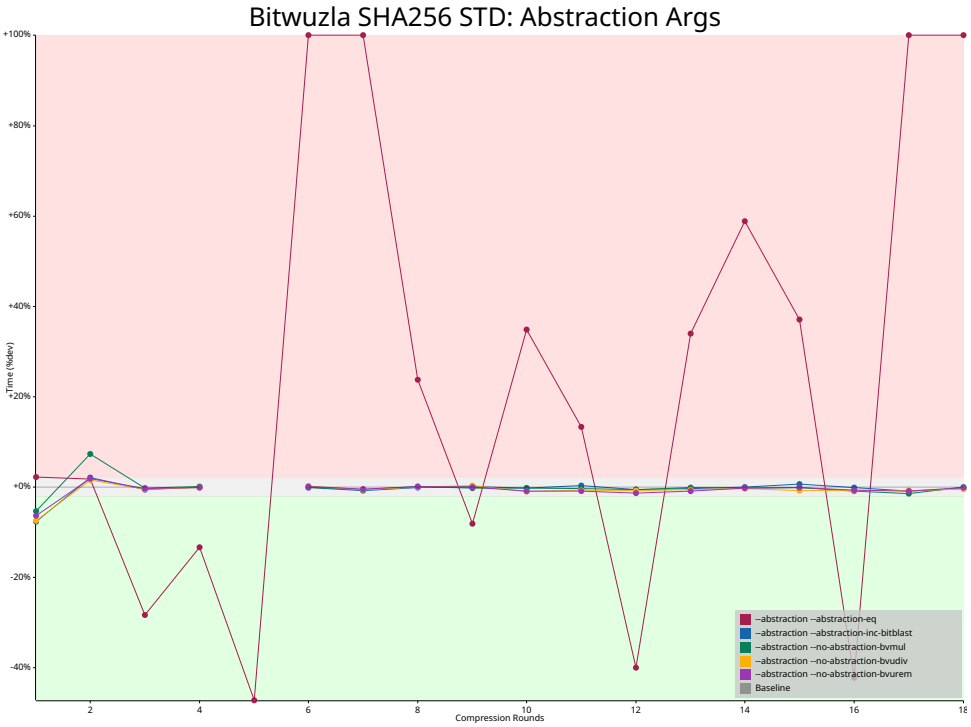


**Figure 13:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to abstraction, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

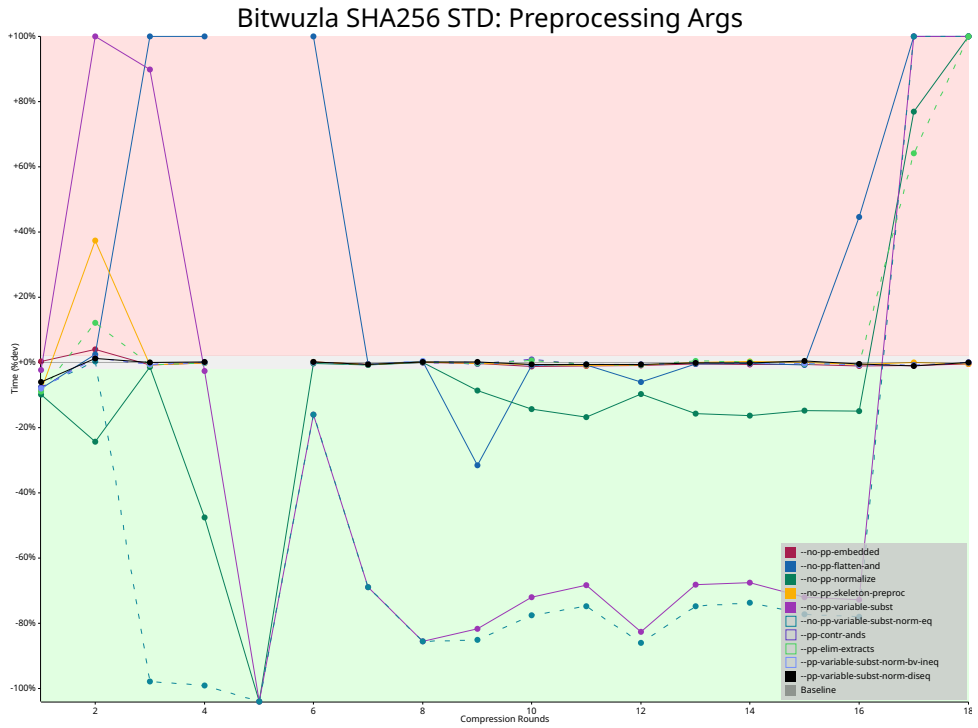


**Figure 14:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to preprocessing, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

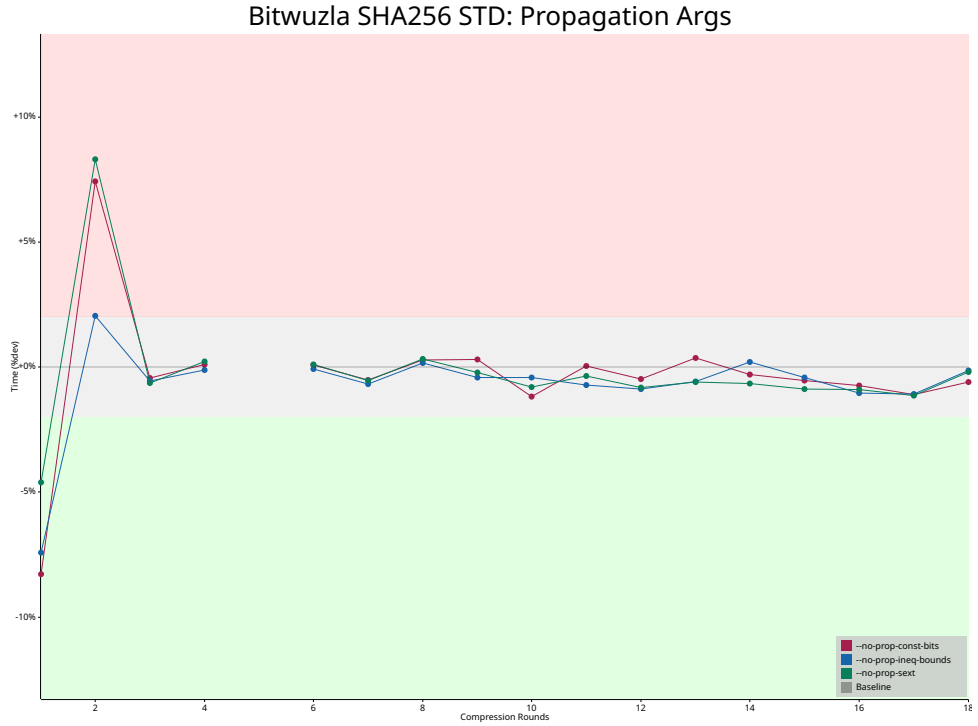


**Figure 15:** Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to propagation. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

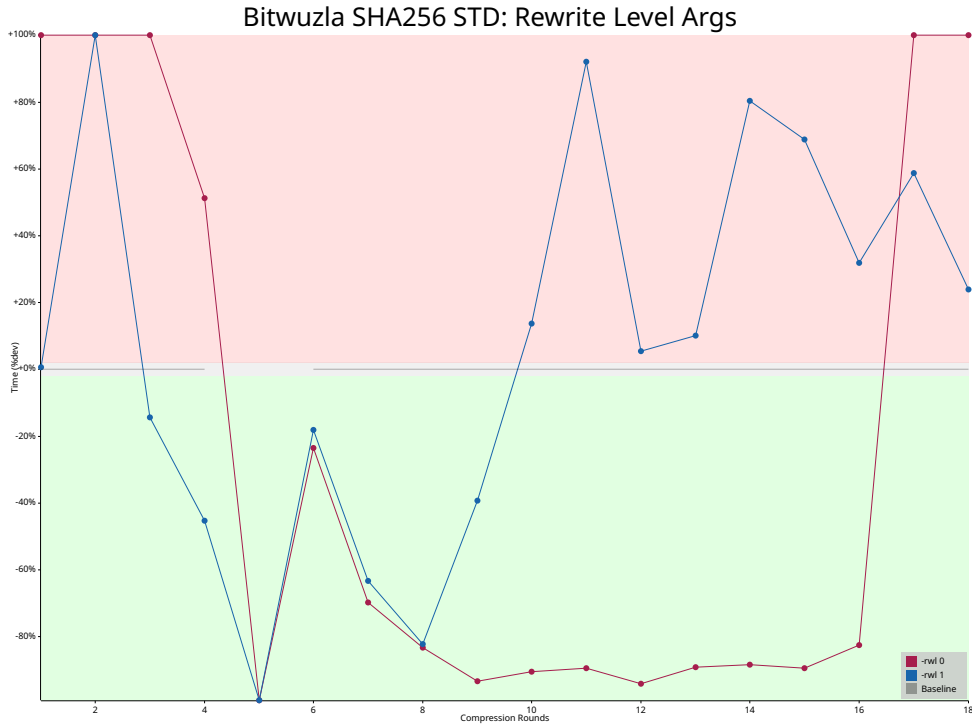


**Figure 16:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to rewrite level, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

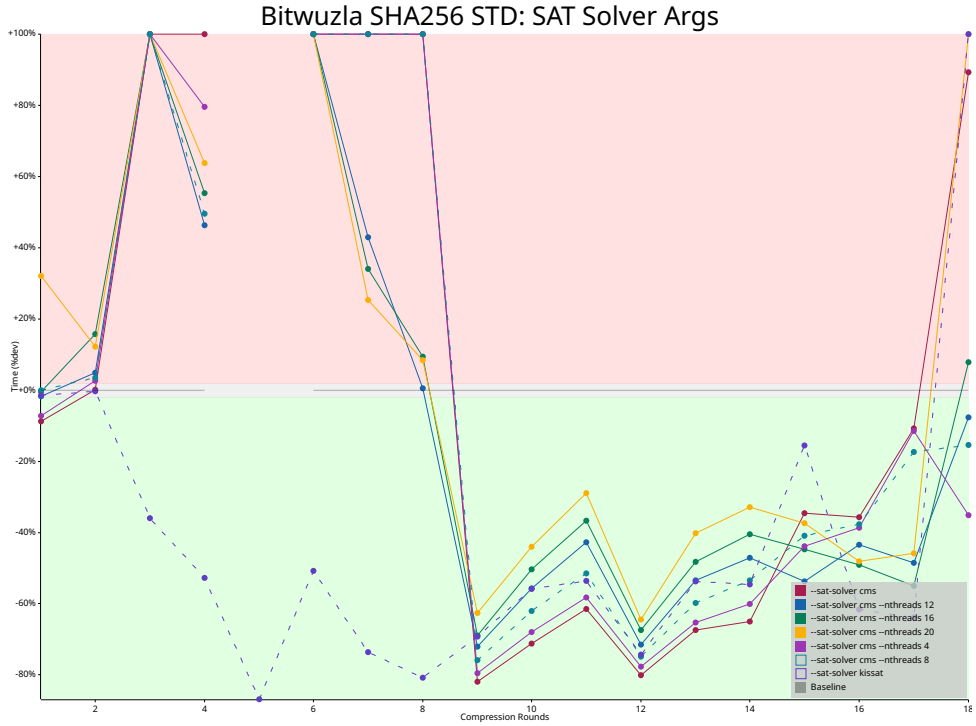


**Figure 17:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to SAT solver backend, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

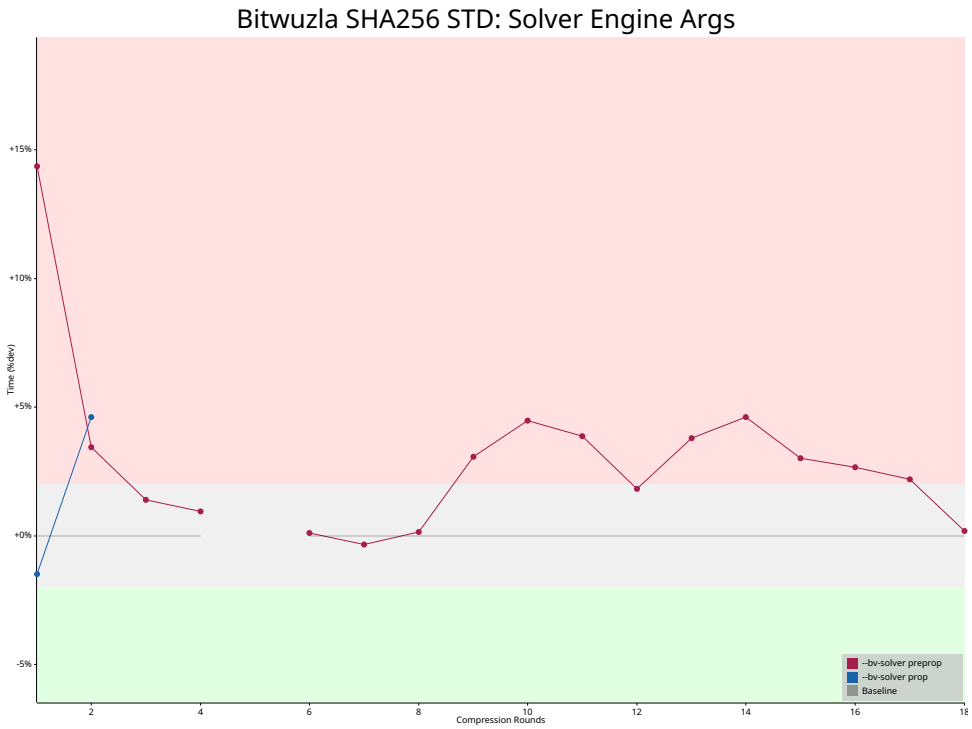


**Figure 18:** Baseline graph showcasing Bitwuzla SHA-256 standard collisions from 1 to 18 rounds using brute-force, where each colour line represents a different solver parameter related to solver engine, compared to Bitwuzla defaults. Results ran with parameters `--round-range 1..19 --hash-function sha256 --collision-type std --continue-on-fail true --arg-set <ARG>`.

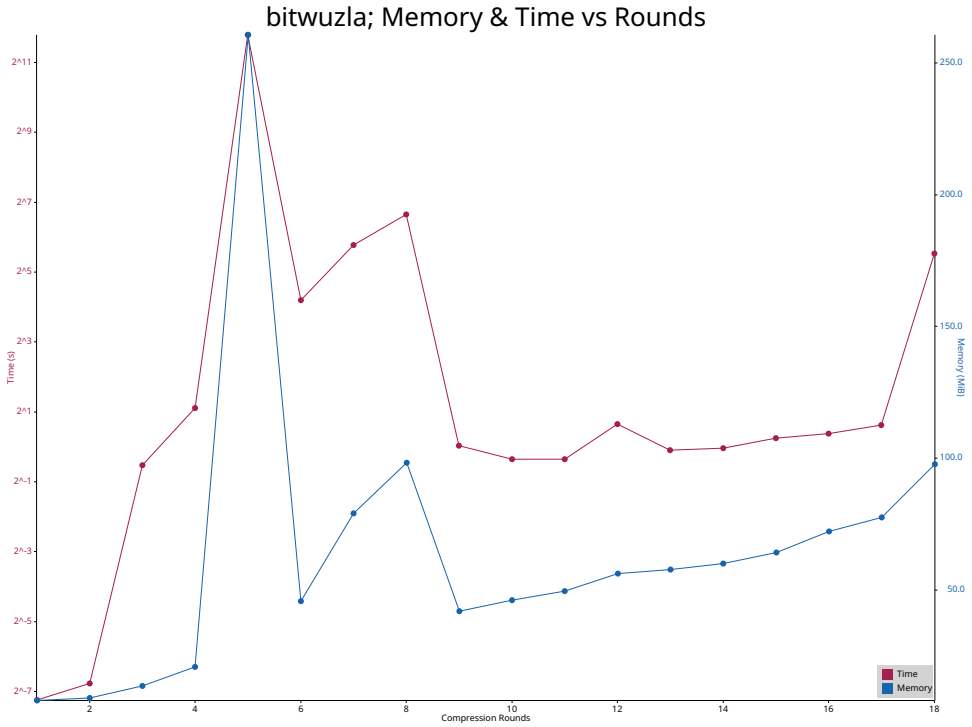


**Figure 19:** Detailed graph showcasing Bitwuzla SHA-256 standard collision from 1 to 18 rounds using brute-force. Results ran with parameters `--round-range 1..20 --hash-function sha256 --collision-type std --continue-on-fail true --timeout-sec 36000`.

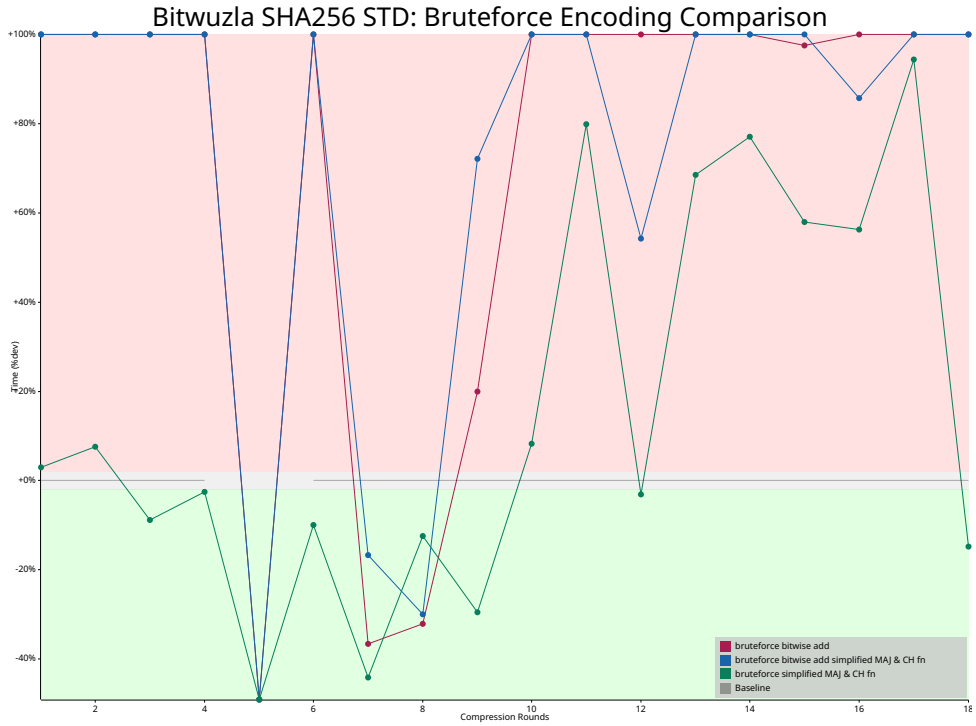


**Figure 20:** Baseline graph showcasing Bitwuzla SHA-256 standard collision graph showcasing rounds 1 to 18, comparing brute-force (baseline) to DSub and DXOR encodings. Results ran with parameters `--round-range 1..20 --hash-function sha256 --collision-type std --continue-on-fail true --encoding-type <ENCODING>::`.

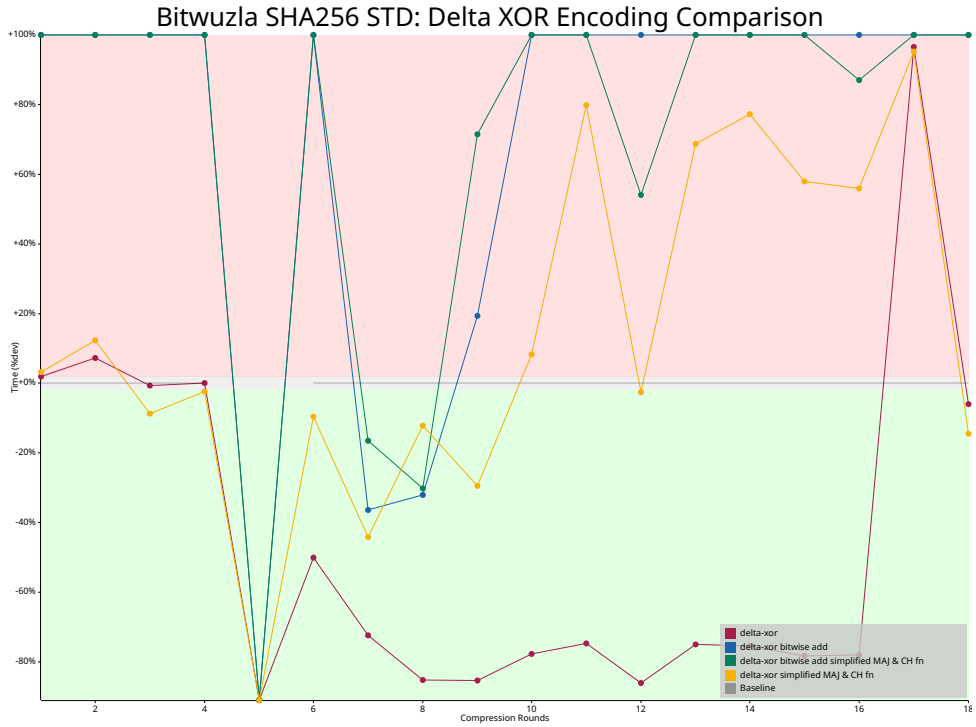


**Figure 21:** Baseline graph showcasing Bitwuzla SHA-256 standard collision graph showcasing rounds 1 to 18, comparing brute-force (baseline) to DXOR with variations. Results ran with parameters `--round-range 1..20 --hash-function sha256 --collision-type std --continue-on-fail true --encoding-type dxor:<simpl. ch/maj>:<a. add>`.

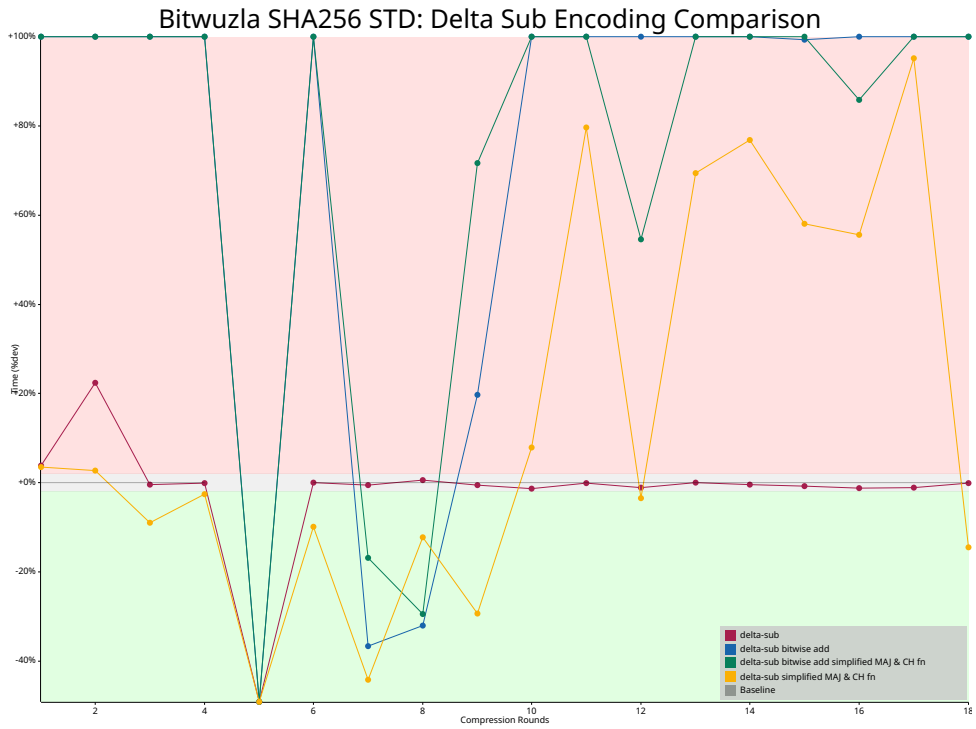


**Figure 22:** baseline graph showcasing Bitwuzla SHA-256 standard collision graph showcasing rounds 1 to 18, comparing brute-force (baseline) to Dsub with variations. Results ran with parameters `--round-range 1..20 --hash-function sha256 --collision-type std --continue-on-fail true --encoding-type dxor:<simpl. ch/maj>:<a. add>`.