# IMPLEMENTATION OF AN OPTIMIZED AUTHENTICATED ENCRYPTION SCHEME

*Alessandro Giaconia, Lorenzo Paleari, Diana Khimey, Filippo Visconti*

Department of Computer Science
ETH Zürich
Zürich, Switzerland

## ABSTRACT

You should write the abstract last.

## 1. INTRODUCTION

**Motivation.** In an era characterized by the exponential growth of data and the escalating prevalence of digital communication, safeguarding sensitive information has become an indispensable priority. The continuous evolution of technology introduces new challenges in upholding the confidentiality and integrity of data exchanged over networks.

This project embarks on a journey focused on implementing a state-of-the-art, optimized, high-performance authenticated encryption scheme. The motivation driving this initiative arises from the necessity for robust cryptographic solutions that not only ensure security but also exhibit efficient runtimes. Authenticated encryption schemes serve as fundamental pillars in modern cryptography, offering a comprehensive solution for ensuring the confidentiality, integrity, and authenticity of data. These schemes find widespread application in various contexts, including TLS, SSH, IPsec, and numerous others.

Our project leverages the combined strengths of ChaCha20 and Blake3, prioritizing efficiency and speed.

It's essential to note that the key distinction between authenticated encryption schemes and traditional encryption schemes lies in the former's ability to not only ensure confidentiality but also provide data integrity and authenticity, addressing the comprehensive security needs of modern applications.

**Related work.** Next, you have to give a brief overview of related work. For a report like this, anywhere between 2 and 8 references. Briefly explain what they do. In the end contrast to what you do to make now precisely clear what your contribution is.

## 2. BACKGROUND

In this section, we give a brief overview of the background necessary to understand the rest of the report. In particular, we introduce the concepts of encryption and authentication, and we explain the algorithms we use.

**Encryption.**

**Authentication.** Authentication is the process through which the integrity and origin of data are verified, to ensure that t it remains untampered and confirming that it originated from the expected sender. This verification process holds significant importance in diverse fields, including secure communication, electronic commerce, and various other domains.

To achieve authentication, Message Authentication Code (MAC) is commonly employed. A MAC is a concise piece of information utilized to authenticate a message, working in conjunction with a secret key. The recipient of the message can validate its authenticity by recalculating the MAC and comparing it with the original MAC. This mechanism provides a secure means of ensuring that the received data has not been altered and originates from the expected source.

**Blake3.** We opted for Blake3 as the authentication component of our scheme, driven by its notable combination of high performance and robust security guarantees. Blake3 is inherently parallelizable, enhancing its suitability for our implementation. The implementation, carried out in the C language, employs two distinct approaches: one utilizing a stack for versatility, and the other employing a more efficient divide-and-conquer strategy, albeit with slightly reduced flexibility, but significantly improved performance.

Blake3 operates as a cryptographic hash function with a built-in MAC function, the specific feature we are leveraging. At a high level, Blake3 processes input by dividing it into chunks of up to 1024 bytes and arranges them as the leaves of a binary tree. These chunks are then compressed using a compression function, conducting a series of operations on the input and producing a 64-byte output known as the chaining value. The chaining value from each chunk becomes the input for the subsequent chunk, and this process continues until all chunks are processed. The chunks effectively represent the leaves of a binary tree, and the tree is traversed from the bottom up. At each level, the chaining values of two children nodes are combined until reaching the root node. The final result is the chaining value of the

root node, obtained through iterations of this process.

To obtain an output of arbitrary length, the last call to the compression function can be repeated with an increased counter, ensuring adaptability to various application requirements.

## 3. YOUR PROPOSED METHOD

In this section, we dive into the details of our implementation. We describe how we implemented the algorithms, and we explain the optimizations we used to achieve high performance. **Blake3.** In this subsection, we describe our implementations of the Blake3 algorithm, which are based on and closely follow the original Blake3 paper that describes the algorithm in detail [1]. Given their different strenghts and characteristics, we also included a dispatcher that chooses the best implementation based on the input size, to always guarantee a correct output. **Stack version.** TODO

**Divide-and-conquer version.** The divide-and-conquer version of the algorithm relies on the premise that the entire input is accessible at the outset of the computation. This enables an equitable distribution of work among multiple threads, each operating on distinct portions of the input, and subsequently eliminating dependencies among them. The threads function concurrently until they reach the base case, wherein each thread is left with one node.

Upon reaching this base case, the threads transmit their individual results to the parent thread, which consolidates these outcomes and returns the final result. This parallelized approach enhances the algorithm's efficiency, leveraging the availability of the complete input early in the computation to facilitate concurrent and independent processing by multiple threads.

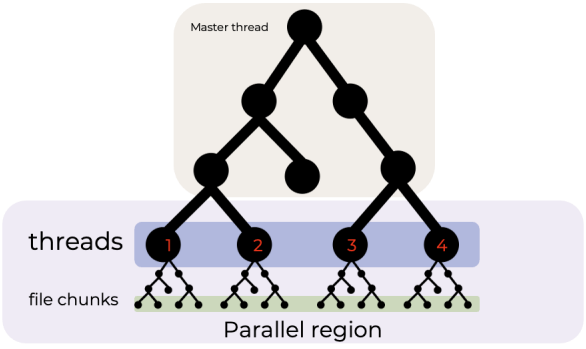


**Fig. 1**. High-level visualization of the divide and conquer approach

To facilitate parallelization in our algorithm, we leverage the OpenMP library, streamlining the process of distributing computation across threads once a parallel region is established. Additionally, we enhance performance by vectorizing the compression function, the most computationally intensive component of the algorithm, using the AVX instruction set and adopting code available on GitHub from the reference implementation.

The primary advantage of this approach lies in its scalability, attributed to the even distribution of work among threads with no inter-thread dependencies. This characteristic allows us to achieve commendable speedups, as detailed in the results section. However, it's important to note that this approach, while highly scalable, has limitations in flexibility. Specifically, it necessitates the availability of the entire input at the commencement of the computation (which aligns with our requirements) and is most effective when the input conforms to a full tree structure. These considerations, although relevant, are mitigated by the suitability of our use case.

TODO Mention and cite any external resources that you used including libraries or other code.

## 4. EXPERIMENTAL RESULTS

In this section, we will describe the experiments we conducted to evaluate our implementations. We will first describe the experimental setup, then we will present the results of our experiments, and finally, we will discuss the results.

**Experimental setup.** We ran all of our experiments on the Ault Cluster of CSCS. In particular, we chose two nodes with the following specifications:

- Node 1: 2x AMD EPYC 7501 32 cores @ 2.0 GHz
- Node 2: 2x AMD EPYC 7742 64 cores @ 2.25 GHz
- 512 GB RAM
- OS: Rocky Linux release 8.4 (Green Obsidian)
- Kernel: Linux 4.18.0-305.19.1.el8_4.x86_64

We used the following compiler and flags:

- `gcc 10.3.0`
- `-O3 -march=native -fopenmp`

We decided to benchmark our implementations over varying input sizes, ranging from 1KB to 128GB, to evaluate their performance across a wide range of use cases. We also included the reference implementation of the respective functions in our benchmarks to provide a single-threaded baseline for comparison.

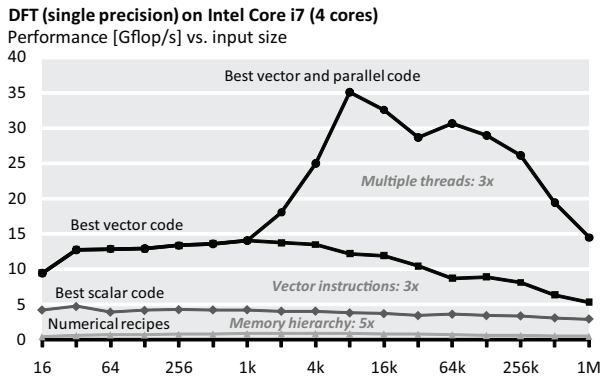


**Fig. 2**. Performance of four single precision implementations of the discrete Fourier transform. The operations count is roughly the same. The labels in this plot are maybe a little bit too small.

**Results.** Next divide the experiments into classes, one paragraph for each. In each class of experiments you typically pursue one questions that then is answered by a suitable plot or plots. For example, first you may want to investigate the performance behavior with changing input size, then how your code compares to external benchmarks.

For some tips on benchmarking including how to create a decent viewgraph see pages 22–27 in [2].

**Comments:**

- Create very readable, attractive plots (do 1 column, not 2 column plots for this report) with readable font size. However, the font size should also not be too large; typically it is smaller than the text font size. An example is in Fig. 2 (of course you can have a different style).
- Every plot answers a question. You state this question and extract the answer from the plot in its discussion.
- Every plot should be referenced and discussed.

## 5. CONCLUSIONS

Here you need to summarize what you did and why this is important. *Do not take the abstract* and put it in the past tense. Remember, now the reader has (hopefully) read the report, so it is a very different situation from the abstract. Try to highlight important results and say the things you really want to get across such as high-level statements (e.g., we believe that .... is the right approach to .... Even though we only considered x, the .... technique should be applicable ....) You can also formulate next steps if you want. Be brief. After the conclusions there are only the references.

## 6. FURTHER COMMENTS

Here we provide some further tips.

**Further general guidelines.**

- For short papers, to save space, I use paragraph titles instead of subsections, as shown in the introduction.
- It is generally a good idea to break sections into such smaller units for readability and since it helps you to (visually) structure the story.
- The above section titles should be adapted to more precisely reflect what you do.
- Each section should be started with a very short summary of what the reader can expect in this section. Nothing more awkward as when the story starts and one does not know what the direction is or the goal.
- Make sure you define every acronym you use, no matter how convinced you are the reader knows it.
- Always spell-check before you submit (to us in this case).
- Be picky. When writing a paper you should always strive for very high quality. Many people may read it and the quality makes a big difference. In this class, the quality is part of the grade.
- Books helping you to write better: [3] and [4].
- Conversion to pdf (latex users only):

  dvips -o conference.ps -t letter -Ppdf -G0 conference.dvi

  and then

  ps2pdf conference.ps

**Graphics.** For plots that are not images *never* generate the bitmap formats jpeg, gif, bmp, tif. Use eps, which means encapsulate postscript. It is scalable since it is a vector graphic description of your graph. E.g., from Matlab, you can export to eps.

The format pdf is also fine for plots (you need pdflatex then), but only if the plot was never before in the format jpeg, gif, bmp, tif.

## 7. REFERENCES

[1] S. Neves Z. Wilcox-O'Hearn J. O'Connor, J. Aumasson, "Blake3 one function, fast everywhere," online: https://blake3.io, 2019.

[2] M. Püschel, "Benchmarking comments," online: http://people.inf.ethz.ch/markusp/teaching/263-2300-ETH-spring11/slides/class05.pdf.

[3] N.J. Higham, *Handbook of Writing for Mathematical Sciences*, SIAM, 1998.

[4] W. Strunk Jr. and E.B. White, *Elements of Style*, Longman, 4th edition, 2000.