FPGA-Accelerated Password Cracking

Eric Britten
Department of Computer Science
California State University, Fullerton
Fullerton, USA
ebritten@csu.fullerton.edu

Mikhail Gofman
Department of Computer Science
California State University, Fullerton
Fullerton, USA
mgofman@fullerton.edu

Yu Bai
Department of Electrical and Computer Engineering
California State University, Fullerton
Fullerton, USA
ybai@fullerton.edu

***Abstract*—Passwords are a common way of securing systems and applications from unauthorized access. However, passwords can be vulnerable to attackers who try to crack them by using random guesses, common patterns (e.g., password topologies), dictionary words, or leaked passwords. In this paper, we propose a hardware-accelerated password cracking method that leverages field-programmable gate array (FPGA) technology to recover passwords hashed with the sha512crypt algorithm. This algorithm is widely used on Linux systems. Our approach focuses on fast development to simulate a casual attacker who wants to boost their password cracking performance by using the speed and parallelism of FPGAs, which can outperform traditional CPUs and GPUs.**

Using C++ high-level language (a technique called high-level synthesis), we created a hardware device (i.e., a core) that runs the sha512crypt algorithm on the Zynq Z-7020 CPU-FPGA hybrid chip. We then tested the password cracking speed of our core (passwords/sec) and compared it to the AMD Ryzen 9 and Apple M1 Max CPUs. Based on the results of one core and two parallel sha512crypt cores on our chip, the maximum that could fit, we estimated that an FPGA chip with more than 8 sha512crypt cores and the same parameters as our chip could crack 10-character passwords at 360 passwords/sec, which is faster than a single-threaded sha512crypt on either CPU. We also projected that with 15 cores, we could achieve a speed of 675 passwords/sec, which is twice as fast as either of the CPUs.

We think this work is a useful addition to the research on cybersecurity applications of FPGAs, as few works have tried to break sha512crypt on the FPGA. In the future, we will implement our design on modern FPGA chips that can accommodate more than 10 cores, optimize our designs and evaluation methodology, and perform a more extensive evaluation.

***Index Terms*—Security, cybersecurity, FPGA, passwords, password cracking, field-programmable gate array**

I. Introduction

Password-based authentication is a common method to protect systems and applications from unauthorized access. However, this method is vulnerable to attacks that can crack passwords by using random guesses, common patterns (e.g., password topologies), dictionary words, or leaked passwords. Defenders try to frustrate these attacks by enforcing password complexity rules and frequency of change, and by storing one-way hashes of passwords instead of plaintext passwords. One-way password hashes make it difficult for the attacker to find out the password even if they breached the system password database. Some secure password hashing algorithms are scrypt [1], bcrypt [2], and sha512crypt [3]. The attacker cannot reverse the hash value to get the password and can only guess the password that results in the same hash.

We present a scheme for cracking passwords stored with the sha512crypt algorithm, which is widely used on Linux systems, using field-programmable gate array (FPGA) technology. FPGAs are chips that can be programmed to create custom on-chip hardware circuits. FPGAs have advantages over traditional central processing units (CPUs) and graphical processing units (GPUs), such as faster speeds, higher computational throughput, and lower power consumption. These advantages can be exploited by attackers to bruteforce passwords efficiently. However, they can also be used by defenders to speed up the process of finding weak passwords, restoring lost passwords, and testing security.

The contributions of this paper are as follows:

1) We present a prototype design of the FPGA-based sha512crypt algorithm to bruteforce passwords stored in the sha512crypt format. Although multiple works have attempted similar FPGA-based attacks against passwords stored using other approaches, such as scrypt, our work is among the few (see Section VII) to implement sha512crypt on the FPGA.
2) We implement our design on the Zynq Z-7020 CPU-FPGA hybrid unit and compare the cracking speeds of two parallel FPGA-implemented sha512crypt devices (cores) to the single-threaded sha512crypt on the AMD Ryzen 9 and Apple M1 Max CPUs. Two cores is the maximum we can fit on our FPGA due to resource limitations and our rapid development driven design that emphasizes reuse of existing components.
3) We then projected that an FPGA with more than 8 cores can crack 10-character passwords faster than both CPUs, based on the linear performance gains of two parallel sha512crypt devices. We also estimated that 15 cores can crack passwords twice as fast as either of the CPUs

[4].

4) We believe that the simplicity of our design and rapid development process illustrate that it is viable for casual attackers with limited FPGA programming knowledge to use FPGAs for cracking passwords.

The paper is structured in the following manner. Section II discusses the preliminaries. Sections III, IV, and V present design, implementation, and performance results, respectively. Section VI discusses the implications of our results and projects results for a larger number of sha512crypt cores, and Section VII compares our work to the existing related literature. Finally, our concluding remarks and plans for future work appear in Section VIII.

II. Preliminaries

In this section, we present the preliminaries of secure password storage, password cracking, and the FPGA technologies.

A. Secure Password Storage

Hashing and salting are standard approaches for securely storing passwords. If the attacker gains access to the password database, hashing and salting make password recovery a computationally challenging task. Let $(uid, salt, H(P||salt))$ represent an entry in the password database where uid is the unique ID of the user, $salt$ is a unique random value associated with the user's account during the account creation time, and $H(P||salt)$ is the output of a one-way, collision-resistant hash function H where $P||salt$ is a concatenation of user's password P and $salt$. Examples of real-world hash functions include SHA-512 [5] and BLAKE3 [6].

The non-invertibility of H will prevent the attacker from directly recovering P even if they gain access to $(uid, salt, H(P||salt))$. The randomness of the $salt$ value also helps to prevent the attacker from using precomputed tables of hashed passwords, known as *rainbow tables* [7], to recover P indirectly. The system can still verify the password by appending the user's $salt$ to the entered password, hashing the result, and comparing it to the hash in the database. The password is valid only if the hashes match.

We targeted the sha512crypt database entry format that is based on the SHA-512 hashing algorithm [8]. We also assumed that each password had up to 64 characters in length and was salted with 16-bytes.

B. Password Cracking

Assume the attacker has a copy of the system password database comprising pairs (P, $H(P)$) (assuming no salting). To recover the password P' that is equivalent to P, the attacker can guess P' values until $H(P') = H(P)$. P' values can come from lists of weak, common, or leaked passwords, be randomly generated, or be generated based on common password patterns (password toplogies).

Attacker can also speed up the attack by using a rainbow table. However, if a random m-bit salt value is added to the password prior to hashing, the rainbow table grows 2^m times larger, which makes it harder or impossible to make a table. Salting also makes the hashes different even when two users have the same passwords.

The most computationally demanding aspect of the attack is repeatedly computing $H(P'||salt)$ for many guesses of P' (assuming the attacker knows the salt). The metric used to measure the speed of the attack is often password hashes per second or simply passwords per second. Faster CPU speeds, more CPU cores, and highly parallel GPUs help attackers to crack passwords faster. FPGAs give attackers the extra advantage of creating custom, on-chip hardware that is parallel, fast, and power efficient compared to CPUs and GPUS [9], and is specially optimized for hashing. However, programming an FPGA is generally more challenging than writing programs for CPUs and GPUs.

C. The sha512crypt Algorithm

The sha512crypt algorithm is based on the SHA-512 hash function [3]. It is designed to be computationally resource intensive to make password cracking harder. This algorithm takes three inputs: the password, the salt, and the number of hashing rounds. It uses the SHA-512 algorithm to hash these inputs repeatedly for the given number of rounds, resulting in a final hash. The number of rounds can vary, but 5000 is a common choice. Increasing the number of rounds makes the password verification slower for both the system and the attacker, but it affects the attacker more who has to verify many password guesses. The algorithm is described in more detail at [3].

D. Field-Programmable Gate Arrays (FPGA)

An FPGA is an integrated circuit whose internal structure and function can be programmatically altered after the circuit is manufactured. For example, FPGA programmers can create custom, on-chip hardware, including mathematical function units, memory, and other complex hardware-based functions including hashing algorithms [10].

FPGA consists of an array of programming logic blocks that can do basic computations such as arithmetic, logic, and storage; and a hierarchy of reconfigurable interconnects that can connect the programming logic blocks. FPGA programming is usually done using a low-level hardware description language (HDL). However, many modern FPGAs can be programmed using high-level languages such as C, C++, and Python, which are then converted into HDL by the software development kit for the FPGA. This is called high level synthesis (HLS). We use high level synthesis with C++ for our implementation.

The flexibility of FPGAs can allow an attacker to create many parallel, optimized hash function units on a chip and use them to crack passwords efficiently. We used the Zedboard with a Zynq Z-7020 CPU-FPGA hybrid chip (Figure 1) and the Vitis Unified Software Platform 2022.2 [11] for development. The platform includes Vitis HLS, Vivado, and Vitis IDE which we discuss in Section IV. The CPU in the chip was a 667 MHz dual core ARM Cortex-A9 processor. The FPGA was an Artix-7 with 53,200 look-up tables (LUTs) [12].

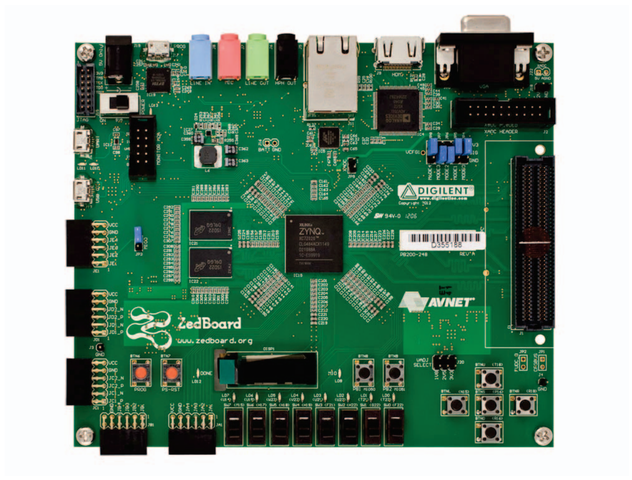

Fig. 1. Zedboard with Zynq Z-7020 [13]

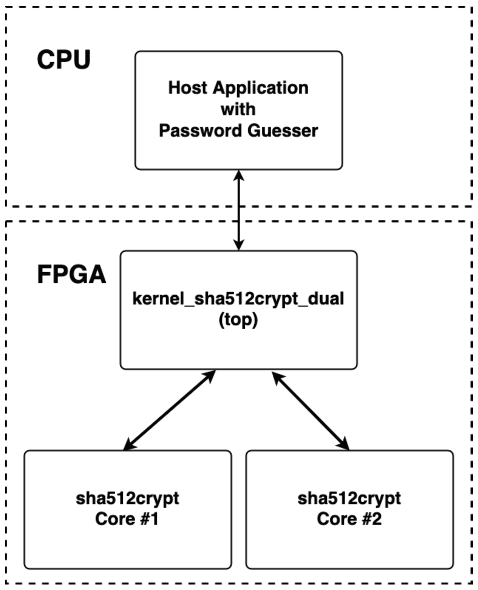
CPU
Host Application with Password Guesser
FPGA
kernel_sha512crypt_dual (top)
sha512crypt Core #1
sha512crypt Core #2

Fig. 2. Architecture with Dual Core Sha512Crypt

III. Design

By integrating an ARM-based CPU and an Artix FPGA on a single die, the Zynq Z-7020 chip enables faster communication between the two units than separate chips. We leverage this feature to design a password-guessing system that runs on both the CPU and the FPGA. The CPU generates candidate passwords and sends them to the FPGA, which performs the sha512crypt algorithm to compute the hash. The CPU then checks if the hash matches the target. We implement the guessing logic on the CPU to facilitate rapid development, as it is easier to program than the FPGA, but with some performance trade-off.

Our overall design is shown in Figure 2. The design has two main components:

1) As shown at the top of Figure 2, the **host application** running on the CPU generates and sends the guessed password and salts to the FPGA hardware, which hashes them. The host application then compares the hash with the target hash to determine if the password is cracked. The password-guessing algorithm generates candidate passwords based on the user-supplied password topology. A password topology is the pattern of the password. For example *ulllllldd* represents a topology of a 9-character password where *u*is an upper-case letter, *l* a lower-case letter, and *d* a digit. Attackers can guess passwords faster by using the most common topologies [14].
2) The second part is the **sha512crypt algorithm** implemented as a hardware device (i.e., a core), on the FPGA (sha512crypt in the figure). The core executes the sha512crypt algorithm and produces a hash. Multiple parallel cores can be implemented on a single FPGA chip if the FPGA has sufficient resources. We were able to physically fit two sha512crypt cores on the FPGA.

The CPU-based host application calls the top-level kernel function on the FPGA. This function is the interface that transfers data between the host program and the FPGA. The top-level kernel function (kernel_sha512crypt_dual in Figure 2) copies the passwords and salt values to data buffers for each FPGA sha512crypt core and triggers them to run in parallel and return the password hash values. When both modules finish, the top-level function returns the hash values to the host program.

Our design can also handle larger FPGAs that can fit more than two cores. In such cases, we can modify the top-level function to distribute n passwords among the n sha512crypt cores on the FPGA. The top-level function then collects the password hash values from all the sha512crypt cores and sends them back to the host application for comparison.

IV. Implementation

We realized our design using the Vitis Unified Software Platform 2022.2 for the Zedboard development board with the Zynq Z-7020.

First, we used Xilinx Vitis HLS software to develop the FPGA sha512crypt module using HLS with C++. Next, we used Xilinx Vivado software to configure the internal structure of the FPGA on the Zedboard. The resulting design is shown in Figure 3. The FPGA design includes our sha512crypt module and components that handle communication between the FPGA and the main memory. Those components include the Zynq processing system which connects to the board's main memory and two AXI interconnects and a processor system reset module that are used to give the sha512crypt module access to that memory. Vivado software was then used to generate the bitstream file of our FPGA design that was used to program the FPGA. Lastly, we then used the Xilinx Vitis IDE software to develop the host program in C++ and then run the program on the board CPU.

We ran the host program of the password cracker in standalone mode. This means that it was the only program running on the Zedboard CPU, without the embedded Linux operating system supported by the board. This way, we avoided OS overhead and simplified the development process.

We based our FPGA implementation of the sha512crypt algorithm in the glibc C library [15], with some modifications. We limited the password length to 64 characters and the salt value to 16 bytes, while glibc could handle larger inputs. To reduce the development time, we used the SHA-512 subroutine from the Vitis HLS library [16] and called it for multiple rounds as required by sha512crypt. To accommodate our password and salt size restrictions, we increased the buffer size of the SHA-512 subroutine from 256 bytes to 4336 bytes, which is the maximum amount of data that sha512crypt needs to hash. However, this caused timing violations on the Zynq Z-7020 FPGA, so we reduced the clock rate from 100MHz to 70MHz. We plan to optimize our implementation further to achieve a higher clock speed in the future.

Only two sha512crypt cores fit on the FPGA, and this limitation was imposed by the number of available LUTs as seen in Table I. LUTs are the basic computation units of the FPGA, and LUTRAM is a special type of LUT in Xilinx FPGAs that can handle complex computations. Flipflops (FFs) are used for the storage of data produced by logical operations. Block RAM is placed on chip memory that is used to transfer data between different modules using a first in, first out (FIFO) manner. The columns ”Single Core” and ”Dual Core” show the resources used by the single and dual core FPGA implementations, respectively. The column ”Available” shows the number of resources available on the Zedboard.

	Single Core	**Dual Core**	**Available**
LUTs	18,624	35,679	53,200
LUTRAM	4,170	8,123	17,400
FF	22,456	41,811	106,400
BRAM	3	5	140

TABLE I
ZEDBOARD UTILIZATION

We used a shared DDR memory space that was accessible to both the CPU host program and the FPGA sha512crypt core to communicate. The host program writes passwords and salt values to this memory area, and then the FPGA reads them and writes out the password hash values to the same memory area. We used two AXI Interconnects [17], which are modules that enable memory mapped data transfers, to facilitate communication between the host program and the sha512crypt module.

For comparison purposes, the sha512crypt function from the standard Linux glibc library was used when the running the password cracker on the desktop CPUs. Machine code generated for the CPUs ran in single thread on a single core. Two machines were used for CPU testing. First, a Linux system with Ubuntu 22.04.2 and a 3.7 GHz AMD Ryzen 9 5900X processor with 128 GB of RAM. Second, a MacBook Pro with Mac OS 12.5 and a 3.2 GHz Apple M1 Max processor with 64 GB of RAM.

V. Performance Results

We compared the passwords/sec speed of our approach when using an FPGA with a single sha512crypt core and two sha512crypt cores against sha512crypt running on the system with AMD Ryzen 9 5900X processor [18] and against sha512crypt running on a system with the Apple M1 Max [19] processor. The algorithm running on the CPUs was single-threaded.

In each experiment, 100 passwords of a specified length and topology were randomly generated. Each of the passwords was then respectively hashed along with the 16-byte salt value using sha512crypt. The experiment for each password length was repeated 10 times, after which the hashing speeds were averaged. The experiment was conducted for password lengths between 1 - 64 characters. The average speeds of the FPGA with one sha512crypt core and two sha512crypt cores and the speeds of the two CPUs were then plotted against password size in Figure 4.

As can be observed from the plot, dual core FPGA implementation of the sha512crypt algorithm was twice as fast as the single core implementation. n cores are therefore expected to result in speed of ns where s is the speed of the single core. The next section examines this assumption in detail and uses these results to estimate performance for implementations with more than two sha512crypt cores.

Also, we see that the speed drops for all implementations as the password size grows. The biggest drop happens between 15 and 16 characters. This is because of the sha512crypt algorithm's behavior and how it deals with data of different lengths.

Since the Zedboard FPGA has less space and is older than the CPUs, the CPUs were expected to outperform the FPGA. The AMD Ryzen 9 5900X was 3.18 times faster, while Apple M1 Max was 3.7 times faster than the dual core FPGA implementation. The next section shows how many sha512crypt FPGA units are needed to beat both Ryzen 9 and Apple M1 Max.

VI. Discussion

We used the results from the previous section to project the performance of our approach on the FPGAs with the same specification as Zynq Z-7020 but more room for additional sha512crypt units design similar to ours.

We could only fit two sha512crypt cores on the Zynq Z-7020 chip due to limited LUTs. Larger FPGAs can fit more cores and achieve a linear increase in hashing throughput. Specifically, as observed from results in the previous section, n cores will yield performance of ns where s is the speed of a single core. We also assume that there is no significant diminishing return for up to 20 cores. Based on the FPGA platform characteristics and our design, we believe these assumptions to be reasonable.

The linear relationship does not hold for traditional CPUs, which face several performance challenges. They have to communicate between cores, which takes time. In traditional systems, the CPU runs an operating system and multiple

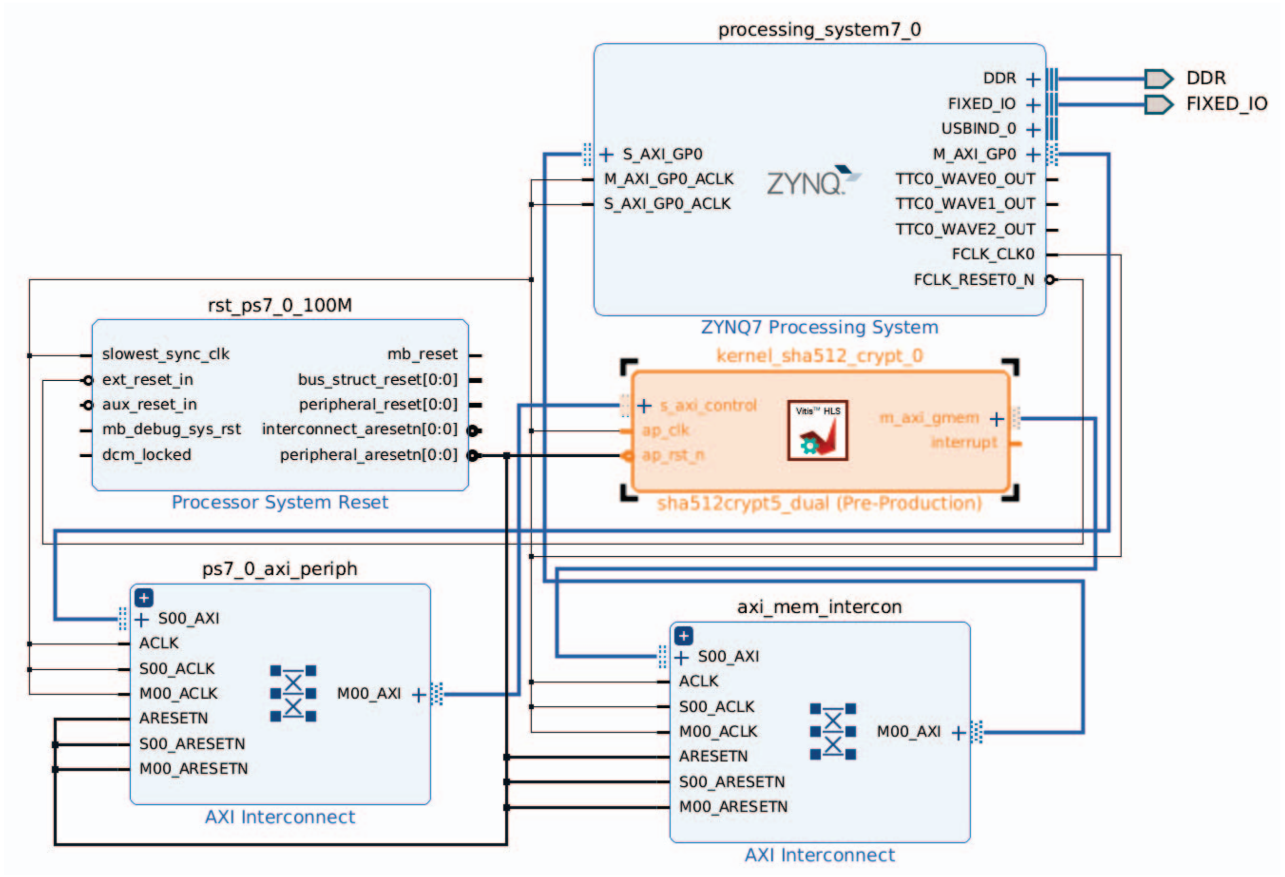

Fig. 3. Vivado Board Configuration

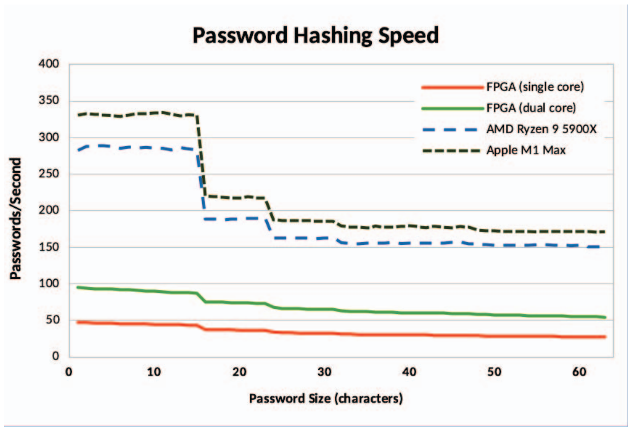

Fig. 4. Password Hashing Speed

processes and threads, which lowers the performance of a single process such as the password cracker. Moreover, CPUs are limited by the number of cores that the manufacturers offer, while FPGAs can scale up the number of cores as the attacker needs, as long as the core design efficiency and the FPGA resources allow it. We plan to implement parallel multithreaded CPU versions of the sha512crypt and compare their performance to our FPGA-based approach in the future.

Using our assumptions, we estimated the number of cores needed to outperform the AMD Ryzen 9 and Apple M1 Max CPUs in single-threaded sha512crypt performance for 10-character and 60-character passwords. These lengths represent a common minimum password length and a long password (e.g., for an administrative account), respectively.

A. *Projections for 10-Character and 60-Character Passwords*

For a 10-character password, our single-core implementation achieved a speed of 45 passwords/sec, 2-core FPGA implementation achieved a speed of 90 passwords/sec, while the AMD Ryzen 9 and the Apple M1 Max achieved speeds of 286 and 334 passwords/sec, respectively.

Considering the above results, we projected the potential password cracking speeds for larger core counts. Assuming a linear relationship between the number of cores (n) and the cracking speed (s), we can infer the following: If s = 45 passwords/sec for n = 1, then for n = 7, the projected speed would be 315 passwords/sec, surpassing the performance of the AMD Ryzen 9. Furthermore, with n = 8, the projected speed is 360 passwords/sec, outperforming the Apple M1 Max. Moreover, when $n = 15$, we estimated a cracking speed of 675 passwords/sec, which is more than twice as fast as either of the CPUs.

For a 60-character password, our single-core implementation achieved a speed of 27 passwords/sec and our 2-core FPGA implementation achieved 55 passwords/sec. Therefore, outperforming AMD Ryzen 9, which achieved a speed of 152 passwords/sec, will require 6 cores. Similarly, to match the performance of the Apple M1 Max, which reached a speed of 189 passwords/sec, will require 7 cores.

As we discuss in Section VIII, our next research step is to implement our approach on larger FPGA chips such as the Kintex Ultrascale+ XCKU5P [20], which we believe can accommodate up to 10 sha512crypt cores and empirically prove our projections. We also believe that by further optimizing our implementation, we can reduce the physical footprint for each sha512crypt unit, and thus be able to accommodate even more units per chip.

VII. Related Works

Table II shows previous studies on FPGA-based password cracking. The ”Algorithm” column gives the password storage scheme and the ”Language” column gives the FPGA language. The table shows that studies [2], [21]–[24] do not give the language. So, it is unclear whether the FPGA was programmed with HDL or HLS.

Works [25] and [26] used high level synthesis. In [27] the authors briefly mentioned sha512crypt, while mostly focusing on sha256crypt. This work used the Zynq Z-7030 FPGA and it could accommodate two cores of sha256crypt or one sha512cryptc core. Our work was able to accommodate two sha512crypt cores on the same board, although their core achieved a speed of 220 MHz while our core ran at the speed of 70 MHz. Their system was about 711 times faster compared to ours. We attribute this to a higher clock frequency that was 3.1x faster than ours and other factors from the more powerful hardware and more complex design choices.

Specifically, Zynq Z-7030 (XC7Z030) has about 50% more logic cells than the Z-7020 that we use. The design in [27] also employs a custom designed sophisticated finite state machine, while our approach emphasized rapid development and reuse of existing components. Additionally their system was developed using Xilinx Vivado 2016.3, while ours was with Vivado 2022.3.

John the Ripper [28] is another software that can use the FPGA for password cracking and is programmed with VHSIC Hardware Description Language (VHDL; a form of HDL) while we used the HLS approach. It’s design [28] differs from ours and the previously described works significantly. It’s design uses SHA-512 hashing cores along with a soft core processor that executes a sha512crypt program on the FPGA. The ZTEC 1.15y [29] board that it uses contains four FPGA chips, each of which contains 12 units. Each unit contains 4 SHA-512 cores allowing 192 passwords to be processed at the same time. It should be noted that the FPGA model (XC6SLX150) on the ZTEC 1.15y has significantly more logic cells than our FPGA, and it therefore expected that ZTEC 1.15y can support more cores. The XC6SLX150 [30] contains 147,443 logic cells, where as the Zedboard’s Zync Z-7020 only has 85,000.

VIII. Conclusions and Future Works

We designed and implemented a prototype FPGA-based sha512crypt bruteforce password cracking scheme using C++ on the Zedboard with a Zynq Z-7020 CPU/FPGA, and we compared its performance to the single-threaded implementation of sha512crypt AMD Ryzen 9 and Apple M1 Max modern CPUs. Due to the physical limitations of our FPGA size and rapid development driven design, we could accommodate two sha512crypt computing cores. However, additional units could be easily added on a larger chip, and we would expect the hashing performance to increase linearly. The same is not expected on the CPUs where the performance can often be bottlenecked by the operating system.

Projections based on our performance results indicate that an FPGA chip capable of accommodating more than 8 cores (with all other variables held constant) would outperform the single-threaded algorithm running on the AMD Ryzen 9 and the Apple M1 Max for cracking 10-character passwords. Fifteen cores would make our approach more than twice as fast as both CPUs.

Our work is among the first to provide performance results for a high level synthesis approach of a sha512crypt implementation on the FPGA and is among the few to accommodate more than one sha512core on the FPGA. Furthermore, this is the first work to explicitly experiment the rapid development approach to FPGA-based password cracking.

We find the preliminary results encouraging. Our next steps are as follows:

1) Further optimize the layout and time efficiency of our FPGA-based sha512crypt implementation.
2) Move the password guessing logic to the FPGA to completely eliminate the CPU-imposed bottlenecks.
3) Experiment with other schemes, such as scrypt and bcrypt, and evaluate our approach against them.
4) Implement our approach on the Kintex Ultrascale+ KCU116 development board and empirically validate the projections made in this paper. We estimate that the XCKU5P FPGA on the board can accommodate around 12 units or more.
5) Compare the performance of our approach to the multithreaded implementation of sha512crypt algorithm on the modern CPU.
6) Compare our approach to the performance of the GPU-based password cracking schemes.
7) Utilize more LUTRAM and less LUTs to optimize our approach and fit more password hashing cores on the FPGA.

References

[1] C. Percival, “Stronger key derivation via sequential memory-hard functions,” 2009.

	Year	**Algorithm**	**Language**	**Description**
[23]	2005	25DES	N/A	The Xilinx XC4VLX200 FPGA was used to hash passwords.
[31]	2010	LM Hash, md5, sha1	VDHL	Builds rainbow tables using a Xilinx XCV5VLX330T FPGA.
[24]	2014	bcrypt	N/A	Password generation and hashing on the Zedboard with the Z-7020 FGPA.
[2]	2014	bcrypt	N/A	Several different FPGAs including the Zedboard with the Z-7020 FGPA are used to hash passwords in parallel.
[22]	2015	bcrypt	N/A	Compares password hashing speeds of bcrypt on CPU's, GPU's and the Zedboard with the Z-7020 FGPA.
[21]	2016	PBKDF2, SHA1	N/A	Password generator and hasher are implemented on Xilinx Spartan-6 and Artix-7 FPGAs.
[26]	2017	SHA-3	C	Hashing algorithm was implemented on a Zedboard with the Z-7020 FGPA.
[28], [32]	2018	bcrypt, descrypt, scrypt, sha256crypt, sha512crypt	VHDL	Password cracker that generates and hashes passwords on a ZTEC 1.15y FPGA board with four XC6SLX150 FPGAs.
[25]	2018	md5	C++	An OpenCL approach with an Intel Arria 10 GX1150 FPGA is compared to CPUs and a GPU.
[33]	2019	md5	VHDL	Brute force password cracker with password guesser and MD5 hasher implemented on a Xilinx Virtex-7 FPGA.
[27]	2020	Sha256crypt, Sha512crypt	N/A	Password recovery with a CPU/FPGA hybrid approach was used with a Zynq Z-7030.
[34]	2021	SHA1, SHA-256, SHA-512, MD5	N/A	Password recovery comparison between FPGAs, a GPU and a CPU. FPGA's were the Xilinx Kintex XCKU060, Xilinx U280, Huawai FX600 and the Intel Cyclone IV.

TABLE II
COMPARISON OF RELATED WORKS

[2] K. Malvoni, S. Designer, and J. Knezovic, "Are your passwords safe: Energy-efficient bcrypt cracking with low-cost parallel hardware.," in *WOOT*, 2014.

[3] "Unix crypt using sha-256 and sha-512." https://www.akkadia.org/drepper/SHA-crypt.txt, 2016-8-31. Accessed on June 7, 2023.

[4] "Ultrascale+ fpgas product tables and product selection guide." https://docs.xilinx.com/v/u/en-US/ultrascale-plus-fpga-product-selection-guide, 2023. Accessed on June 16, 2023.

[5] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)." Federal Information Processing Standards Publication, 2015. FIPS PUB 180-4.

[6] "Blake3." https://github.com/BLAKE3-team/BLAKE3/. Accessed: 2013-06-08.

[7] P. Oechslin, "Making a faster cryptanalytic time-memory trade-off," in *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pp. 617–630, Springer, 2003.

[8] M. Kerrisk, "crypt(3) - linux manual page." https://man7.org/linux/man-pages/man3/crypt.3.html. Accessed: 2021-12-17.

[9] Z. Zhang, P. Liu, W. Wang, S. Li, P. Wang, and Y. Jiang, "High-performance password recovery hardware going from gpu to hybrid cpu-fpga platform," *IEEE Consumer Electronics Magazine*, vol. 11, no. 1, pp. 80–87, 2022.

[10] I. Kuon, R. Tessier, J. Rose, *et al.*, "Fpga architecture: Survey and challenges," *Foundations and Trends® in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[11] "Vitis unified software platform." https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html, 2023. Accessed on June 7, 2023.

[12] "Zynq-7000 soc data sheet: Overview (ds190)." https://docs.xilinx.com/v/u/en-US/ds190-Zynq-7000-Overview, 2018. Accessed on June 4, 2023.

[13] https://www.xilinx.com/products/boards-and-kits/1-8dyf-11.html. Accessed: 2023-06-30.

[14] H. Leininger, "Pathwell: Password topology histogram wear-leveling," in *BSides Asheville*, 2014.

[15] "The gnu c library (glibc)." https://www.gnu.org/software/libc/, 2023. Accessed on June 3, 2023.

[16] "sha512: Vitis hls library." https://docs.xilinx.com/r/en-US/Vitis_Libraries/security/guide_L1/hw_api.html_98, 2023. Accessed on June 3, 2023.

[17] "Axi interconnect." https://www.xilinx.com/products/intellectual-property/axi_interconnect.html, 2023. Accessed on June 16, 2023.

[18] "Amd ryzen 9 5900x desktop processors." https://www.amd.com/en/products/cpu/amd-ryzen-9-5900x, 2020. Accessed on June 3, 2023.

[19] "Macbook pro (16-inch, 2021) - technical specifications." https://support.apple.com/kb/SP858?locale=en_US, 2021. Accessed on June 7, 2023.

[20] "Kintex ultrascale+ product table." https://www.xilinx.com/products/silicon-devices/fpga/kintex-ultrascale-plus.html, 2023. Accessed on June 7, 2023.

[21] M. Kammerstetter, M. Muellner, D. Burian, C. Kudera, and W. Kastner, "Efficient high-speed wpa2 brute force attacks using scalable low-cost fpga clustering," in *Cryptographic Hardware and Embedded Systems–CHES 2016: 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings 18*, pp. 559–577, Springer, 2016.

[22] M. Dürmuth and T. Kranz, "On password guessing with gpus and fpgas," in *Technology and Practice of Passwords: International Conference on Passwords, PASSWORDS'14, Trondheim, Norway, December 8-10, 2014, Revised Selected Papers 7*, pp. 19–38, Springer, 2015.

[23] N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, "Cracking unix passwords using fpga platforms," 2005.

[24] F. Wiemer and R. Zimmermann, "High-speed implementation of bcrypt password search using special-purpose hardware," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, pp. 1–6, IEEE, 2014.

[25] Z. Jin and H. Finkel, "Evaluation of md5hash kernel on opencl fpga platform," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1026–1032, IEEE, 2018.

[26] H. S. Jacinto, L. Daoud, and N. Rafla, "High level synthesis using vivado hls for optimizations of sha-3," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 563–566, IEEE, 2017.

[27] Z. Zhang and P. Liu, "A hybrid-cpu-fpga-based solution to the recovery of sha256crypt-hashed passwords," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–23, 2020.

[28] "John the ripper password cracker." https://www.openwall.com/john/, 2023. Accessed on June 7, 2023.

[29] "Usb-fpga module 1.15y." https://www.ztex.de/usb-fpga-1/usb-fpga-1.15y.e.html, 2023. Accessed on June 24, 2023.

[30] "Spartan-6 family overview." https://docs.xilinx.com/v/u/en-US/ds160, 2011. Accessed on June 24, 2023.

[31] K. Theocharoulis, I. Papaefstathiou, and C. Manifavas, "Implementing rainbow tables in high-end fpgas for super-fast password cracking," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 145–150, IEEE, 2010.

[32] "John the ripper - fpga-sha512crypt." https://github.com/openwall/john/tree/bleeding-jumbo/src/ztex/fpga-sha512crypt, 2018. Accessed: 2023-06-17.

[33] M. Gillela, V. Prenosil, and V. R. Ginjala, "Parallelization of brute-force attack on md5 hash algorithm on fpga," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pp. 88–93, IEEE, 2019.

[34] B. Li, F. Feng, X. Chen, and Y. Cao, "Reconfigurable and high-efficiency password recovery algorithms based on hrca," *IEEE Access*, vol. 9, pp. 18085–18111, 2021.