TEA/XTEA brute-force cracker for CPU, GPU and FPGA

1st Costin-Emanuel Vasile
Dept. of Devices, Circuits and Architectures
National 77506a568db31a5b724d8d82c1d43bc77b7b28c130854a6625e45a36faeeb2fd
Bucharest, Romania

2nd Călin Bîră
Dept. of Devices, Circuits and Architectures
National 77506a568db31a5b724d8d82c1d43bc77b7b28c130854a6625e45a36faeeb2fd
Bucharest, Romania

3rd George-Vlăduț Popescu
Dept. of Devices, Circuits and Architectures
National 77506a568db31a5b724d8d82c1d43bc77b7b28c130854a6625e45a36faeeb2fd
Bucharest, Romania

2024 International Semiconductor Conference (CAS) | 979-8-3503-5207-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/CAS62834.2024.10736715

***Abstract*—This work presents a study comparing Tiny Encryption Algorithm (TEA/XTEA) brute-force cracker C++ implementations on FPGA, GPU and CPU. Our HLS FPGA implementation outperformed by at least 2.5x core-per-core state-of-the art for TEA and 20x for XTEA. For a similar 75W TDP, our FPGA implementation outperformed our GPU implementation by at least 3x.**

***Index Terms*—TEA, XTEA, brute-force cracker, FPGA, GPU, CPU.**

I. Introduction

Data security and integrity are critical for applications relying on personal information and data transmission channels. Issues such as eavesdropping, cloning, counterfeiting, and tracking can compromise user privacy and security agreements. Online banking, which stores usernames and passwords on bank servers, is particularly vulnerable. Unauthorized access to customer login information can lead to financial losses. Similarly, applications like smart grid data transmission and Unmanned Autonomous Vehicles (UAVs) require robust data protection measures.

Cryptographic algorithms are essential for defending against cyber-attacks, securing login information in online banking, securing electronic cash transactions [1], and protecting communications from eavesdropping. Cryptography involves encrypting data to safeguard it from hackers and spammers, typically using secret keys for encryption and decryption. There are two main types of cryptographic algorithms: symmetric (using just a private key), which uses the same key for both encryption and decryption, and asymmetric, which uses different keys (a public key used for encryption and a private key used for decryption). These algorithms are designed to be computationally complex, making them impractical to break with current resources [2]. Attempts have been made to analyze the algorithms and find weaknesses [3][4][5]; a good cryptographic algorithm is considered the one where the best attacks are of similar effort (in order of magnitude) with bruteforcing the key-search.

The effectiveness of cryptographic algorithms is measured by their encryption and decryption speed and their ability to withstand attacks. Various encryption algorithms were developed to improve the speed [6][7][8].

Even in the advent of quantum computers, which have the potential to dramatically reduce the time required to break symmetric encryption through brute force attacks (leveraging their capability to perform many calculations simultaneously), Grover's algorithm can theoretically reduce the time complexity of brute force attacks on symmetric key cryptography from $O(2^n)$ to $O(2^{n/2})$. This may be compensated efficiently by doubling the key size (the use of AES-256 instead of AES-128 is expected until 2050 [9]).

II. Related Works

The Tiny Encryption Algorithm (TEA) [10] and its successor, the Extended Tiny Encryption Algorithm (XTEA), which was designed to correct some weaknesses of the original algorithm, are two block cipher algorithms that use 128-bit keys and 64-bit input data blocks. Those algorithms offer a small code footprint and require a small amount of memory compared to other encryption algorithms, thus making them suitable for resource-constrained systems. Both algorithms are recommended to be used in 32 rounds.

Various software and hardware implementations were proposed and evaluated in several previous studies. In [11], TEA and XTEA implementations for several CPU, GPU, and FPGA (Zynq 7020) platforms are presented, using for evaluation plaintext data blocks of different sizes. The results show that the GPU (which, usually, works at very high frequencies) and FPGA implementations are faster than the CPU ones in all tested cases. The FPGA implementation showed quite similar throughputs for all tested cases (∼5300 Mbps). The performance of GPUs increases as the plaintext size increases. Nevertheless, compared to the GPUs, the FPGA showed better throughput for small plaintext sizes and similar throughput for large plaintext sizes.

979-8-3503-5207-8/

In [12], a hardware implementation of XTEA on an Altera Cyclone-V-based SoC is presented. Three approaches were considered: implementation of the encryption algorithm only in software, hosted by a Nios II processor, implementation on hardware, developing an XTEA accelerator, and a co-design implementation, in which Nios II soft processor controls the XTEA hardware accelerator. For the latter approach, two cases were considered: first, with only one XTEA accelerator, and second, with multiple XTEA accelerators controlled by the same processor. The throughput for the co-design approach with multiple XTEA accelerators reaches 113 Mbps.

Several studies were dedicated to FPGA-based hardware TEA and XTEA architectures, obtaining different throughput performance depending on the design approach and target device. In [13], three different TEA algorithm design approaches (sequential, combinational, and hybrid) on the Spartan 6-XC6SLX45 FPGA are presented. The evaluation shows a throughput varying from 55.86 Mbps to 7713.09 Mbps, with the best one being obtained for the hybrid approach. Nevertheless, the high throughput comes with costs in terms of area and power. A pipelined architecture for TEA, implemented on a Spartan 3E XC3S400 FPGA, is presented in [14]. The evaluation shows a throughput of 19.52 Mbps, for a clock frequency of 59.3 MHz.

Hardware architectures for TEA and XTEA algorithms are also presented in [15], where a pipelined approach implemented on a Virtex-7 FPGA (XC7VX330T-2) and ASIC (UMC 0.18 μm) is evaluated. The evaluations show a higher maximum frequency for the FPGA implementation, therefore a better throughput: 805 Mbps on FPGA (frequency of 415.16 MHz) and 432.89 Mbps on ASIC (frequency of 223.21 MHz) for TEA and 960 Mbps on FPGA (frequency of 494.51 MHz) and 345.20 Mbps on ASIC (frequency of 177.93 MHz) for XTEA.

III. Methods

A. Hardware setups

We implemented our brute-force cracker of TEA / XTEA algorithms in C++ on three different platforms: a CPU, a GPU and an FPGA.

TABLE I
Hardware platforms used for brute-force cracker

Type	Name	TDP(W)	Cores/Thr	Top Freq (GHz)
CPU	Ryzen9 5900HS	35	8 / 16	4.6
GPU	RTX3050 Ti	75	20 / 2560	1.035
FPGA	Alveo U50	75	50	0.5

B. Software

When using the CPU, we utilized OpenMP directives, to run on each hardware thread (Ryzen 9 5900HS is an 8-cores, 16-threads processor), whereas on the GPU, we used CUDA extensions to split the work into workpackages of 256 to optimally map on all 20 SMs (nVIDIA RTX 3050 Ti Mobile has a total of 2560 CUDA cores). We used the 555.42.02 drivers with CUDA 12.5

The same main loop (listed in Fig. 1) was utilized, but various code adaptations were made to accommodate the three hardware platforms.

```
for (uint8_t i = 0; i < 32; i++) {

 #ifdef XTEA
    v0 += (((v1 << 4)^(v1 >> 5)) + v1) ^
             (sum + k[sum & 3]);

    sum += DELTA;

    v1 += (((v0 << 4)^(v0 >> 5)) + v0) ^
             (sum+ k[(sum >> 11) & 3]);

 #else // TEA
    sum += DELTA;

    v0 += ((v1 << 4) + k[0])^(v1 + sum) ^
             ((v1 >> 5) + k[1]);

    v1 += ((v0 << 4) + k[3])^(v0 + sum) ^
             ((v0 >> 5) + k[2]);

 #endif
}
```

Fig. 1. Main loops of XTEA and TEA algorithms

The brute-force craker used the following sequence:

- load the start key (128 bits).
- load the plaintext (64 bits).
- load the expected cryptotext (64 bits).
- evaluate each key from start key to end:
  - encrypt plaintext.
  - compare result with expected cryptotext.
  - increment the key value.
  - break if a match is found.

C. Hardware

The FPGA-based approach involves developing the algorithm in C++ and using High-Level Synthesis (HLS) to leverage the pragma-driven optimization features, thereby reducing the overall implementation time. The hardware setup consists of:

- One **AMD Alveo U50 Data Center Acceleration Card**, running multiple parallel brute-force TEA breaking kernels.
- An **Intel Xeon W-3223** serving both for kernel development and as the host in the application, handling kernel management and data transfer.

The development environment uses the Vitis Unified Software Platform for kernel high-level synthesis, kernel linking, and host application development. Although the unified platform utilizes Vitis HLS and Vivado tools in the backend for HLS and linking, we discovered that using these tools independently during critical development steps can be beneficial, as they offer improved simulation and results analysis capabilities compared to Vitis IDE.

IV. Benchmarks and results

Using the sourcecode we developed, we ran 4 billion encryptions (using 4G different keys) and timed it.

For the TEA algorithm, we obtained the results listed in Table II.

TABLE II
TEA BRUTE-FORCE CRACKER SPEED

Type	Name	Speed*(Gbps)	Speed(Mkeys/s)
CPU	Ryzen9 5900HS	3.8	60.12
GPU	nVIDIA RTX3050 Ti	390	6100
FPGA	Alveo U50 (1C)	19.4	303
FPGA	Alveo U50 (60C)	1163	18180
FPGA[13]	Spartan 6 LX45	7.7	n/a
FPGA[15]	Virtex-7 XC7VX330T	0.8	n/a
ASIC[15]	custom	0.43	n/a

For XTEA algorithm, we obtained the results listed in Table III.

TABLE III
XTEA BRUTE-FORCE CRACKER SPEED

Type	Name	Speed*(Gbps)	Speed(Mkeys/s)
CPU	Ryzen9 5900HS	3.0	46.93
GPU	nVIDIA RTX3050 Ti	281	4394
FPGA	Alveo U50 (1C)	19.4	303
FPGA	Alveo U50 (60C)	1163	18180
FPGA[15]	Virtex-7 XC7VX330T	0.96	n/a
ASIC[15]	custom	0.34	n/a

Since the cracker uses the same plaintext encrypted with a different key, we avoided the term of throughput and chose the speed to be measured in Gbps.

Table IV shows the performance evolution from the first iteration, through two intermediate iterations, to the final iteration. We detail the implementation techniques used and evaluate each iteration based on the following metrics: correctness of results, number of keys evaluated per second, and resource utilization of the kernel (relative to the total available resources on the Alveo U50).

TABLE IV
METRICS OF DIFFERENT FPGA IMPLEMENTATIONS

Iteration	Kernel freq.	Used Area	Speed (Mkeys/s)
Naive approach	100MHz	1%	2.5
One parallel kernel	100MHz	70%	142
60 naive kernels	350MHz	60%	526
60 optimized kernels	400MHz	50%	18180

The final HLS design implements one kernel consisting of a pipeline with 66 stages and an ideal initiation interval of 1, achieving a maximum estimated frequency of 450 MHz. A single instance of this kernel occupies less than 1% of the total available resources on Alveo U50 and can process 303 million keys per second. Considering that the maximum number of 60 instances can fit on the Alveo U50 board, we estimate that such a system can process 18 billion keys per second.

By following the same implementation approach for XTEA, we achieved identical results as for the TEA algorithm. This was because the HLS tool successfully scheduled the additional operations required for XTEA within the same pipeline stages implemented for the TEA algorithm.

V. DISCUSSION

In the initial attempt at high-level synthesis of the algorithm, we utilized the simplest C++ implementation and allowed the HLS tool to apply its default optimizations. The only constraint specified was the maximum kernel frequency, which was limited to 100 MHz to reduce the overhead on the synthesis tools. This approach produced a pipelined kernel that outputs correct results and processes approximately 2.5 million keys per second. One instance of this kernel occupies around 1% of the total available resources on the Alveo U50.

Given the low resource utilization, we focused on increasing the kernel's parallelization in subsequent attempts to reduce execution time. We pursued two strategies: parallelizing operations within the pipeline and parallelizing multiple pipeline instances. These strategies did not yield favorable results. The resulting kernels either failed to produce correct outputs or exhibited an unsatisfactory ratio between performance and resource utilization. One of the noteworthy kernels from these attempts, which provided correct results, processed 142 million keys per second, occupying over 70% of the total resources. Compared to the initial attempt, and without accounting for the frequency increase, this resulted in a speed-up of 56.8 with 70 times more resource utilization. Based on these results, we concluded that the datapath could not be sufficiently parallelized in an efficient manner. Thus, we decided to maintain a simple kernel implementation and maximize the number of instances during the link stage.

Two additional attempts were made to simplify the kernel implementation. We observed that the break statement significantly degraded the pipeline's performance. The HLS tool would either generate a pipeline with an ideal initiation interval of 1 but a low frequency of 8 MHz, or a high-frequency pipeline with a suboptimal initiation interval of 64. To address this issue, the break statement was removed, and two static variables were introduced to maintain persistent output register values. Removing the break statement caused only a minor performance decrease, which was acceptable since the estimated time to complete all 20 million loop iterations was sufficiently small.

The host application was fairly consistent throughout the FPGA implementation steps. It is responsible for loading the XCLBIN file onto the Alveo board, initializing all kernels, and polling for results. It divides the key space into equal intervals based on the number of kernels, assigning each kernel a different starting key. For instance, to search through 1 billion keys using 50 kernels, the first kernel would start from key 0, the second from key 20 million, and so forth.

VI. FUTURE WORK

As future work, we intend to port our code TEA/XTEA code breaker on accelerators like [16] or map-reduce as in [17] or map-scan as in [18] or one with partial reconfiguration like [19]

VII. Conclusion

We successfully implemented a brute-force cracker for breaking the TEA and XTEA encryption algorithms across three devices: CPU, GPU, and FPGA. The benchmarks demonstrate that our FPGA implementation outperforms previously reported results in the literature by at least 2.5x core-per-core for TEA, and 20x for XTEA. This is attributed to the fact that we implemented an optimized accelerator and emphasized high parallelism by instantiating as many kernels as the FPGA resources allowed. Our FPGA implementation has similar performance regardless of the algorithm (TEA or XTEA) whereas our CPU and GPU implementations show a 38% performance degrade between TEA and XTEA. For a similar 75W TDP, our FPGA implementation outperformed our GPU implementation by at least 3x.

References

[1] Daxing Wang and Leying Xu. “Multi-bank Electronic Cash Scheme Based on Elliptic Curve Cryptosystem”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 82.4 (2020), pp. 131–142.

[2] Eugen Neacsu and Paul Schiopu. “A Security Analysis of Public Key Cryptographic Systems Used For Electronic Signature”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 83.1 (2021), pp. 135–146.

[3] Marian Cretu. “Analysis of a cryptosystem based on a modified Rijndael algorithm implementation”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 75.1 (2013), pp. 163–178.

[4] Wei Li et al. “Security Analysis of the Lightweight Cryptosystem TWINE in the Internet of Things”. In: *KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS* 9.2 (Feb. 2015), pp. 793–810.

[5] Yu-Chi Chen. “Security Analysis of Public Key Encryption with Filtered Equality Test”. In: *ADVANCES IN MATHEMATICS OF COMMUNICATIONS* 17.6 (Dec. 2023), pp. 1358–1363.

[6] Corina Macovei, Adina-Elena Lupu (Blaj), and Mircea Raducanu. “Enhanced Cryptographic Algorithm Based on Chaotic Maps and Wavelet”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 82.4 (2020), pp. 119–130.

[7] Masroor Hajari et al. “Impossible differential cryptanalysis of reduced-round TEA and XTEA”. In: *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*. 2015, pp. 58–63.

[8] Yi Cai and Mingsheng Feng. “A Dynamic Symmetric Seachable Encryption Algorithm With An Extended Dual-Index Structure Leveraging IPFS”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 86.1 (2024), pp. 191–206.

[9] “Quantum-Safe Cryptography (QSC); Limits to Quantum Computing applied to symmetric key sizes”. In: URL: https://www.etsi.org/deliver/etsi_gr/QSC/001_099/006/01.01.01_60/gr_QSC006v010101p.pdf.

[10] David J. Wheeler and Roger M. Needham. “TEA, a tiny encryption algorithm”. In: *Fast Software Encryption*. Ed. by Bart Preneel. Berlin, Heidelberg: Berlin Heidelberg, 1995, pp. 363–366. ISBN: 978-3-540-47809-6.

[11] Vivek Venugopal and Devu Manikantan Shila. “High throughput implementations of cryptography algorithms on GPU and FPGA”. In: *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. 2013, pp. 723–727.

[12] Mohamed H. AlMeer. “FPGA implementation of a hardware XTEA light encryption engine in co-design computing systems”. In: *2017 Seventh International Conference on Innovative Computing Technology (INTECH)*. 2017, pp. 26–30.

[13] Muhammad Awais Hussain and Rabiah Badar. “FPGA Based Implementation Scenarios of TEA Block Cipher”. In: *2015 13th International Conference on Frontiers of Information Technology (FIT)*. 2015, pp. 283–286.

[14] Kiran V.G et al. “Design And Implementation Of Tiny Encryption Algorithm”. In: *Int. Journal of Engineering Research and Applications* 5 (June 2015), pp. 94–97.

[15] Zeesha Mishra and Bibhudendra Acharya. “High throughput novel architectures of TEA family for high speed IoT and RFID applications”. In: *J. Inf. Secur. Appl.* 61.C (Sept. 2021). ISSN: 2214-2126.

[16] Mihaela Maliţa, George Vlăduţ Popescu, and Gheorghe M. Ştefan. “Heterogeneous Computing System for Deep Learning”. In: *Deep Learning: Concepts and Architectures*. Cham: International Publishing, 2020, pp. 287–319. ISBN: 978-3-030-31756-0.

[17] Mihaela Malita, George Vladut Popescu, and Gheorghe M. Stefan. “Pseudo-Reconfigurable Heterogeneous Solution for Accelerating Spectral Clustering”. In: *2020 IEEE International Conference on Big Data(BIG DATA)*. Ed. by XT Wu et al. 2020, pp. 5138–5145. ISBN: 978-1-7281-6251-5.

[18] Mihaela Malita and Gheorghe M. Stefan. “Map-Scan Node Accelerator for Big-Data”. In: *2017 IEEE International Conference on Big Data(BIG DATA)*. Ed. by JY Nie et al. 2017, pp. 3524–3529. ISBN: 978-1-5386-2715-0.

[19] Alexandru Gheolbanoiu et al. “A Software-defined FPGA vector processor with application-aware reconfiguration”. In: *UPB Scientific Bulletin Series C-Electrical Engineering and Computer Science* 78.4 (2016), pp. 43–56.