A Fast and Resource Efficient FPGA Implementation of Secret Sharing for Storage Applications

Jakob Stangl*†, Thomas Lorünser†, Sai Manoj Pudukotai Dinakarrao *
*Technical University of Vienna, Austria
jakob.stangl@tuwien.ac.at, saimanoj.p.2013@ieee.org
†AIT Austrian Institute of Technology GmbH, Digital Safety & Security, Austria
thomas.loruenser@ait.ac.at

***Abstract*—Outsourcing data into the cloud gives wide benefits and opportunities to customers. Beside these advantages, new challenges such as confidentiality and accessibility have to be addressed. One approach to overcome these challenges is by applying secret sharing in a distributed storage setting, known as cloud of clouds approach. For this purpose we present a new hardware architecture of a wide parametrizable secret sharing core. Performance metrics for various applied bit-widths of secret words are given, which are crucial for benefits of higher level protocols in the cloud of clouds approach. Additionally, a complete system which is able to operate in a network environment is presented. The achieved throughputs are in the order of Gbit/s. It is significantly faster than similar comparable hardware architectures and orders of magnitude higher than software implementations.**

***Keywords*-Secure cloud computing, cryptography, privacy, information theoretic security, usability, privacy by design**

I. Introduction

In recent years, cloud computing has emerged as a new way to access information technology (IT) and has transformed the digital landscape in many ways. The basic idea of the cloud is to centralize IT and leverage economies of scale to provide cheaper services. One major trend in cloud computing is away from dedicated monolithic storage solutions of specific vendors towards the use of cheap commercial off-the-shelf (COTS) hardware to reduce costs and avoid vendor lock-in. However, because of the high failure rate of COTS hardware, an additional layer of redundancy is needed to reach the desired reliability and availability of the overall storage system.

The application of erasure coding is considered the most efficient solution [1] to get high availability in modern storage solutions. In this work we demonstrate the use of secret sharing, similar to erasure coding. In the concept of secret sharing, a secret, the data, is transformed into multiple smaller so-called shares and stored on different servers. To restore the original data, only a configurable subset of these shares, k with $k \leq n$, is required. It is of no importance which of these shares are used, as long as sufficient shares are recombined. The secrecy is accomplished by the fact that any number smaller than this boundary k reveals no information of the secret.

Confidentiality of data is protected as long as the different shares cannot be collected easily. In practice, the shares should be distributed over multiple administrative trust zones, which can be anything from separate servers within data centres to dedicated cloud providers in multi-cloud settings. Contrary to applying a cipher to the secret and performing erasure coding, this approach is completely key-less and therefore avoids issues of key management.

While various software solutions exist, few have been proposed for hardware. However, implementations in dedicated hardware offer the possibility of higher performance and have the potential to expand its applicability.

When a file is shared, it is first split into words of a certain bit-width, on which a mathematical algorithm is applied. Although the security of the concept is not directly influenced by the word size, there is a trade-off in terms of efficiency and capabilities. Present solutions usually work on word sizes of one byte, which lower the mathematical complexity compared to greater bit-widths and provide higher performance. However, if additional verifiability techniques for auditing and verifiability procedures such as [2], [3] are to be supported, small word sizes do not provide adequate security and efficiency. Therefore in this work the feasibility and efficiency of hardware implementations using different bit-widths are investigated and efficient hardware architectures, optimised for a certain bit-width, are proposed. All these investigations are targeted at Field Programmable Gate Arrays (FPGA). A fully parametrizable architecture is the result. In particular, word-widths of 8, 16, 32, 64 and 128 bits were investigated.

A. About this Work

The contribution of this work is threefold: Firstly we present, to the best of our knowledge, the fastest and most resource efficient implementation of information theoretical secure secret sharing and information dispersal. The implementation is orders of magnitudes faster than known hardware implementations and tested software libraries. Secondly, it is also the first parameterizable core in terms of bit-width to be used and the core automatically adapts to the best implementation strategy for the given word size.

978-3-9819263-0-9/DATE18/©2018 EDAA

Various optimization strategies have been investigated to reach this goal. Thirdly, we present the first optimized and fully integrated computational secret sharing core dedicated for storage applications.

Furthermore a full prototype of a dispersed secure storage application was developed for demonstration purposes and to evaluate the developed IP core. The Zedboard , containing a Xilinx Zynq-7000 AP SoC XC7Z020-CLG484, was selected as the FPGA platform. It is partitioned into processing system (PS) and programmable logic (PL). Synthesis and implementations were performed with Vivado® 2015.3.

B. Organization of the paper

In section II, an overview of the concepts applied in this work is given. The related work is summarized in section III. In section IV, the proposed architecture is presented while in section V a complete system is developed and performance results are presented. Conclusions are drawn in section VI.

II. Preliminaries

The developed cores include both, a perfectly secure secret sharing scheme (PSS) and a computational secure scheme (CSS). The PSS scheme guarantees information theoretical security, which is the strongest form of security possible, but produces substantial storage overhead. In PSS, each of the n shares has the same size as the original unencoded data. Because PSS is not best suited for large storage applications, we chose to support computational secret sharing for bulk data processing. CSS is optimal in terms of storage overhead and still provides good security guarantees for practical usage.

A. Shamir's Secret Sharing (PSS)

For PSS we selected Shamir's algorithm [4], which is based on the generation, evaluation and interpolation of polynomials. In dependency of the threshold value k a polynomial of degree $k-1$ is defined. The lowest coefficient c_0 is the secret, while all higher coefficients $c_1 ... c_n$ are random numbers. To generate the shares, the polynomial is evaluated at n different x-points, $x \neq 0$, where each $x/f(x)$ pair is a share. The size of the share can be minimized by making x public. In order to restore the secret, a polynomial interpolation can be performed if at least k shares are combined. If any shareholder holds fewer than k shares no information about the secret is revealed, which gives Shamir's scheme information theoretical secrecy.

B. Computational Secret Sharing (CSS)

Hugo Krawczyk published the first and very efficient scheme for computational secret sharing [5]. First, the data is encrypted by a secure encryption function ENC and a random generated key. This key is shared using a PSS. The rest of the data is shared with the polynomial concept of Shamir, but replacing random numbers in all of the coefficients of the polynomial with other secrets. This way of using polynomials to encode data is also called information dispersal (IDS) as introduced by Rabin [6] and is mainly used for erasure coding in storage applications.

In the process of reconstructing the secret, first the key is restored followed by the encrypted secret, which is then decrypted by applying the key in ENC^{-1}.

III. Related Work

There are few hardware implementations of secret sharing algorithms known from the literature and no extensive treatment of such has been done so far. The focus in these works is different from ours and — to the best of our knowledge — there exists no speed optimized FPGA implementation for full CSS schemes. Moreover, no work deals with the analysis and evaluation of different word sizes and their trade-offs.

In [7], secret sharing is used in a network monitoring application. Only the front-end sharing part, which handles the data at Gigabit rates, was realized in hardware. A significant performance increase was gained by restricting the bit-width of the x-value. However, the multiplier in the computational core was not optimized for this application and gives potential for further performance increases. The work was implemented on a network FPGA card using a Virtex-II Pro 50 FPGA. The isolated examination of the share generation reveals a throughput of 2359 Mbit/s with the usage of 1633 slices in this design. These are roughly 3266 4-input look-up-tables. The full key share units reach a throughput of 343 Mbit/s with 3687 slices and 18 BRAMs.

Another implementation from [8] focuses on secure secret sharing. The target architectures are application-specific integrated circuits (ASICs), synthesized using Cadence Encounter RTL Compiler with the Nangate 45 nm Opencell library. They apply robust codes and algebraic manipulation detection to resist strong cheating attacks. Besides the size of the implementation the results are focusing on the efficiency of cheating detection and correction and the causing area-overhead. There are no performance results given in terms of throughput or any bit-width dependencies.

There are various software implementations of secret sharing. The crucial parts of software implementation are the time-consuming polynomial multiplications limiting the performance. Multiplications of small words can be processed efficiently with look-up-tables, but processing time grows exponentially with the greater bit-widths needed by larger Galois fields (GF). Therefore, software implementations are only efficient for widths of up to 16 bits.

In [9], a collection of libraries supporting secret sharing was gathered. The GFShare library [1] operates in a GF(2^8) field and was analysed in detail on an Intel i5-2500K, 3.3 GHz, 8 Gbit RAM, computer system. The achieved

[1] https://launchpad.net/libgfshare (Accessed: 03.07.2017)

throughput for sharing large files for an applied $5/3$ threshold scheme was 7.4 Mbit/s. This value is relatively small compared to Gigabit networks and small extension fields.

In [10], secret sharing schemes were analysed in terms of their performance. The focus of their work is the comparison of different schemes according to the threshold parameters n and k. There was no information about the computer system. However, with Shamir's secret sharing scheme a throughput of about 9 Mbit/s could be achieved for generating 10 shares. The computational secret sharing achieved a throughput of about 18 Mbit/s for the same amount of shares.

To increase the throughput, efforts were made to use the Graphics Processing Unit (GPU) for better performance. In [11] it was shown that the benefit of a GPU increases with higher thresholds. A threshold of 4 achieves a throughput of 48 Mbit/s for the calculation of one share. Another implementation of secret sharing based on cellular automata on a GPU [12] reveals a speed of 40-160 Mbit/s in a $5/5$ threshold scheme.

The most comprehensive secret sharing library for storage applications was presented in [13] as part of a full multi-cloud storage application. However, the implementation is in Java and the reported performance figures are of the order of about 500 Mbit/s in a $4/3$ threshold scheme.

IV. Architecture

The main functional block is structured into a Share Generation Unit (SGU), a Secret Reconstruction Unit (SRU) and an Advanced Encryption Standard Core (AES), which performs the ENC and ENC^{-1} functions of the CSS algorithm. Moreover, a True Random Number Generator (TRNG) based on the design of Wold and Tan [14] was developed, but excluded from the core to enable different TRNGs in order to reach the individual space security trade-off. The SGU and SRU are designed to work in Shamir's mode as well as in the computational mode. They are capable of sharing the key and the payload, selectable via a single signal. Due to the high resource occupation of the AES, only one AES is shared between the SGU and SRU.

While the AES, SGU and SRU operate on streams, independently processing words of a certain bit-width, the overall architecture is packet based for better communication on the external interfaces and a better combination of payload shares with their according key share. Figure 1 represents this architecture. Header signals are added to each packet to enable fragmenting and identification of packets, which are additionally passed through the CSS core.

While the AES statically works with 128 bits, the SGU and SRU are designed to work generically at a bit-width of 8, 16, 32, 64 or 128 bits. The bit-width of the SGU and SRU is selected before synthesis via generic parameters. The design is then adapted to certain implementation strategies for the set bit-width. All operations are performed within a Galois field GF(2^n), where n corresponds to the bit-width.

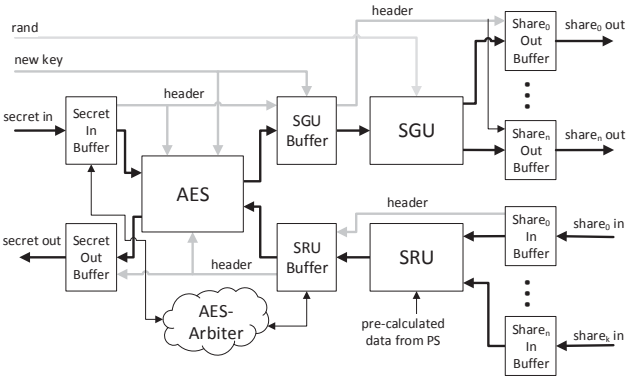

Figure 1: Architecture of the CSS core.

A. Share Generation and Secret Reconstruction

If the Secret In Buffer contains a whole packet and the AES is not busy, it is passed to the AES. If a new key is required according to the packet's header, it is loaded and forwarded to the SGU Buffer, together with the encrypted packet. If the SGU Buffer is full, the packet is forwarded to the SGU, where the key is first shared using Shamir's mode and then the payload is shared with the computational mode. The reconstruction follows the same sequence.

B. AES Blockcipher Unit

An OpenCores certified AES core[2] having sufficient performance was chosen. It is able to process one new 128 bit block each cycle, with a latency of 22 cycles. The applied core makes use of BRAMs and LUTs and was modified in this project to give more generic design options for their usage. The plain AES core was extended to operate in counter mode (CTR). Because it is shared by the SGU and SRU, the final AES unit consists of two count registers of generic width and two key registers.

C. Share Generation Unit (SGU)

The share generation unit supports two modes, Shamir's scheme and information dispersal. Both require the definition of a unique polynomial and the shares are generated by evaluation at an arbitrary point x with the condition $x \neq 0$, i.e. $f(x) = \sum_{i=0}^{n} c_i x^i$. In Shamir's mode the lowest coefficient c_0 is a secret word and the other coefficients are filled with random words, while in the information dispersal mode all coefficients $c_0 ... c_n$ are filled with secret-words.

Evaluating the polynomial directly requires $\sum_{i=0}^{n}(i)$ multiplications and n additions. Conversely, in the Horner scheme the coefficients are processed from c_n to c_0 downwards which allows a significant reduction to n multiplications and n additions, as shown in equation (1).

$$f(x) = (...(c_n x + c_{n-1})x + ... + c_2)x + c_1)x + c_0 \qquad (1)$$

[2]https://opencores.org/project,tiny_aes (Accessed: 16.9.2017)

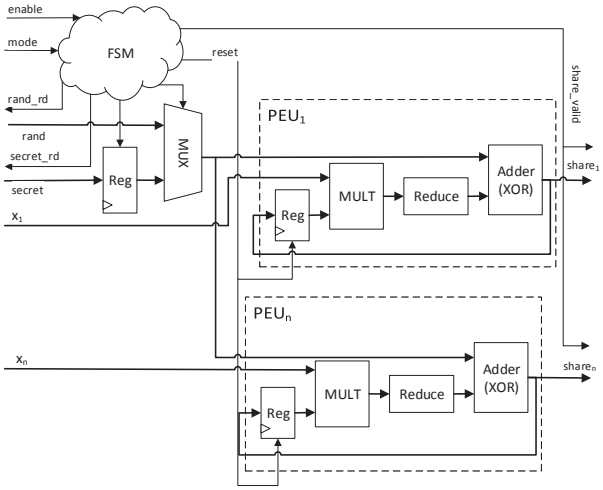

Figure 2: Architecture of the SGU.

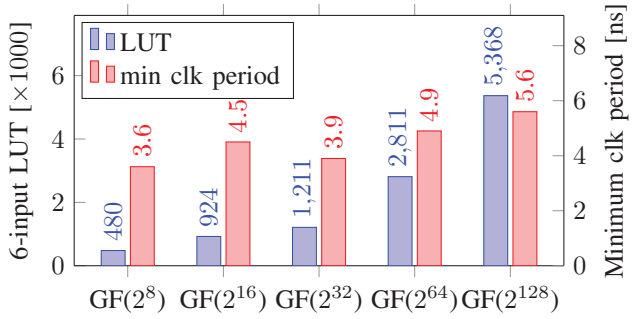

Figure 3: Implementation results of the SGU, where 10 shares can be generated in parallel.

The SGU architecture is illustrated in figure 2. Multiple Polynomial Evaluation Units (PEUs) perform the evaluation of the Shamir polynomial to generate the shares in parallel. Each PEU needs $k-1$ cycles to construct the share; the coefficients are loaded sequentially. Each PEU consists of an adder, realizable with XORs, a multiplier and a reduction circuit appropriate to the Galois field. By defining the Galois field with an irreducible polynomial of low weight, the reduction circuit can be realised by a minimum number of static-XOR connections. The polynomial multiplier shows the strongest size increase with ascending bit-width. As mentioned in [7], a significant reduction can be realized by restricting the x-value. While this value is generic in the design, it was set to 8 bits in this paper. By holding the x-value at a certain bit-width, the size growth is linear, which results in a similar share generation performance for all the investigated bit-widths. In figure 3 implementation results are shown for a parallel generation of 10 shares.

D. Secret Reconstruction Unit (SRU)

The evaluation process of the Shamir polynomial is a linear equation system. Written in matrix notation it leads to formula (2), with a matrix SEC containing a set of secrets, a matrix SH containing a set of shares and a matrix X with x-values and their powers. Such secrets or shares are always computed together.

$$\underbrace{\begin{pmatrix} share_0 \\ share_1 \\ share_2 \\ share_3 \end{pmatrix}}_{SH} = \underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} secret_0 \\ secret_1 \\ secret_2 \\ secret_3 \end{pmatrix}}_{SEC} \tag{2}$$

By solving the equation for SEC, the matrix X has to be inverted, which leads to equation (3) for the reconstruction of a set of secrets.

$$SEC = X^{-1}SH \tag{3}$$

Because the calculation of the matrix X^{-1} is of high computational effort and is only required if new x-values are applied, it has high potential for HW/SW partitioning. Therefore the PS of the Zynq-7000 SoC calculates all matrix coefficients and loads these values into the SRU. The hardware architecture of the SRU is presented in figure 4.

Every row of the matrix X is calculated in parallel, where each multiplication takes place in a $share_n - subreconstruction$ block. The multiplication results are added and reduced in order to obtain one secret of the set. Since there is no feedback loop, the whole process can be pipelined. To meet a custom design trade-off of resources against timing requirements, the number of pipelining stages can be set at synthesis time by generics. The rows of the matrix are processed sequentially by applying different values of the matrix X but the same share for each multiplier. These values are previously calculated from the PS.

While the polynomial multiplier becomes the bottleneck of the SRU, a bit-width limitation of one input, as it was done for the SGU, is not applicable here. However, because these multipliers essentially limit the applicability of higher bit-widths, further optimisations were necessary. One approach is applying Karatsuba's algorithm [15] to decrease the mathematical complexity. A multiplication of n bits is broken down to 3 multiplications of $n/2$ bits and 4 additions, which reduces the complexity from $O(n^2)$ to $O(2^{\log_2(3)})$. This method is applied recursively. In the case of 6-input LUTs a resource reduction is observable upon a width of 16 bits. Therefore, in the proposed design, Karatsuba's algorithm is applied recursively until a 16×16 bit sub-multiplication is reached.

Since polynomial multiplications are performed, arithmetic multipliers of FPGA-DSPs are not directly applicable. The main difference is the absence of the carry bit. Designs

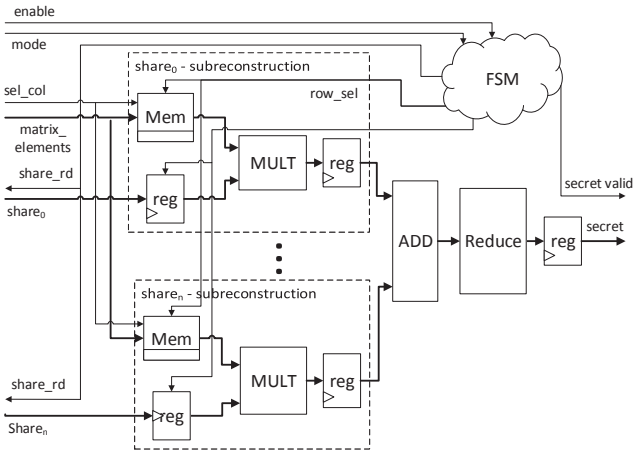

Figure 4: Architecture of the SRU.

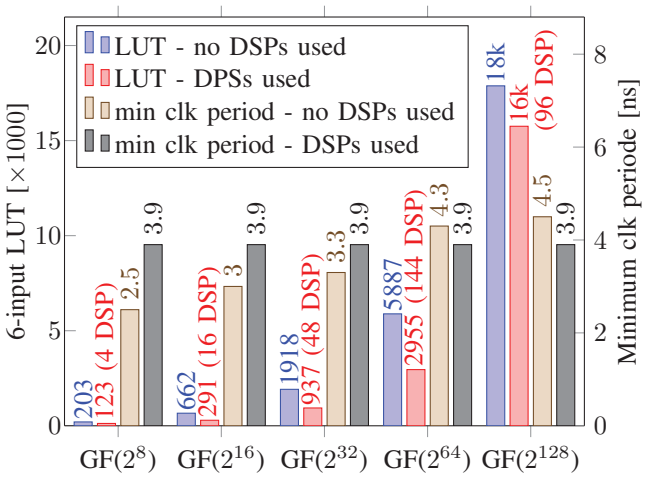

Figure 5: Implementation results of the CSS SRU with and without the inclusion of DSPs.

to include DSP multipliers were investigated, even if potentially less efficient. In practice an 18×25 DSP-multiplier was adapted to a 9×6 polynomial multiplier by skipping all the bits influenced by a carry. From these a 16×16 bit polynomial multiplier was formed to serve as a base multiplier block, which saves about 15-20 LUTs per DSP.

The number of FPGA-DSPs used can be controlled by generics, allowing excess DSPs to save LUTs. The implementation results for the SGU, assuming a threshold of 4, are presented in figure 5, comparing the involvement of DSPs in the polynomial multipliers against pure LUT utilization.

V. Results

In order to evaluate the complete architecture in a real setup, a complete system was developed, embedded in a network environment to manage, share and reconstruct complete files. As target platform the Zedboard was extended

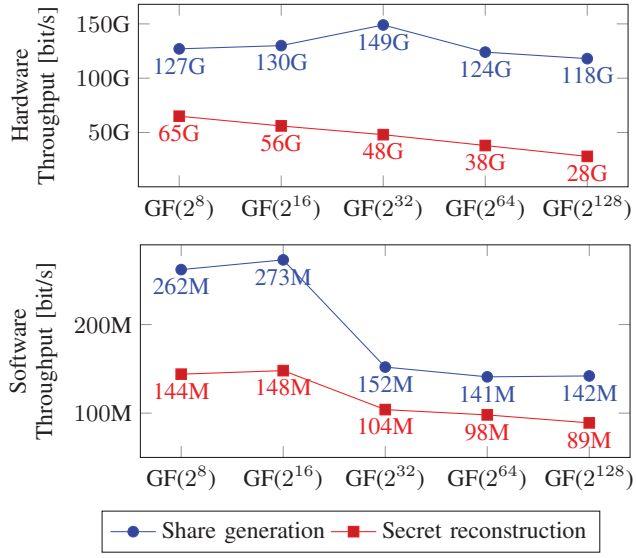

Figure 6: The theoretical maximal throughput of a SW and an FPGA implementation for an $8/4$ threshold scheme. The throughput is measured in share-bit/s for the share generation and secret-bit/s for the secret reconstruction.

by an Ethernet FMC, where 3 Gigabit Ethernet connections are used. The CSS core was encapsulated in a wrapper, to correctly distribute each packet of the 3 physical connections to a buffer, as prepared dynamically by the PS. A specially developed protocol carried over UDP is used to fragment and identify all packets.

The result is a resource optimized architecture, widely parametrizable for parameters such as the bit-width, x-value range, buffer size, packet size, n/k threshold scheme, inclusion of DSPs, BRAM/FF usage, pipelining stages etc.

In order to obtain the theoretical throughput, the developed cores were evaluated independently and the results were extrapolated for a 50% utilization of the target FPGA, listed in figure 6. In this theoretical estimation, the data flows and buffers were neglected. DSPs were not included in this design to allow a more universal comparison. The counterpart was a software-implementation, which also neglected all data flows and simply performs all of the required calculations, performed on an Intel i5-4590 Quadcore running at 3.3 GHz and full utilization of all cores. The share generation rate in the FPGA design could be held almost constant due to the fixed bit-width of the x-point. The reconstruction shows a significant performance decrease with ascending bit-width, for both the software and hardware implementations. However, overall the FPGA design shows the capability of throughputs 100 to 1000 times faster than its software counterpart. This was observed for the software applied in figure 6 as well as the previously described existing software libraries summarized in section III.

The complete core in the prototype was built with a 64

Functional Unit	LUT	(%)	FF	(%)	BRAM	DSP	P(W)
Complete Design	27745	(100)	35405	(100)	90	144	2.99
PS	—	(–)	—	(–)	—	—	1.53
3 Ethernet IP	7317	(26)	3984	(11)	4	0	0.88
TRNG	146	(0.5)	52	(0.1)	0	0	0.001
Share Switch	119	(0.7)	151	(0.7)	0	0	0.014
CSS Core	17135	(62)	20448	(58)	77.5	144	1.02
AES	2588	(9.3)	8851	(43)	36	0	0.46
SRU	2942	(11)	7467	(36)	0	144	0.3
SGU	1638	(6)	1095	(5.3)	0	0	0.033
CSS Buffers	3154	(11)	9728	(48)	39	0	0.15

Table I: FPGA utilization results for the functional units of a complete CSS system in a $4/8$ threshold scheme.

bit architecture, as it is a good trade-off in terms of resource utilization and capabilities. It is capable of processing 6.4 Gbit/s, working with 64 bit words and an $8/4$ threshold scheme. The bottleneck is the external Ethernet connection, which limits the speed to 1 Gbit/s. The resource utilization and power consumption, structured in terms of functional components, is summarized in table I.

VI. Conclusion

We presented the first hardware based computational secret sharing core including information theoretical secret sharing and information dispersal as dedicated components. Furthermore the proposed core is widely parametrizable and the influence of the secret word-width on the performance has been evaluated and shown in detail.

We show the feasibility of significant performance increase when performing secret sharing on hardware. While a significant performance drop for higher bit-widths is observable, as it is in software, higher bit-widths are still processable with throughput orders of magnitudes higher than in software solutions.

Therefore, FPGA implementations of secret sharing are applicable in multi-cloud storage solutions to enable the efficient usage of higher protocols, such as auditing and verification. It widens the application of a secure storage solution on top of existing cloud infrastructure in a distributed setup. Single point of trust or failure are avoided and secrecy is achieved without key-management issues. This work is a step towards practical high-speed low latency storage solutions based on secret sharing, which has been considered in academia for a long time but was lacking efficient implementations.

In the future work the core has to be adapted to high-bandwidth storage interfaces and cloud storage protocols in order to increase its applicability. Moreover, the extension to a verifiable secret sharing core is of interest.

Acknowledgements

This work was partly funded by the European Commission through grant agreement no 644962 (PRISMACLOUD). The authors would also like to thank Xilinx University program (XUP) for their support.

References

[1] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002.

[2] D. Demirel, S. Krenn, T. Loruenser, and G. Traverso, "Efficient and Privacy Preserving Third Party Auditing for a Distributed Storage System," in *International Conference on Availability, Reliability and Security*, 2016.

[3] S. Krenn, T. Loruenser, and C. Striecks, "Batch-verifiable Secret Sharing with Unconditional Privacy," in *International Conference on Information Systems Security and Privacy*, 2017.

[4] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, nov 1979.

[5] H. Krawczyk, "Secret sharing made short," in *International Cryptology Conference on Advances in Cryptology*, 1994.

[6] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM (JACM)*, vol. 36, no. 2, pp. 335–348, 1989.

[7] J. Wolkerstorfer, "Secret-sharing hardware improves the privacy of network monitoring," in *International Workshop on Data Privacy Management, and 3rd International Conference on Autonomous Spontaneous Security*, 2011.

[8] P. Luo, A. Y. L. Lin, Z. Wang, and M. Karpovsky, "Hardware implementation of secure shamir's secret sharing scheme," in *International Symposium on High-Assurance Systems Engineering*, 2014.

[9] M. Kirchner, "On the applicability of secret sharing cryptography in secure cloud services," Master's thesis, Technische Universität Wien, 2014.

[10] A. Abdallah and M. Salleh, "Secret sharing scheme security and performance analysis," in *International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE)*, 2015.

[11] S. Chen, L. Bai, Y. Chen, H. Jiang, and K. C. Li, "Deploying scalable and secure secret sharing with gpu many-core architecture," in *IEEE International Parallel and Distributed Processing Symposium Workshops PhD Forum*, 2012.

[12] R. A. Hernandez-Becerril, A. G. Bucio-Ramirez, M. Nakano-Miyatake, H. Perez-Meana, and M. P. Ramirez-Tachiquin, "A gpu implementation of secret sharing scheme based on cellular automata," *The Journal of Supercomputing*, 2016.

[13] T. Loruenser, A. Happe, and D. Slamanig, "Archistar: towards secure and robust cloud based data sharing," in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE International Conference on*, 2015.

[14] K. Wold and C. H. Tan, "Analysis and enhancement of random number generator in fpga based on oscillator rings," in *International Conference on Reconfigurable Computing and FPGAs*, 2008.

[15] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet Physics - Doklady*, 1963.