Exploring the Application of Homomorphic Encryption to a Cross Domain Solution

Cody Tinker
Department of Computer Engineering
Rochester Institute of Technology
cwt9976@rit.edu

Kevin Millar
Department of Computer Engineering
Rochester Institute of Technology
kdm8162@rit.edu

Alan Kaminsky
Department of Computer Science
Rochester Institute of Technology
ark@cs.rit.edu

Michael Kurdziel
Harris Corporation
mkurdzie@harris.com

Marcin Lukowiak
Department of Computer Engineering
Rochester Institute of Technology
mxleec@rit.edu

Stanisław Radziszowski
Department of Computer Science
Rochester Institute of Technology
spr@cs.rit.edu

***Abstract*—A Cross Domain Solution (CDS) is a means of secure information exchange that provides the ability to access or transfer digital data between varying security domains. Most existing CDS methods focus on risk management policies that rely on using protected or trusted parties to process the information in order to solve this problem. A CDS that is able to function in the presence of untrusted parties is a challenge.**

We apply the concepts of homomorphic encryption (HE) to explore a new solution to the CDS problem. We built a practical software case study application using the Yet Another Somewhat Homomorphic Encryption Scheme (YASHE) around the specific challenge of evaluating the gateway bypass condition on encrypted data. We assess the feasibility of such an application through performance and memory profiling in order to find a parameter selection that ensures proper homomorphic evaluation. The correctness of the application was assured for 64-, 72-, 96-, and 128-bit security parameter selections of YASHE resulting in high latency performance. The computing time required by our proof-of-concept implementation may be high but this approach allows the manual process employed in current systems to be eliminated.

***Index Terms*—Homomorphic Encryption, Cross Domain Solution**

I. Introduction

The Cross Domain Solution (CDS) problem is concerned with the ability to access or transfer information between multiple but different security domains [1]. An application is said to be a CDS if it solves this problem with regard to the transfer or access of the information. Most current solutions are based on a system of risk management that usually relies on protected party members. However, there still exists an unsolved problem of how to create an autonomous system that transfers highly classified data through a lower or untrusted classified network without revealing any information about the data to the intermediate parties. This task is difficult because of the lack of solutions that allow an untrusted or semi-trusted party to operate on the encrypted data to determine its routing endpoint without first decrypting the data. Homomorphic encryption (HE) schemes provide a potential solution to the CDS problem as they allow computations to be performed on encrypted data without the need to decrypt it. For example, as shown in Figure 1, if a party wants to perform an operation with two plaintext operands, the homomorphic version of the operation can be calculated using encrypted operands. The homomorphic operation will return an encrypted output. The decryption of this output is the same as the output which would have been received if the normal operations were performed on the unencrypted operands.

The objective of this work was to explore and assess the viability of using homomorphic encryption to create a CDS that ensures security while transferring information across different security domains. The Yet Another Somewhat Homomorphic Encryption Scheme (YASHE) [2] is used as an encryption scheme basis for building a proof-of-concept application that determines routing endpoints for several pieces of encrypted data based on a hierarchy of data attributes. The application was profiled and analyzed to evaluate potential problems that come with using homomorphic encryption as a solution, as well as to identify methods to improve upon these issues. The correctness of the application was assured for 64-, 72-, 96- and 128-bit security parameter selections of YASHE with the lightweight block cipher SIMON [3], [4]. Note that this resulted in high latency performance. SIMON was selected due to its hardware implementation simplicity relative to the AES. The homomorphic computation times, as reported in Section VI, may be high, but this approach allows the manual process often employed in the current systems to be eliminated. Using the most popular block cipher AES instead of SIMON would increase the cost of homomorphic processing by YASHE significantly.

This paper is organized as follows: The CDS problem is reviewed in Section II. Related work to the CDS problem is presented in Section III. An overview and the rationale for selecting YASHE and SIMON as cryptographic schemes used in our experimentation are briefly described in Section IV. In the same section, computation and security costs resulting from the selection of the HE parameters are also discussed. The case study describing the framework for achieving a CDS

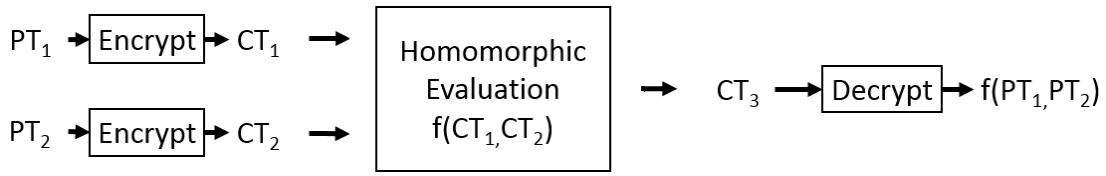

Fig. 1. The process flow for using homomorphic encryption. "PT" represents plaintext, and "CT" represents ciphertext. The "Encrypt," "Evaluation," and "Decrypt" operations are all defined by the homomorphic encryption scheme being used. Key management is not included in this diagram.

solution using HE is established in Section V. The implementation of the case study and the experimentation results are provided and analyzed in Section VI. The conclusions of our work are presented in Section VII.

II. The Cross Domain Problem

Multiple data producers exist that want to relay information of varying classifications to specific endpoints. The data they relay is broadcasted on an untrusted network where the data will be received by all parties who act as network gateways. The purpose of the gateways is to only relay information whose classification matches that of the network at its endpoint. These gateways must be treated as untrusted parties. They must not learn anything about the type of data they relay or the actual classification of its endpoint. Once a gateway has processed the data and confirmed the access level it has observed, it will relay the data to its endpoint. The system can include multiple types of classifications, such as: Top Secret, Secret, Confidential, Restricted, Official, Unclassified. An endpoint network may have one or more associated users. A figure that represents the concept of the problem is shown in Fig. 2.

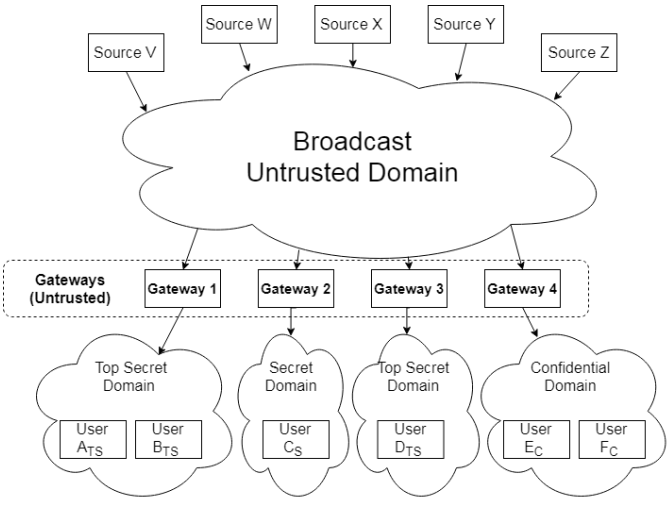

Fig. 2. The Cross Domain Network Case Study

The gateway will not learn anything about the classification of the data. It only produces an encrypted result which is decrypted and used by the corresponding router (for simplicity the routers are not shown in the diagram). As such, the gateway learns nothing about the data, or its intended destination. This prevents a rogue gateway and network from searching for specific information and gathering data not intended for its endpoint. The summary of desired properties for CDS is as follows:

- Payload data must remain encrypted at all times,
- The Payload must have associated attribute data,
- Required network bandwidth should be minimized,
- The Gateway must be treated as an untrusted party, i.e. it must not be able to decrypt homomorphically encrypted data,
- The Router can be semi-trusted at most.

III. Related Work

The difficulty of CDS is that, in order to operate on the plaintext data, any party on the network must be protected and trusted [1]. The National Institute of Standards and Technology (NIST) provides a large catalog of security frameworks. These include a number of non-mutually exclusive scenarios for CDS, in the information flow enforcement section, from which one can select to fulfill their requirements [5]. However, the methods and policies used are primarily determined by the implementing organization. Most CDS related definitions focus on security policy and risk management using protected domains with authorized human personnel to manually evaluate and control the flow of information. Therefore, a system that is able to automatically use information while maintaining security inside an untrusted domain is a challenging problem. The use of homomorphic encryption to achieve this objective is a mostly unexplored research area and could provide an interesting alternative solution.

Homomorphic encryption refers to a type of encryption, which allows a user to operate on the encrypted data, and can theoretically allow untrusted parties to operate on the data while maintaining security. The concept of a homomorphic encryption scheme was introduced by Rivest, Adleman, and Dertouzous in 1978 [6]. However, the first construction of a fully homomorphic encryption (FHE) scheme was not proposed until 2009 in Craig Gentry's PhD dissertation [7]. An FHE scheme is one that allows an arbitrary set of operations to be evaluated on encrypted data for an unlimited number of times. This is in contrast to other systems that were previously developed, which only permitted a limited number of operations.

After 2009, several FHE schemes have been explored. These include systems based on the Learning With Errors (LWE) and Ring-Learning With Errors (R-LWE) [8], [9]

problems. The Brakerski-Gentry-Vaikuntanathan (BGV) [10] proposed a novel leveled fully homomorphic scheme with large improvements in performance. The BGV scheme focuses on reducing the per-gate computation of the homomorphic operations. BGV builds on the techniques introduced by the BV scheme [11], such as avoiding the squashing method and by effective managing the noise of the ciphertext in terms by improving the modulus switching technique.

The Brakerski/Fan-Vercauteren (BFV) scheme [12], also commonly abbreviated as FV, is a part of the scale-invariant B12 scheme by Brakerski [13]. This scheme is based on the LWE and R-LWE. FV also proposes two methods for relinearization that are more efficient than the technique used in B12. In addition, despite the avoidance of the technique in [13], FV introduces modulus switching from [11] as an optimization just as in the BGV scheme for bootstrapping purposes. Like BGV and a few others, FV avoids the use for the squashing technique.

IV. YASHE AND SIMON

YASHE is a leveled FHE scheme developed by Bos et. al [2] of the Microsoft Research Team, and it is based on the Stehlé and Steinfeld [14] and López-Alt et al. [15] encryption algorithms. It was chosen over other popular schemes like BGV and BFV because of its relative simplicity and performance. YASHE's security is based on the R-LWE hardness assumption [8]. This approach has been gaining acceptance in the cryptographic community due to its promising performance and security properties. The R-LWE schemes offer an optimization technique to perform a key switching operation after multiplication to transform the ciphertext to one encrypted with the original key, and to prevent the need to expand the ciphertext size after the multiplicative operations are performed [2].

A high-level simplified description of the YASHE scheme is as follows. A security parameter λ that determines a ring R of polynomials with modular arithmetic. There exists public parameters h and γ and a secret parameter f. Parameters h and f are single ring elements and γ is an array of ring elements. Encryption of a message $m \in R/tR$ to ciphertext $c \in R$ follows $c = [\lfloor \frac{q}{t} \rfloor [m]_t + e + hs]_q \in R$ where s, e are ring elements sampled from a random distribution, and t and q are the plaintext and ciphertext coefficient moduli, respectively. Decryption of a ciphertext $c \in R$ to message $m \in R/tR$ follows $m = [\lfloor \frac{t}{q} [fc]_q \rceil]_t \in R$. Homomorphic addition of two ciphertexts, c_1 and c_2, follows $c_{add} = [c_1 + c_2]_q$. Homomorphic multiplication of two ciphertexts, c_1 and c_2, follows $c_{mult} = \text{KeySwitch}(\tilde{c}_{mult}, \gamma)$, where $\tilde{c}_{mult} = [\lfloor \frac{t}{q} c_1 c_2 \rceil]_q$ and KeySwitch($\tilde{c}_{mult}, \gamma$) refreshes the ciphertext to one with lower noise.

The parameters chosen for the implementation of an HE scheme have an effect on security, ciphertext size, performance, bound on the circuit evaluation depth, and the plaintext space. The main choices that influence overall performance of the system are the polynomial ring degree n and the coefficient modulus q, which are defined by the HE schemes, in particular such as YASHE. Table I presents a selection of parameters for n and the maximum size of q, in bits $\log_2(q)$, that achieve 64-, 72-, 96-, and 128-bit overall system security. Other parameters, like the number of rounds in SIMON, were chosen to minimize computations achieving given security level [2], [3], [4]. The last two columns show the amount of memory required to store a ring element and the expected maximum circuit evaluation depth L.

TABLE I
R-LWE BASED FHE PARAMETERS FOR VARYING SECURITY LEVELS

n	$\log_2(q)$	Security (bits)	Ring Element Size (kB)	L
16384	885	64	1,770	28
	770	72	1,540	24
	570	96	1,140	17
	440	128	880	13
32768	1,770	64	7,080	53
	1,540	72	6,160	46
	1,139	96	4,556	34
	880	128	3,520	26
65536	3570	64	28,560	106
	3130	72	25,040	90
	2310	96	18,480	66
	1760	128	14,080	50

SIMON, which is a symmetric key block cipher, has a lightweight architecture and was publicly released by the National Security Agency (NSA) in 2013 [3], [4]. The cipher consists of a balanced Feistel network [16] with a round operation that includes a mix of three simple, bitwise operations: shift, XOR, and AND. This makes the algorithm better suited to homomorphic implementations [17], [18]. Furthermore, when compared to other block ciphers such as the AES, SIMON's structure is much simpler, and provides tweaking parameters such as block size, number of rounds and security level.

V. CASE STUDY

Our case study focused on the ability to route data from sources to proper endpoints without revealing any information about the data or its routing path over an untrusted network. The proof-of-concept application was implemented using the YASHE homomorphic encryption scheme and the SIMON block cipher. Since computational performance and memory usage are the main concerns with HE systems, the best approach is to work with small but descriptive metadata that is sufficient to determine the type of payload that is encrypted and its destination. In our case study the metadata was restricted to be a 32-bit random value. This allowed only one ring R element to store the encrypted metadata. The assigned values of each classification in the context of the application are represented in Table II.

As a proof-of-concept, our application used only a single data producer, gateway, and router. A top level design of data flow is shown in Fig. 3. The producer gathers data and builds metadata describing collected data and its destination constraints. Both data and metadata are encrypted (using different schemes, see Fig. 4). The gateway evaluates

TABLE II
ASSIGNED METADATA CLASSIFICAION VALUES

Description	Hex Value
Top Secret	0xE7191C86
Secret	0x72CCDDC8
Confidential	0xC989E663
Restricted	0x636C6E0C
Official	0x99BB94C7
Unclassified	0xCCCCE185

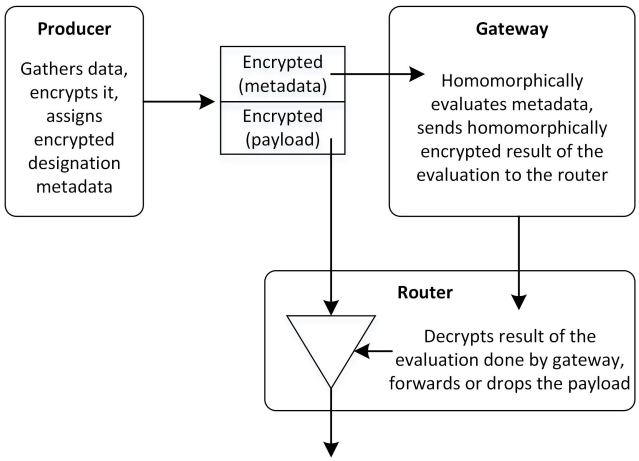

Fig. 3. Top Level Flow

encrypted metadata homomorphically and then forwards to the router encrypted routing constraint. The router decrypts it and according to the outcome, either forwards the payload, which remains encrypted, or drops it.

Details of the top-level data flow in Fig. 3 are presented in Fig. 4. All keys are shown as generated by the Key Distribution Center (KDC). However in particular scenarios, keys such as the symmetric key `AESsk` for the AES payload encryption, may be pre-placed at the producer and the destination domain and not use the KDC. Three distinct encryption schemes are color coded as follows. Red: AES is used to encrypt a (large) payload from each producer (only one shown) using `AESsk`. Yellow: the lightweight scheme SIMON, using the `SIMONsk` key, is first used to encrypt (small) metadata classification values which define the destination domains. Then, the SIMON algorithm and keys are used by YASHE within the homomorphic operations. Blue: a homomorphic scheme implemented by YASHE, using a public/private key pair `Ypk/Ysk`, supports gateways and router functionalities (only one gateway and router shown). First, using homomorphic evaluation with the key `Yevk`, the gateway produces inputs for homomorphic comparison of metadata to routing conditions. Second, the YASHE comparison result, encrypted with public key `Ypk`, is passed on to the router (only one shown). The router extracts the routing flag, using the secret key `Ysk` and determines whether to forward or drop the AES encrypted payload.

TABLE III
MAIN PARAMETER SELECTION FOR CONFIGURATIONS ACHIEVING CORRECTNESS FOR CHOSEN SECURITY LEVEL

YASHE configuration	Security (bits)	n	$\log_2(q)$	SIMON (#rounds)
α	64	16384	885	32
β	72	32768	1139	36
γ	96	32768	1139	36
δ	128	65536	1760	44

VI. RESULTS

We performed the experiments using a software application implemented in C++. Lepoint and Naehrig [18] provided a preliminary open source implementation of the YASHE scheme. Their implementation offers an easy to follow class architecture for YASHE which includes parameter set up, key generation, encryption, decryption, and ciphertext addition and multiplication operations. This implementation also provides homomorphic evaluation of SIMON. Our application is based on [18] and uses the Fast Library for Number Theory (FLINT) [19] and the GNU Multiple Precision (GMP) [20] arithmetic libraries. These libraries are optimized for large data structures such as the ones used in the YASHE scheme. Our application utilizes multithreading for YASHE and homomorphic SIMON operations. The application was compiled using gcc 7.2.1 and executed on an AMD Opteron 6272 running at 2.1GHz with 128GB of RAM, for all experiments.

A small collection of parameter configurations were established that supports a variety of requirements. Each configuration was established using the smallest parameters (n, q) that allowed correct homomorphic evaluation of SIMON by YASHE, for standard values of the number of rounds of SIMON and overall system security of 64–128 bits. The parameter selection for the configurations α, β, γ, and δ are represented in Table III. The memory profiling results for each configuration are presented in Table IV and the performance profiling results for each configuration are presented in Table V.

VII. CONCLUSIONS

This work demonstrated that, provided the application process is kept lightweight and simple, a CDS can be achieved using homomorphic encryption. We demonstrated that a practical application that securely transfers cross domain information across an untrusted network can be achieved under a parameter selection that ensures 64-bit security under both the YASHE and SIMON. These configurations require an execution time of about one hour and no more than a 220 MB of total processing memory. Both are easily achievable using modern computing systems. Our method exhibits high latency due to the homomorphic evaluation of the SIMON decryption circuit. However, the method permits the required user to gateway network bandwidth to remain low.

The performance of current fully homomorphic encryption schemes, especially for large parameters, can still be improved

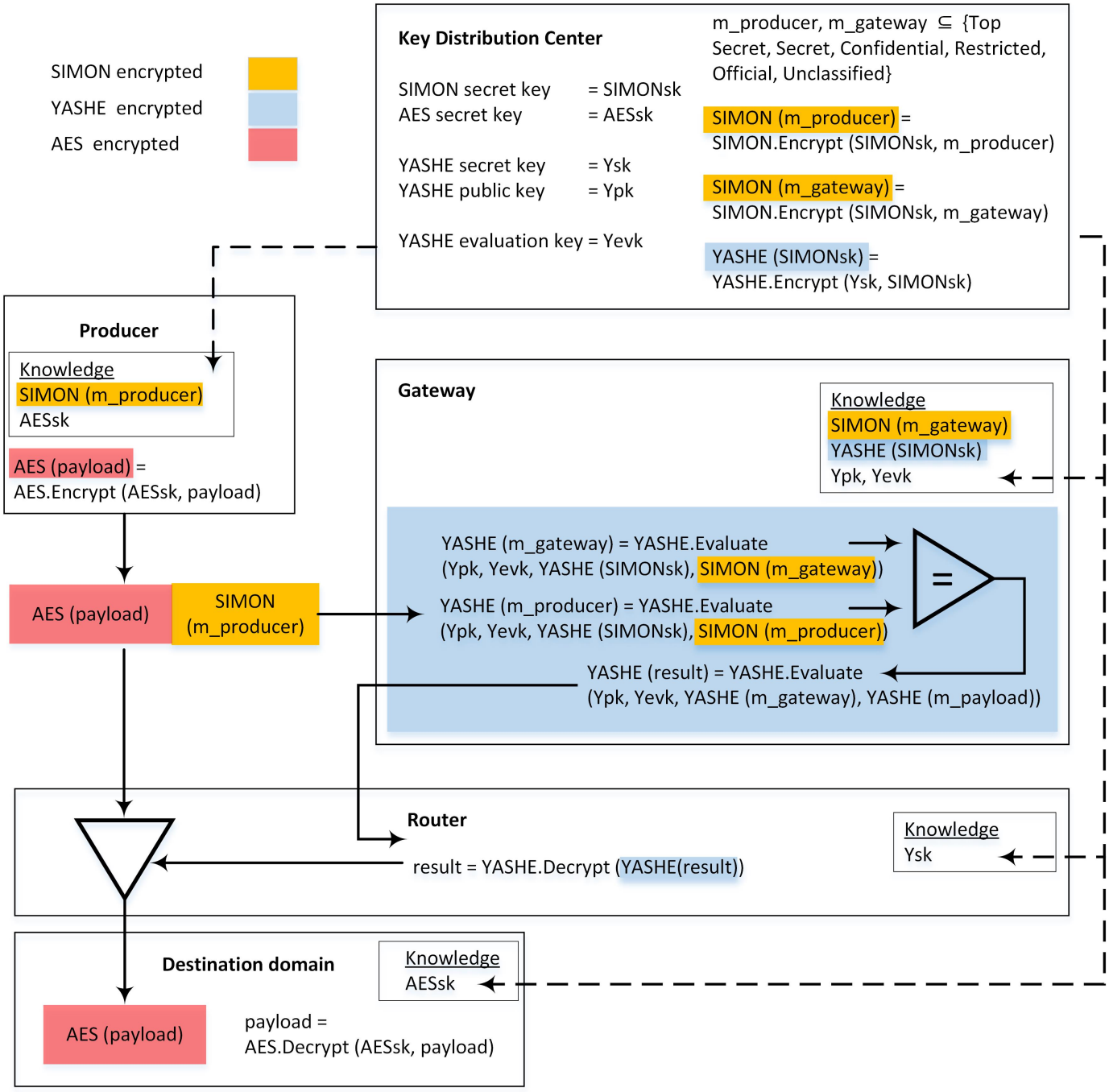

Fig. 4. Application Design Details. Three distinct encryption schemes are color coded as follows. Red: the AES is used to encrypt (large) payload of each producer (only one shown). Yellow: the lightweight scheme SIMON is first used to encrypt (small) metadata classification values which define routing constraints, and then used by YASHE within homomorphic operations. Blue: a homomorphic scheme implemented by YASHE which aids gateways and routers functionalities (only one gateway and router shown). First, using homomorphic evaluation the gateway produces inputs to homomorphic comparison of metadata to routing conditions. Second, YASHE encrypted comparison result is passed on to the router (only one shown) which has capacity to extract the final result determining whether to forward or drop the AES encrypted payload.

TABLE IV
MEMORY PROFILE RESULTS OF CDS APPLICATION

YASHE configuration	secret key `Ysk` (kB)	public key `Ypk` (kB)	evaluation key `Yevk` (kB)	encrypted `SIMONsk` (kB)	encrypted metadata (kB)	encrypted result (kB)
α	1,792	1,792	50,176	114,688	57,344	1,792
β	4,608	4,608	165,888	331,776	221,184	4,608
γ	4,608	4,608	165,888	442,368	221,184	4,608
δ	14,336	14,336	802,816	1,835,010	917,504	14,336

TABLE V
AVERAGE PERFORMANCE PROFILE RESULTS OF CDS APPLICATION

YASHE configuration	encrypt SIMON key (s)	SIMON key expansion (s)	encrypt metadata (s)	homomorphic SIMON decryption (s)	homomorphic metadata evaluation (s)	decrypt result (s)
α	5.4	112.8	3.2	2433.0	612.0	1.0
β	19.0	419.4	13.3	12149.1	1966.0	3.0
γ	21.4	514.3	13.3	12150.6	1971.2	3.0
δ	78.6	1958.7	48.2	64367.6	8079.1	8.9

upon. Hardware acceleration has been explored previously in research [21]–[23]. The next step would be to explore and apply these optimizations to this work.

REFERENCES

[1] Committee on National Security Systems. Committee on National Security Systems (CNSS) Glossary. (4009):160, 2015.

[2] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. Cryptology ePrint Archive, Report 2013/075, 2013. https://eprint.iacr.org/2013/075.

[3] Ray Beaulieu, Douglas Shors, Jason Smith, and Stefan Treatman-clark. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.

[4] Hoda A. Alkhzaimi and Martin M. Lauridsen. Cryptanalysis of the SIMON family of block ciphers. Cryptology ePrint Archive, Report 2013/543, 2013. https://eprint.iacr.org/2013/543.

[5] NIST. Security and Privacy Controls for Federal Information Systems and Organizations Security and Privacy Controls for Federal Information Systems and Organizations. *Sp-800-53Ar4*, pages 400+, 2014.

[6] Ronald L Rivest and Michael L Dertouzos. On Data Banks and Privacy Homomorphisms. 1978.

[7] Craig Gentry and Dan Boneh. *A Fully Homomorphic Encryption Scheme*, volume 20. Stanford University Stanford, 2009.

[8] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[9] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 7, 2010.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*, 43(2):831–871, 2014.

[12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[13] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Advances in cryptology–crypto 2012*, pages 868–886. Springer, 2012.

[14] Damien Stehlé and Ron Steinfeld. Making NTRUEncrypt and NTRUSign as Secure as Standard Worst-Case Problems over Ideal Lattices. *Eurocrypt*, 6632:27–47, 2011.

[15] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[16] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[17] Craig Gentry and Nigel P Smart. *Homomorphic Evaluation of the AES Circuit*. 2015.

[18] Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes FV and YASHE. In *International Conference on Cryptology in Africa*, pages 318–335. Springer, 2014.

[19] W. Hart, F. Johansson, and S. Pancratz. FLINT: Fast Library for Number Theory, 2013. Version 2.4.0, http://flintlib.org.

[20] The GNU Multiple Precision Arithmetic Library, 2016. Version 6.1.2, http://gmplib.org.

[21] Michael J Foster. Accelerating Homomorphic Encryption in the Cloud Environment through High-Level Synthesis and Reconfigurable Resources. 2017.

[22] Wei Dai, Yarkın Doröz, and Berk Sunar. Accelerating NTRU based homomorphic encryption using GPUs. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1–6. IEEE, 2014.

[23] Erdinç Öztürk, Yarkin Doröz, Berk Sunar, and Erkay Savas. Accelerating somewhat homomorphic evaluation using fpgas. *IACR Cryptology ePrint Archive*, 2015:294, 2015.