Towards an Energy-Efficient Hash-based Message Authentication Code (HMAC)

Cesar E. Castellon
School of Engineering
University of North Florida
Jacksonville - USA
n01453427@unf.edu

Swapnoneel Roy
School of Computing
University of North Florida
Jacksonville - USA
s.roy@unf.edu

O. Patrick Kreidl
School of Engineering
University of North Florida
Jacksonville - USA
patrick.kreidl@unf.edu

Ayan Dutta
School of Computing
University of North Florida
Jacksonville - USA
a.dutta@unf.edu

Ladislau Bölöni
Department of Computer Science
University of Central Florida
Orlando - USA
ladislau.boloni@ucf.edu

***Abstract*—Hash-based message authentication code (HMAC) involves a secret cryptographic key and an underlying cryptographic hash function. HMAC is used to simultaneously verify both integrity and authenticity of messages and, in turn, plays a significant role in secure communication protocols e.g., Transport Layer Security (TLS). The high energy consumption of HMAC is well-known as is the trade-off between security, energy consumption, and performance. Previous research in reducing energy consumption in HMAC has approached the problem primarily at the system software level (e.g. scheduling algorithms). This paper attempts to reduce energy consumption in HMAC by applying an energy-reducing algorithmic engineering technique to the underlying hash function of HMAC, as a means to preserve the promised security benefits. Using pyRAPL, a python library to measure computational energy, we experiment with both the standard and energy-reduced implementations of HMAC for different input sizes (in bytes). Our results show up to 17% reduction in energy consumption by HMAC, while preserving function. Such energy savings in HMAC, by virtue of HMAC's prevalent use in existing network protocols, extrapolate to lighter-weight network operations with respect to total energy consumption.**

***Index Terms*—HMAC, Energy, Security.**

I. Introduction

Hashed-based Message Authentication Code (HMAC) is a well known machine authentication code extensively used in different cybersecurity applications. Advocates for such uses cite the HMAC's ability to provide both integrity and authentication. [1], [2], [3], [4], [5]. One particularly promoted use case is the *Transport Layer Security (TLS)* [1], [3], [6], [7], [8], which is the standard, widely deployed protocol for securing client-server communications over the Internet. The Transport Layer Security (TLS) protocol, sometimes referred to, the Secure Sockets Layer (SSL) protocol, is a stateful, connection-oriented, client-server protocol. TLS is the most widely deployed communications security protocol on the Internet, providing confidentiality, integrity and authentication for parties in communication.

Other usages of HMAC include error correction codes [9], remote attestation [10], VANETs [5], [11], [12], [13], security and privacy in systems [14], and Internet of Things (IoT) [15]. A balance exists in the amount of security, performance, and energy consumption one wants to achieve. In general, increasing security generally causes additional energy consumption while decreasing performance. Finding perfect balance between energy consumption, performance and security level — becomes a challenge in HMAC based applications, where supply of energy is often very limited [16].

To be viable for an application that values autonomy for greater lengths of time, any system must be configured to make more efficient use of energy. Disabling HMAC integrity and authentication will certainly save energy, but also weaken security: it is in such contexts that the exploration of ways to reduce energy consumption of HMAC alone can be of tremendous practical significance.

A. Related Work

Energy efficiency in computation is a widely studied topic, with numerous points-of-view: hardware-specific platforms, operating systems, hypervisors and containers [17]; software development and security [18]; and algorithms [19], [20]. Energy measurements are sometimes obtained by uniquely instrumented equipment [21], while other times can leverage hardware providers' Application Programmer Interfaces (APIs) in which firmware counters are recalled to provide near real-time information e.g., Running Average Power Limit (RAPL) technology [22]. Energy-efficient (*greener*) HMAC implementations have been actively researched as a means to improve overall energy efficiency in secured systems [23].

This research study is supported in part by NSF CPS #1932300, NSF CPS #1931767, Fidelity National Financial Distinguished Professorship in CIS #0583-5504-51, and Cyber Florida #220408 grants.

978-1-6654-6550-2/22/$31.00 ©2022 IEEE

These research have mostly focused to optimize energy efficiency of HMAC in a variety of scenarios (e.g., scheduling [24], [25], [26], system software [27], [28], [29], hardware implementation [30], [31], and hypervisors [32]). However, all of these works treat HMAC as a black box and to the best of our knowledge, no research exist in the literature that deals directly with the hash algorithms of HMAC and attempts to engineer them to reduce energy consumption in HMAC.

B. Our Scope and Contributions

We study the extent to which the underlying hash function (SHA256), a principal element of HMAC, can be made more energy efficient. Our approach employs an energy-reducing algorithmic engineering technique, based upon an Energy Complexity Model (ECM) proposed by Roy et al. [19], [20], on the SHA256 encryption algorithm, which is central to HMAC. Using pyRAPL, a python library to measure an executable's Runtime Average Power Limit, we experiment with both the standard and energy-reduced implementations of HMAC for input sizes (in bytes) that are commonly seen within applications using HMAC. Our results show significant reductions in energy consumption, up to 13.5% but on average around 12.7% across the tested input sizes. At present, it is only a conjecture that reduced energy consumption in the HMAC module itself extrapolates to comparable reduction of an application using HMAC on the whole. In any case, to the best of our knowledge our work is the first to address energy optimization of HMAC by engineering the implementation of one of its component algorithms (SHA256). Moreover, the proposed energy-reducing technique is similarly applicable to other key elements of a secured system, potentially affording even "greener" secured application systems than implied by only the HMAC results obtained thus far.

This paper builds on top of research done in [33]. While [33] experiments on energy efficiency of Merkle Trees in Blockchain, this work experiments on energy efficiency of HMAC, a different algorithm. Also, part of this work was done while the first author pursued his Masters in EE as presented in this M.S thesis [34].

II. Methodology

An Energy Complexity Model (ECM) [19], [20] has been applied to the underlying SHA256 function of HMAC. We first describe how the general HMAC function works, followed by a brief discussion of the ECM and its application to the underlying SHA256 of HMAC.

A. HMAC Message Digest Generation

As mentioned before, HMAC implements both integrity checking and authentication of messages using cryptographic hash functions. Any hash function (e.g. MD5, SHA128, SHA256, etc.) can be used in HMAC combined with a shared secret key. HMAC's strength cryptography-wise is dependent on the strength of its underlying hash function [35], [36].

Fig. 1 shows a graphic representation of a simple HMAC message digest generation. The input to HMAC is a message M containing $\ell-1$ blocks ($Y(1)\cdots Y(\ell-1)$), each of size b. A signature S_i is concatenated to the left of M before it is input to the underlying hash function (e.g. SHA256) to produce a temporary message digest MD'. MD' is further concatenated with output signature $S_o = K^+ \oplus PAD$, which is then hashed again using the underlying hash (e.g. SHA256) to produce MD, the final message digest.

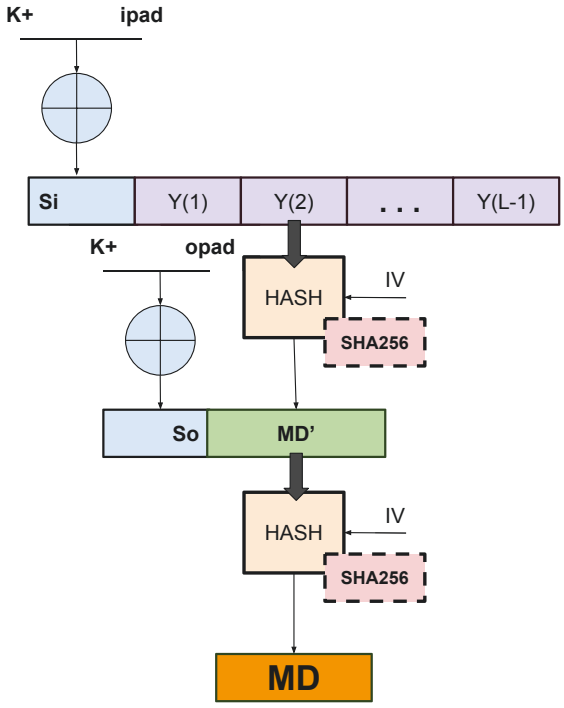

Figure 1: Basic HMAC generation

For a recap, in Fig. 1, HASH stands for the hash function function (SHA256), M is the input message, S_i and S_o are respectively the input and output signatures, $Y(i)$ is the i^{th} block of M, i ranges from $[1, \ell)$, ℓ is the number of blocks in M. K is the secret key used for the hash. IV is an initial vector (constant values used by SHA256).

B. The Energy Complexity Model (ECM)

The Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) is the reference architecture for the energy complexity model (ECM) [19], which has applied to HMAC in this work. As illustrated in Fig. 2, the main memory of DDR is divided into *banks*, containing a fixed number of *chunks*[1].

Allocation of data happens in each bank over chunks. Additionally, every bank contains a special chunk called the *sense amplifier*. For any data access, the chunk containing the data to be accessed has to be brought inside the corresponding bank's sense amplifier. Each sense amplifier can house one

[1]The term "block" is used in DDR specifications, but we use the term "chunk" to avoid confusion within our HMAC context.

chunk at a given time, so the present chunk has to be returned to its bank before a new one can be brought in for the next access. At a given time, therefore, only one chunk of a given bank can be accessed; however, chunks of different banks can be accessed in parallel (within each bank's own sense amplifier). Hence, for a P bank DDR memory (e.g., $P = 4$ in Fig. 2), at any point of time we can access P chunks. The sense amplifier is called per-bank cache in DDR3 version of the DDR architecture.

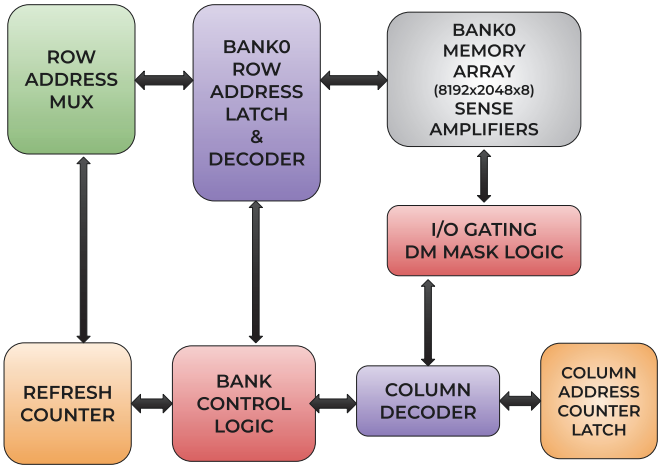

Figure 2: Internal DDR SDRAM memory chip block diagram.

The P banks of a given DDR3 SDRAM resource is denoted by $M_1, M_2, \ldots, M_P$ by the ECM. There are multiple chunks of size-B (in bytes) and a cache C_i respectively in each bank M_i. Fig. 3 illustrates an example with $P = 4$ banks similar to the case in Fig. 2 with each bank having only four chunks. Labels in numbers $1, 2, \ldots, 16$ were assigned to the chunks. Given the constraint in DDR that a single chunk may be put inside a given cache C_i at any time, examples of completely serial execution are the access patterns $(1, 2, 3, 4)$ or $(5, 6, 7, 8)$, while $(1, 5, 9, 13)$ or $(3, 8, 10, 13)$ are examples of completely parallel execution. The authors of [19] discovered (1) accessing same number of chunks (sequentially or in parallel) account for very similar amount of power consumption and (2) execution time of an algorithm is reduced significantly when chunks are accessed in parallel than when chunks are accessed sequentially. Since energy consumption of an algorithm is dependent on both time and power, it was implied that energy consumption in any algorithm is potentially reduced by parallelizing chunk accesses during the execution of that algorithm. Formally, as derived by Roy *et al.* [19], the energy consumption (in Joules) of an algorithm $\mathcal{A}$ with execution time τ, assuming a P-bank DDR3 architecture with B bytes per chunk, is given by

$$E(\mathcal{A}) = \tau + (P \times B)/I \tag{1}$$

where the so-called *parallelization index* is denoted by I, which is essentially the number of parallel block accesses across memory banks per P block accesses made by $\mathcal{A}$ on the whole. In other words, an algorithm's potential for energy reduction is inversely proportional to the degree it can be engineered for parallelization of its memory accesses, according to ECM.

Figure 3: ECM for DDR3 Resource with $P = 4$ Banks

C. Engineering Hash Calculations Using ECM

The energy consumption of the underlying hash algorithm (SHA256) of HMAC has been reduced by engineering it based on ECM in this work. First, how any algorithm $\mathcal{A}$ can be parallelized based on ECM is described. Then we illustrate how SHA256, the underlying hash algorithm for HMAC is engineered for parallelization based on ECM.

1) Parallelizing any algorithm: For algorithm $\mathcal{A}$, the most common access sequence of $\mathcal{A}$ on execution for a given input is first identified. The vector formed by this access sequence is then engineered to achieve the desired level of parallelism by framing a logical mapping over chunks of memory that store data accessed by $\mathcal{A}$. Physical location of the input (chunks) is static in the memory and is controlled by the memory controller of DDR. But order of access over chunks is different for different levels of parallelization. Different page table vectors $\mathbf{V}$ is framed each time for implementing different levels of parallelization of access over physical chunks. $\mathbf{V}$ defines the ordering of access among chunks (Fig. 4).

For 1-way access, the page table vector $\mathbf{V}$ has the pattern $(1, 2, 3, 4, \ldots)$ and for 4-way access it has the pattern $(1, 5, 9, 13, \ldots)$. A function is then created to map the pattern of the page table vector $\mathbf{V}$ to the original physical locations of the input. Algorithm 1 shows the function to create an ordering among the chunks. The ordering is based on the way we want to access the chunks (P-way would mean full parallel access). The page table is populated by picking chunks with *jumps*. For P-way access, jumps of P are selected that ensure the consecutive chunk accesses lie in P different banks. Going by the above example, for $P = 1$, jumps of 1 ensure that 4 consecutive chunk accesses lie in the same bank (bank 1 of Fig. 3). On the other hand, for $P = 4$, jumps of 4 ensures that 4 consecutive chunk access lie in 4 different banks (banks 1 through 4 of Fig. 3).

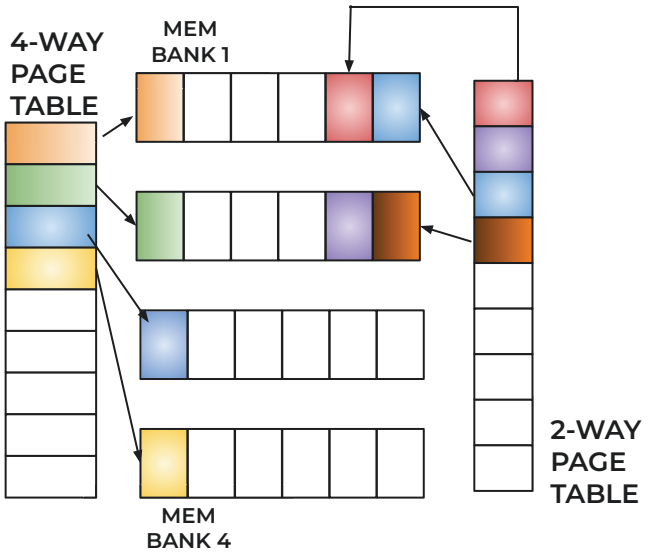

Figure 4: Memory Layout ($P = 4$) and Role of Page Tables

Input: Page table vector $\mathbf{V}$, jump amount *jump*.
factor $= 0$;
for $i = 0$ *to* $\frac{N}{B} - 1$ **do**
  if $i > 1$ *and* $(i \times jump) \mod \frac{N}{B} = 0$ **then**
    factor $=$ *factor* $+1$;
  end
  $\mathbf{V}_i = (i \times jump + factor) \mod \frac{N}{B}$;
end

Algorithm 1: Create a Page Table for N Chunks

2) Parallelizing SHA256: As illustrated in Fig. 1, HMAC generates the final message digest (MD) by applying SHA256 twice. The SHA256 algorithm partitions its input into fixed size message blocks, presented in sequence to separate compression functions, as shown in Fig. 5. This block sequence is identified in correspondence with the access pattern of the SHA256 algorithm, which we subject to engineering based on the ECM. The SHA256 input vector (see Fig. 5), is pre-processed into another vector by applying Algorithm 1. The mapping is then stored in a page table to be used in subsequent hash calculations. An example of this operation for 16 blocks and a parallelization index (jump) of 4 is shown in Fig. **??**.

Fig. 6 shows the outcome of engineering the SHA256 algorithm based on ECM. In our experimentation, an 8-bank DDR3 SDRAM is used and the parallelization index is set to $I = 8$. This essentially means that for any set of eight consecutive block access in SHA256, we created a virtual mapping using techniques described in [20] to ensure that each size-8 access occurs across all eight banks.

Theorem 1. *The engineered SHA256 algorithm has the same computational complexity as the original SHA256 algorithm.*

Proof. SHA256 has a computational complexity of $\Theta(N)$, where N is the number of blocks in Fig. 5 [37]. Algorithm 1 has a computational complexity of $\Theta(N)$ since the for loop of line 2 executes exactly $\frac{N}{B}$ times. Therefore applying Algorithm 1 to SHA256 as illustrated in Fig. **??** does not change the overall computational complexity of SHA256. □

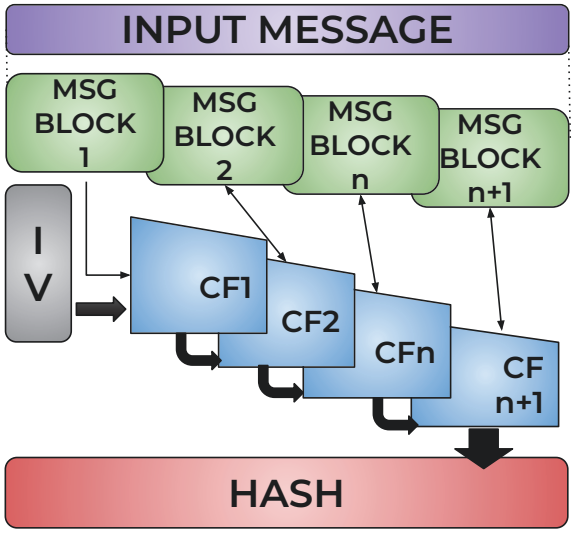

Figure 5: The SHA256 Algorithm

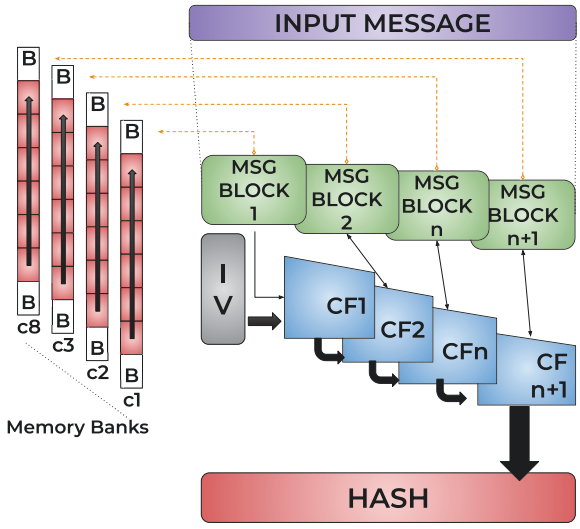

Figure 6: ECM-Enhanced SHA256

III. Experiments

This section describes experiments performed to measure energy efficiency of HMAC out of the engineering illustrated in the previous section. The ECM required a hardware with a DDR RAM architecture. According to the ECM, maximum energy efficiency is attained by the parallelization index set to the number of memory banks, which depends upon the DDR version: 4 for DDR2, 8 for DDR3 and 16 for DDR4 and higher. We used a machine with a 64-bit dual-core processor

(Intel i5-2410M @ 2900MHz with cache size L2 256KB and L3 3072KB), running Linux Mint version 19.3 with a 8GB DD3 RAM and 500GB SSD storage. Also, pyRAPL, a software toolkit, was used to measure the host machine's energy footprint along the execution of Python code for comparing energy consumption between HMAC with standard and ECM-enhanced of the underlying SHA256. pyRAPL is built upon Intel's Running Average Power Limit (RAPL) technology that estimates a CPU's power consumption; depending on the hardware and operating system configurations, pyRAPL can measure energy consumption of the following CPU domains: CPU socket, GPU, and DRAM [22].

A. Implementation Details and Setup

Standard and ECM-enhanced versions of the SHA256 algorithm have been implemented in two different C language programs, these are called from a master Python program via the `ctypes` module) as an external command. This permits the use of Python pyRAPL to measure energy events having at the same time the low-level memory control to implement the ECM-enhanced SHA256 functionality.

Our experiments simulated the HMAC calculation with Python code that runs one complete round of Message Digest generation having pyRAPL methods invoked yielding in a single energy measurement per event. Since measurement implementation is subject to noise we have invoked 1000 repetitions for the process and report the average energy (mean and deviation). Our experiments also vary the input size (i.e., the message size) to the HMAC calculations, choosing 64, 128, 256, 384, 512, 768, and 1024 bytes motivated by having standard message's not to exceed traditional MTU limit of 1500 bytes and selecting standard steps of size increase.

B. Results and Discussion

Our experimental setup features two implementations of HMAC calculations, the standard one (which we label by "O" as it uses the standard SHA256) and the engineered one using ECM (which we label by "E" as it uses the energy efficient SHA256), as well as seven different input sizes.

Per implementation and per input size, our experimental Python script leverages the pyRAPL toolkit to measure the average energy (mean and deviation over 1000 trials) of simulated HMAC calculations. Fig. 7 summarizes the seven average energy measurements in a bar chart, per input size comparing the Standard HMAC (O) and the Enhanced HMAC (E) average energy (in μJoules). We observe that the ECM-enhanced implementation consistently consumes less energy that the standard implementation.

The first set (Fig. 7) compares average energy measurements between the energy efficient ('E') and the original ('O') HMAC implementations, starting from input size 64 byte to 1024 bytes. To summarize, the ECM-engineered HMAC shows an average energy consumption increment of around 100% with increase in input size from 64 byte to 1024 bytes.

The standard ('O', non-enhanced) model in comparison showed an average energy consumption increment of around

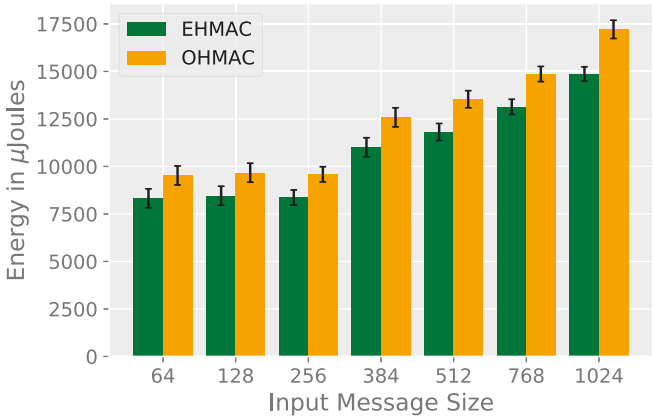

Figure 7: Comparison of Average Energy Consumption in HMAC per Message Size (with 1-sigma standard deviation over 1000 trials)

75% over the same input sizes. For example, with input sizes of 64,128, and 256 bytes, the average energy consumption are 8380 and 9600 $\mu Joules$ for 'E' and 'O' respectively, while for input size 1024 bytes, they are 14866 and 17213 $\mu Joules$ respectively. It can be concluded that memory parallelism implementation in SHA256 based on ECM has an overhead impact on energy consumption of HMAC. This is in line with the ECM model proposed in [19].

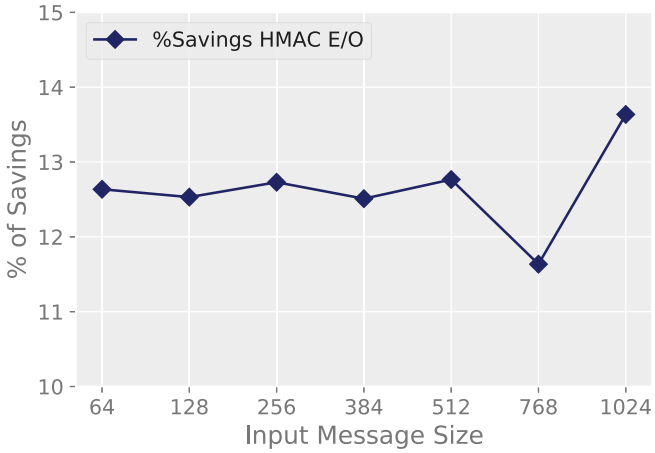

Figure 8: Percentage of Energy Saving per process in HMAC per Message Size

Fig. 8 presents average energy savings on more relative terms, namely as a percent reduction achieved by the ECM-enhanced implementation over the standard implementation of HMAC over all seven input sizes. The energy savings for the smaller input sizes range between $12-13\%$, while the energy savings for the larger input sizes range between 13% and 14%,. As observed, the 768B input renders a savings lower than the average, yet it is still around the margins over 10% for single operation.

IV. Conclusion

This work considers reducing the energy consumption of HMAC by engineering the underlying hashing algorithm (SHA256), based on the Energy Complexity Model (ECM) [19]. The ECM-enhanced implementation was compared to the standard implementation via experimental energy measurements with various input sizes of practical significance. The results show up to about 14% energy savings for input sizes typically used by HMAC. It remains conjecture that reduced energy consumption in HMAC extrapolates to comparable reduction to applications using HMAC. Future work can also assess the energy savings in different applications of HMAC mentioned in Section I.

References

[1] L. Beringer, A. Petcher, Q. Y. Katherine, and A. W. Appel, "Verified correctness and security of {OpenSSL}{HMAC}," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 207–221.

[2] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 479–498.

[3] T. Ylonen, "Ssh–secure login connections over the internet," in *Proceedings of the 6th USENIX Security Symposium*, vol. 37, 1996, pp. 40–52.

[4] J. Lettner, B. Kollenda, A. Homescu, P. Larsen, F. Schuster, L. Davi, A.-R. Sadeghi, T. Holz, and M. Franz, "{Subversive-C}: Abusing and protecting dynamic message dispatch," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 209–221.

[5] G. Sang, J. Chen, Y. Liu, H. Wu, Y. Zhou, and S. Jiang, "Pacm: Privacy-preserving authentication scheme with on-chain certificate management for vanets," *IEEE Transactions on Network and Service Management*, 2022.

[6] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting {SSL/TLS} implementations: New bleichenbacher side channels and attacks," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 733–748.

[7] N. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt, "On the security of {RC4} in {TLS}," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 305–320.

[8] I. Ristic, *Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2013.

[9] T. Lawrence, F. Li, I. Ali, M. Y. Kpiebaareh, C. R. Haruna, and T. Christopher, "An hmac-based authentication scheme for network coding with support for error correction and rogue node identification," *Journal of Systems Architecture*, vol. 116, p. 102051, 2021.

[10] L. Wilke, J. Wichelmann, F. Sieck, and T. Eisenbarth, "undeserved trust: Exploiting permutation-agnostic remote attestation," in *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2021, pp. 456–466.

[11] M. A. Ferrag and L. Shu, "The performance evaluation of blockchain-based security and privacy systems for the internet of things: A tutorial," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17 236–17 260, 2021.

[12] S. S. Moni and D. Manivannan, "Crease: Certificateless and reused-pseudonym based authentication scheme for enabling security and privacy in vanets," *Internet of Things*, vol. 20, p. 100605, 2022.

[13] V. O. Nyangaresi, A. J. Rodrigues, and N. K. Taha, "Mutual authentication protocol for secure vanet data exchanges," in *International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures*. Springer, 2021, pp. 58–76.

[14] A. A. Khaliq, A. Anjum, A. B. Ajmal, J. L. Webber, A. Mehbodniya, and S. Khan, "A secure and privacy preserved parking recommender system using elliptic curve c cryptography and local differential privacy," *IEEE Access*, 2022.

[15] L. C. Thungon, M. Hussain *et al.*, "Performance evaluation of zero knowledge and hmac-based authentication in fog-based internet of things," in *Proceedings of the International Conference on Computing and Communication Systems*. Springer, 2021, pp. 633–641.

[16] S. Litzinger, O. Körber, and J. Keller, "Reducing energy consumption of hmac applications on heterogeneous platforms," in *2019 International Conference on High Performance Computing & Simulation (HPCS)*, 2019, pp. 152–158.

[17] J. Westin, "Evaluation of energy consumption in virtualization environments: proof of concept using containers," 2017.

[18] P. D. Harish, "Towards designing energy-efficient secure hashes," Master's thesis, University of North Florida, 2015.

[19] S. Roy, A. Rudra, and A. Verma, "An energy complexity model for algorithms," in *Proc. of the 4th Conf. on Innovations in Theoretical Computer Science*, 2013, pp. 283–304.

[20] ——, "Energy aware algorithmic engineering," in *Proc. 22nd IEEE Int. Symp. on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, 2014, pp. 321–330.

[21] C. A. Roma and M. A. Hasan, "Energy consumption analysis of XRP validator," in *Proc. of 2020 IEEE Int. Conf. on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–3.

[22] M. Santos, J. Saraiva, Z. Porkoláb, and D. Krupp, "Energy consumption measurement of C/C++ programs using Clang tooling," in *Proc. of 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications*, 2017.

[23] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the energy consumption of security protocols," in *Proceedings of the 2003 international symposium on Low power electronics and design*, 2003, pp. 30–35.

[24] L. S. P. Annabel and K. Murugan, "An energy efficient wakeup schedule and power management algorithm for wireless sensor networks," in *2012 International Conference on Recent Trends in Information Technology*. IEEE, 2012, pp. 314–319.

[25] J. A. Haqbeen, T. Ito, M. Arifuzzaman, and T. Otsuka, "Traffic adaptive hybrid mac with qos driven energy efficiency for wsns through joint dynamic scheduling mode," in *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*. IEEE, 2018, pp. 3–9.

[26] S. Yu, Q. Yugui, and L. Zhiting, "Hmac: An energy efficient mac protocol for wireless sensor networks," *Journal of University of Science and Technology of China*, vol. 40, no. 10, p. 1054, 2010.

[27] A. Apavatjrut, W. Znaidi, A. Fraboulet, C. Goursaud, K. Jaffrès-Runser, C. Lauradoux, and M. Minier, "Energy efficient authentication strategies for network coding," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 10, pp. 1086–1107, 2012.

[28] P. Kalaivaani and A. Rajeswari, "An energy efficient analysis of s-mac and h-mac protocols for wireless sensor networks," *International Journal of Computer Networks & Communications*, vol. 5, no. 2, p. 83, 2013.

[29] H. Wang, X. Zhang, and A. Khokhar, "Wsn05-6: An energy-efficient low-latency mac protocol for wireless sensor networks," in *IEEE Globecom 2006*. IEEE, 2006, pp. 1–5.

[30] M. Juliato and C. Gebotys, "Fpga implementation of an hmac processor based on the sha-2 family of hash functions," *University of Waterloo, Tech. Rep*, 2011.

[31] C. Haas, S. Munz, J. Wilke, and A. Hergenröder, "Evaluating energy-efficiency of hardware-based security mechanisms," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE, 2013, pp. 560–565.

[32] W. Itani, A. Chehab, and A. Kayssi, "Energy-efficient platform-as-a-service security provisioning in the cloud," in *2011 International Conference on Energy Aware Computing*. IEEE, 2011, pp. 1–6.

[33] C. Castellon, S. Roy, P. Kreidl, A. Dutta, and L. Bölöni, "Energy efficient merkle trees for blockchains," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2021, pp. 1093–1099.

[34] C. Castellon, "Energy considerations in blockchain-enabled applications," Master's thesis, University of North Florida, 2021.

[35] H. Krawczyk, M. Bellare, and R. Canetti, "Rfc2104: Hmac: Keyed-hashing for message authentication," USA, 1997.

[36] R. Canetti, M. Bellare, and H. Krawczyk, "Keyed hash functions and message authentication," in *Proceedings of Crypto*, vol. 96.

[37] D. Rachmawati, J. Tarigan, and A. Ginting, "A comparative study of message digest 5 (md5) and sha256 algorithm," in *Journal of Physics: Conference Series*, vol. 978, no. 1. IOP Publishing, 2018, p. 012116.