

PROCEEDINGS OF
INTERNATIONAL CONFERENCE ON INTELLIGENT SYSTEMS AND NEW
APPLICATIONS

<https://proceedings.icisna.org/>

2nd International Conference on Intelligent Systems and New Applications (ICISNA'24), Liverpool, April 26-28, 2024.

Enhancing Privacy and Security in IoT Environments through Secure Multiparty Computation

Rik van de Haterd¹, Mohammed Elhajj²

¹*EEMCS, University of Twente, Netherlands*
r.vandehaterd@student.utwente.nl, ORCID: 0009-0007-4252-2033

²*EEMCS, University of Twente, Netherlands*
m.elhajj@utwente.nl, ORCID: 0000-0002-5549-4357

Abstract— With the increasing influence of IoT devices in our daily lives, secure data-sharing is becoming ever more important. Sensors and other devices are communicating vast amounts of possibly unencrypted data, which poses a significant privacy concern. To tackle this problem, this research implements two Partially Homomorphic Encryption (PHE) schemes, RSA and the Paillier cryptosystem, to perform Secure Multiparty Computation (SMPC) in the resource-constrained IoT environment. The environment consists of a laptop connected to an Arduino Uno through a serial connection. The RSA-based SMPC protocol has an average completion time of 2007ms. However, due to the inability to use padding, RSA lacks semantic security. Conversely, the Paillier-based protocol is semantically secure but cannot complete the encryption due to dynamic memory issues. Even if resolved, the estimated encryption time exceeds 103.3 minutes. Despite the potential of SMPC in IoT environments for secure data handling, the results from this research suggest that directly implementing PHE schemes on Arduino is not practical based on the observed limitations.

Keywords— Secure Multiparty Computation, Internet of Things, Resource-constrained device, Partially Homomorphic Encryption

I. INTRODUCTION

The Internet of Things (IoT) has steadily grown over the last couple of decades[1]. With this growth, IoT devices are increasingly integral in daily life. These interconnected devices share sensitive data, making the confidentiality and integrity of information a substantial concern in the field of IoT security [2]. Furthermore, due to the nature of the IoT, there are additional constraints, such as limited resources, diversity of standards, and network vulnerabilities[3]. In this resource-constrained environment, challenges regarding privacy during data aggregation and transport encryption emerge[4][5]. Traditional cryptographic algorithms encounter challenges when applied to IoT scenarios due to inherent resource limitations such as power constraints, limited battery capacity, and the need for

real-time execution[6]. Thus, this research focuses on addressing the critical privacy and security challenges in resource-constrained IoT environments by applying Secure Multiparty Computation (SMPC) techniques.

A. Motivation

The field of SMPC has flourished with the rise of cloud computing and data-sharing in IoT environments. However, the literature on the practical application of SMPC protocols on resource-constrained devices is lacking. Prior research has mostly been focused on developing secure protocols and testing them in virtual environments, rather than on devices like an Arduino. Therefore, this research aims to contribute to the literature by gaining insights into the practical application of SMPC in resource-constrained devices within the IoT environment.

B. Background

In the context of SMPC, a group of parties aims to collectively compute a function based on their private inputs whilst ensuring only the output is disclosed[7]. The problem was formally introduced as the Millionaires' Problem by Andrew Yao (1982)[8] and describes two millionaires who want to know which one of them is richer, without disclosing their actual wealth. Yao's millionaire problem is a Boolean predicate but was proven to be computationally feasible for any function in 1986, again by Yao[9]. One of the possible building blocks of SMPC is Homomorphic encryption (HE). HE allows for computations to be performed on encrypted data without the need of having to first decrypt it. The three main types of HE are:

- **Partially Homomorphic Encryption (PHE):** Partially Homomorphic Encryption is the most computationally practical form of HE but is also the most mathematically limited. PHE schemes only support the evaluation of one

gate and the two operations of additive homomorphic encryption or multiplicative homomorphic encryption.

- **Somewhat Homomorphic Encryption (SHE):** Somewhat Homomorphic Encryption can evaluate two types of gates but only for a subset of operations.
- **Fully Homomorphic Encryption (FHE):** Fully Homomorphic Encryption allows the evaluation of arbitrary circuits made up of multiple gate types of unbounded depth. FHE is the strongest type of HE but is also the most computationally heavy.

Middleware in IoT manages communication between hardware and applications, handling data collection, storage, analysis, processing, and forwarding to consumers[5]. However, it poses security risks as a high-value target for attackers. If not under the owner's control, the middleware may be untrustworthy, and outsourcing data to third parties raises concerns about data usage control, exemplified by potential privacy breaches in aggregating data from different sensors[7]. Additionally, IoT devices' resource-constrained nature limits their ability to perform complex data encryption compared to non-IoT devices[10]. Through extensive literature review, Secure Multi-Party Computation (SMPC) emerges as a promising solution for secure data-sharing in the resource-constrained IoT environment.

C. Research questions

This research will investigate the following research questions (RQ).

- (1) How can a customized cryptographic protocol based on SMPC be implemented to ensure confidentiality and integrity of data shared among IoT devices, considering the specific constraints and requirements of IoT environments?
- (2) What is the practicality, efficiency, and security of the implemented SMPC-based cryptographic protocol in real-world IoT use cases, and how do they compare with existing security solutions in terms of computation time, resource utilization, and power consumption?

D. Structure

Section II provides an overview of what is presented in the literature regarding the usage of SMPC to enhance privacy and security in IoT applications. Then section III documents the hardware and software configurations, implemented PHE schemes, and research metrics. The section concludes with an in-depth review of the testing environment and design choices. Next, section IV details the performance- and security analysis of the implemented PHE schemes. The performance analysis delves into computation time, power consumption, and memory usage, whilst the security analysis examines the security model, cryptographic key size and semantic security. Additionally, RQ1 and RQ2 are answered at the end of the section. The paper concludes with section V, which summarizes key findings and outlines potential avenues for future research.

II. RELATED WORK

The field of SMPC has been rapidly evolving over the years and many efficient protocols have been published. With the rise of more efficient protocols, the application of SMPC in IoT is also becoming more relevant. A study by authors in [11] offers two optimized SMPC protocols for the Internet-like setting. Their protocols are based on multiparty garbled circuits as described in the paper of Beaver, Micali and Rogaway[12]. Furthermore, they provide a protocol based on the paper from Ben-Or, Goldwasser and Wigderson[13] that incorporates the free-XOR technique as well as reducing round complexity. With these optimizations they reduced overall runtime from 355 seconds to 25 and the online time from 330 seconds to <0.5 seconds compared to the 1987 paper from Goldreich, Micali and Wigderson[13]. Another paper by authors in [15] looks at optimizing Shamir-Secret Sharing[16] (SSS) to achieve privacy-preserving data aggregation in the IoT environment. They optimized the sharing phase of SSS by lowering the degree of the polynomial. With this optimization, they managed to make their aggregation time 6 and 9 times faster as well as using 7 and 10 times less radio-on time in their testing environments (Flocklab and DCube respectively). Cloud computing has a lot of potential for SMPC in the IoT setting. Authors in [10] discussed the possibility of using the cloud to utilize Homomorphic encryption in the IoT environment. In their architecture, the IoT device encrypts their data and sends it to a cloud. Then a 'data user' can query for the data, which is then computed by the cloud and delivered. They concluded that resource-constrained IoT devices might not be able to afford the costs of Fully Homomorphic encryption. However, they did find that Partial Homomorphic encryption has great potential for resource-constrained IoT devices. The cloud can be used by IoT devices to offload their complex computations. Authors in [17] proposed a protocol based on a modified SSS[16] scheme where the source node can outsource its computation to a set of workers. The paper showed that the proposed protocol met the requirements of full anonymity, confidentiality, verification, and computation synchronization while also shifting most of the computational costs to the workers, ensuring correct results. These papers show that SMPC has great potential to be used in the IoT environment.

III. METHODOLOGY

This section details the methodology employed in the research. It begins with a description of the hardware and software setup, providing the foundation for algorithm implementation. Then, the research metrics are presented, followed by an outline of the chosen PHE schemes. The section concludes with insights into the practical implementation and testing environments used in the research.

A. Hardware and software setup

- 1) *Tools:* The hardware configuration includes an Arduino Uno with an ATmega328 microcontroller (32K bytes of flash memory), a MacBook Air powered by a 1.3 GHz Dual-Core Intel Core i5 CPU (121 GB flash memory), and an AVHzY USB-Meter C3 designed for power consumption measurement. The software environments comprise of the

Arduino IDE 2.2.1, utilizing C/C++ and the microRSA and arduino-cryptographic-library libraries. Pycharm CE 2021.3 supports Python 3.11, integrating libraries such as OpenSSL and python-paillier.

B. Algorithms

This paper focuses on implementing two PHE encryption schemes to cover both mathematical operations. RSA for its multiplicative property and the Paillier cryptosystem for its additive property.

1) *RSA*: RSA is a public-key cryptosystem that relies on the practical difficulty of integer factorization[21] specifically for the product of two large prime numbers. RSA encryption is described by the following equation:

$$E(m) = m^e \text{ mod } n \quad (1)$$

In this equation, m represents the plaintext message, e is the public exponent, and n is the product of two large prime numbers, ensuring the security of the encryption scheme. The multiplicative homomorphic property of RSA is described by the equation:

$$\begin{aligned} E(m_1) * E(m_2) &= m_1^e m_2^e \text{ mod } n \\ &= (m_1 * m_2)^e \text{ mod } n \\ &= E(m_1) * E(m_2) \end{aligned}$$

This property highlights the ability to perform multiplication on the cyphertexts directly without the need for decryption.

2) *Paillier cryptosystem*: The Paillier cryptosystem is a probabilistic public-key encryption scheme based on a discrete logarithm trapdoor modulo a large integer that is hard to factor[22]. Paillier encryption is described by the following equation:

$$E(m) = g^{m r^n} \text{ mod } n^2 \quad (2)$$

In this equation, m represents the plaintext message, g is a generator of the multiplicative group modulo n^2 , and r is a random value chosen from the set $\{0, \dots, n-1\}$. The Paillier cryptosystem exhibits an additive homomorphic property, described by the equation:

$$\begin{aligned} E(m_1) * E(m_2) &= (g^{m_1 r_1^n})(g^{m_2 r_2^n}) \text{ mod } n^2 \\ &= (g^{m_1 + m_2})(r_1 r_2)^n \text{ mod } n^2 \\ &= E(m_1) + E(m_2) \end{aligned}$$

This property enables the computation of the encryption of the sum of corresponding cyphertexts directly.

C. Research metrics

This research benchmarks implemented protocols based on performance and security metrics. Performance evaluation includes memory usage, power consumption, and computation time measured in bytes, volts, milliwatts, milliamperes, and milliseconds respectively. Security assessment focuses on analyzing security models, cryptographic key sizes, and semantic security to ensure robustness against adversarial behavior and encryption security.

D. Implementation

The laptop and Arduino Uno are connected via USB for Serial port communication, facilitating reliable data exchange. The laptop handles key generation tasks with a 1024-bit key size, using the Python subprocess library and OpenSSL for RSA operations and the python-paillier[20] library for Paillier encryption. The Arduino Uno employs a custom 'bignum8' structure for numbers up to 512 bytes in Paillier encryption. Implementation on the Arduino includes the RNG library for secure random number generation, modular arithmetic, and modular exponentiation to enhance computational efficiency for a robust and secure cryptographic system.

1) *RSA environments*: For RSA encryption two different environments interact with each other, the Python- and Arduino environments. The Python environment generates a 1024-bit RSA key pair in PEM format by utilizing OpenSSL. Then the modulus n (Formula 2) is extracted from the public key and sent to the Arduino to be used in encryption. After the encryption is finished, the script receives the cyphertext from the Arduino and can calculate the final result by multiplying the two cyphertexts. Finally, the resulting cyphertext can be decrypted to check the correctness of the computation.

The Arduino environment utilizes a modified version of the microRSA library to ensure compatibility with an IoT environment. The Arduino receives the modulus from the laptop and encrypts its plaintext message. The following cyphertext is then sent to the laptop for further calculations.

2) *Paillier environments*: The Paillier encryption uses the same environment structure as the RSA encryption. The Python environment generates a 1024-bit Paillier key pair using the python-paillier library. The script then extracts the modulus n and squared modulus n^2 (Formula 2) After extraction, the two values are sent over the serial connection. From here the same steps apply as in the RSA environment. The script waits for the encrypted Arduino message, computes the sum, and decrypts the final result.

The Arduino environment uses an extended version of the microRSA library to implement the missing mathematical operations. Once the Arduino receives n and n^2 it can start encrypting its plaintext message. Finally, the Arduino sends the cyphertext back to the laptop.

IV. RESULTS

This section includes an examination of the SMPC protocols, shedding light on the performance and security aspects of the PHE schemes. With a primary focus on RSA and the Paillier cryptosystem, this section delves into the computation time, power consumption, and memory usage analyses. The performance analysis is followed by a security analysis which goes into the security model, cryptographic key size and semantic security. Finally, the section concludes with answering the research questions posed in the introduction.

A. Performance Analysis

1) *Computation time*: This section explores the computation time of cryptographic operations, providing a comprehensive perspective on minimum, maximum, and average runtimes based on 100 sampled instances. Table 3 presents the runtime analysis of the SMPC protocol utilizing RSA, covering key generation (Keygen), Python message encryption (EncryptP), laptop decryption (Decrypt), and Arduino message encryption (EncryptA). Notably, the overall protocol runtime is significantly influenced by the Arduino's encryption time, averaging 1807ms, approximately 100 times slower than its Python counterpart. Key generation shows notable variability in both minimum and maximum times, but its infrequency is unlikely to significantly impact the overall protocol performance.

TABLE 1: COMPUTATION TIME RSA

RSA	Min	Average	Max
Keygen	191ms	256ms	578ms
EncryptA	1792ms	1807ms	1834ms
EncryptP	12ms	17ms	32ms
Decrypt	13ms	19ms	29ms
Total	2008ms	2099ms	2473ms

Table 4 provides a comprehensive breakdown of the computation time for the Paillier-based SMPC protocol on the laptop. Key generation (Keygen), encryption of Python messages (Encrypt), and decryption (Decrypt) are the key components analyzed. Key generation, similar to the RSA protocol, displays notable variability in both minimum and maximum times, but, as mentioned earlier, this variability is not expected to significantly impact the overall performance of the protocol due to the infrequency of key generation.

TABLE 2: COMPUTATION TIME PAILLIER PYTHON

Paillier Python	Min	Average	Max
Keygen	108ms	356ms	1054ms
Encrypt	24ms	32ms	51ms
Decrypt	7ms	10ms	24ms
Total	139ms	398ms	1129ms

TABLE 3: COMPUTATION TIME ARDUINO MODULO

Modulo in bytes	128	256	512
128	1ms	922ms	2766ms
256	0ms	2ms	3668ms

TABLE 4: COMPUTATION TIME ARDUINO MULTIPLICATION

Multiply in bytes	64	128	256
64	25ms	46ms	92ms
128	46ms	99ms	183ms
256	92ms	183ms	367ms

Due to memory issues, the Paillier encryption on the Arduino faced constraints, impeding the acquisition of runtime data. An estimate, based on the computational demands of the most intensive task, modular exponentiation involving 256- and 512-byte numbers, can be derived. With 1024 cycles and additional executions, the modular exponentiation step comprises a combined 1536 multiplications and modulo operations. Using data from Tables 5 and 6, the estimate suggests a total runtime of approximately 103.3 minutes for the function.

2) *Power consumption*: The power analysis, outlined in Tables 7 and 8, sheds light on the energy demands of cryptographic algorithms implemented on the Arduino Uno. At 5.1 volts, the Arduino exhibited an average amperage of 19.0mA when idle. During cryptographic operations (RSA, Multiply, and Modulate), the amperage only slightly increased to an average of 20.4mA, peaking at 21.2mA during modulation. These values remain well below the Arduino Uno's maximum power draw capacity of at least 200.0mA[23]. The correlation between recorded amperage and wattage, validated through Ohm's Law ($W = V * I$), underscores the consistency of data across cryptographic functions. Consequently, the power consumption of these PHE schemes on the Arduino Uno is deemed negligible, indicating that power constraints should not pose significant challenges in implementing or discussing these cryptographic algorithms on the Arduino platform.

TABLE 5: AMPERE USAGE ARDUINO

Function	Min	Average	Max
RSA	20.1mA	20.4mA	20.8mA
Multiply	20.2mA	20.3mA	20.5mA
Modulate	20.1mA	20.4mA	21.2mA

TABLE 6: WATTAGE ARDUINO

Function	Min	Average	Max
RSA	102.9mW	103.8mW	105.8mW
Multiply	104.9mW	106.3mW	110.2mW
Modulate	103.7mW	104.2mW	106.2mW

3) *Memory Usage*: In the evaluation of memory usage for the RSA and Paillier implementations on the Arduino, Table 9 provides a comprehensive overview of Flash and dynamic memory consumption before runtime. Both implementations stay well below the maximum thresholds of 32K and 2K respectively, with the RSA utilizing 6076 bytes of Flash and 330 bytes of dynamic memory, while the Paillier implementation uses 10176 bytes of Flash and 493 bytes of dynamic memory. Since the Arduino IDE does not support real-time memory analysis, the subsequent calculations are estimates based on line-by-line code analysis.

TABLE 7: MEMORY BEFORE RUNTIME IN BYTES

Memory type	Flash	Dynamic
-------------	-------	---------

RSA	6076	330
Paillier	10176	493

To gain insights into the dynamic memory usage during runtime, the memory flow over specific events is depicted in Figures 2 and 3. For the RSA implementation (Figure 1), the memory usage grows during key generation, message generation, and conversion to the custom bignum8 structure. The encryption phase introduces two temporary variables, reaching an estimated highest memory usage of 1226 bytes. The RSA implementation does not encounter memory issues. The Paillier encryption on the other hand (Figure 2) involves higher memory demands due to the reception of two large variables, n (128 bytes) and n^2 (256 bytes). Additionally, during modular exponentiation, the memory usage variable of r peaks at 512 bytes. The estimated highest memory usage for the Paillier implementation is 1773 bytes, occurring during the modular exponentiation and the final cyphertext calculation. Despite these estimates not reaching or exceeding the Arduino's maximum memory capacity of 2048 bytes, the Paillier implementation experiences program breakdowns and outputs zero values, indicating potential memory-related issues.

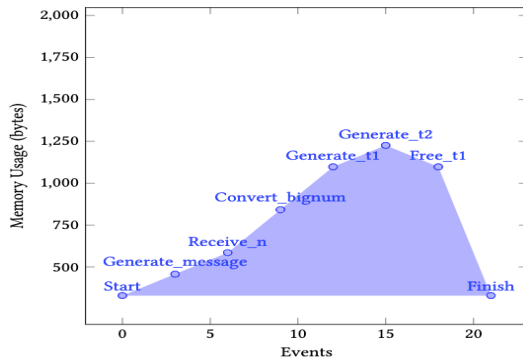


Figure 1: Estimated memory usage RSA

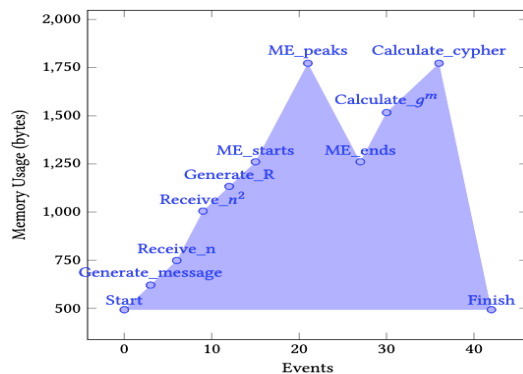


Figure 2: Estimated memory usage Paillier

B. Security analysis

1) *Security model*: The security of a protocol is meaningful only when discussed under a specific security model, as the capabilities of the adversary define the security requirements. This paper assumes a Semi-honest Adversary Model as the security model during the execution of the protocol. In this model, corrupted parties must execute the

protocol correctly. The adversary can obtain information on corrupted parties but will attempt to use this information discreetly. Protocols that achieve this level of security prevent the leakage of information between collaborating parties.

2) *Key size*: NIST revised its recommendation for RSA key lengths in 2015, now advising a minimum of 2048 bits[24]. This update supersedes the consensus which advocated for a minimum key size of 1024 bits. Consequently, the use of a 1024-bit key in encryption is discouraged for safeguarding sensitive or critical data. However, if the key's lifespan or the protected data spans only days or weeks, the necessity for employing a key resistant to years-long attacks diminishes. The choice of key size and its associated security considerations in the context of the IoT environment would be contingent on specific usage. It's noteworthy that 1024-bit keys represent the upper limit of what an Arduino Uno, without additional memory extensions, can effectively manage.

3) *Semantic security*: The intrinsic homomorphic property of RSA makes the algorithm susceptible to semantic insecurity in the absence of proper padding. The lack of semantic security renders the encryption vulnerable to potential attacks such as Chosen Plaintext and Message Replay. The introduction of padding however destroys the homomorphic property, making performing SMPC impossible. The Paillier cryptosystem does offer semantic security against chosen-plaintext attacks. The successful distinction of the challenge cyphertext boils down to the ability to make decisions about composite residuosity, a task considered computationally intractable under the assumption of decisional composite residuosity. However, despite providing semantic security against chosen-plaintext attacks, the system exhibits malleability due to its homomorphic properties. Thus, it does not achieve the highest level of semantic security, lacking protection against adaptive chosen-cyphertext attacks.

C. Answering RQ1

A customized SMPC protocol based on PHE schemes can ensure the confidentiality and integrity of data through the homomorphic properties of the algorithms. PHE schemes are the least computationally complex type of HE and are therefore a logical consideration for the resource-constrained IoT environment. With these homomorphic properties, one can compute results based on cyphertexts rather than on plaintext. This ensures that the context of the numbers is lost and the parties involved do not learn any information about the other participating parties. This protocol can be further extended by implementing secret-sharing techniques like SSS or Oblivious Transfer when distributing keys or cyphertexts.

D. Answering RQ2

The practicality of implementing the RSA and Paillier PHE schemes on a resource-constrained device is rather lacking, as efficiency and security perform insufficiently for the IoT

environment. For the RSA algorithm, encryption takes on average 1807ms which is too long for the often real-time data needed in the IoT environment. Furthermore, the RSA algorithm cannot make use of padding which makes it semantically insecure. This results in the fact that parties can gain knowledge about other participating parties, which goes against SMPC requirements. The Paillier encryption on the other hand is semantically secure but cannot finish encryption due to running out of dynamic memory. Even if the Paillier algorithm could finish an encryption, this would take at least 103,3 minutes. Again, this is way too long for any practical application in the IoT environment. The security status of the chosen 1024-bit key size is contingent on the usage of the protocol, therefore no concrete conclusion can be made regarding key size other than the fact that the protocols should not be used to store critical data. Power consumption is the only characteristic that both algorithms perform sufficiently. However, this does not weigh up against the lack of computational efficiency and security.

V. CONCLUSIONS

Privacy and security are pivotal in data-sharing within the IoT environment. Despite this, there is a noticeable gap in the literature regarding the practical implementation of SMPC protocols on resource-constrained devices within the IoT [25]. This research details the implementation of two PHE-based SMPC protocols on the resource-constrained Arduino Uno. One protocol is built upon the RSA cryptosystem, and the other is based on the Paillier cryptosystem. The protocols underwent benchmarking and security analysis. For the performance, metrics important to resource-constrained devices such as memory usage, power consumption, and computation time were tested [26]. Whereas the security analysis delves into the security model, semantic security, and cryptographic key size. Based on the results of this research I conclude that SMPC shows great potential for the resource-constrained IoT environment. However, it is not practical to directly implement Partially Homomorphic Encryption schemes on a resource-constrained device like the Arduino Uno with the goal of SMPC. I would discourage attempts to optimize RSA and Paillier encryption on the Arduino and instead focus on offloading complex computations and encryptions through the use of cloud computing and secure data offloading. With these techniques, one could utilize even more robust encryption algorithms more efficiently than the Arduino can.

REFERENCES

- [1] Adam Thierer and Andrea Castillo. Projecting the growth and economic impact of the internet of things. George Mason University, Mercatus Center, June, 15, 2015.
- [2] Lo'ai Tawalbeh, Fadi Muheidat, Mais Tawalbeh, and Muhannad Quwaider. Iot privacy and security: Challenges and solutions. *Applied Sciences*, 10(12), 2020.
- [3] Asma Haroon, Munam Ali Shah, Yousra Asim, Wajecha Naeem, Muhammad Kamran, and Qaisar Javaid. Constraints in the iot: the world in 2020 and beyond. *International Journal of Advanced Computer Science and Applications*, 7(11), 2016.
- [4] Elhajj, M., Jradi, H., Chamoun, M. and Fadlallah, A., 2022, June. Lasii: Lightweight authentication scheme using iot in iot platforms. In 2022 20th Mediterranean Communication and Computer Networking Conference (MedComNet) (pp. 74-83). IEEE.
- [5] Marcel von Maltitz and Georg Carle. Leveraging secure multiparty computation in the internet of things. In Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '18, page 508–510, New York, NY, USA, 2018. Association for Computing Machinery.
- [6] Saurabh Singh, Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–18, 2017.
- [7] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu an Tan. Secure multi-party computation: Theory, practice and applications. *Information Sciences*, 476:357–372, 2019.
- [8] Andrew C. Yao. Protocols for secure computations. In 23rd Annual Symposium on Foundations of Computer Science (sfc82), pages 160–164, 1982.
- [9] Andrew Chi-Chih Yao. How to generate and exchange secrets. In 27th Annual Symposium on Foundations of Computer Science (sfc86), pages 162–167, 1986.
- [10] Wang Ren, Xin Tong, Jing Du, Na Wang, Shan Cang Li, Geyong Min, Zhiwei Zhao, and Ali Kashif Bashir. Privacy-preserving using homomorphic encryption in mobile iot systems. *Computer Communications*, 165:105–111, 2021.
- [11] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. *Cryptology ePrint Archive*, Paper 2016/1066, 2016. <https://eprint.iacr.org/2016/1066>.
- [12] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In Symposium on the Theory of Computing, 1990.
- [13] E. W. van der Wal and M. El-Hajj, "Securing Networks of IoT Devices With Digital Twins and Automated Adversary Emulation," *2022 26th International Computer Science and Engineering Conference (ICSEC)*, Sakon Nakhon, Thailand, 2022, pp. 241-246.
- [14] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.
- [15] Himanshu Goyal and Sudipta Saha. Multi-party computation in iot for privacy-preservation. In 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS), pages 1280–1281, 2022.
- [16] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.
- [17] Oladayo Olufemi Olakanmi and Kehinde Oluwasesan Odeyemi. Trust-aware and incentive-based offloading scheme for secure multi-party computation in internet of things. *Internet of Things*, 19:100527, 2022.
- [18] qqqlab. microrsa. <https://github.com/qqqlab/microRSA>, 2020.
- [19] Rhys Weatherley. Arduino cryptography library. <https://rweather.github.io/arduino/libs/crypto.html>, 2023.
- [20] CSIRO's Data61. Python paillier library. <https://github.com/data61/python-paillier>, 2013.
- [21] Kefa Rabah. Review of methods for integer factorization applied to cryptography. *Journal of applied Sciences*, 6(1):458–481, 2006.
- [22] Dario Catalano, Rosario Gennaro, Nick Howgrave-Graham, and Phong Q. Nguyen. Paillier's cryptosystem revisited. In Proceedings of the 8th ACM Conference on Computer and Communications Security, CCS '01, page 206–214, New York, NY, USA, 2001. Association for Computing Machinery.
- [23] Microchip. megaavr® data sheet. <https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf>, 2019.
- [24] Elaine Barker and Quynh Dang. Recommendation for key management part 3: Application-specific key management guidance, 2015-01-22 2015.
- [25] El-Hajj, M., Fadlallah, A., Chamoun, M., & Serhrouchni, A. (2020, January). Secure PUF: Physically unclonable function based on arbiter with enhanced resistance against machine learning (ML) attacks. In SEIA'2019 Conference Proceedings, Lulu. com (p. 216).
- [26] Fneish, Zein Al-Abidin Mohammad, Mohammad El-Hajj, and Khoulood Samrouth. "Survey on iot multi-factor authentication protocols: A systematic literature review." In 2023 11th International Symposium on Digital Forensics and Security (ISDFS), pp. 1-7. IEEE, 2023