Using Garbled Circuit for Secure Brokering

Lotfi ben Othmane[1] and Noor Ahmed[2]

[1] Iowa State University, Ames, IA
[2] Air Force Research Lab/RISD, Rome, NY USA

Abstract. Many domains such as military and agriculture collect data from sensors and other trusted applications, process them, and distribute them according to predetermined rules to interested entities. These systems need to trust the data broker, which could be hosted in a public environment. This paper assesses the use of garbled circuits technique to protect the collected time-sensitive data and filtering and distribution policy from suspicious brokers. It reports about the design of a secure brokering protocol and preliminary performance evaluation of the implementation.

1 Introduction

Current information systems collect data from sensors and other trusted applications, process them, and distribute them according to predetermined rules within given time-constraints to interested entities. Figure 1 shows an example of such systems, an area monitoring system. The parties of the scenario are: data broker B, data producers P_i and data consumers C_i, where index i refers to the ith participant. Let P_1 be a drone that takes photos and sends them along with its ID P_1 and its latitude and longitude position (L_1, L_2) to a broker B. Let the consumer C_1 sends a request to the broker to get photos collected in the area (L_A^1, L_B^2) and (L_B^1, L_B^2), where (L^i, L^j) are the latitude and longitude of area limit point A.

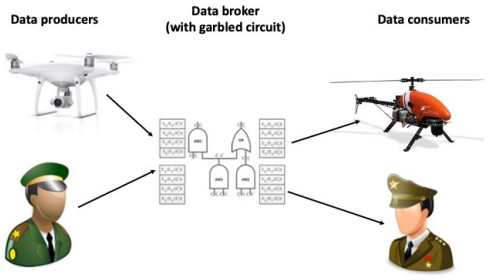

Fig. 1. Secure evaluation of access and dissemination policy on untrusted environments.

The broker needs often to be located close to the time-sensitive data producer entities and may need to operate in a suspicious or malicious environment, e.g., to serve a ground monitoring mission. To address this requirement, the broker needs to operate such that it applies the filtering and distribution policy without understanding the data and the policy.

The threat model is as follow: The broker could intercept the execution of the filtering policy and/or apply arbitrary function on the data received from the producers and send the results to the consumers. The producers are trusted to send "good" data. The consumers are trusted to send "good" filtering policy predicate. The broker, however, must know nothing about the application of the filtering policy.

The issue of secure enforcement of access control policies in cross-domain system is applicable to wide-range of domains including e-commerce, supply-chain, and cyber-physical systems. The common approach is to use Attribute-Based Encryption (ABE) [1,2]. However, this approach requires mechanisms to distribute and manage the public/private keys for each participant, which makes it difficult to apply in data dissemination scenarios, where the data senders and receivers do not know each others at the setup phase. In addition, the performance of the proposed homomorphic schemes that address the problem makes them not practical.

An alternative approach is to use the Garbled Circuit (GC) [3] technique to ensure the secure execution of brokering algorithm. GC is not commonly used because most of the constructions require its size to grow exponentially to the size of the input, includes big number of encryption/decryption operations, and requires extensive exchange between the two parties contributing to evaluate the circuit. For instance, Carter et al. evaluated the execution time of garbled circuit for a set of algorithms including Dijkstra with 50 vertices and 100 vertices, which take about 6 hours and 47 hours respectively [4]. Recent work focus on developing non-interactive evaluation of garbled circuit [5,6], which allows to reduce the cost of communication between the two parties participating on evaluating the garbled circuit. These work focused on the security of the algorithms and did not provide empirical performance evaluations.

The contributions of this paper are:

- Design and implementation of a GC-based brokering protocol for protecting input data and filtering policy from suspicious brokers for time-sensitive information sharing application.
- Preliminary security analysis and performance evaluations of the protocol.

The paper is organized as follows. Section 2 discusses related work. Section 3 describes the protocol of using Garbled Circuit (GC) for brokering data that we propose. Section 4 describes the prototype of the protocol *SFEDataShare*. Section 5 discusses the evaluation of SFEDataShare and section 6 concludes the paper.

2 Related work

We discuss in this section examples of work on attribute-based encryption of circuits, implementations of GC and non-interactive constructions of GC.

Attribute-based encryption (ABE) for circuits. In ABE scheme, a ciphertext is associated with public indexes of the attributes *ind*, a message m, a private/public key, and Boolean predicate P over the attributes *ind* [7]. Several ABE schemes have been proposed. For instance, Gorbunov et al. [1] proposed a construction that uses the Learning With Errors (LWE) assumption [8], which allows a receiver to decrypt only data that comply with its predicate and no more. The scheme uses the public key to encrypt the wires inputs and allows to compute the ciphertext and the circuit representing the predicates over the attributes at the same time. In this scheme, the parameters and ciphertext grow linearly with the depth of the circuit. Garg et al. proposed another construction of ABE scheme but using multilinear maps [2].

Bonah et al. [9] proposed an attribute based encryption systems that relies on LWE problems [8] and supports functions representing arithmetic circuits. This scheme allows to transform an encryption c of message m with attribute vector x into an encryption under public key $< f(x), f >$. It can then decrypt the result only if $f(x) = 0$, that is, the policy is valid. The performance of the scheme is expected to be not practical for time critical data sharing in military applications.

Implementations of garbled circuit. Fairplay is the first known implementation of garbled circuit for two-party computation system [10]. Several implementations have been published since then, which provide better performance and security. We report about the main ones in the following.

Kreuter et al. implemented the garbled circuit protocol for the malicious model [11] in the two-party context using multiple-copies to detect malicious GC evaluator. They were able to execute secure AES in 1.4 seconds and 4095-bit edit distance circuit in about 8.2 hours on 256 cores for each of the two parties. We note, however, that about 40% of the execution time is spent on the communication between the parties.

Bellare et al. [12] implemented the primitives of GC using AES to generate the tokens for the wire signals. They also improved their implementation to benefit from the free-xor [13] and garbled row reduction [14] performance improvement techniques. The performance of the implementation [15] was assessed using AES circuit and showed excellent performance: 637 μs for the circuit garbling primitive and 264 μs for the evaluation primitive.

Almeida et al. [16] proposed a software stack for Secure Function Evaluation (SFE) that consists of a verified compiler that translates C programs into Boolean circuits, a verified implementation of Yao's SFE protocol based on GCs and oblivious transfer [17], and a transparent application integration and communications. The security of the protocol implementation is formally verified using EasyCrypt [18], a tool-assisted framework for building high-confidence cryptographic proofs.

Non-interactive GC. Bellare et al. [19] proposed a security formulation of GC as four algorithms: garble the circuit, encrypt input, evaluate GC, and decrypt the results. In addition, they formulated three security properties for GCs: (1) privacy – the host shouldn't learn anything impermissible beyond that which is revealed by knowing just the final output, (2) obliviousness–leaks nothing about the original function and input beyond known information such as the topology of the circuit and (3) authenticity–inability of adversary to create an output from the GC and input that is different from its output. The authors provided one garbling scheme that comply with the security property and could be instantiated using AES.

Goldwasser et al. proposed a construction of an encrypted decryption function that decrypts an encrypted message without a key [20]–they embed the key in the GC. The construction uses GC of the decryption algorithm and LWE-based ABE [1]. The construction uses the encrypted message to identify recursively the labels to be used in evaluating the GC and outputs the plain-text message.

3 Design of the Secure Brokering Protocol

Garbled Circuit (GC) has been proposed as an approach for secure function evaluation [3,21]. A short description of garbled circuit follows. Let C be a circuit that implements function f. A garbled circuit of C is an "encryption" of C such that the decryption of its output using input x is equivalent to the output of function f using the same input x. The steps of the algorithm are [3,22,23]:

1. $Gb(1\lambda, f) \rightarrow (F, e, d)$ – The garbling algorithm Gb takes in the security parameter λ and a circuit f, and returns a garbled circuit F, encoding information e, and decoding information d.
2. $En(e, x) \rightarrow X$ – The encoding algorithm En takes in the encoding information e and an input x, and returns a garbled input X.
3. $Ev(F, X) \rightarrow Y$ – The evaluation algorithm Ev takes in the garbled circuit F and the garbled input X, and returns a garbled output Y.
4. $De(d, Y) \rightarrow y$ – The decoding algorithm De takes in the decoding information d and the garbled output Y, and returns the plain text output y.

Figure 2 shows the GC of a simple circuit. In this construction, we associate to each of the wires two labels, e.g., wire 2 is associated with label C for signal value 1 and label D for value 0. We also provide the truth table of each of the garbled gates and associate the labels for outputs 0 and 1, where the values 0 indicates that the predicate is valid and 1 indicates otherwise. We observe that in the basic construction, the encoding algorithm generates labels for the input wires signals and associates them to the wire input and the decoding algorithm generates labels for output wires signals and associates them to the output values [19].

We propose the use of GC [3,21] for the secure evaluation of policy functions in malicious environments. Figure 3 depicts the adapted protocol that we propose

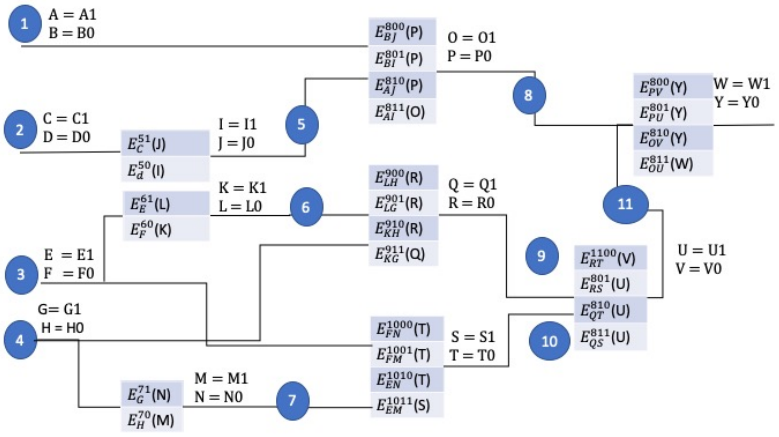

Fig. 2. An Illustration of Garbled circuit for a simple circuit

for secure brokering. The protocol uses the four roles: (1) the *Owner*, (2) the *Produce*, (3) the *Broker*, and (4) the *Consumer*. The *Owner* generates the input labels using the key SK_i and the GC using the key SK_v and sends the the key SK_i to the *Producer*, and the GC and the key SK_v to the *broker*. In addition, the *Owner* constructs a decoding map that matches the possible output of the circuit to the possible output labels of the GC, which it sends to the *Consumer*. The *Producer* uses the key SK_i to generate the input labels associated with its input, which it sends to the *Broker*. The *Broker* evaluates the GC using the input labels it receives from the *Producer* and the key SK_v and sends the output labels of the GC to the *Consumer*. The *Consumer* uses the output map to decode the output labels it receives from the *Broker* and generates the plain-text data of the circuit.

We adapt the cryptographic primitives of Bellare et al. [19] to generate and evaluate the GC for our use case as illustrated in the Procedure `Gb` and Procedure `Ev` respectively. Procedure `Gb` generates a GC from function f. The first loop sets up the GC in line 2 and 3, and the second loop computes the q garbled tables for each gate g in $P[g, b, a]$ in lines 4 to 5, where the set garbled tables F is given to the broker. Figure 4 shows the partial content of GC at the Owner, including the parameters of the circuit, and a part of the garbled table.

For the evaluation of the GC, The procedure `Ev` evaluates the GC F using encrypted input X by extracting the labels X_a and X_b of the inputs to each of the gates and uses them to compute the output labels. Then, the output labels are given to the consumer to decrypt them.

A short security analysis of our GC weaknesses is as follows: Each of the producers receives a secret key to encrypt its data then send to the broker. An attacker who compromises a producer can impersonate the producer by using the producers' secret key to send data of their choice to the broker. Another

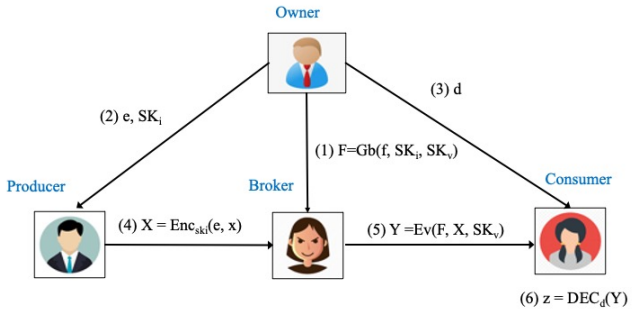

Fig. 3. Secret-key-based Protocol for secure data dissemination. Note: f: function, F: garbled function, e: encoding alg., SKi: secret key for generating input labels, SKv: secret key for garbling circuit C representing f, input labels x: producer input, X: encryption of x, y: output, Y: encrypted output, and d: map of possible circuit outputs to possible GC output labels.

Procedure Gb(1^k,f) [19]

$(n, m, q, A', B', G) \leftarrow f$
for $i \in \{1, ..., n+q-m\}$ **do**
  $t \twoheadleftarrow \{0,1\}, X_i^0 \twoheadleftarrow \{0,1\}^{k-1} t, X_i^1 \twoheadleftarrow \{0,1\}^{k-1} \bar{t}$
for $(g, i, j) \in n+1, ..., n+q \times 0, 1 \times 0, 1$ **do**
  $a \leftarrow A'(g), b \leftarrow B'(g)\ A \leftarrow X_a^i, a \leftarrow lsb(A), B \leftarrow X_b^j, b \leftarrow lsb(B)$
  $T \leftarrow g \,\|a\|\, b, P\,[g, a, b] \leftarrow E_{A,B}^T(X_g^{G^{(i,j)}})$
$F \leftarrow (n, m, q, A', B', P)$
$e \leftarrow (X_1^0, X_1^1, ..., X_n^0, X_n^1)$
$d \leftarrow (X_{n+q-m+1}^0, X_{n+q-m+1}^1, ..., X_{n+q}^0, X_{n+q}^1$
return (F, e, d)

weakness of the first version of our solution is that the size of a predicate could be small enough for a malicious broker to bypass it by mapping the input and the output. We consider to address these limitations in our future work.

4 Description of `SFEDataShare` Framework

We developed a framework for secure brokering, referred to as *SFEDataShare*. We use *JustGarble* [24,19,12,15], an open source code base for garbling and evaluating Boolean circuits, as the core component of the framework. *JustGarble* [15] uses two AES keys in constructing the GC. The first key is used to generate the input labels that correspond to possible inputs to the circuit. The second key is used to generate the GC wires labels and to evaluate the GC using the provided input labels.

Procedure Ev(F, X) [19]

$(n, m, q, A', B', P) \leftarrow F, (X_1, ..., X_n) \leftarrow x$
for $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A'(g), b \leftarrow B'(g)\ A \leftarrow X_a, a \leftarrow lsb(A), B \leftarrow X_b, b \leftarrow lsb(B)$
  $T \leftarrow g \,\|a\|\, b, Xg \leftarrow \mathbb{D}^T_{A,B}(P\,[g, a, b])$
return $x_{n+q-m+1}, ..., x_{n+q}$

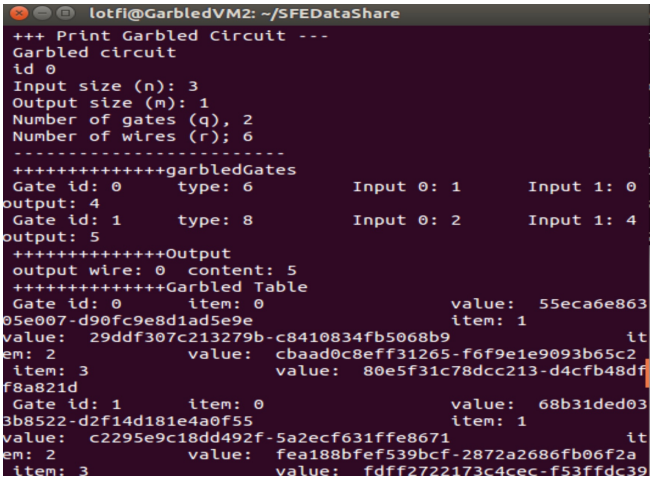

Fig. 4. Data Owner GC Output

`justGarble` allocates memory for the GC dynamically. The operating system allocates a big memory size for the GC as the memory is fragmented. For instance, the OS reserves effectively 2 GB memory (considering memory fragmentation) for a garbled circuit of 51 input wires while the effective needed memory is 9544 bytes, excluding the memory for storing the input labels. This problem prevented us from easily serializing the GC. Thus, we developed serialization and deserialization functions for the GC, input labels, and output labels. The serialization consists of performing a memory copy of the data in its data type form to a reserved memory of type `void`. The deserialization consists of performing a memory copy of the received data of type `array of voids` to the variables addresses and performing type casting of the variables or using function `__mmload_si128()` [25] to store the data into `__m128i` variables. This solves the problem as we are consuming only the needed memory zones.

Figure 4 shows part of the log of the *Owner* module. The log shows the input/output labels of a circuit composed of three input wires and two gates XOR(.) and AND(.).

Table 1. Performance of the protocol.

# of inputs wires	# of gates	GC Size (in bytes)	Garbling time (in ticks)	Evaluation time (in ticks)
3	2	520	80584	569
10	9	1836	81076	1137
15	14	2776	82134	901
22	21	4092	171404 (0.017 seconds)	1990 (0.2 msec)

*10.000 ticks correspond to 1 millisecond

5 Performance Evaluation

We evaluated the framework using a MacBook Air laptop of 1.1 GHz Dual-Core Intel i3 processor with 16 GB RAM. The framework's modules run in a virtual box VM that runs Ubuntu 14.04.6 LTS. Table 1 provides the GC serializing size, circuit garbling time, and GC evaluation time. We observe that garbling a circuit of 21 gates and 22 wires takes about 17 milliseconds to garble and 0.2 milliseconds to evaluate. The timing values are considered acceptable, compared to the timing of e.g., the construction of Carter et al. [4], although the sizes of the circuits are small.

The time to send GC from the *Owner* to the *Broker* depends mainly on the size of the serialized GC. Equation 1 provides the size of the serialized garbled circuit, where q is the number of gates, n is the number of input wires, and m is the number of output wires of the circuit. Column 3 of Table 1 provides examples of the sizes the serialized GCs. We believe that the time to send GC will be reasonably low.

$$s = (84 \times q) + ((n + q + 1) \times 52) + (4 \times m) + 16 + 20. \quad (1)$$

where:
q: number of gates.
n: number of input wires.
m: number of output wires.

The framework crashes when we use circuits of large size because of the enormous effective size of the memory allocated to the GC in `justGarble`. This issue requires further debugging of the code to make it more robust and useful.

We note that the implemented protocol uses two AES keys to construct the labels for the GC. The schema is secure because the broker, who evaluates the GC, cannot associate the input to the input labels and cannot associate the output of the circuit to the output labels generated from the GC.

6 Conclusion

We discussed in this paper the results of assessing the use of Garbled Circuit (GC) to protect time-sensitive input data and filtering and distribution rules from suspicious brokers. We designed a protocol for secure data dissemination that uses GC and implemented a prototype of it, *SFEDataShare*. Then, we evaluated the framework using a small randomly generated circuits of up to about 21 gates.

We found that the size of data exchanged between the parties is small and the time to garble a circuit and evaluate it are also reasonably small. The results show that GC could be used for secure data brokering with acceptable performance.

7 Acknowledgement

This work is supported by the Air Force Research Lab, Rome, USA.

References

1. S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Attribute-based encryption for circuits," *J. ACM*, vol. 62, Dec. 2015.
2. S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters, "Attribute-based encryption for circuits from multilinear maps," in *Advances in Cryptology – CRYPTO 2013* (R. Canetti and J. A. Garay, eds.), pp. 479–499, 2013.
3. A. C. Yao., "How to generate and exchange secrets," in *In Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, p. 162–167, 1986.
4. H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, vol. 24, no. 2, pp. 137–180, 2016.
5. S. Badrinarayanan, A. Jain, R. Ostrovsky, and I. Visconti, "Non-interactive secure computation from one-way functions," in *IACR Cryptology ePrint Archive*, 2018.
6. A. Morgan, R. Pass, and A. Polychroniadou, "Succinct non-interactive secure computation." Cryptology ePrint Archive, Report 2019/1341, 2019. `https://eprint.iacr.org/2019/1341`.
7. D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Theory of Cryptography* (Y. Ishai, ed.), (Berlin, Heidelberg), pp. 253–273, Springer Berlin Heidelberg, 2011.
8. O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, Sept. 2009.
9. D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy, "Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits," in *Advances in Cryptology – EUROCRYPT 2014* (P. Q. Nguyen and E. Oswald, eds.), pp. 533–556, 2014.
10. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay—a secure two-party computation system," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (USA), p. 20, USENIX Association, 2004.
11. B. Kreuter, A. Shelat, and C. hao Shen, "Billion-gate secure computation with malicious adversaries," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, (Bellevue, WA), pp. 285–300, USENIX, 2012.

12. M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *2013 IEEE Symposium on Security and Privacy*, pp. 478–492, 2013.
13. V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Automata, Languages and Programming* (L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, eds.), pp. 486–498, 2008.
14. B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Proc. International Conference on the Theory and Application of Cryptology and Information Security – ASIACRYPT 2009*, (Berlin, Heidelberg), pp. 250–267, Springer Berlin Heidelberg, 2009.
15. M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Get justgarble." `http://cseweb.ucsd.edu/groups/justgarble/`. accessible on July 2020.
16. J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, B. Grégoire, V. Laporte, and V. Pereira, "A fast and verified software stack for secure function evaluation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 1989–2006, 2017.
17. M. O. Rabin, "How to exchange secrets with oblivious transfer," technical report TR-81, Aiken Computation Lab, Harvard University, 1981.
18. G. Barthe, F. Dupressoir, B. Grégoire, C. Kunz, B. Schmidt, and P.-Y. Strub, *Foundations of Security Analysis and Design VII FOSAD 2012-2013 Tutorial Lectures*, ch. EasyCrypt: A Tutorial, pp. 146–166. 2014.
19. M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proc. of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, p. 784–796, 2012.
20. S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption," in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, p. 555–564, 2013.
21. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proc. of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, p. 218–229, 1987.
22. Ágnes Kiss and T. Schneider, "Valiant's universal circuit is practical." Cryptology ePrint Archive, Report 2016/093, 2016. `https://eprint.iacr.org/2016/093`.
23. L. Ben Othmane, *Active Bundles for Protecting Confidentiality of Sensitive Data throughout Their Lifecycle*. PhD thesis, Western Michigan University, USA, 2010.
24. S. Keelveedhi and M. Bellare, "Justgarble." `https://github.com/irdan/justGarble`, 03 2014.
25. Intel Corporation, "Intel® c++, intrinsic reference, document number: 312482-003us," Sep. 2021.