symmetry

MDPI

Article

A Ciphertext Reduction Scheme for Garbling an S-Box in an AES Circuit with Minimal Online Time

Xu Yan [1,2], Bin Lian [1,*], Yunhao Yang [1,2], Xiaotie Wang [3], Jialin Cui [1], Xianghong Zhao [1] and Fuqun Wang [4] and Kefei Chen [4]

1 School of Information Science and Engineering, NingboTech University, Ningbo 315100, China; yx2000@zju.edu.cn (X.Y.); 22331136@zju.edu.cn (Y.Y.); cuijl_jx@163.com (J.C.); zxh@nit.net.cn (X.Z.)
2 College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou 310058, China
3 School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China; wangxiaotiey@163.com
4 School of Mathematics, Hangzhou Normal University, Hangzhou 311121, China; fqwang@hznu.edu.cn (F.W.); kfchen@hznu.edu.cn (K.C.)
* Correspondence: lianbin_a@163.com

Abstract: The secure computation of symmetric encryption schemes using Yao's garbled circuits, such as AES, allows two parties, where one holds a plaintext block m and the other holds a key k, to compute $Enc(k, m)$ without leaking m and k to one another. Due to its wide application prospects, secure AES computation has received much attention. However, the evaluation of AES circuits using Yao's garbled circuits incurs substantial communication overhead. To further improve its efficiency, this paper, upon observing the special structures of AES circuits and the symmetries of an S-box, proposes a novel ciphertext reduction scheme for garbling an S-box in the last SubBytes step. Unlike the idea of traditional Yao's garbled circuits, where the circuit generator uses the input wire labels to encrypt the corresponding output wire labels, our garbling scheme uses the input wire labels of an S-box to encrypt the corresponding "flip bit strings". This approach leads to a significant performance improvement in our garbling scheme, which necessitates only 2^8 ciphertexts to garble an S-box and a single invocation of a cryptographic primitive for decryption compared to the best result in previous work that requires 8×2^8 ciphertexts to garble an S-box and multiple invocations of a cryptographic primitive for decryption. Crucially, the proposed scheme provides a new idea to improve the performance of Yao's garbled circuits. We analyze the security of the proposed scheme in the semi-honest model and experimentally verify its efficiency.

Keywords: garbled circuits; secure computation of AES; oblivious pseudo-random function; secure two-party computation

Citation: Yan, X.; Lian, B.; Yang, Y.; Wang, X.; Cui, J.; Zhao, X.; Wang, F.; Chen, K. A Ciphertext Reduction Scheme for Garbling an S-Box in an AES Circuit with Minimal Online Time. *Symmetry* **2024**, *16*, 664. https://doi.org/10.3390/sym16060664

Academic Editors: Sergei D. Odintsov and Xiaoyang Dong

Received: 29 March 2024
Revised: 20 May 2024
Accepted: 21 May 2024
Published: 28 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Yao's Garbled Circuits and Secure AES Computation

Yao's garbled circuits, introduced in [1], remain a cornerstone method for facilitating secure two-party computation tasks. This approach enables two parties to jointly compute a function without leaking their input to each other. In the traditional garbled circuits protocol, the two parties play a circuit generator and a circuit evaluator, respectively. Initially, the two parties decompose the function that they want to evaluate into a Boolean circuit, and then the circuit generator produces the wire labels and garblings (ciphertexts) for the Boolean circuit. The wire labels are used to conceal the truth values, and the garblings are used to decrypt the output wire labels. Finally, with input wire labels and garblings, the circuit evaluator can decrypt the output wire labels and learn the output of the function. During the process, the evaluator does not learn which truth values the wire labels correspond to because they are all produced by the generator, while the generator does not learn which wire labels the evaluator uses to decrypt the garblings because the

generator is not involved in the decryption process. Thus, neither side knows what the other's input is.

Among the myriad applications of garbled circuits [2–4], the secure computation of the Advanced Encryption Standard (AES) [5] has garnered significant interest. In this setup, the evaluator, possessing a plaintext message m, can encrypt m using the generator's key k without disclosing m to the generator while remaining oblivious to k. Secure AES computation has found numerous applications [5], notably in the realms of side-channel protection, blind message authentication codes (MACs), blind encryption, third-party operations on encrypted data, etc. A particularly notable application is the construction of the oblivious pseudo-random function (OPRF), which is fundamental to a host of privacy-preserving technologies, including private set intersection (PSI) [6–8], private information retrieval (PIR) [9,10], private keyword search (PKS) [11,12], location sharing [13], etc.

When these privacy-preserving protocols are constructed using secure AES computation, their runtime performance can be significantly improved. For example, the study in [14] demonstrates a PSI protocol based on the OPRF, which is constructed using secure AES computation, achieving the highest efficiency compared to the RSA blind signature-based PSI (RSA-PSI), Diffie–Hellman-based PSI (DH-PSI), and Naor–Reingold PRF-based PSI (NR-PSI). However, this method incurs the most communication overhead, primarily due to the use of Yao's garbled circuits to securely compute AES.

1.2. Gaps and Motivation

The huge communication overhead and deployment difficulty are two major obstacles to the practical application of Yao's garbled circuits. Although there have been various schemes proposed to reduce the communication overhead of garbled circuits in recent years, their communication overhead is still not ideal due to the fact that (1) the essence of the garbled circuits is to use a bit string (wire label) to mask a bit, and (2) the Boolean circuit that realizes the function is complex. In addition, there are few mature frameworks that can efficiently convert the function to a Boolean circuit, and there is no systematic literature on how to implement the garbled circuit schemes, which adds challenges to the deployment of Yao's garbled circuits in real applications.

Motivation. In view of these challenges, it is important to shift attention from universal Yao's garbled circuits to a commonly used cryptographic module realized by Yao's garbled circuits, such as secure AES computation. The optimization of a specific module is much easier than the optimization of universal garbled circuits. Therefore, in this paper, we focus on how to optimize and implement secure AES computation, which is realized using Yao's garbled circuits.

1.3. Our Idea

An AES circuit needs to perform four algorithms: SubBytes, MixColumns, ShiftRows, and AddRoundKey. After extensive optimization of the circuit structure, only the SubBytes step requires the transmission of ciphertexts (i.e., incurs communication overhead). The remaining steps—MixColumns, ShiftRows, and AddRoundKey—can be efficiently performed using only free XOR gates [15].

Central to the SubBytes step is the S-box, which is a critical nonlinear element to create turmoil. There are various studies on the optimization [16,17] of the S-box and its applications [18–21]. For the original AES S-box, Huang et al. [22] proposed two garbling schemes aimed at minimizing the total and online times, respectively. The first scheme involves decomposing the process of the S-box into a circuit, which completes GF(2^8) inversion and bit transformation calculations, ultimately resulting in 58 [22] non-free gates. The second scheme treats the S-box as a "gate" with eight input and eight output wires; thus, the generator needs to garble this "gate" using eight input wire labels to encrypt eight output wire labels. This results in 8×2^8 ciphertexts, of which the evaluator only needs to decrypt eight (eight output wire labels). The former approach, while less communication-intensive, incurs longer online times, as the computation cannot be pre-processed and requires multi-

ple invocations of cryptographic primitives. Many cryptographic primitives can be used to produce ciphertexts such as hash functions [23], fixed-key AES [24], etc. The latter, while necessitating the generation of more ciphertexts, allows for pre-processing by the generator and reduces online time by requiring only four invocations of a cryptographic primitive.

It should be noted that if we use the second idea in [22] to garble an S-box that has eight input and eight output wires, instead of using the garbling scheme in [22] where the generator uses eight input wire labels to encrypt eight corresponding output wire labels, the generator can encrypt the corresponding "flip bit string" according to the mapping law of the S-box. For example, for an S-box that maps 00111100 to 11010101, the flip bit string is $00111100 \oplus 11010101 = 11101001$. These flip bit strings are used to flip the least significant bit (lsb) of the input wire labels of the S-box. Since the final truth values of the output are determined by the lsb of the output wire labels, and there are only XOR operations between wire labels in other parts of the AES circuit, flipping the lsb of the input wire labels of the S-box can eventually produce the correct output, thus reducing the number of ciphertexts by converting the encryption of eight output wire labels into only one flip bit string. Furthermore, by introducing optimized S-box structures [16–21], where each flip bit string is almost exclusively mapped to an input byte, the security of our garbling scheme can be effectively improved without additional overhead.

Contributions. The current state of optimizing performance for garbling the AES circuit seems to be approaching its limits unless there is a significant breakthrough in the field of garbled circuits. In this paper, we leverage the unique structure of the S-box and AES circuit to propose a ciphertext reduction scheme. The contributions of this paper are summarized as follows:

- We propose a novel garbling scheme, applicable to the 16 S-boxes in the final SubBytes step, which requires only 2^8 ciphertexts to garble each S-box, with only one call necessary for a cryptographic primitive. A comparison of the communication and computational cost between our garbling scheme and existing schemes is shown in Table 1. It is important to note that regardless of the S-box structure used, the overhead of our scheme does not change. However, the overhead of the minimal total time scheme in [22] increases because the optimized S-box has more nonlinear gates.
- In our experiments, we avoid using any hardware description language to instantiate the AES circuit. Instead, we show how to only use the C++ class to construct the structure of the AES circuit. In order to reuse circuit units, we introduce the concept of the circuit layer, where we design a circuit layer for each algorithm in AES, and each circuit layer stores only one copy in memory. Our implementation can help researchers better understand how to deploy Yao's garbled circuits in reality.

Table 1. Performance comparison between our scheme and existing schemes (single S-box).

Scheme	Num. of AND Gates	Num. of Ciphetexts	Calls for Crypt.
Min. online time scheme in [22]	-	2048	4
Min. total time scheme in [22]	58	116 (half-gate [23])	116 (half-gate [23])
		174 (GRR3 [25])	58 (GRR3 [25])
Ours	-	256	1

Organization. Our paper is organized as follows. In Section 2 we discuss related works on garbled circuits and secure AES computation. In Section 3, we give the necessary concepts and notations for understanding our design. In Section 4, we demonstrate our ciphertext reduction scheme for garbling an S-box, illustrating its potential extra overhead and possible extensibility, and provide the whole algorithm for garbling and evaluating the AES circuit. In Section 5, we prove the security of our scheme, and in Section 6, we show the circuit constructions and analyze the efficiency of our garbling scheme.

2. Related Works

Yao's garbled circuits have been at the forefront of cryptographic research, particularly in the domain of secure multi-party computation (MPC) since their introduction by Andrew Yao in the 1980s. This period has seen a burgeoning interest from the research community in refining and optimizing the performance of this pivotal technique.

The "point-and-permute" optimization, first proposed by Beaver in 1990 [26], marked a significant advancement in reducing the communication and computational overhead associated with garbled circuits. This technique leverages the last bit of the wire label, enabling the evaluator to discern which ciphertext to decrypt. This innovation not only minimizes the computational burden of decryption but also obviates the need for communicating MACs to verify decryption outcomes. Furthermore, a suite of techniques has been developed to reduce the number of ciphertexts required: the 4-to-3 garbled row reduction (GRR3) [25], 4-to-2 garbled row reduction (GRR2) [5], free XOR [15], flexible XOR [27], half-gate [23], and slicing and dicing [28] methods. Each of these technologies contributes to the goal of optimizing communication efficiency within garbled circuits, as detailed in Table 2.

Table 2. Comparison between efficient garbling schemes [28].

	Communication (K bits/per Gate)		Calls to H per Gate			
			Generator		Evaluator	
Scheme	**AND**	**XOR**	**AND**	**XOR**	**AND**	**XOR**
Yao	8	8	4	4	2.5	2.5
Point and permute	4	4	4	4	1	1
GRR3	3	3	4	4	1	1
GRR2	2	2	4	4	1	1
Free XOR	3	0	4	0	1	0
FleXOR	2	{0,1,2}	4	{0,2,4}	1	{0,1,2}
Half-gate	2	0	4	0	2	0
Slicing and dicing	1.5	0	$\leq$6	0	$\leq$3	0

AES is a globally adopted symmetric encryption standard known for its efficiency and security. It operates on block sizes of 128 bits with key sizes of 128, 192, or 256 bits, executing several rounds of transformation to securely encrypt plaintext into ciphertext. When combined with garbled circuits, secure AES computation offers expansive application potential in privacy-preserving fields.

Initially introduced by Pinkas et al. [5], secure AES computation has been incorporated into various secure multi-party computation (MPC) frameworks, including Fairplay [29], Obliv-C [30], and TASTY [31]. It continues to stand as a prominent benchmark for evaluating MPC systems.

Its appeal arises from its potential for various applications, including the construction of the OPRF, a critical component in cryptographic operations. Although the OPRF derived using secure AES computation is less efficient than that constructed via oblivious transfer (OT) [32,33], it offers a unique advantage. This advantage stems from the inherent efficiency of AES encryption. If a privacy-preserving protocol employing secure AES computation-based OPRF could utilize AES encryption somewhere, its efficiency could be significantly enhanced. This point has been confirmed in the work of Kiss et al. [14]. The authors compared the performance of various kinds of PSI protocols constructed using different schemes, as detailed in Table 3, where the GC-PSI protocol is constructed using secure AES computation-based OPRF. The findings suggest that although the PSI protocol built using secure AES computation-based OPRF is less efficient in the base phase for the secure evaluation of AES circuits, it is the most efficient from a global perspective.

Table 3. Runtime performance of different kinds of PSI protocols. The experimental data were obtained from [14], with all other parameters kept constant.

Scheme	Base (ms)	Setup (ms)	Online (ms)
RSA-PSI	56	3,441,906	7.38
DH-PSI	1	462,496	3.49
ECC-DH-PSI	1	1,325,400	2.91
NR-PSI	119	758,400	10.28
GC-PSI	**1132**	**70**	**2.49**

3. Preliminaries

3.1. Notations

Wire labels in the garbled circuits are denoted as W_i^b, where i represents the index of the wire and b signifies the binary value of 0 or 1. For wire i, W_i^0 and W_i^1 correspond to the false and true labels, respectively. Each gate shares the same index with its output wire. Additionally, wire labels can be also represented by a capital letter along with its least significant bit (lsb), facilitating the exposition of some concepts. For example, $(A,1)$ indicates that the wire label is A and its lsb is 1. The concatenation of two wire labels is denoted by $||$, and $||_{\{0,1,2...n\}}^{m_i} W$ denotes the concatenation of multiple labels $W_0^{b_0}, W_1^{b_1}, W_2^{b_2}...W_n^{b_n}$, where the sequence $b_0 b_1 b_2 ... b_n = m_i$. m_i is the 8-bit representation of i, such as $m_3 = 00000011$. "$\leftarrow \$$" denotes the random sampling, and "$\leftarrow$" denotes the value assignment.

3.2. Garbled Circuit

The basic two-party garbled circuit evaluation scheme involves a generator and an evaluator. The generator is responsible for producing wire labels and ciphertexts for the circuit, while the evaluator uses wire labels in hand to decrypt ciphertexts according to the circuit's topology. The security of the scheme is based on the idea that the generator does not learn which wire labels the evaluator holds, and the evaluator does not learn the truth value that the wire labels represent.

The process for the generator to garble a Boolean circuit is as follows: The generator randomly samples two labels, W^0 and W^1, for each wire, representing bits 0 and 1, respectively. For a binary gate g with input wires i and j and output wire k, the generator arranges the ciphertexts as follows:

$$En^k_{W_i^{b_i}, W_j^{b_j}}(W_k^{g(b_i, b_j)})$$

for all inputs $b_i, b_j \in \{0, 1\}$. The equation means that the generator encrypts the output wire label using the corresponding combination of input wire labels. For example, for an AND gate with input value $(0, 1)$, the output value should be 0; thus, the generator encrypts W_k^0 using W_i^0 and W_j^1, where the input wires are i and j and the output wire is k. In this way, the generator generates four ciphertexts successively for each gate according to the topology order of the circuit.

A universal representation of the garbling scheme can be derived from [34], where a garbling scheme is denoted as a five-tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$, as shown in Figure 1.

The function Gb maps f and k to (F, e, d), where (F, e, d) are the stings that represent the encoding garbled function, the encoding function, and the decoding function. Possession of e and x allows one to compute the garbled input $X = \mathsf{En}(e, x)$; F and X enable the calculation of the garbled output $Y = \mathsf{Ev}(F, X)$; and d and Y allow for the recovery of the final output value $y = \mathsf{De}(d, Y)$, which must be equal to $\mathsf{ev}(f, x)$.

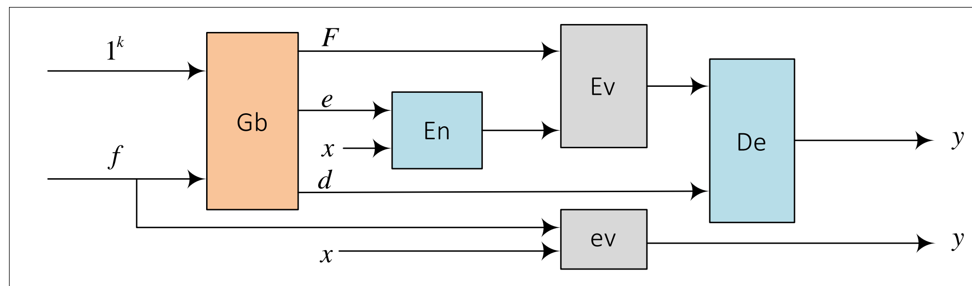

Figure 1. Components of a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$.

3.3. Free XOR Gate

The free XOR technique, initially proposed in [15], eliminates the need for a ciphertext to evaluate an XOR gate, significantly reducing the communication overhead of garbled circuits.

The idea of free XOR is based on the observation that it is unnecessary to randomly generate false (representing bit 0) and true (representing bit 1) labels on a wire. Instead, the false and true labels on a wire can establish a relationship such that the false label = the true label $\oplus$ the offset value. The offset value is globally present and is secretly kept by the generator.

As depicted in Figure 2, A and B denote the false labels on their respective wires, and Δ represents the global offset value. Thus, the corresponding true labels on their respective wire are $A \oplus \Delta$ and $B \oplus \Delta$. The false label on the output wire is computed as $C = A \oplus B$. Similarly, the true label on the output wire is $C \oplus \Delta$. When the generator produces wire labels for the entire circuit, it only needs to randomly sample a false label for every input wire and a global offset value Δ, and the false label of other wires can be computed gate by gate according to the topology of the circuit.

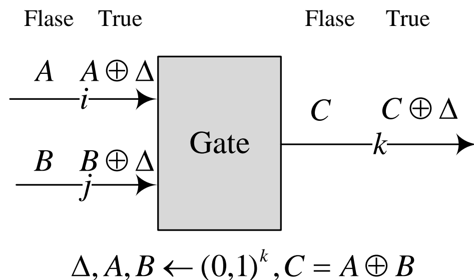

Figure 2. How to produce wire labels in the free XOR scheme.

This method of generating wire labels allows the evaluator, holding two input labels of an XOR gate, to simply XOR the two labels to compute the output wire label without decrypting any ciphertext. The correctness is demonstrated in Table 4.

Table 4. Free XOR correctness verification.

i	j	k
A	B	C
A	$B \oplus \Delta$	$C \oplus \Delta$
$A \oplus \Delta$	B	$C \oplus \Delta$
$A \oplus \Delta$	$B \oplus \Delta$	C

3.4. Reusable Circuit Layers in the AES Circuit

There are primarily four algorithms involved in encrypting one plaintext block using AES: SubBytes, ShiftRow, MixColumn, and AddRoundKey. The KeyExpansion algorithm can be processed locally by the generator.

The circuit design of each of these algorithms has been extensively studied [22]. However, to the best of our knowledge, there is no existing literature on how to reuse the circuit units in the AES circuit to generate a garbled circuit. Although some automated compilation tools have been proposed [29,31], the structure of the auto-compiled circuit is not the simplest. This underscores the significance of manually designing circuit structures and understanding how to reuse circuit units.

To enable the reuse of circuit units, we propose designing a circuit layer for each algorithm and setting an input and output 128-bit register at both ends of the circuit layer call area. In the AddRoundKey layer, an additional 128-bit register is needed for the generator to input the round key.

As depicted in Figure 3, each circuit layer is stored in memory only once and is directly connected between the input register and the output register when it needs to be called. After evaluating a circuit layer, the wire labels are transmitted from the output register to the input register. By reusing circuit layers, the memory usage of both parties can be greatly reduced.

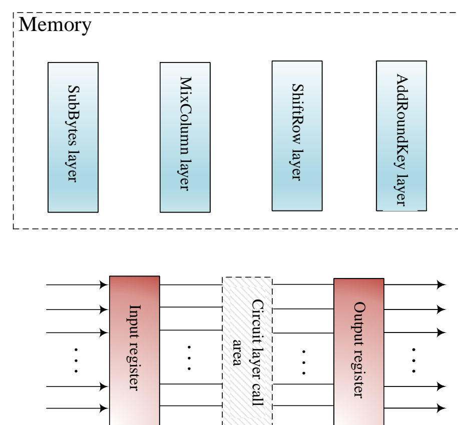

Figure 3. Structure for reusing circuit layers in the AES circuit.

4. Construction

4.1. Intuitive Description

Initially, we recall how the evaluator computes the values on the output wires of a garbled circuit. The origin input x is transformed into input wire labels through $X \leftarrow \mathsf{En}(\hat{e}, x)$. Subsequently, the output wire labels are computed as $Y \leftarrow \mathsf{Ev}(\hat{F}, X)$. This process involves the evaluator decrypting the ciphertext $\hat{F}$ of the garbled circuit using the input wire labels X. Finally, holding the output wire labels Y and the decoding vector $\hat{d}$, the evaluator can compute the output values as $y \leftarrow \mathsf{De}(\hat{d}, Y)$. The decoding vector $\hat{d}$ usually comprises the lsbs of the false wire labels corresponding to each output wire and is transmitted from the generator to the evaluator at the beginning of the protocol. After acquiring all the output wire labels by evaluating the garbled circuit, the evaluator can compute the output value y by comparing the lsbs of the output wire labels with the corresponding bits in $\hat{d}$. If the lsb of an output wire label matches the corresponding bit in $\hat{d}$, it means that this wire label represents the value 0; otherwise, it represents the value 1.

For example, let us assume there is only one output wire, with the false wire label $(A, 1)$ (where 1 is the lsb) and the true wire label $(A \oplus \Delta, 0)$. The generator sends the lsb (1) of the false wire label (A) as the decoding vector $\hat{d}$ to the evaluator. When the evaluator computes an output wire label $(A, 1)$ and finds that the lsb of A matches the bit in $\hat{d}$, it learns that A represents the value 0. Conversely, if the evaluator computes an output wire label $(A \oplus \Delta, 0)$ and finds that the lsb of the output wire label does not match the bit in $\hat{d}$, it learns that the computed output wire label represents the value 1.

An important observation here is that if the lsb of the output wire label is flipped but the bit in $\hat{d}$ remains unchanged, the evaluator will output the opposite value. For instance,

if the output wire label computed by the evaluator is $(A, 0)$ (assuming the lsb of A is flipped during the evaluation) instead of $(A, 1)$, but the bit in $\hat{d}$ remains 1, the evaluator will think that it is acquiring a true wire label and output value 1, even though A represents a false wire label from the generator's perspective.

This example illustrates a key point: if the evaluator ultimately obtains an output value of 1 (0), it is not necessary for them to acquire the true (false) label. By flipping the lsb of the output wire label, the correct output value can still be achieved. Therefore, for an S-box with eight input wires and eight output wires, the generator does not need to encrypt the output wire labels using input wire labels like in the traditional garbled circuits protocol. Instead, it can simply encrypt eight flip bits. When the evaluator decrypts these flip bits using the input wire labels they possess, they use these flip bits to flip the lsbs of these input wire labels. This way, the evaluator can still output the correct values, even without the correct true or false label.

If using this method to garble an S-box, the generator can produce ciphertexts as follows: Consider a possible input value of the S-box 0×10, and the output will be $S(0 \times 10)$=0× CA. The corresponding binary representations are 0×10 = 0b00010000 and $0 \times$ CA = 0b11001010. Thus, the bits that need to be flipped are $00010000 \oplus 11001010 = 11011010$, where 1 denotes the need for flipping and 0 denotes no flipping. As depicted in Figure 4, for the input value 0x10, the generator knows the evaluator will hold the input wire label combination $\{A, B, C, D \oplus \Delta, E, F, G, H\}$. Thus, the generator uses $A||B||C||D \oplus \Delta||E||F||G$ to encrypt the flip bit string (fbs) 11011010 (all possible fbs are shown in Appendix A Figure A1). Similarly, the generator can produce ciphertexts for all possible input wire label combinations, resulting in a total of 2^8 ciphertexts for an S-box. The evaluator only needs to decrypt one of them. It is important to note that in the AES circuit, except for the S-box, the rest of the gates are free XOR gates, where the evaluator only needs to perform the XOR operation between wire labels locally and does not need to decrypt any ciphertext. Thus, the behavior of flipping lsbs does not affect the circuit evaluation. The encryption scheme is drawn from [23]. It is important to note that the length of the output H is σ, and the length of the fbs is 8. Thus, we need a reversible injective mapping pad: $(0,1)^8 \leftrightarrow (0,1)^\sigma$, and we use H to encrypt $pad(fbs)$. However, for the convenience of presentation, in the description of the scheme, we omit the process of mapping the fbs to $(0,1)^\sigma$.

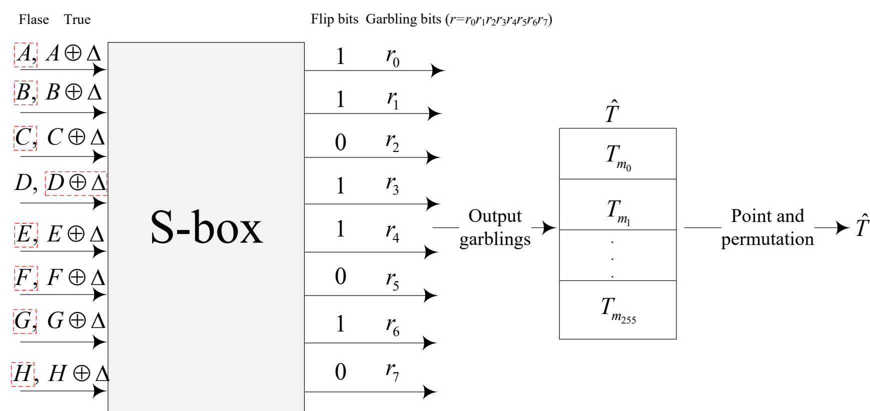

Figure 4. Garbling an S-box with its input wire labels.

However, the fbs will lead to a significant risk of information leakage due to the public mapping mode of the S-box. For example, if the evaluator decrypts the $fbs = 11011010$, it can promptly deduce that the input value is 0×10 and the output value is $0 \times$ CA. This inference is feasible because the fbs almost exclusively corresponds to the input and output values.

To address this concern, the generator needs to conceal the real fbs and let the evaluator decrypt a garbled fbs. The detailed process is as follows: The generator randomly samples an 8-bit string r, which we refer to as the garbling bits, and records each bit of r on the corresponding wire. Then, for all 2^8 possible fbss, the generator XORs r and fbs to obtain the garbled fbs^* and uses corresponding input wire label combinations to encrypt fbs^*. As shown in Figure 4, for the $fbs = 11011010$, the generator encrypts $fbs^* = 11011010 \oplus r_0r_1r_2r_3r_4r_5r_6r_7$ using the corresponding wire label combination: $A||B||C||D \oplus \Delta||E||F||G$. As a result, when the evaluator holds this input wire label combination, it will decrypt the $11011010 \oplus r_0r_1r_2r_3r_4r_5r_6r_7$ instead of the real $fbs = 11011010$. Although every possible fbs is garbled by the same r, based on the security of the free XOR gate scheme, the evaluator can only decrypt one of them, effectively preventing the leakage of the original fbs.

To ensure the correctness of the final output, the generator transmits the garbling bits to the final output wire according to the circuit's topology. Finally, instead of sending the original $\hat{d}$ to the evaluator, the generator sends $\hat{d}^* = \hat{d} \oplus r^*$, where r^* represents the garbling bits that are finally recorded on the output wire. Holding $\hat{d}^*$, the evaluator can still output the correct value as $y \leftarrow De(\hat{d}^*, Y)$.

Unfortunately, our garbling scheme for S-boxes can only be applied in the final SubBytes layer among the 10 SubBytes layers involved in the AES-128 circuit. This limitation arises from the fact that flipping the lsbs in the S-boxes of the preceding SubBytes layers would adversely impact the decryption process in subsequent SubBytes layers.

Despite this restriction, the S-box garbling scheme above ensures a reduction in the number of ciphertexts from $8 \times 16 \times 2^8$ to 16×2^8, specifically in the final SubBytes layer while maintaining minimal online time. It should be noted that a SubBytes layer consists of 16 S-boxes.

4.2. Garbling Scheme for the Final SubBytes Layer

Our garbling scheme for AES circuits is shown in Figure 5. For each gate i (i is also its output wire), the $\mathsf{GateInputs}(f, i)$ function returns its input wire indices. For each S-box i, the $\mathsf{GateInputs}(f, i)$ function returns eight input wire indices, and the $\mathsf{GateOutputs}(f, i)$ function returns eight output wire indices. r_i denotes the garbling bit recorded on wire i.

Here, we mainly describe the GbSbox algorithm in detail. In the initial step, the generator randomly samples garbling bits r_i for each input wire i. For every conceivable input (i.e., $m_0 - m_{255}$), the generator first computes the fbs and garbles it by XORing the fbs with garbling bits recorded on the wire. Then, the generator encrypts these garbled fbss using the corresponding combination of input wire labels. Finally, the generator uses the point-and-permutation technique to sort the order of ciphertexts. The transmission of garbling bits is the same as the evaluation of a gate. For example, for an XOR gate with input wires a and b and output wire c, $r_c = r_a \oplus r_b$ (there is no AND gate in the AES circuit). After transmission of garbling bits from the S-boxes to the output wire, the generator produces the decoding vector $\hat{d}$, where for the output wire i, $d_i = lsb(W_i^0) \oplus r_i$.

procedure Gb($1^k, f$):
$\Delta \leftarrow \$(0,1)^{k-1}$
for $i \in$ Inputs(f) **do**
$W_i^0 \leftarrow \$(0,1)^{k-1}$, $lsb(W_i^0) \leftarrow \$(0,1)$
$W_i^1 \leftarrow W_i^0 \oplus \Delta$, $lsb(W_i^1) \leftarrow lsb(W_i^0) \oplus 1$
$e_i \leftarrow W_i^0$
in {AddRoundKey, ShiftRow, MixColumn layers}
for $i \notin$ Inputregister (in topology order) **do**
$\{a, b\} \leftarrow$ GateInputs(f, i)
$W_i^0 \leftarrow W_a^0 \oplus W_b^0$
$W_i^1 \leftarrow W_i^0 \oplus \Delta$
$r_i \leftarrow r_a \oplus r_b$
for $i \in$ Inputregister **do**
$a \leftarrow$ GateInputs(f, i)
$W_i^0 \leftarrow W_a^0$
$W_i^1 \leftarrow W_i^0 \oplus \Delta$
$r_i \leftarrow r_a$
in {previous SubBytes layers}
for j-th S-box **do**
$\hat{T}_j \leftarrow$ GbSbox*
$F_j \leftarrow \hat{T}_j$
for the last SubBytes layer **do**
for j-th S-box **do**
$\hat{T}_j \leftarrow$ GbSbox
$F_j \leftarrow \hat{T}_j$
for $i \in$ Outputs(f) **do**
$d_i = lsb(W_i^0) \oplus r_i$
return $(\hat{F}, \hat{e}, \hat{d})$

procedure GbSbox
for $i \in$ GateInputs(Sbox) **do**
$r_i \leftarrow \$(\mathbf{0},\mathbf{1})$
for $i = \mathbf{0}$ **to 255 do**
$fbs = m_i \oplus S(m_i)$
$T_{m_i} \leftarrow H(\|_{\{j \in \text{Input(Sbox)}\}}^{m_i} W) \oplus pad\left(fbs \oplus (\|_{\{j \in \text{Input(Sbox)}\}} r)\right)$
$\hat{T} \xleftarrow{\text{point and permutaion}} \{T_{\mathbf{0}}, T_{\mathbf{1}} \ldots T_{\mathbf{255}}\}$
return $\hat{T}$

procedure Ev($1^k, f, \hat{F}$)
for $i \in$ Inputs(f) **do**
$W_i \leftarrow X_i$
in {AddRoundKey, ShiftRow, MixColumn layers}
for $i \notin$ Inputregister **do**
$\{a, b\} \leftarrow$ GateInputs(f, i)
$W_i \leftarrow W_a \oplus W_b$
$lsb(W_i) \leftarrow lsb(W_a) \oplus lsb(W_b)$
for $i \in$ Inputregister **do**
$a \leftarrow$ GateInputs(f, i)
$W_i \leftarrow W_a$
$lsb(W_i) \leftarrow lsb(W_a)$
in {previous SubBytes layers}
for j-th S-box **do**
$\{a, b, c, d, e, f, g, h\} \leftarrow$ GateInputs(f, j)
$\{a^*, b^*, c^*, d^*, e^*, f^*, g^*, h^*\} \leftarrow$ GateOutputs(f, j)
$(W_{a^*}, lsb(W_{a^*})) \leftarrow$ EvSbox*, $(W_{b^*}, lsb(W_{b^*})) \leftarrow$ EvSbox*$\ldots(W_{h^*}, lsb(W_{h^*})) \leftarrow$ EvSbox*
in {the last SubBytes layer}
for j-th S-box **do**
$\{a, b, c, d, e, f, g, h\} \leftarrow$ GateInputs(f, j)
$\{a^*, b^*, c^*, d^*, e^*, f^*, g^*, h^*\} \leftarrow$ GateOutputs(f, j)
$W_{a^*} \leftarrow W_a, W_{b^*} \leftarrow W_b \ldots W_{g^*} \leftarrow W_g, W_{h^*} \leftarrow W_h$
$fbs \leftarrow pad^{-1}\left(H(W_a \| W_b \| W_c \| W_d \| W_e \| W_f \| W_g \| W_h) \oplus F_{(j, lsb(W_a) \| lsb(W_b) \| \ldots lsb(W_h))}\right)$
$lsb(W_{a^*}) \leftarrow lsb(W_a) \oplus fbs_0$, $lsb(W_{b^*}) \leftarrow lsb(W_b) \oplus fbs_1 \ldots lsb(W_{h^*}) \leftarrow lsb(W_h) \oplus fbs_7$
for $i \in$ Outputs(f) **do**
$Y_i \leftarrow W_i$
$lsb(Y_i) \leftarrow lsb(W_i)$
return $\hat{Y}$

procedure En($\hat{e}, \hat{x}$)
for $e_i \in \hat{e}$ **do**
$X_i \leftarrow e_i \oplus x_i \Delta$
return $\hat{X}$

procedure De($\hat{Y}, \hat{d}$)
for $d_i \in \hat{d}$ **do**
$y_i \leftarrow d_i \oplus lsb(Y_i)$
return $\hat{y}$

Figure 5. Garbling scheme for AES circuit. GbSbox* and EvSbox* denote the original S-box garbling scheme, and GbSbox denotes our S-box garbling scheme. *pad* denotes a reversible injective mapping: $(0,1)^8 \leftrightarrow (0,1)^\sigma$.

4.3. Discussions on the Additional Cost and Universality

The only additional cost in our S-box garbling scheme is that the generator needs to sample some garbling bits, record them, and transmit them to the output wire. In fact, compared to the overhead incurred by the generator to produce the ciphertexts for the AES circuit, this additional cost is almost negligible.

Apart from the S-box, our garbling scheme can also be applied to other combinational circuits. However, whether it is worthwhile depends on the depth of the combinational circuit and the number of input and output wires. There are two scenarios where our scheme becomes impractical: (1) when the combinational circuit inherently contains few AND gates and (2) when the combinational circuit has too many input wires. In the first

scenario, the cost of evaluating gate by gate is minimal, rendering the integration of the combinational circuit unnecessary. In the second scenario, an excessive number of input wires results in a substantial number of ciphertexts, making it impractical. An S-box has eight input wires, resulting in 2^8 ciphertexts, which is acceptable. However, if we decompose the S-box into various gates for evaluation, it may contain more than 58 AND gates, leading to a high cost of evaluating gate by gate. Therefore, our scheme is highly suitable for garbling S-boxes.

5. Security

5.1. Cryptographic Assumption and Security Model

5.1.1. Random Oracle

Random oracle (RO) is an ideal hash function that maps $(0,1)^* \rightarrow (0,1)^\sigma$. For any distinct query, RO outputs a random $\sigma-$bit string, which cannot be predicted. For the hash function we used, $H(t_0 \oplus a\Delta || t_1 \oplus b\Delta || ... || t_7 \oplus h\Delta)$ must be indistinguishable from a random σ-bit string for any randomly chosen values of $\{t_0, t_1 ... t_7\}$, $\{a, b, c ... h\}$.

5.1.2. Semi-Honest Model

The adversary in the semi-honest model strictly follows the protocol but tries to learn more information from the message it receives. For a protocol π: $(x, y) \rightarrow (f_{P_0}(x, y), f_{P_1}(x, y))$, P_0 inputs x and outputs $f_{P_0}(x, y)$, while P_1 inputs y and outputs $f_{P_1}(x, y)$. The security for party P_i can be proven if there is a simulator that can use P_{1-i}'s input and output to compute the complete view of P_i. That is to say, adversary $\mathcal{A}$ cannot distinguish the distribution of the simulator's view $View^\pi_\mathcal{S}(x, y)$ and the real party P_i's view $View^\pi_{P_i}(x, y)$ in polynomial time, except for a negligible possibility. $View^\pi_{P_i}(x, y)$ consists of its input, output, randomness, and all messages it receives.

5.2. Security Analysis of Our Garbling Scheme for S-Boxes

Intuitively speaking, the security of our garbling scheme is totally based on the security of free XOR [15]. The only difference is that we use input wire labels to encrypt garbled fbss instead of output wire labels. We now analyze whether there is any information leakage when the evaluator decrypts a garbled fbs. Based on the security analysis in [15], the evaluator can only decrypt one of the ciphertexts, while the rest of the ciphertexts look random in its view. Therefore, the real fbs is equal to being encrypted by a one-time pad r, and the evaluator cannot learn what the real fbs is.

In the following, we demonstrate the simulation process for our garbling scheme. Instead of showing the simulation of the whole garbling scheme, we mainly focus on the process of garbling the S-box, where the evaluator inputs the eight input wire labels of the S-box and the generator inputs 2^8 real fbss. Finally, the evaluator outputs a garbled fbs, and the generator outputs nothing. For the security requirement, the evaluator cannot learn the real fbs. Thus, we only need to simulate the view of the evaluator.

The simulator $\mathcal{S}$ randomly samples $2^8 - 1$ σ-bit strings as the rest of the garblings for the S-box. Upon receiving the input wire labels and the garbled fbss from the evaluator, $\mathcal{S}$ encrypts the garbled fbss using the input wire labels. $\mathcal{S}$ arranges the ciphertexts in the corresponding position according to the lsbs of the input wire labels and outputs garblings F_S such that $View^\pi_\mathcal{S}(x, y) = F_S$.

Theorem 1. *For any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*:*

$$\begin{aligned} &\left|\Pr[\mathcal{A}^\mathcal{L}(View^\pi_\mathcal{S}(x, y)) = 1]\right| - \\ &\left|\Pr[\mathcal{A}^\mathcal{L}(View^\pi_E(x, y)) = 1]\right| < \varepsilon(\sigma) \end{aligned} \tag{1}$$

where ε *is a negligible function and* σ *is the security parameter.*

Proof. First, the encrypted garbled fbss are the same in the views of both $\mathcal{S}$ and the evaluator. Furthermore, $H(t_0 \oplus a\Delta||t_1 \oplus b\Delta||...||t_7 \oplus h\Delta)$ is indistinguishable from a random σ-bit string for any randomly chosen values of $\{t_0, t_1...t_7\}, \{a, b, c...h\}$. Thus, the rest of the garblings F in $View^{\pi}_{E}(x, y)$ are indistinguishable from the corresponding garblings F_S in $View^{\pi}_{S}(x, y)$. Finally, we can conclude that $View^{\pi}_{S}(x, y)$ is indistinguishable from $View^{\pi}_{E}(x, y)$. □

6. Experiment

In this section, we experimentally implement our garbling scheme. Our platform is an R7-7840HS at 3.80GHz running on Windows 11. We write our codes in C++. We do not use any hardware description language to instantiate the AES circuit. Instead, we construct the topology structure of the AES circuit using C++ objects. To reuse the circuit units, we use the circuit layer model introduced in Section 3.4.

6.1. Gate Class

In our implementation, we create a **Gate** class to maintain all gate objects. Each gate object contains an output wire and a number of input wires, and the gate shares the same index with its output wire. For a gate with only one input wire, the value on the output wire equals that on the input wire. For a gate with more than one input wire, the value on the output wire equals the XOR of the values of all input wires.

6.2. AES Circuit

We demonstrate the construction of the MixColumn, ShiftRow, AddRoundKey, and SubBytes layers in detail. As shown in Figure 3, each circuit layer is connected between an input register and an output register. These registers are composed of 128 gate objects. Each group of eight gates, sequentially arranged from 0 to 127, constitutes a byte within the AES state matrix, as illustrated in Figure 6.

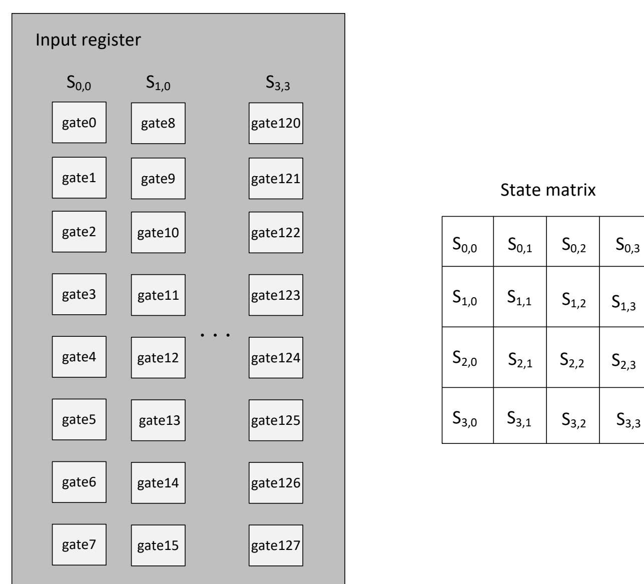

Figure 6. Input/output register.

6.2.1. ShiftRow Layer

The ShiftRow operation only involves the conversion of positions between individual bytes, so we only need to concatenate the gate in the input register with the corresponding gate in the output register according to the shift rule.

Here, we give the concatenation rule (Figure 7) between the input register and the output register. igate[i] denotes the i-th gate of the input register, ogate[i] denotes the i-th gate of the output register, and $\leftarrow$ denotes the concatenation.

```
for (int i = 0; i < 4; i++)
{
      for (int j = 0; j < 4; j++)
      {
            for (int z = 0; z < 8; z++)
            {
                  ogate[z + j * 8 + i * 32] ← igate[z + j * 8 + (i+j mod 4) * 32];
            }
      }
```

Figure 7. Concatenation rule in the ShiftRow layer.

6.2.2. MixColumn Layer

The core computation of the MixColumn layer involves performing a matrix multiplication as follows:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{2,3} \end{pmatrix} = \begin{pmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{2,3} \end{pmatrix}$$

where the computation of a single column is

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

Here, the multiplication is over GF(2^8). The computation above is equivalent to

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (2 \cdot s_{1,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (2 \cdot s_{2,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (2 \cdot s_{3,j}) \oplus s_{3,j} \\ s'_{3,j} &= (2 \cdot s_{0,j}) \oplus s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

Furthermore, the operation 02 that multiplies a byte x can be divided into

$$\begin{aligned} &y_7 = x_6, \quad y_6 = x_5, \; y_5 = x_4, \quad y_4 = x_3 \oplus x_7, \\ &y_3 = x_2 \oplus x_7, \; y_2 = x_1, \; y_1 = x_0 \oplus y_7, \; y_0 = x_7 \end{aligned}$$

where $\{02\} \cdot x = y$, $x = x_7x_6x_5x_4x_3x_2x_1x_0$, $y = y_7y_6y_5y_4y_3y_2y_1y_0$.

After the decomposition of the computation, the MixColumn layer can be executed only by the XOR gate. We show the MixColumn layer in Figure 8, where Xtimes executes the $\{02\} \cdot x$ computation, and 5wayXOR is a gate with five input wires, executing the XOR operation. The construction of Xtimes and 5wayXOR is shown in Figure 9.

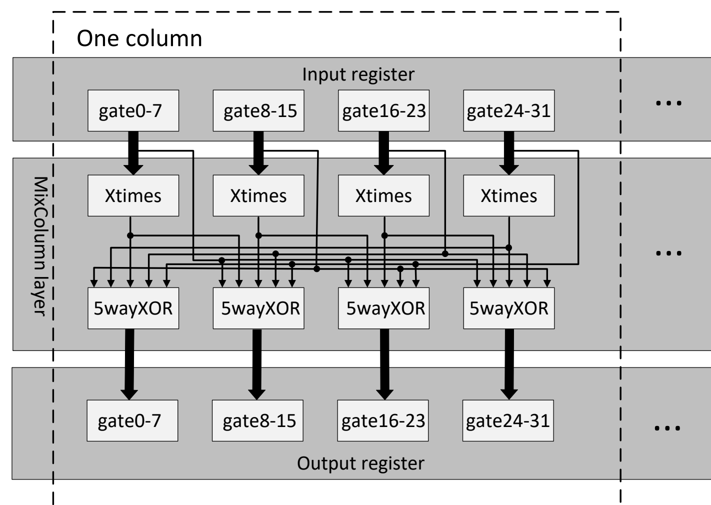

Figure 8. MixColumn layer.

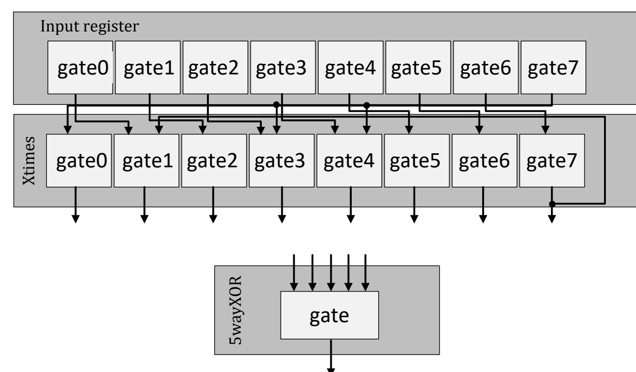

Figure 9. Xtimes and 5wayXOR.

6.2.3. AddRoundKey Layer

The core computation of the AddRoundKey layer is the XOR operation between two bytes. Except for the input register, which stores the state matrix, an extra 128-bit register is needed for the generator to input the round key. It is important to note that the key expansion algorithm can be executed locally by the generator. The AddRoundKey layer is shown in Figure 10.

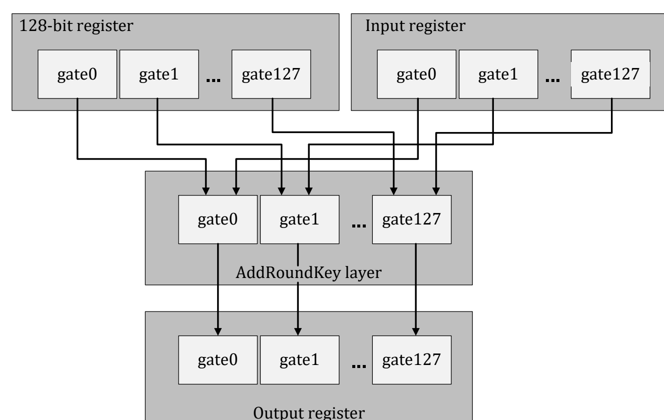

Figure 10. AddRoundKey layer.

6.3. Performance Evaluation

In our experimental setup, we use $\sigma = 256$-bit wire labels and instantiate H as SHA-256 (the output length is 256 bits). The performance of our garbling scheme is shown in Table 5, which shows the respective runtimes of the two parties in each circuit layer. We disregard the cost of OTs for the wire label transmission from the generator to the evaluator.

Table 5. The runtime performance of our garbling scheme for AES circuits.

			Evaluator		
	AddRoundKey	ShiftRow	MixColumn	SubBytes(the last)	Overall online time
Time ($\times 10^{-6}$ s)	2.8	2.2	4.1	83	3171
			Generator		
	AddRoundKey	ShiftRow	MixColumn	SubBytes(the last)	Overall offline time
Time ($\times 10^{-6}$ s)	2.1	1.4	0.5	21,248	788,061

On the generator's side, the main cost includes two parts: (1) producing the output wire labels for each XOR gate, which includes the XOR operation between the input wire labels and offset value, and (2) producing the ciphertexts for the S-boxes. Both parts can be executed completely offline. On the evaluator's side, the main cost in the AddRoundKey, ShiftRow, and MixColumn layers is the XOR operation between the input wire labels to compute the output wire labels. In the SubBytes layer, the main cost is the hash function invocations, which must be executed online. Therefore, fewer hash calls lead to better online performance of the protocol.

We also compare our garbling scheme with schemes in previous works (the data are derived from Table 1 in [22]). The results (Table 6) suggest that our garbling scheme has minimal online time due to the fewer calls for the hash function. However, the overall time increases, which we believe is mainly because we do not use any hardware description language to instantiate the AES circuit. Furthermore, since it is impossible to reproduce the scheme in [22], and the implementation platform is also different, this comparison can only be used as a general reference to show that the runtime performance of the proposed scheme is comparable with the state of the art.

Table 6. Comparison between our garbling scheme and schemes in previous works.

Scheme	Online Time (s)	Overall Time (s)
[31]	0.4	3.3
[22]	0.008	0.2
Ours	0.003	0.8

7. Conclusions

In conclusion, taking into account the special structure of the S-box and the AES circuit, this paper proposes a garbling scheme for S-boxes in the final SubBytes layer, which further reduces the ciphertext size of secure AES computation. Compared to the best result in previous works, which requires 2048 ciphertexts and 4 hash calls (minimal online time) or 116(174) ciphertexts and 116(58) hash calls (minimal total time), our garbling scheme only requires 256 ciphertexts and 1 hash call. In addition, if the optimized S-box structure is used instead of the original AES S-box to enhance security, it would increase the number of ciphertexts required by the minimum total time scheme, unlike our scheme.

In our implementation, we introduce the circuit layer model to reuse circuit units in the AES circuit, where each algorithm is designed into a circuit layer, and only one copy is stored in memory. Finally, we demonstrate the construction of each circuit layer and experimentally evaluate the performance. The experimental data show that our garbling scheme achieves better online performance compared to schemes in previous works. However, the non-optimal overall time may be due to the fact that we did not use any hardware description language to implement the AES circuit.

To be honest, the extensibility of the proposed garbling scheme in this paper is relatively limited. In future work, we will focus on applying the proposed garbling scheme to all SubBytes layers to further improve the efficiency of secure AES computation. Future work will also focus on extending the idea of the ciphertext reduction scheme proposed in this paper to universal Yao's garbled circuits.

Author Contributions: Conceptualization, X.Y. and B.L.; methodology, X.Y.; software, X.Y.; validation, X.Z. and J.C.; formal analysis, X.W.; investigation, Y.Y. and F.W.; resources, B.L. and K.C.; writing—original draft preparation, X.Y. and F.W.; writing—review and editing, X.W. and Y.Y.; visualization, K.C.; supervision, J.C. and F.W.; project administration, B.L. and J.C.; funding acquisition, B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant 61972350, 61972124, 11974096; and in part by the Zhejiang Provincial Natural Science Foundation of China under Grant No. LY23F020013 and Zhejiang Provincial basic public welfare research project of China (No. LGG22F030019) and Ningbo City's Key Technology Breakthrough Plan for "Science and Technology Innovation Yongjiang 2035" (No. 2024Z261) and the Ningbo 2025 Major Project of Science and Technology Innovation under Grant 2021Z109, 2020Z021, 2021Z010, 2023Z040 and Major Technological Innovation Projects of Ningbo High tech Zone (No. 2022BCX050002).

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to (specify the reason for the restriction).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Flip Bit String for All Possible Inputs of the S-Box

Figure A1 shows the corresponding fbs for each possible input of the AES S-box. The red text indicates that the input and fbs are injective, and the number characterizes the security of our garbling scheme.

flip bit string

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	01100011	01111101	01110101	01111000	11110110	01101110	01101001	11000010	00111000	00001000	01101101	00100000	11110010	11011010	10100101	01111001
1	11011010	10010011	11011011	01101110	11101110	01001100	01010001	11100111	10110101	11001101	10111000	10110100	10000000	10111001	01101100	11011111
2	10010111	11011100	10110001	00000101	00010010	00011010	11010001	11101011	00011100	10001100	11001111	11011010	01011101	11110101	00011111	00111010
3	00110100	11110110	00010001	11110000	00101100	10100011	00110011	10101101	00111111	00101011	10111010	11011001	11010111	00011010	10001100	01001010
4	01001001	11000010	01101110	01011001	01011111	00101011	00011100	11100111	00011010	01110010	10011100	11111000	01100101	10101110	01100001	11001011
5	00000011	10000000	01010010	10111110	01110100	10101001	11100111	00001100	00110010	10010010	11100100	01100010	00010110	00010001	00000110	10010000
6	10110000	10001110	11001000	10011000	00100111	00101000	01010101	11100010	00101101	10010000	01101000	00010100	00111100	01010001	11110001	11000111
7	00100001	11010010	00110010	11111100	11100110	11101000	01001110	10000010	11000100	11001111	10100000	01011010	01101100	10000010	10001101	10101101
8	01001101	10001101	10010001	01101111	11011011	00010010	11000010	10010000	01001100	00101110	11110100	10110110	11101000	11010000	10010111	11111100
9	11110000	00010000	11011101	01001111	10110110	10111111	00000110	00011111	11011110	01110111	00100010	10001111	01000010	11000011	10010101	01000100
A	01000000	10010011	10011000	10101001	11101101	10100011	10000010	11111011	01101010	01111010	00000110	11001001	00111101	00111000	01001010	11010110
B	01010111	01111001	10000101	11011110	00111001	01100000	11111000	00011110	11010100	11101111	01001110	01010001	11011001	11000111	00010000	10110111
C	01111010	10111001	11100111	11101101	11011000	01100011	01110010	00000001	00100000	00010100	10111110	11010100	10000111	01110000	01000101	01000101
D	10100000	11101111	01100111	10110101	10011100	11010110	00100000	11011001	10111001	11101100	10001101	01100010	01011010	00011100	11000011	01000001
E	00000001	00011001	01111010	11110010	10001101	00111100	01101000	01110011	01110011	11110111	01101101	00000010	00100010	10111000	11000110	00110000
F	01111100	01010000	01111011	11111110	01001011	00010011	10110100	10011111	10111001	01100000	11010111	11110100	01001100	10101001	01000101	11101001

Figure A1. fbs lookup table.

References

1. Yao, A.C.C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986), Toronto, ON, Canada, 27–29 October 1986; IEEE: Piscataway, NJ, USA, 1986; pp. 162–167. [CrossRef]
2. Huang, Y.; Shen, C.H.; Evans, D.; Katz, J.; Shelat, A. Efficient secure computation with garbled circuits. In Proceedings of the Information Systems Security: 7th International Conference, ICISS 2011, Kolkata, India, 15–19 December 2011; Procedings 7; Springer: Berlin/Heidelberg, Germany, 2011; pp. 28–48. [CrossRef]
3. Mohassel, P.; Riva, B. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In Proceedings of the Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013; Proceedings, Part II; Springer: Berlin/Heidelberg, Germany, 2013; pp. 36–53. [CrossRef]
4. Frederiksen, T.K.; Nielsen, J.B.; Orlandi, C. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 191–219. [CrossRef]

5. Pinkas, B.; Schneider, T.; Smart, N.P.; Williams, S.C. Secure two-party computation is practical. In Proceedings of the Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, 6–10 December 2009; Proceedings 15; Springer: Berlin/Heidelberg, Germany, 2009; pp. 250–267. [CrossRef]
6. Pinkas, B.; Schneider, T.; Zohner, M. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur. (TOPS)* **2018**, *21*, 1–35. [CrossRef] [CrossRef]
7. Pinkas, B.; Schneider, T.; Zohner, M. Faster private set intersection based on {OT} extension. In Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA, 20–22 August 2014; pp. 797–812. [CrossRef]
8. Rindal, P.; Schoppmann, P. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, 17–21 October 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 901–930. [CrossRef]
9. Chor, B.; Kushilevitz, E.; Goldreich, O.; Sudan, M. Private information retrieval. *J. ACM (JACM)* **1998**, *45*, 965–981. [CrossRef] [CrossRef]
10. Chor, B.; Gilboa, N.; Naor, M. Private information retrieval by keywords. *Citeseer* **1997**. [CrossRef]
11. Freedman, M.J.; Ishai, Y.; Pinkas, B.; Reingold, O. Keyword search and oblivious pseudorandom functions. In Proceedings of the Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, 10–12 February 2005; Proceedings 2; Springer: Berlin/Heidelberg, Germany, 2005; pp. 303–324. [CrossRef]
12. Yang, Y.; Lu, H.; Weng, J. Multi-user private keyword search for cloud computing. In Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Athens, Greece, 29 November–1 December 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 264–271. [CrossRef]
13. Lian, B.; Cui, J.; Chen, H.; Zhao, X.; Wang, F.; Chen, K.; Ma, M. Trusted Location Sharing on Enhanced Privacy-Protection IoT Without Trusted Center. *IEEE Internet Things J.* **2024**, *11*, 12331–12345. [CrossRef] [CrossRef]
14. Kiss, Á.; Liu, J.; Schneider, T.; Asokan, N.; Pinkas, B. Private set intersection for unequal set sizes with mobile applications. *Proc. Priv. Enhancing Technol.* **2017**, *2017*, 177–197. [CrossRef] [CrossRef]
15. Kolesnikov, V.; Schneider, T. Improved garbled circuit: Free XOR gates and applications. In Proceedings of the Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, 7–11 July 2008; Proceedings, Part II 35; Springer: Berlin/Heidelberg, Germany, 2008; pp. 486–498. [CrossRef]
16. Artuğer, F.; Özkaynak, F. A new post-processing approach for improvement of nonlinearity property in substitution boxes. *Integration* **2024**, *94*, 102105. [CrossRef] [CrossRef]
17. Sokolov, A.; Radush, V. A method for synthesis of S-boxes with good avalanche characteristics of component Boolean and quaternary functions. *J. Discret. Math. Sci. Cryptogr.* **2022**, *26*, 561–572. [CrossRef] [CrossRef]
18. Khan, H.; Hazzazi, M.M.; Jamal, S.S.; Hussain, I.; Khan, M. New color image encryption technique based on three-dimensional logistic map and Grey wolf optimization based generated substitution boxes. *Multimed. Tools Appl.* **2023**, *82*, 6943–6964. [CrossRef] [CrossRef]
19. Alali, A.S.; Ali, R.; Jamil, M.K.; Ali, J.; Gulraiz. Dynamic S-Box Construction Using Mordell Elliptic Curves over Galois Field and Its Applications in Image Encryption. *Mathematics* **2024**, *12*, 587. [CrossRef] [CrossRef]
20. Ali, J.; Jamil, M.K.; Alali, A.S.; Ali, R. A medical image encryption scheme based on Mobius transformation and Galois field. *Heliyon* **2024**, *10*, e23652. [CrossRef] [CrossRef] [PubMed]
21. Ali, R.; Jamil, M.K.; Alali, A.S.; Ali, J.; Afzal, G. A robust S box design using cyclic groups and image encryption. *IEEE Access* **2023**, *11*, 135880–135890. [CrossRef] [CrossRef]
22. Huang, Y.; Evans, D.; Katz, J.; Malka, L. Faster secure Two-Party computation using garbled circuits. In Proceedings of the 20th USENIX Security Symposium (USENIX Security 11), San Francisco, CA, USA, 8–12 August 2011. [CrossRef]
23. Zahur, S.; Rosulek, M.; Evans, D. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In Proceedings of the Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015; Proceedings, Part II 34; Springer: Berlin/Heidelberg, Germany, 2015; pp. 220–250. [CrossRef]
24. Bellare, M.; Hoang, V.T.; Keelveedhi, S.; Rogaway, P. Efficient Garbling from a Fixed-Key Blockcipher. In Proceedings of the 2013 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, 19–22 May 2013; pp. 478–492. [CrossRef]
25. Naor, M.; Pinkas, B.; Sumner, R. Privacy preserving auctions and mechanism design. In Proceedings of the 1st ACM Conference on Electronic Commerce, Denver, CO, USA, 3–5 November 1999; pp. 129–139. [CrossRef]
26. Beaver, D.; Micali, S.; Rogaway, P. The round complexity of secure protocols. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 14–16 May 1990; pp. 503–513. [CrossRef]
27. Kolesnikov, V.; Mohassel, P.; Rosulek, M. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Proceedings of the Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2014; Proceedings, Part II 34; Springer: Berlin/Heidelberg, Germany, 2014; pp. 440–457. [CrossRef]
28. Rosulek, M.; Roy, L. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Proceedings of the Annual International Cryptology Conference, Virtual Event, 16–20 August 2021; Springer: Berlin/Heidelberg, Germany, 2021; pp. 94–124. [CrossRef]
29. Malkhi, D.; Nisan, N.; Pinkas, B.; Sella, Y. Fairplay-Secure Two-Party Computation System. In Proceedings of the USENIX Security Symposium, San Diego, CA, USA, 9–13 August 2004; Volume 4, p. 9. [CrossRef]

30. Zahur, S.; Evans, D. Obliv-C: A Language for Extensible Data-Oblivious Computation. *Cryptol. Eprint Arch.* **2015**, 1153. Available online: https://eprint.iacr.org/2015/1153 (accessed on 30 November 2015). [CrossRef]
31. Henecka, W.; Kögl, S.; Sadeghi, A.R.; Schneider, T.; Wehrenberg, I. TASTY: Tool for automating secure two-party computations. In Proceedings of the 17th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 4–8 October 2010; pp. 451–462. [CrossRef]
32. Naor, M.; Pinkas, B. Oblivious transfer and polynomial evaluation. In Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, Dallas, TX, USA, 23–26 May 1999; pp. 245–254. [CrossRef]
33. Naor, M.; Pinkas, B. Efficient oblivious transfer protocols. In Proceedings of the SODA, Washington, DC, USA, 7–9 January 2001; Volume 1, pp. 448–457. [CrossRef]
34. Bellare, M.; Hoang, V.T.; Rogaway, P. Foundations of garbled circuits. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 784–796. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.