# Perfect/Statistical 2-Party Computation with Improved Circuit Privacy

Jonathan Trostle

Independent Consultant

**Abstract.** We present protocols where one entity, the server, evaluates a circuit with encrypted inputs from the second party, the client. We give secret key somewhat additive homomorphic schemes where the client has perfect privacy (server is computationally unbounded). Our scheme is somewhat additive homomorphic and we extend it to support multiplication. The server handles circuit multiplication gates by sending the multiplicands to the client which does the multiplication and updates the decryption key so that the original ciphertext vector includes the encrypted multiplication gate outputs. The key idea for client privacy is the permutation table which consists of rows of vectors modulo a prime integer $m$. The initial row is $(1, d_2, \ldots, d_c)$ where $d_{i-1}|d_i$, $d_i > N(a+1)d_{i-1}$, for an integer $N$ which is a power of 2 and integer $a$, $2 \leq i \leq c$. Subsequent rows are integer multiples of the first row, modulo $m$. The permutation table has a subset of rows (vectors) whose components are relatively short (facilitating addition without overflowing $m$) and which map to every possible vector modulo $N$ (giving perfect privacy since every plaintext vector is possible given a ciphertext vector from the table.)

We give a 2-party computation (2PC) protocol that also incorporates server inputs where the client has perfect privacy, assuming the server is semi-honest in the setup protocol. Client privacy is statistical if the server is malicious during the setup protocol. Server privacy only holds against a computationally bounded adversary since it depends on the hardness of a variant of the HSSP (Hidden Subset Sum Problem) and the DDH (Decisional Diffie Hellman Assumption). We leverage the Castagnos Laguillaumie linear homomorphic public key encryption for setup. The 2PC protocol maintains circuit privacy, without circuit expansion, except for leaking the number of AND gates to the client. Scaling the 2PC protocol via separate encryption parameters for smaller subcircuits allows the ciphertext size to remain constant as circuit size grows.

## 1 Introduction

Homomorphic encryption schemes with respect to a single operation (addition or multiplication) include Goldwasser-Micali, Paillier [36],and textbook RSA [38]. Rivest et al. posed the question of homomorphic encryption [37] in 1978. Gentry's breakthrough work [21] presented a fully homomorphic scheme and accelerated

the study of homomorphic schemes that can compute arbitrary functions in a model with circuits that have both addition and multiplication gates.

Gentry's scheme along with follow-up work [18], [8], [9], [3], [19] features schemes where security depends on computationally hard problems in lattices or number theory (e.g. lattice SVP, approximate GCD problem). The security of these schemes may be affected by advances in algorithms to more efficiently solve these problems.

In this work, we present somewhat additive homomorphic encryption schemes (a limited number of ciphertexts can be added and decryption will still obtain the sum of the plaintexts, see Appendix A for the definition). These schemes are the basis for our protocols that provide perfect privacy for one party, which we call the client. The other party that evaluates the circuit given encrypted inputs from the client is called the server. One use case is when the server has a proprietary algorithm (the circuit) that is not to be shared with the client. Our schemes are secret key based and provide security for the client against a computationally unbounded attacker. A new secret key $b$ is generated for each client request; since the key is not shared with the server this one-time key use does not create key management overhead.

Our basic scheme includes a prime modulus $m$, key $b$, and exponents $e_i$ corresponding to each client plaintext input for the circuit. The ciphertext tuple consists of the vector $(be_1 \bmod m, \ldots, be_c \bmod m)$. For encrypting bits, the parity of each $e_i$ determines the plaintext bit. The size of the $e_i's$ is bounded using the max norm so that the client can decrypt the returned result by multiplying by $b^{-1} \bmod m$. This scheme has two interesting properties:

1. it's additive homomorphic
2. as illustrated in Figure 1, it allows distinct (plaintext key) pairs to map to the same ciphertext. (The key is $b$.)

The 2nd property is the basis for perfect privacy. Our main results for somewhat additive homomorphic encryption are Theorem 1 and Theorem 4 which prove that our Perfect Somewhat Additive Homomorphic Encryption (P-SWAHE) scheme has perfect privacy.

A key concept is the permutation table which consists of rows which are integer multiples modulo $m$ of the first row. The ciphertext vector $(be_1 \bmod m, \ldots, be_c \bmod m)$ is a row in the table as is the vector $(e_1, \ldots, e_c)$ (the e-vector). The e-vector is selected from the set of short vectors in the permutation table and these vectors map (for the bit case the map is the parity of each e-vector component) to every possible binary plaintext vector. A short vector's components sum to an integer less than $m$. See Section 2.1 for an example. Thus encryption is selection of a plaintext vector $(s_1, \ldots, s_c)$ (see Figure 3), mapping the plaintext vector to an e-vector $(e_1, \ldots, e_c)$ such that $e_i \bmod N = s_i, 1 \leq i \leq c$, and selection of a uniform random one-time secret key $b$ to obtain ciphertext vector $(be_1 \bmod m, \ldots, be_c \bmod m)$. For bits, $N = 2$.

We extend P-SWAHE to handle multiplication gates (Section 3.3). In P-SWAHE with multiplication, the server returns the multiplicands to the client

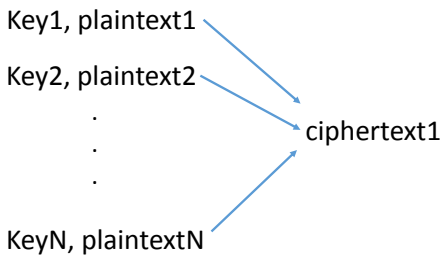



**Fig. 1.** Distinct Plaintext Key Pairs Map to Same Ciphertext

which updates the secret key so that decryption produces the correct output. By leveraging the concept from Figure 1, we are able to modify the underlying secret key and plaintext on the client without any change to the original ciphertext sent to the server. The modified plaintext then includes the outputs for each of the circuit multiplication gates. The additions and multiplications of the underlying plaintexts are modulo $N$ where $N$ is a power of 2. The P-SWAHE with multiplication protocol provides perfect privacy for the client. Subsequent requests from the client can be processed locally on the client given the stored state from the initial reply from the server. Thus the server reply can be viewed as an encoding of the circuit. This circuit encoding may be of independent interest. P-SWAHE with multiplication is a non-compact perfect security Somewhat Homomorphic Encryption algorithm.

We give a 2-party MPC (2PC) scheme (see Algorithm 1) where client privacy is perfect or statistical and server privacy is based on the hardness of a variant of the Hidden Subset Sum Problem (HSSP) (see Section 2.4) and the Decisional Diffie Hellman (DDH). We assume a semi-honest client while the server can be malicious. The server processes addition (XOR) gates on its own but leverages the client's help to process multiplication (AND or NAND) gates. The server sends two elements to the client where each element is computed as a multiplicand plus a hidden subset sum of noise elements. The client can compute four products from the two elements sent by the server and then the client sends the indices of the products in the encrypted vector to the server. The server is able to compute the encrypted vector from the first component of the vector which is sent by the client during the setup phase. The server selects the correct product from the four possible indices. Thus the server evaluates the circuit obliviously except for knowledge of the circuit and its own inputs. We cover these ideas in more detail in Section 2.2.

Intuitively, our 2PC protocol client security is based on the multiple plaintext key combinations mapping to the same ciphertext vector discussed above, Server security is based on the random noise (hidden subset sums) added to each

multiplicand and output sent by the server to the client. Thus the client cannot deduce the plaintexts in the actual multiplicands or the ciphertext row elements.

The setup protocol for the 2PC scheme leverages Castagnos Laguillaumie (CL) linear homomorphic public key encryption [11], [10]. To simplify, the server encrypts the first component of its portion of the e-vector, $e_s$, with its public key and sends it to the client. The client encrypts the first component of its portion of the e-vector, $e_c$, with the server public key and creates a new ciphertext that will decrypt to $b(e_s + e_c) \bmod m$ where $b$ is the secret key. The server decrypts to obtain $b(e_s + e_c) \bmod m$. The server and client can now execute the circuit evaluation protocol since the server can compute the other elements of the ciphertext vector. Although the permutation table can be very large, the server only stores a single element from the table and the client stores the plaintext bits for the multiplication elements (part of a single row in the table). For setup, the client $e_c$ value is protected by the client's secret key $b$, and server $e_s$ is protected by the server CL encryption private key.

There exist 2-party schemes [30], [4] where one party enjoys statistical privacy and the other party is protected from a computationally bounded adversary. Combining GMW [24] with statistical private OT (Oblivious Transfer) for one party also achieves the same security property. Our 2PC protocol provides perfect or statistical privacy for one party and computational security for the other party. It also provides limited circuit privacy (number of AND gates and approximate number of non-AND gates is leaked) without circuit expansion whereas the above approaches achieve circuit privacy with a large blow up in circuit size by using a universal circuit.

In our protocol, client privacy is perfect and protected unconditionally if the server is semi-honest. If the server is in the malicious model, client privacy is protected if the client does not share the output with the server, but the malicious model server can also execute a low probability guessing attack during setup against client input privacy. Thus client perfect security when the server is malicious requires that the server is semi-honest during the setup protocol, else client security is only protected statistically (see Theorem 12.)

### 1.1 Our Contributions

We have the following results:

1. We give a secret key somewhat additive homomorphic encryption algorithm (P-SWAHE) that provides perfect client privacy, and extend this to include homomorphic multiplication. We refrain from calling our protocol somewhat homomorphic encryption since it is not compact. We present an extension which allows subsequent requests to be satisfied on the client without interaction with the server based on the stored multiplicands from the initial client request. An unanswered question is whether the server's circuit can be reconstructed from the stored multiplicands. The resulting protocol protects the client privacy from a computationally unbounded server. To the best of our knowledge, our somewhat additive homomorphic encryption is the first

such algorithm that provides perfect privacy. The use case for P-SWAHE is when the server has a proprietary circuit that is not to be shared with the client and the client desires perfect privacy, or as part of the 2PC protocol below.

2. We show that P-SWAHE has perfect privacy in Theorem 1, Theorem 4, Theorem 6, and Theorem 8.
3. Based on the somewhat additive homomorphic encryption, we construct a 2-party protocol (2PC) where both the client and server provide inputs and the server processes the encrypted inputs through the circuit returning the output to the client. The 2PC protocol properties are
   - client semi-honest and computationally bounded
   - if server semi-honest, client has perfect privacy
   - if server malicious, client has statistical privacy
   - server privacy requires DDH and DPT-HSSP assumptions
   - correctness not assured if server is malicious
   - server has circuit privacy except for leaking number of AND gates

   We show that this protocol provides perfect client privacy (Theorems 10 and 11.) Client privacy is protected from a malicious server if the server does not receive output (Theorem 10) (but there is a low probability guessing attack that a malicious server can execute during setup, see Section 4.2). Thus perfect client privacy requires the server to follow the setup protocol, otherwise client security is statistical (Section 4.4 Theorem 12.)
4. We show the 2PC protocol protects server privacy assuming hardness of our Decisional Permutation Table HSSP (DPT-HSSP) and DDH given a semi-honest client. We show that solving Permutation Table HSSP implies an adversary with non-negligible advantage against CPA security of CL encryption given the initial setup. Thus we conjecture that DPT-HSSP is hard as well (see Section 4.1).
5. Our 2PC protocol provides limited circuit privacy for the party owning the circuit (our server) since only the number of AND gates and approximate number of non-AND gates are known to the client. Our results are a step towards efficient 2PC with statistical or perfect security for one party and circuit privacy for the other party. Unlike existing approaches we do not require a universal circuit ($3 \times n \log(n)$ resulting circuit size when the original circuit has $n$ gates [31].)

### 1.2 Related Work

There has been extensive work in homomorphic encryption since the breakthrough work of Gentry [21] (e.g., [22], [23], [18], [8], [9], [3], [19].

Foundational work in multiparty computation includes [40], [24], [5], [12], [17].

The general case for 2-party perfect security where both parties are computationally unbounded and semi-honest is resolved in [13] and [32]. [29] shows that perfect security against a malicious adversary is possible in the 2-party model given correlated randomness where the output is delivered to only one party.

[2] characterizes perfect security for 2-party symmetric functionalities with up to four possible outputs in the correlated randomness and plain models, given malicious adversaries.

Dakshita and Mughees [30] give 2-party secure protocols where one party is statistically secure and the other party is computationally secure. Their protocols use garbled circuits [40] in combination with Oblivious Transfer to obtain a 5 round protocol. Our 2PC protocol provides limited circuit privacy (number of AND gates and approximate number of non-AND gates is leaked) whereas [30] exposes the circuit to both parties.

Badrinarayanan [4] allows the results of [30] to be based on additional assumptions such as CDH. Acharya [1] generalizes [30] from 2 parties to $n$ parties; one party can be protected with statistical security and their fallback security provides computational security for the other parties.

Rothblum [39] gives constructions for building public key encryption from additively homomorphic secret key encryption. These constructions do not apply to our scheme since they require $l = 4m$ or $l = 8m$ where $m$ is the modulus bit length. Thus security for the Rothblum constructions requires the underlying secret key encryption to be secure when the number of ciphertexts for a given key exceeds the bit length of the homomorphically evaluated ciphertext. For security, our secret key scheme requires that the number of original ciphertexts (parameter $c$ in our scheme) is less than the bit length of $m$ where $m$ is the modulus in our scheme (see Remark 3). Thus the Rothblum constructions do not apply to our scheme.

The Hidden Subset Sum Problem (HSSP) was discussed in [34] and an algorithm to solve it for moderate number of weights, $n$, where the number of samples is $2n$ was given. In [14] and [15] algorithms are given to solve HSSP in polynomial time given a larger number of samples ($n^2$). Our work will limit the number of samples $w$ to less than the number of hidden weights.

## 2 Technical Overview

We give an overview of (P-SWAHE) client privacy and 2PC server privacy. We then discuss preliminaries, CL encryption and our security assumptions.

### 2.1 Client Privacy

We now give an example to motivate and briefly illustrate our ideas for client privacy. Client privacy is perfect if the server is semi-honest and statistical if the server is malicious.

Suppose we desire to homomorphically evaluate the circuit C in Figure 2. C has two addition gates and a single output; the output is obtained by adding (over the field $\mathbb{Z}_2$) the three input bits.

The client will send a ciphertext vector with 3 components to the server which will use the ciphertext components as inputs into the circuit. The server

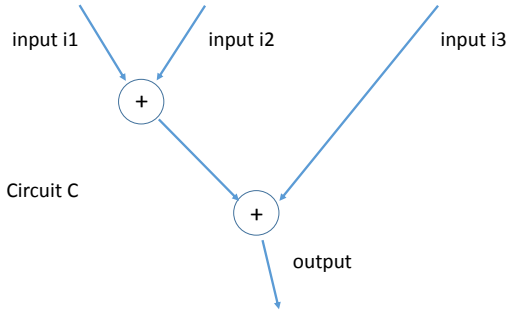



**Fig. 2.** Circuit C with Two Addition Gates

returns the (encrypted) output of the circuit to the client which decrypts to obtain the one bit output of the circuit.

We will show an additive homomorphic encryption algorithm for the circuit C that has perfect security. Consider the **permutation table** $T$ (see Definition 3) in Table 1, with $m-1=96$ rows and $c=3$ columns, where each column is a permutation of the nonzero elements in the additive group $\mathbb{Z}_{97}$. The element in row $i$ and column $j$ is $id_j \bmod m$ where $d_1=1$, $d_2=4$, and $d_3=16$; we take representatives to be integers between 1 and 96. There are other possible choices for the $d_i$ (the column start elements) but one property is that $d_i|d_{i+1}$ which results in the table having some short vectors (components sum to less than $m$.) (Note there are 3 columns of the permutation table in each column of Table 1 and the $ith$ column of the permutation table is the concatenation of the $ith$ columns in each of the 6 columns of Table 1, $1 \leq i \leq 3$.)

We associate with each three component vector in the permutation table a three component associated binary vector where each component is the parity (least significant bit) of the corresponding component in the table. For example, the associated binary vector for the first row vector $(1,4,16)$ is $(1,0,0)$.

We define a distinguished subset of vectors in the permutation table as the **target set of short vectors**; these are the bold faced vectors in Table 1 (see Definition 4). This set has two properties:

1. The vectors are short (the sum of the row components is less than $m=97$.) This property facilitates decryption.
2. The associated binary vectors for this set is the set of all possible binary vectors in 3 components (see Table 2).

We now describe encryption and decryption for this example:

1. The plaintext is a 3 component binary vector. Suppose the plaintext is $(1,1,0)$. The client selects the corresponding target set vector (see Table 2). In this case the target set vector is $(25,3,12)$. The general case is Algorithm 2.

**Table 1.** Permutation Table $T$ for $m = 97$, $c = 3$, $d_1 = 1$, $d_2 = 4$, $d_3 = 16$; Target Set Vectors are Bolded

| | | | | | |
|---|---|---|---|---|---|
| **1 4 16** | 17 68 78 | 33 35 43 | 49 2 8 | 65 66 70 | 81 33 35 |
| **2 8 32** | 18 72 94 | 34 39 59 | 50 6 24 | 66 70 86 | 82 37 51 |
| 3 12 48 | 19 76 13 | 35 43 75 | 51 10 40 | 67 74 5 | 83 41 67 |
| 4 16 64 | 20 80 29 | 36 47 91 | 52 14 56 | 68 78 21 | 84 45 83 |
| 5 20 80 | 21 84 45 | 37 51 10 | 53 18 72 | 69 82 37 | 85 49 2 |
| 6 24 96 | 22 88 61 | 38 55 26 | 54 22 88 | 70 86 53 | 86 53 18 |
| **7 28 15** | 23 92 77 | 39 59 42 | 55 26 7 | 71 90 69 | 87 57 34 |
| **8 32 31** | 24 96 93 | 40 63 58 | 56 30 23 | 72 94 85 | 88 61 50 |
| 9 36 47 | **25 3 12** | 41 67 74 | 57 34 39 | 73 1 4 | 89 65 66 |
| 10 40 63 | **26 7 28** | 42 71 90 | 58 38 55 | 74 5 20 | 90 69 82 |
| 11 44 79 | 27 11 44 | 43 75 9 | 59 42 71 | 75 9 36 | 91 73 1 |
| 12 48 95 | 28 15 60 | 44 79 25 | 60 46 87 | 76 13 52 | 92 77 17 |
| 13 52 14 | 29 19 76 | 45 83 41 | 61 50 6 | 77 17 68 | 93 81 33 |
| 14 56 30 | 30 23 92 | 46 87 57 | 62 54 22 | 78 21 84 | 94 85 49 |
| 15 60 46 | **31 27 11** | 47 91 73 | 63 58 38 | 79 25 3 | 95 89 65 |
| 16 64 62 | **32 31 27** | 48 95 89 | 64 62 54 | 80 29 19 | 96 93 81 |

**Table 2.** Associated Binary Vectors for $T$

| *Target Set Vector from T* | *Associated Binary Vector* |
|---|---|
| 1, 4, 16 | 1, 0, 0 |
| 2, 8, 32 | 0, 0, 0 |
| 7, 28, 15 | 1, 0, 1 |
| 8, 32, 31 | 0, 0, 1 |
| 25, 3, 12 | 1, 1, 0 |
| 26, 7, 28 | 0, 1, 0 |
| 31, 27, 11 | 1, 1, 1 |
| 32, 31, 27 | 0, 1, 1 |

2. The client selects a secret encryption key $b$ that is uniformly distributed between 1 and 96. The key $b$ is only used for this encryption. Suppose $b = 60$.
3. The client computes $b(25, 3, 12) \bmod 97 = 60(25, 3, 12) \bmod 97 = (45, 83, 41)$. as the ciphertext vector. The ciphertext vector is sent to the server. Note that the ciphertext vector is also in the permutation table $T$.
4. The server uses the received vector components as the circuit inputs. It obtains $45 + 83 + 41 = 169$ as the output. It sends the output back to the client.
5. The client decrypts the received ciphertext by computing $b^{-1}(169) \bmod 97 = 76(169) \bmod 97 = 40$. The client completes the decryption by computing $40 \bmod 2 = 0$. Thus the output is 0 as expected.

We now discuss security. Each of the possible distinct secret keys $b$ results in a distinct row vector in the permutation table $T$. Given the ciphertext $C = (45, 83, 41)$, we see from Table 3 that each of the plaintexts is possible.

**Table 3.** Plaintext Possibilities Given Ciphertext $C = (45, 83, 41)$, $m = 97$.

| *Plaintext* $P_i$ | *Target Set Vector from* $T$ | Secret Key $b$ | $Pr(C\|P_i)$ |
|---|---|---|---|
| 1, 0, 0 | 1, 4, 16 | 45 | 1/(m-1) |
| 0, 0, 0 | 2, 8, 32 | 71 | 1/(m-1) |
| 1, 0, 1 | 7, 28, 15 | 48 | 1/(m-1) |
| 0, 0, 1 | 8, 32, 31 | 42 | 1/(m-1) |
| 1, 1, 0 | 25, 3, 12 | 60 | 1/(m-1) |
| 0, 1, 0 | 26, 7, 28 | 95 | 1/(m-1) |
| 1, 1, 1 | 31, 27, 11 | 39 | 1/(m-1) |
| 0, 1, 1 | 32, 31, 27 | 59 | 1/(m-1) |

We have

$$Pr(C) = Pr(C\ AND\ (OR_{i=1}^{8}\ P_i)) = \sum_{i=1}^{8} Pr(C\ AND\ P_i) = \sum_{i=1}^{8} \frac{Pr(P_i)}{m-1} = \frac{1}{m-1}$$

Thus $Pr(C) = Pr(C|P)$ for all of the plaintext vectors $P$. So $C$ and $P$ are independent. $Pr[P|C] = Pr[P\ AND\ C]/Pr[C] = (Pr[P]Pr[C])/Pr[C] = Pr[P]$ for each of the possible plaintexts $P$ in Table 3 and for every possible probability distribution on the plaintexts. Thus we have perfect security. We will prove this assertion formally in Theorem 4.

### 2.2 Server Privacy

In the 2PC protocol, the server also has inputs; the privacy of these inputs depends on the hardness of the Decision Permutation Table HSSP (DPT-HSSP) (see Section 2.4) and the DDH (Decisional Diffie Hellman Assumption); the client is assumed to be computationally bounded and semi-honest. The circuit is not

**Algorithm 1** Summary Overview of 2PC Protocol

1: Inputs: see Table 4

2: SETUP: ($m$, $k$, $c$, one-time secret $b$, $d_1, \ldots, d_c$ generated see Algorithm 5 details)
Server generates its input bits and noise bits $(s_1, \ldots, s_{c_s+k})$
Server computes its part of the e-vector $\boldsymbol{e}'$; 1st component is $e_s$ (Algorithm 2) $(e'_{c_1+i} \bmod 2 = s_i,\ 1 \le i \le c_s + k, e'_i \bmod 2 = 0, 1 \le i \le c_1)$ $e_s = e'_1$ (e-vector is a row in permutation table)
Server encrypts $e_s$ with its CL encryption public key ($E_{pk}(e_s)$); sends to client.
To defend against malicious server, client and server execute modified Okamoto protocol to prove server ciphertext elements are in CL group (Section 4.4, Figure 10.)
Client generates its input bits and random multiplication bits (3 zero bits and 1 one bit per multiplication block) $(i_i, \ldots, i_{c_1})$.
Client computes its part of the e-vector $\bar{\boldsymbol{e}}$; $\bar{e_j} \bmod 2 = i_j$, $1 \le j \le c_1$ (Algorithm 6)
Client encrypts $\bar{e_1} = e_c$ with server CL public key: $E_{pk}(e_c - 2)$
Client computes (homomorphically) $E_{pk}(e_c + e_s - 2)$.
Client computes (homomorphically) $E_{pk}(be)$ where $e = e_c + e_s - 2$; send to server.
Server decrypts to obtain $be \bmod m$. $e$ is 1st component of e-vector. $be \bmod m$ is 1st component of ciphertext vector.

3: CIRCUIT PROCESSING: The $ith$ input is $(be)d_i \bmod m$. Server processes the inputs through the circuit.

4: Multiplication (AND OR NAND gate:) Given multiplicands $be_1 \bmod m$ and $be_2 \bmod m$ : Server sends $x_j = be_j + \sum_{be_i}$ a noise element, $_{t_i^j \in \{0,1\}}\ t_i^j be_i$, $j = 1, 2$ to client where $\sum t_i^j e_i$ has even parity with 1/2 probability, $j = 1, 2$.

Client sets $r_j = parity(b^{-1}x_j)$, $j = 1, 2$. $r_3 = 1 - r_1$, $r_4 = 1 - r_2$;
Client sends pointers to elements in 1st unused multiplication block in order:
1st pointer to $\hat{be_1}$ where parity($\hat{e_1}$) is $r_1 r_2$.
2nd pointer to $\hat{be_2}$ where parity($\hat{e_2}$) is $r_1 r_4$.
3rd pointer to $\hat{be_3}$ where parity($\hat{e_3}$) is $r_3 r_2$.
4th pointer to $\hat{be_4}$ where parity($\hat{e_4}$) is $r_3 r_4$.

If $f_l = r_i r_j$ where $r_i$ and $r_j$ correspond to elements with even subset sums, then server selects the gate output to be the entry in the encrypted vector in location $index(f_l)$. Each pointer or index $j$ satisfies $1 \le j \le 4$. (Algorithm 7)

5: Addition (XOR) gates are processed by the server without client interaction; the two inputs are added as integers to get the output. NOT gates can be processed by adding an element with odd parity to the gate input element.

6: An output $o$ is summed with an even parity subset sum element $s$ to get $s+o \bmod m$ which is sent to the client. The client computes the plaintext output as $parity(b^{-1}(s+o) \bmod m)$. The client does not share the plaintext output with the server unless the server is semi-honest or there is a mechanism for verifying that the server has followed the protocol.

shared with the client and thus the client only learns about the number of AND gates. See Algorithm 1 for an overview summary of the 2PC protocol.

Three key concepts associated with the permutation table $T$ (see the example in Section 2.1 and formal definition in Section 3) are the binary plaintext input vector, the e-vector $\boldsymbol{e}$ which is the row in the table $T$ determined by the binary plaintext input vector (see Algorithm 2) where the parities of $\boldsymbol{e}$'s components equal the binary plaintext input vector components, and the ciphertext vector which is the row in the table $T$ obtained by scalar vector product $b\boldsymbol{e}$ mod $m$ for secret key $b$. The e-vector is in the target set of short vectors (see the example in Section 2.1).

For the 2PC protocol, the plaintext input vector has the structure depicted in Figure 3.

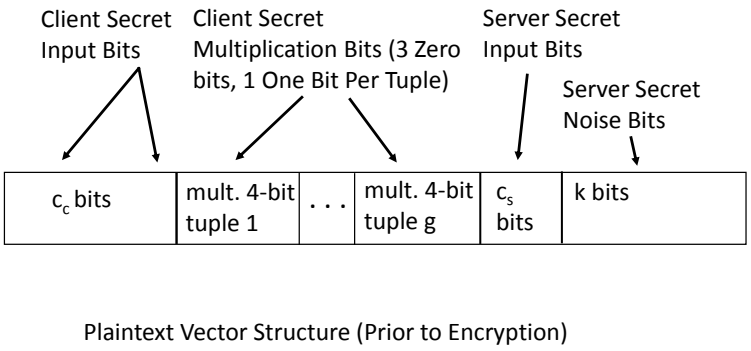



**Fig. 3.** Structure of Plaintext Input Vector for a Circuit $C$ with $g$ Multiplication Gates

The parameters for the 2PC protocol are given in Table 4. The first row of the permutation table $T$ is $d_1, \ldots, d_c$.

**Table 4.** 2PC Protocol Parameters

| |
|---|
| *Shared Parameters for Client and Server*<br>$m$ (prime integer), $d_1, \ldots, d_{c_1}, c_1 = c_c + 4g$, server CL encryption public key |
| *Client Secret Parameters*<br>one-time secret key $b$, $c_c$ client plaintext inputs, $4g$ multiplication inputs |
| *Server Secret Parameters*<br>$c, d_{c_1+1}, \ldots, d_c$, $c_s$ server plaintext inputs, server plaintext noise inputs,<br>server CL encryption private key, circuit $C$, $k$ = number of noise inputs |
| *Unknown to Both Parties*<br>selected e-vector in table $T$ |

Table 5 depicts the structure of the permutation table.

**Table 5.** 2PC Permutation Table Structure, $r = m - 1$, ciphertext row can be any row

| *Row* | *Client Input Columns* | *Server Input Columns* | *Server Noise Columns* |
|---|---|---|---|
| First Row | $d_1 \dots d_{c_1}$ | $d_{c_1+1} \dots d_{c_1+c_s}$ | $d_{t+1} \dots d_c$ |
| Second Row | $2d_1 \dots 2d_{c_1}$ | $2d_{c_1+1} \dots 2d_{c_1+c_s}$ | $2d_{t+1} \dots 2d_c \pmod m$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $lth$ row (e-vector) | $ld_1 \dots ld_{c_1}$ | $ld_{c_1+1} \dots ld_{c_1+c_s}$ | $ld_{t+1} \dots ld_c \pmod m$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| ciphertext row | $bld_1 \dots bld_{c_1}$ | $bld_{c_1+1} \dots bld_{c_1+c_s}$ | $bld_{t+1} \dots bld_c \pmod m$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| last row | $rd_1 \dots rd_{c_1}$ | $rd_{c_1+1} \dots rd_{c_1+c_s}$ | $rd_{t+1} \dots rd_c \pmod m$ |

For processing inputs through a circuit $C$, the server can process addition (XOR) gates without client interaction. The server leverages the client's help for multiplication (AND) gates. The server sends obfuscated (by adding hidden subset sums) versions of $be_1$ and $be_2$ to the client. The hidden subset sums use elements from the server noise columns ciphertext row (see Table 5.) These elements are unknown to the client.[1] Each of $be_1$ and $be_2$ plus the subset sums will have their parity reversed with 1/2 probability.

The client computes the product of the parities of $e_1$ and $e_2$. It also computes the product of $1 - parity(e_1)$ and $e_2$, $e_1$ and $1 - parity(e_2)$, and $1 - parity(e_1)$ and $1 - parity(e_2)$. It sends the indices in the e-vector row that correspond to these products to the server. Each such multiplication operation consumes one multiplication 4-tuple from the e-vector row. The server selects the element that corresponds to the correct product as the output of the multiplication gate. The server is able to compute the encrypted vector from the first component of the vector which is obtained in the CL decryption step (the $ith$ component is $bed_i \bmod m$ where $be \bmod m$ is the first component.)

Our initial setup (Algorithm 5) leverages the linear homomorphic properties of the Castagnos Laguillaumie public key encryption (CLPKE) [11]. We give details in Section 2.4. The server computes the e-vector row in the permutation table corresponding to the plaintext input vector $(0, \dots, 0$, server bits, server noise bits) where the initial prefix has $c_1$ zero bits (see Algorithm 2). The server encrypts the first element of the e-vector row with its CLPKE public key and sends it to the client. To defend against a malicious model server, the client and server execute $z$ runs of a slightly modified Okamoto ZKP (Zero Knowledge Protocol) (see Figure 10) to prove with high probability that the server ciphertext

[1] The classic subset sum problem requires $s$ public sum elements where $s$ is close to $\log(m)$ [28]. Our client security requires $\log(m) \geq c \log(2(a+1))$ and we have $s < c$. Thus subset sum is vulnerable to the low density attacks in [33], [27] and others.

consists of a pair of elements in the CL encryption cyclic group. (The client can efficiently verify that the server element pair is in the larger abelian group $G_1$ but the ZKP is needed to ensure the elements are in the cyclic subgroup $G$.) Theorem 12 uses the result that the server ciphertext pair is in the group $G$ to prove statistical privacy for the client when the server is malicious.

The client computes the e-vector row in the permutation table (Algorithm 6) corresponding to it's plaintext input vector (client input bits, client multiplication bits) and then encrypts the first element with the server CLPKE public key. The client then homomorphically adds the encrypted elements. The client then homomorphically multiplies the result with $b$. The server decrypts using its CLPKE private key to obtain the first element of the ciphertext vector $b\boldsymbol{e}$. The server can then compute the other elements of the ciphertext vector to process through the circuit. See Section 4 for details.

In the setup protocol, a malicious server may deviate and attempt to guess the client plaintext inputs and multiplication bits by encrypting the additive inverse of its guess and sending it to the client. If the guess is correct, the client will send back the generator $g$ raised to zero which is visible to the server, thus confirming the correctness of its guess. This attack succeeds with low probability that decreases as the number of client plaintext inputs and multiplication gates increases. If the guess at the client message, $m_c'$ is incorrect, then the server learns that the guess is not the client message so there is very small leakage in this case. The statistical distance between the conditional distribution (after receiving the encrypted client message) and the distribution prior to setup is very small and goes to zero as the number of multiplication gates increases. Every other client message $m_c \neq m_c'$ is possible given the client encrypted message. See Theorem 12.

We define two variants of existing HSSP [34]. PT-HSSP takes $k$ hidden weights from an e-vector row in the permutation table and adversary's task is to determine the weights given $w$ samples where $w < k$. In Section 4, we show that solving PT-HSSP for $w < k$ with non-negligible advantage given our 2PC initial setup implies CL encryption CPA security can be broken with non-negligible advantage. Intuitively, the function associating hidden weights $\alpha_1, \ldots, \alpha_k$ and $w$ tuples $t_i^j \in \{0,1\}$, $1 \leq j \leq w, 1 \leq i \leq k$ to $w$ samples $\sum_{i=1}^{k} t_i^j \alpha_i$ must associate distinct weight tuples to a single $w$ tuple when $w < k$. In other words, the $w$ samples do not uniquely determine the hidden weights. The 2nd HSSP variant is DPT-HSSP which is the decision version of PT-HSSP. We discuss its hardness in Section 2.4. (The same reasoning shows that distinct server inputs map to the same set of $w$ samples thus making it hard for an adversary to distinguish between a simulation with random server inputs and a real protocol execution.)

For larger circuits, we can improve communication complexity by dividing the circuit into subcircuits. Section 5 discusses the subdivision. Subdividing requires running the setup protocol (Algorithm 5) for each of the subcircuits. Transitioning to a new subcircuit at multiplication gates includes selecting indices for elements in a new ciphertext row corresponding to the new secret key $b'$.

### 2.3 Preliminaries

A vector $\boldsymbol{x}$ of elements in $\mathbb{Z}_m$ (the integers modulo m) will be denoted in boldface. Elements in $\mathbb{Z}_m$ will be taken to be integers between 0 and $m-1$, inclusive. Given $\boldsymbol{x} = (x_1, \ldots, x_c)$. We define $\boldsymbol{x} \bmod m$ to be the vector $\boldsymbol{v} = (v_1, \ldots, v_c)$ where $v_i = x_i \bmod m$, $v_i \in [0, m-1]$, $1 \leq i \leq c$. For $b \in \mathbb{Z}_m$ and $\boldsymbol{x} \in \mathbb{Z}_m^c$, we define $b\boldsymbol{x} = (bx_1, \ldots, bx_c) \bmod m$. $\{0,1\}^n$ denotes the set of integers with $n$ bits.

We will use the terms perfect privacy and perfect security interchangeably: our secret key encryption scheme has perfect security if

$$Pr[Enc_{b_1}(\boldsymbol{r_1}) = \boldsymbol{w}] = Pr[Enc_{b_2}(\boldsymbol{r_2}) = \boldsymbol{w}]$$

for all ciphertexts $\boldsymbol{w}$ and all plaintexts $\boldsymbol{r_1}, \boldsymbol{r_2}$ where $b_1, b_2$ are uniform random secret keys from $[1, m-1]$.

For circuits, we let the parameter $a$ be the maximum number of additions prior to a multiplication gate ($a_1$ in Section 4) and $g$ is the number of multiplication gates. For boolean circuits, the multiplication gates could be either AND or NAND gates, and the addition gates are XOR gates. We will assume the addition and multiplication gates have two inputs. We refer to the party that evaluates the circuit, receiving encrypted inputs from the other party, as the server. The other party that always receives output is the client.

The notation $s \leftarrow S$ is used when $s$ is randomly selected from $S$ via the uniform distribution.

A function $\mu(n)$ is negligible if for every positive polynomial $p(n)$, $\mu(n) < 1/p(n)$ for sufficiently large $n \in \mathbb{N}$.

Let $X = \{X(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ be an infinite family of random variables indexed by $a$ and $\kappa \in \mathbb{N}$. Two families of random variables $X = \{X(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ are computationally indistinguishable if for every non-uniform polynomial-time algorithm $\mathsf{A}$ there exists a negligible function $\mu$ such that for every $a \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$ we have

$$|Pr[\mathsf{A}(X(a,\kappa))] - Pr[\mathsf{A}(Y(a,\kappa))]| \leq \mu(\kappa).$$

We denote computational indistinguishability by $X \equiv^c Y$.

Two families of random variables $X = \{X(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ are perfectly indistinguishable if for every non-uniform algorithm $\mathsf{A}$ and every $a \in \{0,1\}^*$ and $\kappa \in \mathbb{N}$ we have

$$|Pr[\mathsf{A}(X(a,\kappa))] - Pr[\mathsf{A}(Y(a,\kappa))]| = 0.$$

We denote perfect indistinguishability by $X \equiv^{perf} Y$.

We consider two security models in Section 4. For the case where only the client receives output, we have one-sided simulation for the client to show server security, and indistinguishability for the server's view of the client inputs to show client security [26]. We also consider the two-sided simulation model where both parties are semi-honest and the server can receive output. In all cases, the client is semi-honest, but the first model enables client privacy even when the server is malicious.

We assume the client is semi-honest, and we assume a deterministic functionality $f$. Our 2PC protocol has server privacy if there is a probabilistic polynomial-time algorithm $\mathcal{S}_1$ such that

$$\{\mathcal{S}_1(x, f_1(x,y))\}_{x,y\in\{0,1\}^*} \equiv^c \{VIEW^{\Pi}_{client}(x,y)\}_{x,y\in\{0,1\}^*}$$

where $VIEW_{client}$ is the client's view of our protocol $\Pi$'s execution including it's input, randomness and messages received, $f_1(x,y)$ is the client output, and $x, y$ are the client and server inputs respectively.

In the first security model, the server does not receive output. In this model client privacy is defined as

$$\{VIEW^{\Pi}_{srv}(x,y)\}_{x,y\in\{0,1\}^*} \equiv^{perf} \{VIEW^{\Pi}_{srv}(x',y)\}_{x',y\in\{0,1\}^*}$$

where $VIEW_{srv}$ is the server's view of our protocol $\Pi's$ execution including it's input, randomness, and messages received, and $|x| = |x'| = |y|$ where $x$ and $x'$ are client inputs.

In the second security model, the server receives output, both parties are semi-honest, and client privacy is defined via simulation: Our 2PC protocol has client privacy if there is a probabilistic polynomial-time algorithm $\mathcal{S}_2$ such that

$$\{\mathcal{S}_2(y, f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^{perf} \{VIEW^{\Pi}_{srv}(x,y)\}_{x,y\in\{0,1\}^*}$$

where $VIEW_{srv}$ is the server's view of our protocol $\Pi$'s execution including it's input, randomness and messages received, $f_2(x,y)$ is the server output, and $x, y$ are the client and server inputs respectively.

For statistical privacy against a malicious model server during the 2PC setup protocol, we have the following definitions:

**Definition 1.** *(Statistical Distance [16]) Let $X$ and $Y$ be random variables on the same probability space with common range $D$. Then $SD = SD(X,Y) = (1/2)\sum_{d\in D} |Pr[X = d] - Pr[Y = d]|$.*

**Definition 2.** *(Statistical Privacy of 2PC Setup Protocol) The 2PC setup protocol is statistically private against a malicious server if there exists $\delta(g) > 0$ such that $\delta(g)$ goes to zero as $g$ grows where the statistical distance between the conditional distribution on the client messages after the setup protocol is complete and the a-priori distribution is bounded by $\epsilon(g)$ with probability greater than or equal to $1 - \delta(g)$ where $\epsilon(g)$ goes to zero and $g$ grows.*

### 2.4 Computational Hardness Assumptions For Server Security

We present our hardness assumptions here. Hidden Subset Sum [34] is similar to subset sum except the sum elements are kept secret.

**Hidden Subset Sum Problem (HSSP) [34]** Let $m$ be an integer and let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be integers in $\mathbb{Z}_m$. Let $\boldsymbol{x_1}, \ldots, \boldsymbol{x_k} \in \mathbb{Z}^w$ be vectors with entries in $\{0,1\}$. Let $\boldsymbol{h} = (h_1, \ldots, h_w) \in \mathbb{Z}^w$ such that

$$\boldsymbol{h} = \alpha_1 \boldsymbol{x_1} + \alpha_2 \boldsymbol{x_2} + \ldots + \alpha_k \boldsymbol{x_k} \bmod m.$$

The adversary is given $\boldsymbol{h}$ and $m$ and must recover the weights and the $\boldsymbol{x_i}$ vectors, up to a permutation. NOTE: the adversary is not given $k$.

We define two HSSP variants below: PT-HSSP (Permutation Table HSSP) and DPT-HSSP (Decisional Permutation Table HSSP). In our usage of HSSP, we will have $w < n$. In Section 4, we show that solving PT-HSSP for $w < n$ with non-negligible advantage given our 2PC initial setup implies CL encryption CPA security can be broken with non-negligible advantage.

**Permutation Table Hidden Subset Sum Problem (PT-HSSP)** Given $1, d_2, \ldots, d_{c_1}$ and prime modulus $m$ known to the adversary. $d_{c_1+1}, \ldots, d_{c_1+c_s+k}$, $c = c_1 + c_s + k$ are secret values; $d_i | d_{i+1}$, $1 \leq i \leq c-1$, $d_{i+1} \geq 2(a+1)d_i$, $1 \leq i \leq c-1$, $2(a+1)d_c \leq m-1$ where $a$ is an integer equal to $k$ plus a constant. Fix $l$, $1 \leq l \leq m-1$ and hidden subset sum weights $ld_{c_1+i} \bmod m, 1 \leq i \leq c_s+k$. The adversary is given $w$ samples

$$\left( \sum_{i=1}^{c_1+c_s} s_i^j l d_i + \sum_{i=c_1+c_s+1}^{c} t_i^j l d_i \right) \bmod m,$$

$t_i^j$ are uniformly distributed, $t_i^j \in \{0,1\}$, $s_i^j, 1 \leq i \leq c_1$ are nonnegative integers, $1 \leq j \leq w$. The adversary must solve for $ld_{c_1+i} \bmod m, 1 \leq i \leq c_s + k$.

**Decisional Permutation Table Hidden Subset Sum Problem (DPT-HSSP)** Given circuit $C$. We have $\boldsymbol{v_1} = d_1, \ldots, d_{c_1}, d_{c_1+1}, \ldots, d_c$ and $\boldsymbol{v_2} = d'_1, \ldots, d'_{c_1}, d'_{c_1+1}, \ldots, d'_{c'}$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$, $d'_i | d'_{i+1}$, $1 \leq i \leq c'-1$, $d_{i+1} \geq 2(a+1)d_i$, $1 \leq i \leq c-1$, $d'_{i+1} \geq 2(a+1)d'_i$, $1 \leq i \leq c'-1$, $d_i = d'_i$, $1 \leq i \leq c_1$. Also, $2(a+1)d_c \leq m-1$ and $2(a+1)d'_c \leq m-1$ for some integer $a$ equal to $k$ plus a constant. We have $c_s \geq 1$ and $c_1 + c_s < \min\{c, c'\}$

The computationally bounded adversary has negligible probability, as $m$, $k = c - c_1 - c_s$, and $k' = c' - c_1 - c_s$, grow, to distinguish between the distributions

$$y_1^j = \left( \sum_{i=1}^{c_1+c_s} s_i^j l_1 d_i + \sum_{i=c_1+c_s+1}^{c} t_i^j l_1 d_i \right) \bmod m$$

and

$$y_2^j = \left( \sum_{i=1}^{c_1+c_s} s_i^j l_2 d'_i + \sum_{i=c_1+c_s+1}^{c'} t_i'^j l_2 d'_i \right) \bmod m,$$

$j = 1, \ldots, w$. In our application below, $l_1$ ($l_2$) is determined by client inputs, real protocol server inputs (uniform random server inputs), and uniform random noise inputs. $s_i$ are nonnegative integers are determined by the circuit $C$, $t_i, t'_i$ are uniformly distributed, $t_i, t'_i \in \{0,1\}$.

The adversary is given $d_1, \ldots, d_{c_1}, m$, and $y_1^j$ with probability 1/2, else $y_2^j, j = 1, \ldots, w$.

**Castagnos Laguillaumie (CLPKE) Encryption [11]**

The generic scheme is given in Figure 4. In [11], [7], the CLPKE is instantiated with the class group of a non maximal order of an imaginary quadratic field; the authors prove CLPKE is CPA secure. The message space is $\mathbb{Z}q$, $q$ prime; we have $m = q$. $G$ is a group where DDH is hard and $F$ is a subgroup of order $q$ where discrete log is easy. $s$ and $q$ are relatively prime; $n = sq$. $s$ and $q$ are integers with $y1$ and $u1$ bits respectively.

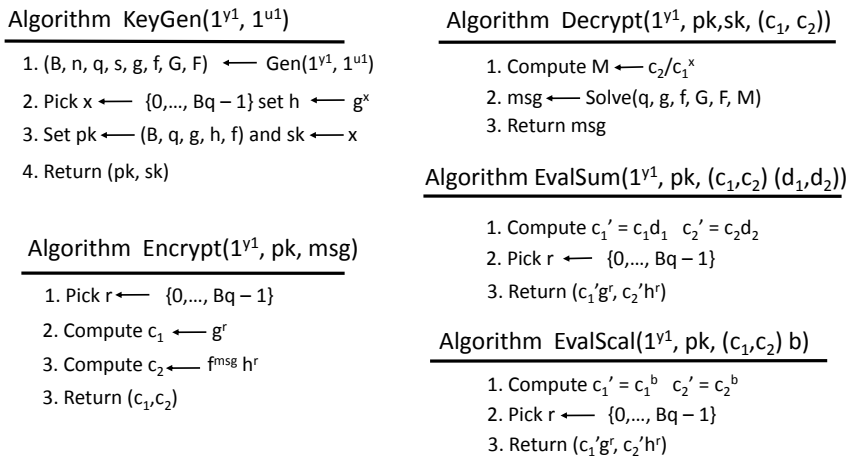



**Fig. 4.** Generic Linearly Homomorphic Public Key Encryption (CLPKE)from [11]

## 3 Somewhat Additive Homomorphic Protocols

In this section we define permutation tables and the target set of short vectors in a permutation table. We prove a sequence of lemmas leading to Theorem 1 which shows a one to one correspondence between the target set of short vectors in a permutation table satisfying certain conditions (a perfect table) and the set of all possible associated vectors in $\mathbb{Z}_N^c$. (In this section, we generalize the plaintext vector space from $Z_2^c$ to $Z_N^c$.)

We then present a somewhat additive homomorphic encryption scheme with perfect security (Definition 5), and we prove our somewhat additive homomorphic encryption (P-SWAHE) scheme has perfect security in Theorem 4. We prove correctness in Theorem 2 and 3. We present an extension to the scheme to support both addition and multiplication (Algorithm 3.) We prove correctness in

Theorem 5 and perfect security in Theorem 6. Our schemes are secret key rather than public key.

### 3.1 Permutation Tables

Unless stated otherwise, vector components are assumed to be nonnegative.

**Definition 3.** *Let $m$ be a prime integer, let $a$ be a positive integer, and let $N$ be an integer that is a power of 2. A vector $\mathbf{d} = (d_1, \ldots, d_c)$ in $\mathbb{Z}_m^c$ generates a* ***permutation table*** *$T$ with $m-1$ rows and $c$ columns. $\mathbf{d}$ is the start vector. The $i$th row of the table, $1 \le i \le m-1$, is the vector $(id_1 \bmod m, \ldots, id_c \bmod m)$ in $\mathbb{Z}_m^c$ where we take representatives to be positive integers less than $m$. A* ***short vector*** *in the table is a vector $(v_1, \ldots, v_c)$ such that $v_i \le (m-1)/(a+1) = \alpha$ for $1 \le i \le c$. We associate with each short vector $\mathbf{v}$ in $T$ a vector $\mathbf{r}$ with $c$ components: $r_i = v_i \bmod N$, $1 \le i \le c$. A* ***perfect*** *permutation table is a table where for each of the possible $N^c$ vectors in $\mathbb{Z}_N^c$ there is some short vector in the table that has that vector as its* ***associated vector****.*

*We will focus on the permutations generated by the columns of a permutation table. We define a* ***permutation run*** *as the increasing or decreasing sequence of elements in a column permutation that are all obtained by subtracting the same multiple of $m$; in other words, any two elements $w_1$ and $w_2$ in the same run can be expressed as $is = qm + w_1$ and $js = qm + w_2$ for some integers $i$ and $j$ where $s$ is the first row element in the column (the start element). The* ***length of a permutation run*** *is the number of elements in the permutation run. The average of the lengths of the permutation runs in a column is the* ***column run period****. (Each column in the permutation table consists of one or more permutation runs in sequence.) We call the first element of a new permutation run a* ***flip****, and we say the column flips in any row that has a flip.*

*Remark 1.* Since $m$ is prime, the last row in a permutation table potentially consists of zeroes; however, we do not include the zero row as part of the permutation table.

**Example:** In Table 1, the 3rd column first permutation run has 6 elements. The second column first permutation run has 24 elements.

Table 6 gives another example of a perfect permutation table.

**Fact 1:** If $s \le (m-1)/2$, then a permutation run increments each element by $s$, where $s$ is the column start element. If $s > (m-1)/2$, then a permutation run decrements each element by $m - s$ where $s$ is the column start element.

**Fact 2:** If $s > (m-1)/2$, then run period(s) $= (m-1)/(m-s)$, where $s$ is the column start element. If $s \le (m-1)/2$, then run period(s) $= (m-1)/s$, where $s$ is the column start element.

**Fact 3:** For each column in a permutation table where $s \le (m-1)/2$, the first permutation run has length $\lfloor (m-1)/s \rfloor$ and succeeding runs have length bounded by $\lfloor (m-1)/s \rfloor + 1$. (see Lemma 3 for a proof).

For the remainder of this paper, we will assume that permutation tables have start elements $s$ where $s \le (m-1)/2$.

**Table 6.** Perfect or No Zeroes Permutation Table for $m = 37$, $c = 3$, $a = 1$, short vectors bolded

| | | | | | |
|---|---|---|---|---|---|
| **1 3 9** | 7 21 26 | **13 2 6** | 19 20 23 | 25 1 3 | 31 19 20 |
| **2 6 18** | 8 24 35 | **14 5 15** | 20 23 32 | 26 4 12 | 32 22 29 |
| 3 9 27 | 9 27 7 | 15 8 24 | 21 26 4 | 27 7 21 | 33 25 1 |
| 4 12 36 | 10 30 16 | 16 11 33 | 22 29 13 | 28 10 30 | 34 28 10 |
| **5 15 8** | 11 33 25 | **17 14 5** | 23 32 22 | 29 13 2 | 35 31 19 |
| **6 18 17** | 12 36 34 | **18 17 14** | 24 35 31 | 30 16 11 | 36 34 28 |

### 3.2 Additive Homomorphic Scheme with Perfect Security

We first prove a lemma that shows that for certain permutation tables, a column flip in a given row implies that the same row flips in columns to the right:

**Lemma 1.** *Let $T$ be a permutation table with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Then the $(i+1)$st column flips when the ith column does.*

*Proof.* There exist integers $k$, $q$, and $r$ such that $kd_i = qm + r$, $kd_i + d_i = qm + r + d_i$ where $r + d_i > m$, $0 \leq r < m$. Let $d_{i+1} = d_i w$. $kd_{i+1} = kd_i w = qmw + rw$, $rw = q_1 m + r_1$, where $0 \leq r_1 < m$. Suppose $w \geq q_1 + 1$. Then $r_1 + d_{i+1} = r_1 + d_i w = rw - q_1 m + d_i w = (r + d_i)w - q_1 m > mw - q_1 m = m(w - q_1) \geq m$. Thus $w \geq q_1 + 1$ implies that the $d_{i+1}$ column flips as claimed. If $w \leq q_1$, then $rw \leq q_1 r < q_1 m$ but $rw = q_1 m + r_1$ so it must be that $w \geq q_1 + 1$. ∎

**Fact 4:** Given a permutation table T satisfying the conditions of Lemma 1. By the lemma, when column $i$ flips, then all columns to the right of column $i$ flip (columns $i{+}1, \ldots, c$) as well. Before column $i$ flips again, column $i{+}1$ flips $d_{i+1}/d_i$ times. (A permutation run has length $(m-1)/d_i$ so $\frac{(m-1)/d_i}{(m-1)/d_{i+1}} = d_{i+1}/d_i$.)

**Definition 4.** *Let $T$ be a permutation table with row vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha = (m-1)/(a+1)$. Given column $j$ in $T$ where $2 \leq j \leq c$. A **left of j row** is a row where column $j-1$ has a flip. The **target set of short vectors** is the set of vectors in $T$ for which the $j$th component, $2 \leq j \leq c$, occurs in a row prior to the $N$th flip of the $j$th column that occurs after some left of $j$ row, and in a row in or after the left of $j$ row, the 1st column component is bounded by $\alpha$, and the $c$th component occurs in a row between zero and $N-1$ rows after a flip of the $c$th column. See Figure 5.*

*Remark 2.* Row 1 is a left of j row for columns $j$, $2 \leq j \leq c$.

**Example:** Row 13 is a left of j row for column 3 in Table 6.

**Lemma 2.** *Let $T$ be a permutation table with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha$. Given $t < d_i$. Then*

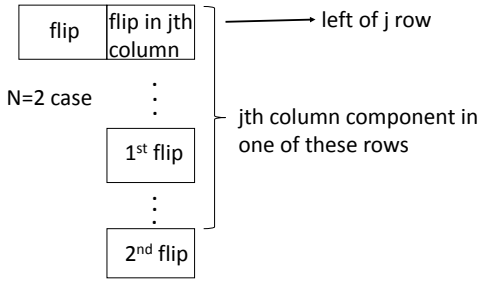



**Fig. 5.** Possible Rows for Target Set Vector Components in Column $j \geq 2$, $N = 2$

1. $\lfloor t\frac{m-1}{d_i} \rfloor + 1$ *is an upper bound for the sum of the lengths of the first* $t$ *permutation runs for the column with start element* $d_i$*, where* $d_i > 1$.
2. $\lfloor t\frac{m-1}{d_i} \rfloor$ *is the lower bound.*

*Proof.* (1) The element $d_i$ is the start element of the first permutation run in the sequence of $t$ permutation runs in column $i$.

$$d_i + wd_i \leq m - 1 + (t-1)(m) = tm - 1$$

where $w$ is the number of $d_i$ values added to $d_i$ to get to the last element in the $t$th permutation run (prior to reducing modulo $m$.) Thus

$$w + 1 \leq \frac{tm-1}{d_i}$$

$$w + 1 \leq \left\lfloor \frac{tm-1}{d_i} \right\rfloor \leq \left\lfloor \frac{tm}{d_i} \right\rfloor \leq \left\lfloor \frac{t(m-1)}{d_i} \right\rfloor + 1$$

since $t < d_i$. Thus it holds for $t = N$.
(2) Let $u$ be the last element in the $t$th permutation run prior to reduction modulo $m$. $u = d_i + wd_i$ where $w$ is the number of $d_i$ values added to $d_i$ to get to the last element. $u + d_i > tm$. $d_i + wd_i + d_i > tm$ by substituting for $u$.

$$w + 2 > tm/d_i > t(m-1)/d_i \geq \lfloor t(m-1)/d_i \rfloor$$

Thus $w + 1 \geq \lfloor t(m-1)/d_i \rfloor$. ∎

**Lemma 3.** *Given a permutation table* $T$*. Let* $x$ *be the start element of a permutation run in column* $i$ *which has* $d_i > 1$ *as the column start element. Let*

$$\frac{m-1}{d_i} - \left\lfloor \frac{m-1}{d_i} \right\rfloor = \frac{k_i}{d_i}$$

*where* $0 \leq k_i \leq d_i - 1$*. If* $1 \leq x \leq k_i$*, then the permutation run has length* $\lfloor \frac{m-1}{d_i} \rfloor + 1$*. If* $x > k_i$ *or* $k_i = 0$*, the permutation run has length* $\lfloor \frac{m-1}{d_i} \rfloor$.

*Proof.*

$$\frac{k_i}{d_i} + \left\lfloor \frac{m-1}{d_i} \right\rfloor = \frac{m-1}{d_i}$$

so

$$k_i + \left\lfloor \frac{m-1}{d_i} \right\rfloor d_i = m-1. \tag{1}$$

Thus if $k_i$ is the start element of the permutation run, then the length is $\lfloor \frac{m-1}{d_i} \rfloor + 1$. Clearly if $x < k_i$, the permutation run must have length at least $\lfloor \frac{m-1}{d_i} \rfloor + 1$. It is also clear that if $x > k_i$, then the permutation run has length bounded above by $\lfloor \frac{m-1}{d_i} \rfloor$.

Suppose $x = d_i$. Then $wd_i \leq m-1$ and $(w+1)d_i \geq m$ where $w$ is the length of the permutation run with $x$ as the start element. So $\frac{m-1}{d_i} \geq w$ and $\lfloor \frac{m-1}{d_i} \rfloor \geq w$. We have $\frac{m}{d_i} \leq w+1$, $\frac{m-1}{d_i} \leq w - \frac{1}{d_i} + 1$. Thus $\lfloor \frac{m-1}{d_i} \rfloor \leq \frac{m-1}{d_i} \leq w - \frac{1}{d_i} + 1$ so $\lfloor \frac{m-1}{d_i} \rfloor \leq w$. Thus $\lfloor \frac{m-1}{d_i} \rfloor = w$. Thus the length of all permutation runs in column $i$ is always at least $\lfloor \frac{m-1}{d_i} \rfloor$ since the start element $x \leq d_i$ will always give at least as long a permutation run as with start element $d_i$.

Since the minimal start element for any permutation run is 1, which will give the longest possible permutation run, it also follows from (1) above that the length of all permutation runs is bounded above by $\lfloor \frac{m-1}{d_i} \rfloor + 1$.

Finally, if $d_i | m-1$, then every permutation run in column $i$ has length $\frac{m-1}{d_i} = \lfloor \frac{m-1}{d_i} \rfloor$.

∎

**Lemma 4.** *Let $T$ be a permutation table with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha$. (Recall $\alpha = (m-1)/(a+1)$.) Then for $2 \leq i \leq c$ we have*

$$\left( \left\lfloor \frac{m-1}{d_i} N \right\rfloor + 1 \right) d_{i-1} < \alpha.$$

*We also have*

$$\left( \frac{Nm + d_i - 2}{d_i} \right) d_{i-1} \leq \alpha$$

*where $\frac{Nm+d_i-2}{d_i}$ is an upper bound on the length of $N$ consecutive permutation runs in column $i$.*

*Proof.* We have

$$\frac{d_i}{d_{i-1}} \geq N(a+1) + 1.$$

Thus

$$1 \geq \frac{d_{i-1}}{d_i} N(a+1) + \frac{d_{i-1}}{d_i}$$

and

$$1/(a+1) \geq \frac{d_{i-1}}{d_i} N + \frac{d_{i-1}}{d_i(a+1)}$$

$$\alpha \geq \frac{m-1}{d_i} N d_{i-1} + \alpha \frac{d_{i-1}}{d_i} > \frac{m-1}{d_i} N d_{i-1} + d_{i-1}$$
$$\geq \left( \left\lfloor \frac{m-1}{d_i} N \right\rfloor + 1 \right) d_{i-1} \tag{2}$$

We have $N(1-\frac{1}{d_i}) \geq \frac{d_i-2}{d_i}$ since $N \geq 2$, $d_i \geq 2$. Thus $\frac{-N}{d_i} + N \geq \frac{d_i-2}{d_i}$. Therefore

$$N \left( \frac{m-1}{d_i} \right) + N \geq \frac{Nm + d_i - 2}{d_i}. \tag{3}$$

From above we have

$$\alpha \geq \frac{m-1}{d_i} N d_{i-1} + \alpha \frac{d_{i-1}}{d_i} \geq \frac{m-1}{d_i} N d_{i-1} + N d_{i-1}$$
$$= \left( \frac{m-1}{d_i} + 1 \right) N d_{i-1} \geq \left( \frac{Nm + d_i - 2}{d_i} \right) d_{i-1}. \tag{4}$$

due to (3) above and $\alpha/d_i \geq N$. It remains to be shown that $\frac{Nm+d_i-2}{d_i}$ is an upper bound on the length of $N$ consecutive permutation runs in column $i$.

For a sequence of permutation runs with start element $x$ of length $N$ we have

$$x + w d_i \leq m - 1 + (N-1)(m) = Nm - 1$$

where $w$ is the number of $d_i$ values added to $x$ to get to the last element in the $Nth$ permutation run (prior to reducing modulo $m$.) Thus

$$w + 1 \leq \frac{Nm + d_i - x - 1}{d_i} \leq \frac{Nm + d_i - 2}{d_i}.$$

∎

**Lemma 5.** *Let $T$ be a permutation table with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $N d_c \leq \alpha$. The target set of short vectors in $T$ has $N^c$ vectors.*

*Proof.* Consider the $Nth$ flip of the $cth$ column that occurs after the first row. All of the $N$ vector components following each of the first $N-1$ flips of the $cth$ column and the first row are target set vector components in the $cth$ column, by definition. By Lemma 4,

$$\left( \left\lfloor \frac{m-1}{d_c} N \right\rfloor + 1 \right) d_i < \alpha$$

for $1 \leq i \leq c-1$. Thus the first column components prior to the $Nth$ flip of the $cth$ column are less than $\alpha$ by Lemma 2. All of the components in column $i$, $2 \leq i \leq c-1$, in rows prior to the $Nth$ flip of the $cth$ column are in rows prior to the $Nth$ flip of column $i$ since $d_i < d_c$. Thus we have $N^2$ target set vectors in the rows prior to the $Nth$ flip of the $cth$ column.

Inductively consider a row prior to the first flip of the $ith$ column, $i < c$, where column $i+1$ has flipped $N$ times and we have collected $N^{c-i+1}$ vectors in the target set. Each of the first $N-1$ flips of the $ith$ column are left of j rows for columns $i+1, \ldots, c$ and each collection of rows between the flips and immediately after the $N-1$ flip yield $N^{c-i+1}$ additional target set vectors for a total of $NN^{c-i+1} = N^{c-i+2}$ target set vectors. The induction is complete and when $i = 2$ we obtain $N^c$ target set vectors in $T$.

∎

We now prove that the vectors in the target set of short vectors are short.

**Lemma 6.** *Let $T$ be a permutation table with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha$. The vectors in the target set of short vectors in $T$ are short.*

*Proof.* We show that the vectors in the target set of vectors (see Definition 4) are short vectors. By Lemma 4, we have

$$\left( \left\lfloor \frac{m-1}{d_2} N \right\rfloor + 1 \right) < \alpha.$$

$\lfloor \frac{m-1}{d_2} N \rfloor + 1$ is at least as large as the the number of rows in the 2nd column that can have target vector components since the 2nd component of the target set vectors must occur prior to the $Nth$ flip of the 2nd column (Lemma 2.) (The only left of j row for the 2nd column is the first row.) Thus all of the target vector components in the first column are short.

We consider the $i-1$ column where $3 \leq i \leq c$. By Lemma 4, we have

$$\left( \left\lfloor \frac{m-1}{d_i} N \right\rfloor + 1 \right) d_{i-1} < \alpha.$$

We consider the vector components in the $i-1$ column prior to the 1st flip after the first row. $\lfloor \frac{m-1}{d_i} N \rfloor + 1$ is at least as large as the number of rows in the $ith$ column that can have target vector components since the $ith$ component of the target set vectors must occur prior to the $Nth$ flip of the $ith$ column. Thus all of the target vector components in the $i-1$ column prior to the first flip after the first row are short. The same argument applies to the target vector components in the later permutation runs of the $i-1$ column since the sum of the permutation run lengths is bounded by $\frac{Nm+d_i-2}{d_i}$ by Lemma 4.

To complete the proof that the target set vector components are short, we examine the $cth$ column. The $cth$ component occurs at most $N-1$ rows after a flip of the $cth$ column. Let $w$ denote the starting element of a $cth$ column permutation run. Then $w \leq d_c$. We have $Nw \leq Nd_c \leq \alpha$ by the conditions of the theorem. Thus the target set vectors are short. See Figure 6 for an illustration of the $c = 4, N = 4$ case. ∎

**Theorem 1.** *Given a circuit with $a$ addition gates, let $N$ be a power of 2, and let $m$ be prime per Definition 3. Let $\alpha = (m-1)/(a+1)$. Given a permutation table $T$ with start vector $(d_1, d_2, \ldots, d_c)$ where $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha$. Then $T$ is a perfect table.*

*Proof.* By Lemma 5, the target set of short vectors has $N^c$ vectors and by Lemma 6 these vectors are short vectors. The set of associated vectors in $\mathbb{Z}_N^c$ has $N^c$ vectors. Thus to prove the theorem, it suffices to show that the mapping from target set vectors defined by $r_i = v_i \bmod N$, $1 \leq i \leq c$ is one to one, given target set vector $(v_1, \ldots, v_c)$.

In order to show a contradiction, suppose there exist vectors $v = (v_1, \ldots, v_c)$ and $w = (w_1, \ldots, w_c)$ in the target set vectors of permutation table $T$ that have the same associated vector in $\mathbb{Z}_N^c$. Then there exists integers $h_v$ and $h_w$ where $h_v < h_w < m$ and $h_v d_i = p_i m + v_i, h_w d_i = q_i m + w_i$, $p_i, q_i \in \mathbb{Z}$, $1 \leq i \leq c$, $1 \leq v_i, w_i < m$, $v_i \cong w_i \bmod N$, $1 \leq i \leq c$. We have $d_1 = 1$ so $p_1 = q_1 = 0$ and $h_v = v_1, h_w = w_1$. Thus $h_v \cong h_w \bmod N$. Thus $p_i \cong q_i \bmod N$, $1 \leq i \leq c$.

Suppose $q_2 \geq N$. Then the 2nd component of $w$ is at a row higher than all of the target set vector components (the first row is the only left of j row for the 2nd column.) So $q_2 < N$. $p_2 \leq q_2$ so $p_2 < N$. Thus $p_2 = q_2 = 0$. Inductively assume $p_i = q_i = 0$ for $i \leq l$ where $2 \leq l < c$. We have $p_{l+1} \leq q_{l+1}$. $q_l = 0$ so the only left of j row for the $l+1$ column that $w$ is below is the first row. Thus $w$ must be in a row prior to the $Nth$ flip of the $l+1$ column. Thus $q_{l+1} < N$. Thus $p_{l+1} < N$ and we have $p_{l+1} = q_{l+1} = 0$. The induction is complete and $p_i = q_i = 0$, $2 \leq i \leq c$.

Since $p_c = q_c = 0$, $v$ and $w$ are located in the first $N$ vectors of the permutation table but $h_v \cong h_w \bmod N$ so $h_v = h_w$ which is a contradiction. Thus the set of target vectors all have distinct associated vectors in $\mathbb{Z}_N^c$. ∎

**Definition 5.** ***Perfect Somewhat Additive Homomorphic Encryption (P-SWAHE)*** *Given a server circuit with $a$ addition gates. $N$ is a positive integer which is a power of 2. See Appendix A for the definition of somewhat additive homomorphic encryption.*

***Key Generation:*** *Select positive prime integer $m$ and uniform random positive integer $b$, the base, where $b < m$. The secret key is $b$. The key is used to encrypt only one plaintext vector. A new key must be selected for each new plaintext vector.*

***Encryption:*** *The input is plaintext vector $\boldsymbol{r} = (r_1, \ldots, r_c)$ where $0 \leq r_i < N$, $1 \leq i \leq c$. Select integers $d_1, \ldots, d_c$ such that $d_i | d_{i+1}$, $1 \leq i \leq c-1$ and $d_1 = 1$. Also $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha = (m-1)/(a+1)$. Let $T$ be the permutation table with $(d_1, \ldots, d_c)$ as its start row vector.*

*For vector $\bar{\boldsymbol{v}}$ in $T$ denote it's associated vector in $\mathbb{Z}_N^c$ by $\boldsymbol{v}$. (Recall that $v_i = \bar{v_i} \bmod N$, $1 \leq i \leq c$.)*

*We compute the vector $(e_1, \ldots, e_c)$ as follows: First we compute the vector $\boldsymbol{v} = (v_1, \ldots, v_c)$ :*

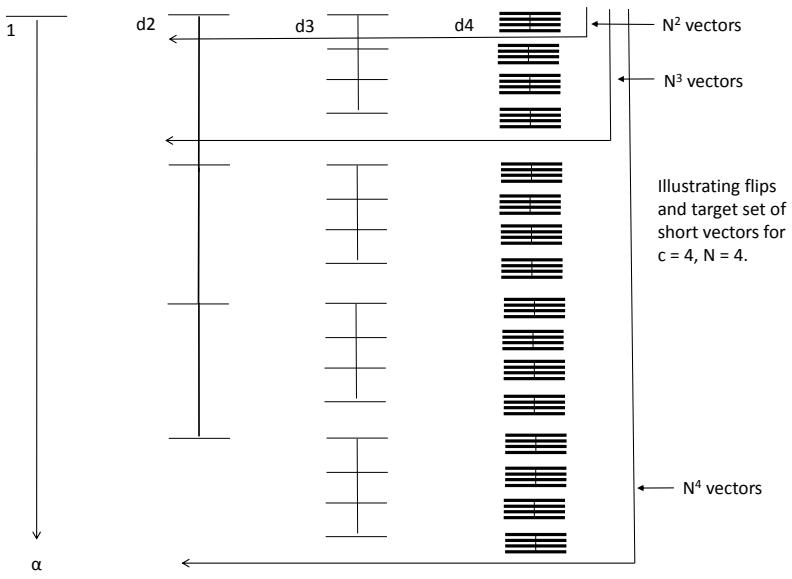



**Fig. 6.** Target Set of Short Vectors in Permutation Table, $c = 4, N = 4$. Short Horizontal Lines Indicate Start of New Permutation Runs (flips). Each Thick Line in Last Column Represents Last Column Components of N vectors in Target Set

1. *If $r_1 = j$, $1 \leq j \leq N-1$, then $\boldsymbol{v}$ is the associated vector of the $j$th row of $T$.*
2. *If $r_1 = 0$, then $\boldsymbol{v}$ is the associated vector of the $N$th row of $T$.*

*Let $\boldsymbol{w} = \boldsymbol{v}$, $s = m \bmod N$, $\xi = s^{-1} \bmod N$, $\boldsymbol{w_{saved}} = zero\ vector$. Then we perform Algorithm 2.*

*Let $\boldsymbol{e} = \bar{\boldsymbol{w}}$ where $\bar{\boldsymbol{w}}$ is the output of Algorithm 2. The components of $\boldsymbol{e}$ are called exponents. $\boldsymbol{e}$ is **the e-vector.***

*The client ciphertext vector is obtained by selecting an integer representative $h_i$, $1 \leq i \leq c$, $1 \leq h_i < m$, where $h_i \equiv be_i \bmod m$. Then $(h_1, \ldots, h_c)$ is the client ciphertext vector.*

***Decryption:** The client decryptor receives an evaluated ciphertext $x$ from the server. The client sets*

$$Decrypt(x) = r = (b^{-1}x \bmod m) \bmod N.$$

***Evaluate:** Evaluate takes the ciphertext tuple from Encrypt and the circuit and outputs another vector of ciphertexts corresponding to the circuit evaluation. Each addition gate is processed by adding the two inputs as integers.*

The client sends the client ciphertext request to the server which uses the input elements as inputs to the circuit. The server returns the output elements to the client.

**Algorithm 2** Encryption Steps for P-SWAHE

1: **while** $\boldsymbol{w} \neq \boldsymbol{r}$ **do**
  $i$ is index of the first component disagreement between $\boldsymbol{w}$ and $\boldsymbol{r}$.
  Let $f = (\xi(w_i - r_i)) \bmod N$.
  Set $\bar{v_j} = \lfloor \frac{m-1}{d_i} f \rfloor d_j \bmod m$, $1 \leq j \leq c$.
  $\bar{\boldsymbol{v}} = (\bar{\boldsymbol{v}} + \boldsymbol{w_{saved}}) \bmod m$
  Advance up or down the rows of $T$ (at most $N$ rows) starting at $\bar{\boldsymbol{v}}$
    until we find $\bar{\boldsymbol{w}}$ where $w_1 = r_1, \ldots, w_i = r_i$.
  $w_j = \bar{w}_j \bmod N$ for $1 \leq j \leq c$, where $\bar{\boldsymbol{w}} = (\bar{w_1}. \ldots, \bar{w_c})$.
  $\boldsymbol{w_{saved}} = \bar{\boldsymbol{w}}$
**endwhile**
**Output:** $\bar{\boldsymbol{w}}$

**Theorem 2.** *Given the P-SWAHE encrypt algorithm which outputs the e-vector* $\boldsymbol{e} = (e_1, \ldots, e_c) \in \mathbb{Z}_m^c$. *Let* $N = 2$. *The vector* $\boldsymbol{e}$ *belongs to the target set of short vectors in the permutation table* $T$ *and* $e_i \bmod N = r_i$, $1 \leq i \leq c$, *where* $\boldsymbol{r} = (r_1, \ldots, r_c) \in \mathbb{Z}_N^c$ *is the plaintext input vector.*

*Proof.* We have $N = 2$ and $\xi = 1$ in Algorithm 2. Thus $f = 1$ when $w_i \neq r_i$. After the initial steps of the encrypt algorithm that compute vector $\boldsymbol{v}$, we have vector $\bar{\boldsymbol{w}}$ in the permutation table where $w_1 = r_1$. Inductively, $\boldsymbol{w_{saved}} = \bar{\boldsymbol{w}}$ where $w_j = r_j$, $1 \leq j \leq i-1$, $w_i \neq r_i$, $i \geq 2$.

Algorithm 2 now obtains a new vector $\bar{w}$ that is in a row after the first flip of the ith column that occurs after the existing $\boldsymbol{w_{saved}}$ vector (row). Note that columns $j < i$ do not flip (since they start with $d_j < d_i$). Thus the columns to the left maintain their existing pattern and there is a vector $\bar{\boldsymbol{v}}$ on the flip row or one row below such that $v_j = w_j = r_j$, $1 \leq j \leq i-1$.

Let $xd_i < m$ and $xd_i + d_i > m$. Let $xd_i \cong l_i \bmod N$. Then

$$(x+1)d_i \bmod m \bmod N = ((x+1)d_i - m) \bmod N = \\ l_i \bmod N + d_i \bmod N - m \bmod N \tag{5}$$

If $d_i$ is odd, then expression (5) equals $l_i \bmod N$. So the *ith* column two row alternating pattern reverses on the flip (we get two rows where the *ith* column element has the same parity). Thus Algorithm 2 obtains vector $\bar{\boldsymbol{w}}$ such that $w_j = r_j, 1 \leq j \leq i$.

If $d_i$ is even, then expression (5) equals $l_i \bmod N+1$. So the existing pattern for the parity of the ith column elements (all even or all odd) now reverses and Algorithm 2 obtains vector $\bar{\boldsymbol{w}}$ such that $w_j = r_j, 1 \leq j \leq i$. By induction, Algorithm 2 obtains vector $\bar{\boldsymbol{w}}$ such that $w_j = r_j, 1 \leq j \leq c$. Thus $e_i \bmod N = r_i, 1 \leq i \leq c$.

It remains to be shown that the Algorithm 2 output vector is in the target set of short vectors. Inductively, the component of $\bar{\boldsymbol{w}}$ in the $i-1$ column when $w_i \neq r_i$ occurs prior to the $Nth$ flip of column $i-1$ that occurs after a flip of column $i-2$. Thus the first $i$ components of $\bar{\boldsymbol{w}}$ will occur prior to the $Nth$ flip

of their respective columns after their left of j rows. Inductively, the argument holds for the $c$ columns.

The $cth$ column component occurs in a row between zero and $N-1$ rows after a flip of the $cth$ column, since a flip of column $j < c$ also flips column $c$.

Consider a binary vector $(r_1, \ldots, r_c)$ where Algorithm 2 finds a disagreement between $r_i$ and $w_i$ for all $i \geq 2$. Such a vector is the deepest row vector that Algorithm 2 can find. So Algorithm 2 slides over in the same row and then down to the next flip of column $i$ for each $i$, $2 \leq i \leq c$. But column $i-1$ does not flip on the row that Algorithm 2 finds, else $d_{i-1} = d_i$ which does not hold. Thus none of the columns $j < i$ can flip during the selection of the row for $r_i \neq w_i$. Inductively, we conclude that the column 2 component of the deepest vector must occur prior to the $Nth$ flip of the 2nd column.

Lemmas 2 and 4 give that the sum of the lengths of these first two permutation runs in the 2nd column is bounded by $\lfloor \frac{(m-1)}{d_2} N \rfloor + 1 \leq \alpha$. So all of vectors output by Algorithm 2 have first column components bounded by $\alpha$. Thus Algorithm 2 always outputs a vector in the target set of short vectors. ∎

**Theorem 3.** *Given $m$ prime, circuit $C$ with $a$ addition gates, $N = 2$, and a permutation table $T$ with start vector $d_1, \ldots, d_c$ as in the setup for P-SWAHE encryption ($d_i | d_{i+1}$, $1 \leq i \leq c$, $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, $\alpha = (m-1)/(a+1)$, and $Nd_c \leq \alpha$.) We also assume $d_i$ is odd, $1 \leq i \leq c$. Given $x$ as input to P-SWAHE decryption corresponding to plaintext input $(r_1, \ldots, r_c)$. Suppose the circuit computes $w_{i_1} + \ldots + w_{i_k} \bmod N$ for some $1 \leq i_i < \ldots < i_k \leq c$ given inputs $w_1, \ldots, w_c$. Then Decrypt$(x) = (r_{i_1} + \ldots + r_{i_k}) \bmod N$.*

*Proof.* The client ciphertext vector is $(h_1, \ldots, h_c)$ where $h_i \cong be_i \bmod m$, $1 \leq i \leq c$. By Theorem 2, $e_i \bmod N = r_i$, $1 \leq i \leq c$ where $(e_1, \ldots, e_c)$ is in the target set of short vectors in $T$. We have $x = \sum_{j=1}^{k} h_{i_j}$. Thus

$$b^{-1}x \cong \sum_{i=1}^{k} b^{-1}h_{i_j} \cong \sum_{i=1}^{k} e_{i_j} \bmod m.$$

$e_i \leq (m-1)/(a+1)$ so $\sum_{i=1}^{k} e_{i_j} \leq k(m-1)/(a+1)$. Now $k \leq a+1$ since there are $a$ addition gates in the circuit so

$$k\,\frac{m-1}{a+1} \leq (a+1)\frac{m-1}{a+1} \leq m-1.$$

Thus

$$b^{-1}x \bmod m = \sum_{i=1}^{k} e_{i_j}$$

and

$$Decrypt(x) = \sum_{i=1}^{k} e_{i_j} \bmod N = \sum_{j=1}^{k} r_{i_j} \bmod N.$$

∎

**Theorem 4.** *The P-SWAHE scheme has perfect security.*

*Proof.* We will show that

$$Pr[Enc_{b_1}(\boldsymbol{r_1}) = \boldsymbol{w}] = Pr[Enc_{b_2}(\boldsymbol{r_2}) = \boldsymbol{w}]$$

where $b_1, b_2$ are uniform random in $[1, m-1]$, $\boldsymbol{w}$ is a ciphertext, and $\boldsymbol{r_1}, \boldsymbol{r_2}$ are plaintexts. $Enc_{b_1}(\boldsymbol{r_1}) = b_1\boldsymbol{e} \bmod m$ where $\boldsymbol{e} \in T$ is computed according to the steps in the P-SWAHE encrypt algorithm. $\boldsymbol{w}$ is a ciphertext so $\boldsymbol{w} \in T$, where $T$ is the permutation table created by the encryption steps (note that $T$ is determined by $m$ and $\boldsymbol{w}$.) We will show that

$$Pr[b_1\boldsymbol{e} = \boldsymbol{w} \bmod m] = \frac{1}{m-1}$$

which suffices for the proof since the same argument proves $Pr[Enc_{b_2}(\boldsymbol{r_2}) = \boldsymbol{w}] = 1/(m-1)$. $\boldsymbol{e} = t_1\boldsymbol{s}$ for some $t_1 \in [1, m-1]$ where $\boldsymbol{s}$ is the start vector for $T$. $\boldsymbol{w} = t_2\boldsymbol{s}$ for some $t_2 \in [1, m-1]$ since $\boldsymbol{w} \in T$. $\boldsymbol{w} = t_2\boldsymbol{s} = t_2t_1^{-1}\boldsymbol{e}$ where $t_1^{-1}$ is the inverse of $t_1$ modulo $m$ ($m$ is prime). So $Pr[b_1\boldsymbol{e} = \boldsymbol{w} \bmod m] \geq 1/(m-1)$. Let $b_1\boldsymbol{e} = y\boldsymbol{e} \bmod m$ for $b_1, y \in [1, m-1]$. Then $b_1^{-1}ye_1 = e_1 \bmod m$ where $e_1$ is the first coordinate of $\boldsymbol{e}$. Since $e_1 \in ]1, m-1]$, we have $b_1^{-1}y = 1 \bmod m$ so $b_1 = y$. Thus

$$Pr[b_1\boldsymbol{e} = \boldsymbol{w} \bmod m] = \frac{1}{m-1}$$

∎

*Remark 3.* For P-SWAHE we have $m \geq (N(a+1))^c + 1$.

### 3.3 Incorporating Multiplication

The multiplication of $be_1 \bmod m$ and $be_2 \bmod m$ where $e_i \bmod N = r_i$, $i = 1, 2$ is defined to be $be \bmod m$ for some $e$ where $e \leq (m-1)/(a+1)$ and $e \bmod N = r_1r_2 \bmod N$. [2]

**P-SWAHE with Multiplication Scheme** We will use the P-SWAHE scheme described above. Each multiplication gate will add 1 to the $c$ parameter.

At a high level, P-SWAHE with Multiplication works by dynamically adjusting the secret key and the plaintext vector in response to each multiplication gate operation. Each such step keeps the ciphertext vector the same. Put another way, we hop across rows in the permutation table to land in the row with all of the correct products for the multiplication gates. Communication between the client and server involves the client sending the ciphertext vector to the server and the server either sending the multiplicands at each multiplication gate operation or batching all of the multiplicands together with the final output. In

**Algorithm 3** P-SWAHE with Multiplication

1: **Initialize** with P-SWAHE parameters: Circuit $C$ with $a$ addition gates, $g$ multiplication gates. $N$ a power of 2, $m$ prime, $c_1$ is number of circuit inputs, $c = c_1 + g$. Client creates $\boldsymbol{r} = r_1, \ldots, r_{c_1}, r_1^{'}, \ldots, r_g^{'}$ where $r_1^{'}, \ldots, r_g^{'}$ are arbitrary, $r_1, \ldots, r_{c_1}$ are client plaintext inputs. Obtain output $\boldsymbol{e^0}$ using $\boldsymbol{r}$ as plaintext input to P-SWAHE encryption. $b$ is secret key. Client sends $b\boldsymbol{e^0}_1 \bmod m$ to the server. Let $b_0 = 1$.

2: **Client/Server Processing:**

Server assigns client inputs $be_1, \ldots, be_{c_1}$ to the circuit $C$ input wires.
Denote the multiplicands for the $g$ multiplication gates by $bw_1^1, bw_2^1, \ldots, bw_1^g, bw_2^g$.
**Addition Gates:** Server adds the inputs as integers to obtain gate output
**Multiplication Gate** $i$**:** Server sends multiplicands $bw_1^i, bw_2^i$ to client.
(Alternatively, multiplicands for the gates can be batched with the final output.)
The client computes $t_j^i = (bb_0 \cdots b_{i-1})^{-1} bw_j^i \bmod m$, $j = 1, 2$
Let $x_j = t_j^i \bmod N$, $j = 1, 2$.
The client computes $s_i = x_1 x_2 \bmod N$; we take $0 < s_i < N$.
Client creates $\boldsymbol{v}_i = r_1, \ldots, r_{c_1}, s_1, \ldots, s_i, r_1^{'}, \ldots, r_{g-i}^{'}$; $r_1^{'}, \ldots, r_{g-i}^{'}$ arbitrary
Client inputs $\boldsymbol{v}_i$ into P-SWAHE encryption step to obtain $\boldsymbol{e^i}$.
Let $b_i \boldsymbol{e^i} = \boldsymbol{e^{i-1}} \bmod m$.
Server assigns $b\boldsymbol{e^0}_{c_1+i} \bmod m$ as output of multiplication gate.
**Output:** Server returns circuit output $o$ to client. Let $b_f = bb_1 b_2 \cdots b_g \bmod m$.
Client computes plaintext output as $b_f^{-1}(o) \bmod m \bmod N$.

the latter case, the protocol has one message from the client to the server and a single return message to the client.

P-SWAHE with multiplication works as follows. Initially, the client executes P-SWAHE key generation and encryption with $c = c_1 + g$ where $c_1$ is the number of client inputs and $g$ is the number of multiplication gates in the circuit. The last $g$ plaintext vector components are arbitrary. The client sends the $c$ vector components $b\boldsymbol{e} \bmod m$ to the server. The server assigns $be_1 \bmod m, \ldots, be_{c_1} \bmod m$ to the circuit input wires. The server processing of addition gates does not create additional overhead whereas multiplication gates require that the server sends the multiplicands to the client. The client uses the multiplicands to create an updated secret decryption key. The outputs for the multiplication gates are the last $g$ elements in the vector $b\boldsymbol{e} \bmod m$. The reader can see Algorithm 3 for details. For P-SWAHE encryption, the approximate communication complexity is $\log(N(a+1))[2g^2 + 2g + 2c_1 + 2c_1 g]$ which follows from Remark 3 and the fact that only the first element of a request vector needs to be sent to the server.

**Theorem 5.** *The P-SWAHE with Multiplication (Algorithm 3) returns the correct output to the client. More precisely, let $o_1$ by the output obtained from the circuit $C$ when the client's $c_1$ plaintext inputs are assigned to $C$'s input wires. Let $o$ be the output obtained by the client using P-SWAHE with Multiplication. Then $o_1 = o$.*

[2] Both addition and multiplication of the plaintexts are in the ring $\mathbb{Z}_N$ which is a field when $N = 2$. $N$ is always a power of 2.

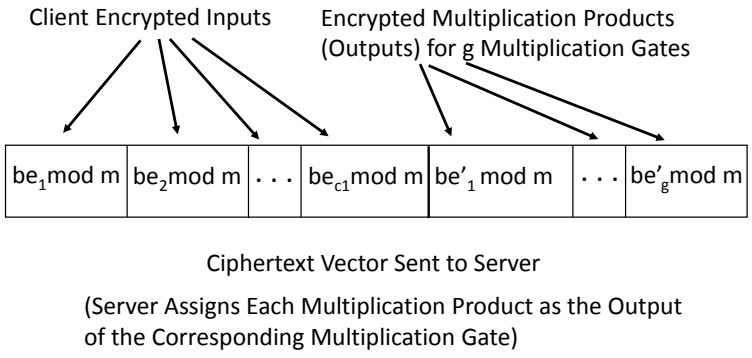



**Fig. 7.** Structure of Ciphertext Vector for P-SWAHE with Multiplication

*Proof.* For the first 2 multiplicands, we obtain $s_1$ on the client ($s_1$ is the product) and then the new vectors $\boldsymbol{v_1}$ and $\boldsymbol{e^1}$. $\boldsymbol{e^1}$ has the correct plaintext components in the first $c_1 + 1$ components since ${\boldsymbol{e^1}}_j \bmod N = r_j$, $1 \leq j \leq c_1$ and ${\boldsymbol{e^1}}_{c_1+1} \bmod N = s_1$. $b_1\boldsymbol{e}^1 = \boldsymbol{e}^0 \bmod m$ so $bb_1\boldsymbol{e}^1 = b\boldsymbol{e}^0 \bmod m$. Thus we may consider the e-vector, secret key pair for the existing ciphertext to be $\boldsymbol{e}^1$ and $bb_1 \bmod m$.

Inductively consider the e-vector, secret key pair for the existing ciphertext as $\boldsymbol{e}^j$ and $bb_1 \cdots b_j \bmod m$ where $\boldsymbol{e}_i^j \bmod N = r_i$, $1 \leq i \leq c_1$ and $\boldsymbol{e}_i^j \bmod N = s_i$, $c_1 + 1 \leq i \leq c_1 + j$. Also, $bb_1 \cdots b_j\boldsymbol{e}^j = b\boldsymbol{e}^0 \bmod m$. Given multiplicands $bw_1^{j+1}$, $bw_2^{j+1} \bmod m$. The multiplicands can be written as $bb_1 \cdots b_j\bar{w}_k^{j+1}$, $k = 1, 2$ where $\bar{w}_k^{j+1}$ are sums of the first $c_1 + j$ components of $\boldsymbol{e}^j$. Then the client computes $(bb_1 \cdots b_j)^{-1}bb_1 \cdots b_j\bar{w}_k^{j+1} \bmod m$, $k = 1, 2$ to obtain $\bar{w}_k^{j+1}$, $k = 1, 2$. Thus $s_{j+1}$ is the plaintext product, mod$N$, of $\bar{w}_1^{j+1}$ and $\bar{w}_2^{j+1}$. Therefore

$$\boldsymbol{v}_{j+1} = (r_1, \ldots, r_{c_1}, s_1, \ldots s_{j+1}, r'_1, \ldots, r'_{g-j-1}),$$

$r'_1, \ldots, r'_{g-j-1}$ arbitrary, is input into the P-SWAHE encryption to get $\boldsymbol{e}^{j+1}$ where $\boldsymbol{e}_i^{j+1} \bmod N = r_i$, $1 \leq i \leq c$, and $\boldsymbol{e}_i^{j+1} \bmod N = s_i$, $c_1+1 \leq i \leq c_1+j+1$. $b_{j+1}\boldsymbol{e}^{j+1} = \boldsymbol{e}^j$ so $bb_1 \cdots b_{j+1}\boldsymbol{e}^{j+1} = b\boldsymbol{e}^0 \bmod m$. The induction is complete.

Now consider the vector $\boldsymbol{e^g}$ in $T$ : $\boldsymbol{e^g} = (e_1^g, \ldots, e_c^g)$ where $\boldsymbol{e}_i^g \bmod N = r_i$ if $1 \leq i \leq c_1$, and $\boldsymbol{e}_{c_1+i}^g \bmod N = s_i$ if $1 \leq i \leq g$. We have $s_i$, $1 \leq i \leq g$, is the product of the multiplicands for the corresponding multiplication gate. Thus

$$b_f\boldsymbol{e^g} = bb_1 \cdots b_g\boldsymbol{e^g} = b\boldsymbol{e^0} \bmod m$$

which is the ciphertext vector sent to the server. Note that the solution $b_i\boldsymbol{e^i} = \boldsymbol{e^{i-1}}$ always exists since any two vectors in a permutation table are both scalar multiples of the start vector. Thus it follows from Theorem 3 and the homomorphic property of the multiplication that the decryption

$$(b_f)^{-1}o \bmod m \bmod N$$

gives the correct output of the circuit. ∎

**Theorem 6.** *The P-SWAHE with Multiplication (Algorithm 3) has perfect security.*

*Proof.* The only information sent to the server is a ciphertext vector from the P-SWAHE encryption algorithm in Algorithm 2. By Theorem 4, P-SWAHE with Multiplication has perfect security. ∎

**Subsequent Client Requests Can Be Processed Locally** In this section, we assume that the client has executed a previous request to the server as described in Algorithm 3, and that the client has stored the multiplicands from this previous request denoted by $bw_1^1, bw_2^1, \ldots, bw_1^g, bw_2^g$. The client can compute the output given new client inputs without communication with the server. See Algorithm 4. Thus the multiplicands can be viewed as an encoding of the circuit.

**Algorithm 4** P-SWAHE with Multiplication: Subsequent Request

1: **Initialize** with P-SWAHE parameters: Circuit $C$ with $a$ addition gates, $g$ multiplication gates. The last gate prior to output in $C$ is a multiplication gate. $N$ a power of 2, $m$ prime, $c_1$ is number of circuit inputs, $c = c_1 + g$.
Client creates $\boldsymbol{r} = r_1, \ldots, r_{c_1}, r_1', \ldots, r_g'$ where $r_1', \ldots, r_g'$ are arbitrary, $r_1, \ldots, r_{c_1}$ are client plaintext inputs. Create $\boldsymbol{f^0}$ using $\boldsymbol{r}$ as plaintext input to P-SWAHE encryption. $b$ is secret key from previous request. $bw_1^1, bw_2^1, \ldots, bw_1^g, bw_2^g$ are multiplicands from previous server reply, and $(e_1, \ldots, e_{c_1+g})$ is the e-vector from the initial request in the previous invocation. Let $o$ be the last component of the vector $b\boldsymbol{f^0} \bmod m$. Let $b_0 = 1$.

2: **Client Processing:**
Let $x = e_1^{-1} f_1^0 \bmod m$.
**for** $i = 1$ **to** $i = g$
Let $t_j^i = (bb_0 \ldots b_{i-1})^{-1} x b w_j^i \bmod m$, and $x_j = t_j^i \bmod N$, $j = 1, 2$.
$s_i = x_1 x_2 \bmod N$; we take $0 < s_i < N$.
$\boldsymbol{v}_i = r_1, \ldots, r_{c_1}, s_1, \ldots, s_i, r_1', \ldots, r_{g-i}'$; $r_1', \ldots, r_{g-i}'$ arbitrary
Input $\boldsymbol{v}_i$ into P-SWAHE encryption step to obtain $\boldsymbol{f^i}$.
Let $b_i \boldsymbol{f^i} = \boldsymbol{f^{i-1}} \bmod m$.
**end for**
**Output:** Let $b_f = bb_1 b_2 \cdots b_g \bmod m$.
Client computes plaintext output as $b_f^{-1}(o) \bmod m \bmod N$.

**Theorem 7.** *The P-SWAHE with Multiplication for Subsequent Requests returns the correct output to the client when subsequent replies are processed via Algorithm 4. More precisely, let $o_1$ by the output obtained from the circuit $C$ when the client's $c_1$ plaintext inputs are assigned to $C's$ input wires. Let $o$ be the output obtained by the client using P-SWAHE with Multiplication for Subsequent Requests. Then $o_1 = o$.*

*Proof.* We must show that

$$(bb_1, \ldots, b_g)^{-1}(bf_c^0) \bmod m \bmod N$$

is the correct plaintext output of the circuit (output of the final multiplication gate).

$$(bb_1, \ldots, b_g)^{-1}(bf_c^0) = f_c^g \bmod m.$$

$f_c^g = s_g \bmod N$.

$$t_j^g = (bb_0, \ldots, b_{g-1})^{-1} e_1^{-1} f_1^0 b w_j^g = f_1^{g-1} e_1^{-1} w_j^g \bmod m$$

Now $f_1^{g-1} e_1^{-1} w_j^g \bmod m$, $j = 1, 2$ gives the input elements at the $gth$ multiplication gate and we have $f_1^{g-1} e_1^{-1} w_j^g \bmod m \bmod N$ $j = 1, 2$ are plaintexts obtained using elements that decrypt to correct plaintexts at the first $g-1$ multiplication gates. Thus $f_g^c = s_g = x_1 x_2 \bmod N$ is the correct plaintext output. ∎

**Theorem 8.** *The P-SWAHE with Multiplication for Subsequent Requests (Algorithm 4) has perfect security.*

*Proof.* The only information sent to the server is a ciphertext vector from the P-SWAHE encryption algorithm in Algorithm 2. By Theorem 4, P-SWAHE with Multiplication for Subsequent Requests has perfect security. ∎

## 4 Two-Party Computation (2PC)

In this section, we consider the case where the server also has inputs for the circuit. As in Section 3, the client provides its (encrypted) inputs to the server. The server processes the client and server inputs through the circuit, and it returns the output to the client. If we assume the server is semi-honest, or another mechanism is used to show the server has not deviated from the protocol, then the client may share the decrypted output with the server (see Theorem 11).

We support boolean ($N = 2$) circuits with both addition (XOR) and multiplication (AND or NAND) gates in this section. We will see that client privacy holds even if the server is fully malicious and computationally unbounded when the client does not share the output with the server (see Theorem 10). (If the client shares the decrypted output with the server then a malicious server could learn additional information about client input values since the server could have returned as output the sum of one of its encrypted input values with a client encrypted input value.)

Server privacy depends on the hardness of the DPT-HSSP problem, DDH, and our assumption that the client is semi-honest.

In this section and Section 5, we will assume $N = 2$.

Our protocol includes a preprocessing step (see Section 4.2) that occurs prior to client and server inputs and subsequent processing. The parties engage in a

protocol that includes using the CL public key algorithm (see Section 2.4) to create and send the ciphertext vector first component, $be_1$ to the server.

For processing a multiplication gate, the server computes random hidden subset sums of the encrypted noise elements and adds them to each multiplicand. Four elements are sent to the client.[3] Each multiplicand is added to either an odd sum (with probability 1/2) or an even hidden subset sum prior to both of the resulting elements being sent to the client. We define an even (odd) sum as a sum where the sum of the e-vector components is even (odd). The client is able to obtain the least significant bit after multiplying by $b^{-1} \bmod m$ for both of the elements and return four products. (To save bandwidth, the client only returns the indices of the first three products in the P-SWAHE encrypted vector. The server knows the location of the fourth product.) Three of the products will be discarded by the server and the client does not know which of the products is the correct one (see Figure 9).

### 4.1 Hardness of PT-HSSP and DPT-HSSP

Here we sketch an argument that solving Permutation Table HSSP with 2PC protocol setup and processing (Algorithm 5 and Algorithm 7) implies an adversary with non-negligible advantage against CPA security of CL encryption.

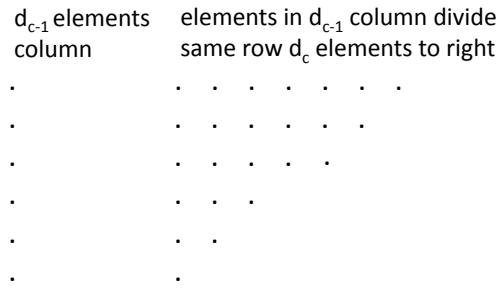



**Fig. 8.** $k = 2$ case: the possible $d_{c-1}$ column elements (represented by dots) are increasing down the column and the larger elements divide a smaller number of possible $d_c$ elements

**Lemma 7.** *Given $d_1 = 1, d_2, \ldots, d_u$ and prime modulus $m$. Let $s = 2(a+1)+1$. We have $d_{u+1}, \ldots, d_c$ where $c = u + k$. $d_i | d_{i+1}, d_{i+1} > 2(a+1)d_i$, $1 \leq i \leq c-1$. $a$ is an integer equal to $k$ plus a constant. Also, $2(a+1)d_c \leq m-1$. The number of possibilities, $\eta$, for $ld_{u+1}, \ldots, ld_c$ where $l$ is an index of a possible e-vector row*

[3] Only two elements have to be sent; one element from each pair is sent. The client knows the other values after decrypting these two elements. The rest of the paper assumes this optimization.

*is approximately* $2^k n^k/k!$, *where*

$$n = \frac{1}{d_u}\left(\frac{m-1}{2(a+1)s^{k-1}} - s^u + 1\right)$$

*Proof.* We have $d_i \leq d_{i+c-i}/s^{c-i}$ since $d_i \leq d_{i+1}/s$, $1 \leq i \leq c-1$. Thus $d_i \leq d_c/s^{c-i} \leq (m-1)/(s^{c-i}(2(a+1)))$. $d_i > 2(a+1)d_{i-1} > \ldots > (2(a+1))^{i-1}$. Thus

$$s^{u+i-1} \leq d_{u+i} \leq \frac{m-1}{s^{k-i}2(a+1)}$$

for $1 \leq i \leq k$, since $k = c-u$. Consider the $k=2$ case which is illustrated in Figure 8. The possible $d_{c-1}$ elements values increase in the later rows so they have fewer $d_c$ elements that they divide. Thus the total number of $d_{c-1}, d_c$ tuples is approximately $n(n+1)/2 \approx n^2/2!$ where $n$ is the number of $d_c$ elements in the first row which follows from the above bounds (note that $sd_{c-2} = d_{c-1}$ on the 1st row). There are 4 possibilities for $l$ giving $2^k n^k/k!$ as claimed. The sum of $n$ integers does not hold exactly (there can be a few gaps) hence the approximation. Faulhaber's formula [20] gives the sum of the $p$th powers of the first $n$ integers; we approximate this sum as $1^p + 2^p + \ldots + n^p \approx n^{p+1}/(p+1)$. For the inductive step, we have the number of possibilities for $d_{u+1}, \ldots, d_{u+k-1}$ to be approximately $n^{k-1}/(k-1)!$. Then $n^{k-1}/(k-1)! + (n-1)^{k-1}/(k-1)! + \ldots + 1/(k-1)! \approx n^k/k(k-1)!$. Incorporating $l$ we have $2^k n^k/k!$.

**Theorem 9.** *Given* $d_1 = 1, d_2, \ldots, d_u$ *and prime modulus* $m$. *Given* $d_{u+1}, \ldots, d_c$ *where* $c = u+k$. $d_i | d_{i+1}, d_{i+1} > 2(a+1)d_i$, $1 \leq i \leq c-1$. $a$ *is an integer equal to* $k$ *plus a constant. Also,* $2(a+1)d_c \leq m-1$. *Fix* $l$ *corresponding to a selection of input values for 2PC protocol. The hidden subset sum weights are* $ld_{u+i} \bmod m$, $1 \leq i \leq k$. *The adversary is gven* $w$ *subset sum samples* $(\sum_{i=1}^{u} s_i^j l d_i + \sum_{i=1}^{k} t_i^j l d_{u+i} \bmod m$, $t_i \leftarrow \{0,1\}$, $1 \leq j \leq w)$, $w < k$, *and must find the hidden weights. Then as* $m$ *and* $k$ *increase the adversary with non-negligible advantage against PT-HSSP with setup from Algorithm 5 has non-negligible advantage against CPA security of CL encryption.*

*Proof.* The number of $d_{u+1}, \ldots, d_c$ tuples is approximately $n^k/k!$, where

$$n = \frac{1}{d_u}\left(\frac{m-1}{2(a+1)s^{k-1}} - s^u + 1\right)$$

from Lemma 7. The Stirling approximation is $k! < e(k/e)^k k^{1/2}$. Thus

$$\frac{n^k}{k!} > \frac{n^k}{e(k/e)^k k^{1/2}} = \frac{(n/k)^k e^{k-1}}{k^{1/2}}$$

If we fix $k > w$, then $((n/k)^k e^{k-1})/(k^{1/2} m^w) \to \infty$ as $m \to \infty$ since $n \to \infty$ as $m \to \infty$ given fixed $k$ and $d_u$. Thus the number of hidden weight tuples exceeds the number of $w$ sample tuples as $m$ grows and the hidden weights are not unique given a $w$ sample tuple. Thus the successful PT-HSSP adversary must have non-negligible advantage against CPA security of CL encryption.

**Hardness of DPT-HSSP:** The existing proof of PT-HSSP hardness does not count the $2^k$ possible $l$ values. If we consider the server plaintext input elements as fixed and included in $d_1, \ldots, d_u$, then it is highly likely that every $w$ sample gets mapped to by many of the $d_{u+1}, \ldots, d_c$ random tuples [4] Thus it is likely that almost all $w$ samples are mapped to by every possible server encoded plaintext vector. Thus we conjecture that DPT-HSSP is hard. We have tested small examples to verify that multiple server input vectors map to the same $w$ sample.

### 4.2 Initial Setup Using CL Public Key Algorithm

**Algorithm 5** Preprocessing Steps for 2-Party Protocol with Perfect Security for Client, Steps 1-4 Occur Prior to Introduction of Data

1: **Client Inputs:** client plaintext inputs, $pk$ (server CL public key), $c_1$
**Server Inputs:** server plaintext inputs, $pk$, $sk$ (server CL private key), circuit $C$
2: Client has $c_1$ parameter: $c_1 = c_c + 4g$ where $g$ is the number of multiplication gates in circuit and $c_c$ is number of client inputs. Client generates $m_{cand1}$ (prime integer) and $a_{cand1}$ ($m$ and $a$ candidates); client sends $m_{cand1}$, $a_{cand1}$, $c_1$ to server.
3: Server sets $a \geq a_1 + k$, where $k$ is number of noise elements and $a_1$ is maximum number of elements that will be added together in circuit (the output of a multiplication gate always counts as one element). Set $k > 2g + 1 = w$ for server security (Theorem 9). Server sends $m_{cand2}$ and $a$ to client.
4: Client accepts $m = m_{cand2}$ (checks that $m_{cand2}$ is prime) or selects $m > m_{cand2}$ where $m$ is prime and sends $m$ to the server. The client sets $d_1 = 1, d_2, \ldots, d_{c_1}$, $d_i > 2(a+1)d_{i-1}$, $d_{i-1}|d_i$, $c_1 \geq i \geq 2$ and sends $d_1, \ldots, d_{c_1}$ to server.
5: Server sets $c = c_c + 4g + c_s + k$; the server creates $d_{c_1+1}, \ldots, d_c$, $2(a+1)d_c \leq m-1$, $d_i|d_{i+1}, d_{i+1} > 2(a+1)d_i, c_1 \leq i \leq c-1$ and creates $k$ random noise plaintext bits, $d_{c_1+1}/d_{c_1}$ is an even integer. The server runs Algorithm 2 with the plaintext binary vector $(0, \ldots, 0,$ plaintext server bits, plaintext noise bits) where the zero bit prefix has $c_1$ bits. Let the first component of the output vector be $e_s$. The server sends $Encrypt(1^l, pk, e_s)$ to the client.
6: Client and server run modified Okamoto protocol to prove server ciphertext consists of a pair of elements in the cyclic group $G$ (see Section 4.4).
7: Client creates $g$ blocks each with 4 multiplication bits (3 zero bits and 1 one bit). The one bit is uniform randomly placed in each block. Client selects uniform random secret key $b$ where $b \leftarrow \{2, \ldots Bm - 1\}$ and $B$ is the constant for CL encryption (Figure 4.) If $m|b$, then reselect $b$. Client runs Algorithm 6 with the $c_1$ component vector (client plaintext bits, multiplication bits). The first component of the output vector is $e_{cli}$. Client computes $y = EvalSum(1^l, pk, Encrypt(1^l, pk, e_s), Encrypt(1^l, pk, e_{cli} - 2))$ (see Figure 4). Client computes $z = EvalScal(1^l, pk, y, b)$ and sends $z$ to server.
8: The server computes $be_1 = Decrypt(1^l, pk, sk, z)$ where $\boldsymbol{e} = (e_1, \ldots, e_c)$ is the e-vector corresponding to the client and server input bits. We have $e_1 = e_s + e_{cli} - 2$.

[4] $d_{u+1}, \ldots, d_c$ map to a $w$ sample $\boldsymbol{v}$ if there exists $t_i \in \{0, 1\}$ such that $\sum_{i=1}^{u} s_i^j l d_i + \sum_{i=u+1}^{c} t_i^j l d_i = \boldsymbol{v_j} \bmod m$, $j = 1, \ldots, w$.

Algorithm 5 describes the initial setup algorithm.

**Proposition 1.** *Given $m$ prime, $d_1 = 1, d_2, \ldots, d_c$ where $d_i < 2(a+1)d_{i+1}$, $1 \leq i \leq c-1$, $d_i | d_{i+1}$, $1 \leq i \leq c-1$. We have $1 < c_1 < c$ and $d_{c_1+1}/d_{c_1}$ is an even integer. Let $(r_1, \ldots, r_{c_1}) \in \{0,1\}^{c_1}$. Algorithm 6 computes $e_1$ where $\boldsymbol{e} = (e_1, \ldots, e_c)$ is a row in the permutation table $T$ satisfying $e_i \bmod 2 = r_i$, $1 \leq i \leq c_1$, and $e_i \bmod 2 = 0$, $c_1 + 1 \leq i \leq c$.*

*Proof.* First consider the case where $d_c | m-1$. Then it is clear we obtain a vector that has a longer prefix that equals the target vector $\boldsymbol{r}$ prefix on each iteration of Algorithm 6. Also, on the iterations after the first iteration, we only slide at most 1 row since the permutation runs have constant size. Thus when $d_c \nmid m-1$, the algorithm also converges since the iteration target vector will be off by at most 1 additional row and we obtain that vector by sliding at most 2 rows. Thus we obtain $\boldsymbol{w}$ such that $w_i = r_i, 1 \leq i \leq c_1$ after at most $c_1$ iterations.

We now show that the last $c - c_1$ components of $\boldsymbol{e}$ have even parity. $d_j/d_{c_1} = (d_j/d_{c_1+1})(d_{c_1+1}/d_{c_1})$ is even for $c \geq j \geq c_1 + 1$. Then $d_j/d_i = (d_j/d_{c_1})(d_{c_1}/d_i)$ is even for $i \leq c_1, j \geq c_1 + 1$. In this case, $d_j/d_i$ is the number of flips of column $j$ when column $i$ flips once. So the latter column elements ($j \geq c_1 + 1$) all return to their original parity by the end of the algorithm. Since $d_{c_1+1}, \ldots, d_c$ are even, then these parities are zero. Note that the client doesn't need to know the last $c - c_1$ components of $\boldsymbol{e}$.

**Algorithm 6** Encryption Steps for Client in 2PC Protocol Setup

**Input:** $\boldsymbol{r}$ a binary vector with $c_1$ components
If $r_1 = 1$, then $\boldsymbol{v}$ is the associated vector of the $1st$ row of $T$.
If $r_1 = 0$, then $\boldsymbol{v}$ is the associated vector of the $2nd$ row of $T$.
$\boldsymbol{w} = \boldsymbol{v}$, $\boldsymbol{w_{saved}} = zero\ vector$.
**while** $\boldsymbol{w} \neq \boldsymbol{r}$ **do**
  $i$ is index of the first component disagreement between $\boldsymbol{w}$ and $\boldsymbol{r}$.
  Set $\bar{v_j} = \lfloor \frac{m-1}{d_i} \rfloor d_j \bmod m$, $1 \leq j \leq c_1$.
  $\bar{\boldsymbol{v}} = (\bar{\boldsymbol{v}} + \boldsymbol{w_{saved}}) \bmod m$
  Advance up or down the rows of $T$ (at most 2 rows) starting at $\bar{\boldsymbol{v}}$ until we find $\bar{\boldsymbol{w}}$ where $w_1 = r_1, \ldots, w_i = r_i$.
  $w_j = \bar{w}_j \bmod 2$ for $1 \leq j \leq c_1$, where $\bar{\boldsymbol{w}} = (\bar{w_1}, \ldots, \bar{w_{c_1}})$.
  $\boldsymbol{w_{saved}} = \bar{\boldsymbol{w}}$
**endwhile**
**Output:** $\bar{w_1}$

### 4.3 Circuit Processing for 2PC Protocol

See Algorithm 7 and Figure 9.

**Algorithm 7** Steps for 2-Party Protocol with Perfect Security for Client: Multiplication, Addition Gates and Output

1: When creating the plaintext vector for the 2PC protocol, the client creates a sequence of contiguous multiplication blocks. Each multiplication block is a random permutation of 4 bits with 3 zero bits and 1 one bit. The server has access to the encrypted vector as described in Algorithm 5.

2: When the server needs to multiply $be_1 \bmod m$ and $be_2 \bmod m$, it creates two elements (in $\mathbb{Z}_m$) to send to the client: $be_1$ plus an even or odd parity hidden subset sum element (using the $k$ encrypted noise elements), and $be_2$ plus an even or odd parity subset sum element where the parity of each hidden subset sum element is even with 1/2 probability.
**for** $i = 1$ to 2 **do**
$\boldsymbol{z} = (be_{c_1+c_s+1}, \ldots, be_c)$
$\boldsymbol{s} \leftarrow \{0,1\}^k$
$sum_i \leftarrow \boldsymbol{s} \cdot \boldsymbol{z} \bmod m$
define parity$(sum_i) = \sum_{s_j=1} parity(e_j)$
**end for**
$x_1 = be_1 + sum_1 \bmod m$.
$x_2 = be_2 + sum_2 \bmod m$.
3: $x_1$ and $x_2$ are sent to the client.
4: Client sets $r_j = parity(b^{-1}x_j)$, $j = 1, 2$. $r_3 = 1 - r_1$, $r_4 = 1 - r_2$;
5: Client sets $f_1 = r_1 r_2$, $f_2 = r_1 r_4$, $f_3 = r_3 r_2$, and $f_4 = r_3 r_4$.
6: Client accesses the first unused multiplication block in the plaintext vector; if $f_i = 0$, $1 \leq i \leq 3$, then $index(f_i)$ is the index of one of the zero bits in the block, selected randomly. If $f_i = 1$, then $index(f_i)$ is the index of the one bit in the block.
7: Client sends $index(f_1), index(f_2), index(f_3)$ to server
8: Server assigns the unused entry in the encrypted multiplication block to $f_4$. If $f_l = r_i r_j$ where $r_i$ and $r_j$ correspond to elements with even subset sums, then let the gate output be the entry in the encrypted vector in location $index(f_l)$.
9: Addition (XOR) gates are processed by the server without client interaction; the two inputs are added as integers to get the output. NOT gates can be processed by adding an element with odd parity to the gate input element. (NOT gates add 1 to the $a_1$ parameter).
10: An output $o$ is summed with an even parity subset sum element $s$ to get $s+o \bmod m$ which is sent to the client. The client computes the plaintext output as $parity(b^{-1}(s+o) \bmod m)$. The client does not share the plaintext output with the server unless the server is semi-honest or there is a mechanism for verifying that the server has followed the protocol.

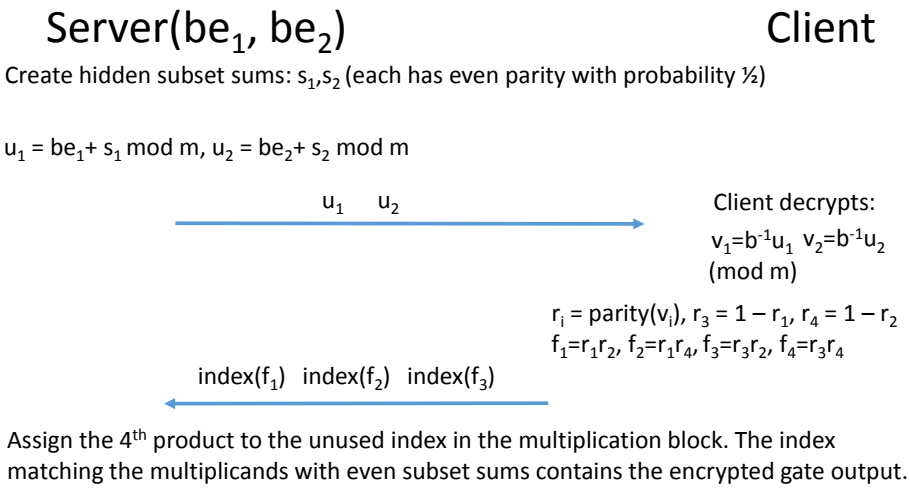



**Fig. 9.** Multiplication Gate Protocol for 2PC Scheme

### 4.4 Proof of Security

**Theorem 10.** *Given the 2-party protocol of Algorithms 5, 6, and 7 where the client is semi-honest and does not share the output with the server. The server is computationally unbounded. If the server is semi-honest in the setup protocol (Algorithm 5), then the protocol has server privacy given hardness of the DPT-HSSP and DDH and perfect client privacy including when the server is malicious during the circuit evaluation protocol (Algorithm 7.)* [5]

*Proof.* We use a simulator argument for server privacy. Our 2PC protocol has server privacy if there is a probabilistic polynomial-time algorithm $\mathcal{S}_1$ such that

$$\{\mathcal{S}_1(x, f_1(x,y))\}_{x,y\in\{0,1\}^*} \equiv^c \{VIEW^{\Pi}_{client}(x,y)\}_{x,y\in\{0,1\}^*}$$

where $VIEW_{client}$ is the client's view of our protocol $\Pi$'s execution including it's input, randomness and messages received, $f_1(x,y)$ is the client output, and $x, y$ are the client and server inputs respectively.

The simulator is given the client inputs and output. It selects uniform random $b$ such that $b < m$ ($m$ prime) and follows P-SWAHE encrypt steps. It creates the server input and noise elements as described in Algorithm 5. The server plaintext bits are random.

The simulator processes the plaintext client input and random server inputs through the circuit and obtains the simulated client output.

[5] Our protocol does not ensure correctness when the server is malicious.

If the simulated output is the same as the actual client output then for the return of the output to the client, as in the real protocol, the output is summed with an even parity hidden subset sum. Otherwise, the output is summed with an odd parity hidden subset sum. Thus we ensure that the client obtains the same output in both the simulation and the real protocol.

The DPT-HSSP assumption shows that at each of the multiplication and output gates the computationally bounded client cannot distinguish between the two probability distributions. Thus the client's view in the real protocol and the simulated protocol are indistinguishable, with high probability.

We now prove that our 2PC protocol has client privacy; we will show that

$$\{VIEW^{\Pi}_{srv}(x,y)\}_{x,y\in\{0,1\}^*} \equiv^{perf} \{VIEW^{\Pi}_{srv}(x',y)\}_{x',y\in\{0,1\}^*}$$

where $VIEW_{srv}$ is the server's view of our protocol $\Pi's$ execution including it's input, randomness, and messages received, and $|x| = |x'| = |y|$ where $x$ and $x'$ are client inputs.

Theorem 4 gives that

$$Pr[Enc_{b_1}(\boldsymbol{r_1}) = \boldsymbol{w}] = Pr[Enc_{b_2}(\boldsymbol{r_2}) = \boldsymbol{w}]$$

for keys $b_1, b_2$ uniformly random, plaintexts $\boldsymbol{r_1}, \boldsymbol{r_2}$ and ciphertext $\boldsymbol{w}$. We have $x$ and $x'$ as a subset of bits of $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$. The server input bits and noise generator bits generate components of $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$. The 2PC protocol messages from the client to the server at the multiplication gates are references to multiplication gate outputs. Consider the reference $i_1, i_2, i_3, i_4$ at an arbitrary multiplication gate where $i_j \in \{1,2,3,4\}$, $1 \le j \le 4$. We show that

$$\begin{aligned} &Pr[\text{reference at gate g} = i_1, i_2, i_3, i_4 \text{ given client input } x] = \\ &\quad Pr[\text{reference at gate g} = i_1, i_2, i_3, i_4 \text{ given client input } x'] \end{aligned} \tag{6}$$

$Pr[i_l = j] = 1/4 \cdot 1/4 + 1/3 \cdot (3/4)^2 = 1/4$, $j, l \in \{1,2,3,4\}$. Our argument depends only on the probability that each of the 4 products is 1 is 1/4 and the distribution of the 3 zero bits and 1 one bit is uniform random. Thus (6) holds. Thus we have

$$\{VIEW^{\Pi}_{srv}(x,y)\}_{x,y\in\{0,1\}^*} \equiv^{perf} \{VIEW^{\Pi}_{srv}(x',y)\}_{x',y\in\{0,1\}^*}$$

Finally we examine the intial setup. The client's view in the real protocol and the simulation is of CLPKE ciphertexts: $E_{pk}(e_s)$ and $E_{pk}(e'_s)$. Thus server privacy is preserved due to the CPA security property. The server's view is identical since there exist $b'$ and $e'$ such that $b'e' = be \bmod m$ where $e'$ is the first coordinate of a permutation table vector that has the server's selection of server and noise inputs along with an arbitrary selection for client inputs. ∎

We now prove security in the semi-honest model where the client may share the output with the server.

**Theorem 11.** *Given the 2-party protocol of Algorithms 5 and 7 where both the client and server are semi-honest and the client may share the output with the server. The server may be computationally unbounded. Then the protocol has server privacy given the hardness of DPT-HSSP and DDH and perfect client privacy.*

*Proof.* The proof for server privacy is the same as the proof of server privacy in Theorem 10.

In order to prove client privacy we will show a probabilistic polynomial-time algorithm $\mathcal{S}_2$ such that

$$\{\mathcal{S}_2(y, f_2(x,y))\}_{x,y\in\{0,1\}^*} \equiv^{perf} \{VIEW^{\Pi}_{srv}(x,y)\}_{x,y\in\{0,1\}^*}$$

where $VIEW_{srv}$ is the server's view of our protocol $\Pi$'s execution including it's input, randomness and messages received, $f_2(x,y)$ is the server output, and $x, y$ are the client and server inputs respectively.

$\mathcal{S}_2$ selects uniform random $b < m$ ($m$ prime) and follows the steps in Algorithm 5. The client plaintext and multiplication bits are random. The difference is the client input and multiplication bits; however, Theorem 4 gives

$$Pr[Enc_{b_1}(\boldsymbol{r_1}) = \boldsymbol{w}] = Pr[Enc_{b_2}(\boldsymbol{r_2}) = \boldsymbol{w}]$$

for keys $b_1, b_2$ uniformly random, plaintexts $\boldsymbol{r_1}, \boldsymbol{r_2}$ and ciphertext $\boldsymbol{w}$. In the real protocol, the client messages to the server are references to multiplication outputs in the ciphertext vector $\boldsymbol{w}$. The same is true for $\mathcal{S}_2$ and the same argument from Theorem 10 shows that the messages occur with the same probability. Finally, $\mathcal{S}_2$ has the same server output. Thus the server's view is replicated with the same probability by $\mathcal{S}_2$ and the proof is complete. ∎

**Proof of Client's Statistical Security Against Fully Malicious Server**
In this section we view the group $G$ as a cyclic subgroup of a larger group $G_1$ following [10]. The benefit of the larger group is that the client can efficiently verify that the pair of received elements are in $G_1$. Then we can run a slightly modified version of the Okamoto Sigma protocol [35], as presented in [6], to prove in zero knowledge that the received ciphertext pair consists of elements in the cyclic subgroup $G$. We then establish statistical security in Theorem 12.

The modified protocol is a Sigma protocol for the relation

$$\mathcal{R} = \{((\alpha,\beta),u) \in \{1,\ldots,2^{\lambda-2}s_3\} \times Z_q \times G : g^{\alpha} f^{\beta} = u\}$$

**Definition 6.** *[10] The Gen algorithm takes inputs parameter $\lambda$ and prime integer $q$ and outputs $(s_3, g, f, g_q, G_1, G, F, G^q)$. $G_1$ is an abelian group of order $q \cdot s_2$ where $s_2$ bitsize is a function of $\lambda$ and $\gcd(q, s_2) = 1$. Gen generates an upper bound $s_3$ for $s_2$. ($s_3 2^{\lambda-2}$ replaces $Bq - 1$ in Figure 4.) Valid encodings of elements of $G_1$ can be efficiently recognized. $F$ is the unique cyclic subgroup of $G_1$ of order $q$ generated by $f$. The set $G$ is a cyclic subgroup of $G_1$ of order $q \cdot s$*

*where $s$ divides $s_2$. We have $F \subset G$ and $G^q$ is the subgroup of order $s$ in $G$ where $G = G^q \times F$. $g_q$ and $g = f \cdot g_q$ are respective generators for $G^q$ and $G$ respectively. The discrete log problem is easy in $F$ so that Solve is a deterministic polytime algorithm for solving discrete log in $F$ :*

$$Pr[x = x^* : (s_3, g, f, g_q, G_1, G, F, G^q) \leftarrow Gen(1^\lambda, q), x \leftarrow Z_q, X \leftarrow f^x, \\ x^* \leftarrow Solve(q, s_3, g, f, g_q, G_1, G, F, G^q, X)] = 1.$$

The main modification to the Okamoto protocol is that the server (the prover) sends $u = g^\alpha f^\beta$ to the client (the verifier) instead of the verifier being given $u$ as input. To compensate for this change we need to run the protocol $z$ times to achieve soundness $2^{-z}$ (but $u$ is fixed across all the runs).

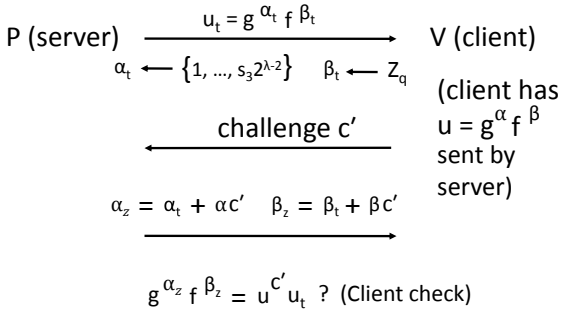



**Fig. 10.** Modified Okamoto Protocol

**Lemma 8.** *The modified Okamoto protocol is zero knowledge and has statistical soundness $2^{-z}$ when run $z$ times sequentially.*

*Proof.* Correctness and zero knowledge are clear. We now prove soundness. Given the check condition $g^{\alpha_z} f^{\beta_z} = u^{c'} \cdot u_t$ where $u$ is initially sent by the server to the client. Since $u, u_t$ are committed to and uniform random $c'$ is sent before this check condition the check holds probabilistically. Note that $g^{\alpha_z} f^{\beta_z} \in G$. Suppose server is honest so $u \in G$. Then each of the $z$ client checks succeeds. Suppose $u \notin G$. If $u^{i+1} u_t \in G$, then $u^i u_t \notin G$, else $u^{i+1} u_t (u^i u_t)^{-1} = u \in G$, Thus the check condition holds with probability at most $1/2$ if $u \notin G$. Thus server probability of success when $u \notin G$ is at most $1/2^z$. (Not even a computationally unbounded adversary can force equality between an element in the subgroup $G$ and an element outside $G$.)

The following theorem leverages the statistical privacy definition from Section 2.3. In the theorem, we have kept the CL notation for group $G$ where $|G| = n = sq$. For the 2PC protocol, $m = q$.

**Theorem 12.** ***(Statistical Privacy Against Fully Malicious Server)*** *Given the 2-party protocol of Algorithms 5, 6, and 7 where the client is semi-honest and does not share the output with the server. In the setup protocol we run the modified Okamoto protocol of Lemma 8 upon receipt of the server ciphertext at the client. The server is computationally unbounded and in the malicious model. Let $m_c$ be the client message sent to the server in the 2PC setup protocol; $m_c$ consists of the client inputs and the multiplication bits. Let $\gamma = Pr[\bar{X} = \text{client inputs for } m_c]$ where $\bar{X}$ is the random variable for the probability of a client input sequence.*

*We assume the client has the properly formatted CL encryption public key of the server.*

*Then the protocol has server privacy given hardness of the DPT-HSSP and DDH and statistical client privacy with $\delta(g) = 2^{-2g}\gamma$ and $\epsilon(g) = \frac{\beta(1-\beta)}{2^{2g}-\beta}$ where $c_c$ is the number of client inputs, $g$ is the number of multiplication gates, and $\beta = Pr[\bar{X} = \text{client input for } m'_c]$. $m'_c$ is the server's guess at a possible client message sent during the setup protocol and satisfies $u_2 + m'_c is - u_1 x = 0 \bmod n$ where $u_1$ and $u_2$ are server controlled parameters, $x$ is the server CL encryption private key, $i$ is an integer, and the CL encryption group has order $n = sq$.*

*Proof.* If $m'_c = m_c$ then this occurs with probability bounded above by $2^{-2g}\gamma$ and accounts for the $\delta(g) = 2^{-2g}\gamma$ term. We now assume $m_c \neq m'_c$.

We show statistical security for a malicious server; by the argument in Theorem 10 the server deviates during the 2PC setup protocol. The client receives $(g^{u_1}, g^{u_2})$ from the server where $g$ generates the group $G$. We can assume the ciphertext pair are elements in $G$ by Lemma 8 since we run the modified Okamoto protocol with soundness error $2^{-z}$. (If one of the protocol checks fails then the client terminates the protocol since the server has been caught trying to cheat. Also, we must repeat the runs for the 2nd component of the ciphertext.) The client encrypts $m_c = e_{cli} - 2$ to obtain $(g^{r_1}, f^{m_c} h^{r_1})$ where $r_1 \leftarrow \{0, \ldots, s_3 2^{\lambda-2}\}, h = g^x, sk = x$ (see Figure 4 for encryption details.) The client then uses the CL encryption EvalSum algorithm to obtain $(g^{u_1+r_1+r_2}, g^{u_2} f^{m_c} h^{r_1+r_2})$ where $r_2 \leftarrow \{0, \ldots, s_3 2^{\lambda-2}\}$.

The client then uses the CL encryption EvalScal algorithm to obtain

$$(g^{b(u_1+r_1+r_2)+r_3}, g^{bu_2} f^{bm_c} h^{b(r_1+r_2)+r_3})$$

where $r_3 \leftarrow \{0, \ldots, s_3 2^{\lambda-2}\}$. We have $f = g^{is}$ for some $i$, $1 \leq i \leq q-1$. Thus we can write the last pair of elements as

$$(g^{b(u_1+r_1+r_2)+r_3}, g^{bu_2+bm_c is+bx(r_1+r_2)+xr_3})$$

which is sent from the client to the server.

Server decryption yields $g^{b(u_2+m_c is-u_1 x)}$. If server computes the discrete log (server is computationally unbounded), then server has

$$b(u_2 + m_c is - u_1 x) \bmod n.$$

Since $m_c \neq m'_c$ and $u_2 + m'_c is - u_1 x = 0 \bmod n$ the server learns that the client message is not $m'_c$. Thus the conditional distribution and the original distribution on client messages are not the same; we calculate the statistical distance (SD), representing the information leakage, between the two distributions:

$$SD = \frac{1}{2} \sum_{\bar{m_c} \text{ has } m'_c \text{ client input}} |Pr[X = \bar{m_c}] - Pr[X = \bar{m_c} | \text{client msg} \neq m'_c]|$$

$$+ \frac{1}{2} \sum_{\bar{m_c} \text{ doesn't have } m'_c \text{ client input}} |Pr[X = \bar{m_c}] - Pr[X = \bar{m_c} | \text{client msg} \neq m'_c]|$$

$$= \frac{1}{2} \left[ \beta - \frac{2^{2g+c_c}\beta - 2^{c_c}\beta}{2^{2g+c_c} - 2^{c_c}\beta} + \frac{2^{2g+c_c} - 2^{2g+c_c}\beta}{2^{2g+c_c} - 2^{c_c}\beta} - (1-\beta) \right]$$

$$= \frac{1}{2} \left[ \frac{\beta(2^{2g+c_c} - 2^{c_c}\beta)}{2^{2g+c_c} - 2^{c_c}\beta} - \frac{\beta 2^{2g+c_c} - 2^{c_c}\beta}{2^{2g+c_c} - 2^{c_c}\beta} + \right.$$

$$\left. \frac{2^{2g+c_c} - 2^{2g+c_c}\beta}{2^{2g+c_c} - 2^{c_c}\beta} - \frac{(1-\beta)(2^{2g+c_c} - 2^{c_c}\beta)}{2^{2g+c_c} - 2^{c_c}\beta} \right] = \frac{1}{2} \left[ \frac{2^{c_c}\beta - 2^{c_c}\beta^2 + 2^{c_c}\beta(1-\beta)}{2^{2g+c_c} - 2^{c_c}\beta} \right]$$

$$= \frac{1}{2} \left[ \frac{2^{c_c}\beta(1-\beta) + 2^{c_c}\beta(1-\beta)}{2^{2g+c_c} - 2^{c_c}\beta} \right] = \frac{\beta(1-\beta)}{2^{2g} - \beta}$$

$X$ is the random variable for the setup protocol client message to the server. We see that $\epsilon(g)$ approaches zero as $g \to \infty$. (Recall Definition 2.)

Given client message $\hat{m_c}$ where $\hat{m_c} \neq m'_c$. Thus $u_2 + \hat{m_c} is - u_1 x \neq 0 \bmod n$. We can assume both $m'_c$ and $\hat{m_c}$ are less than $q$ so $u_2 + \hat{m_c} is - u_1 x \neq 0 \bmod q$. Thus $u_2 + \hat{m_c} is - u_1 x$ is invertible modulo $q$. The system of congruences $x = b \bmod s$ and $x = (u_2 + \hat{m_c} is - u_1 x)^{-1} b(u_2 + m_c is - u_1 x) \bmod q$ has a unique solution $b' \bmod n$ by the Chinese Remainder Theorem ($s$ and $q$ are coprime.) Thus $b'(u_2 + \hat{m_c} is - u_1 x) = b(u_2 + m_c is - u_1 x) \bmod s$ and $b'(u_2 + \hat{m_c} is - u_1 x) = b(u_2 + m_c is - u_1 x) \bmod q$. Thus $b'(u_2 + \hat{m_c} is - u_1 x) = b(u_2 + m_c is - u_1 x) \bmod n$.

## 5 Scaling the 2PC Scheme

In the 2PC scheme, there are $4g$ additional elements of the form $be \bmod m$, where $g$ is the number of multiplication gates in the circuit. We recall $c = c_c + c_s + k + 4g$, $c_1 = c_c + 4g$. The communication complexity for the initial setup is bounded by $2\log(m) + (c_1 - 1)(\log((a+1)N)) + 2\log(\text{CLPKE ciphertext bound})$. The 2nd term for initial setup communication complexity uses $d_{i+1}/d_i \approx N(a+1)$ for $1 \leq i \leq c_1 - 1$. Only the factors $d_{i+1}/d_i$ need to be sent by client to send $d_2, \ldots, d_{c_1}$ to the server. The complexity for the protocol circuit evaluation is $2g\log(m) + 6g + \log(m)$ where the first two terms are for the multiplication gates and the last term is for the output. Thus total communication complexity is

$$(2g+3)\log(m) + 6g + 2\log(\text{CLPKE ciphertext bound}) + (c_1 - 1)^2(\log((a+1)N)$$

We have $\log(m) > (c_c + c_s + 4g + k)\log(N(a+1))$ (see Remark 3). The dominant term in the communication complexity is roughly $8g^2\log(N(a+1))$. If we

partition the circuit into j subcircuits then we exchange $g^2$ for $\sum_{j\ times}(g/j)^2 = g^2/j$. Thus we can improve the communication complexity by partitioning the circuit into subcircuits. This partitioning has to be balanced against the per subcircuit cost of running the setup protocol.

In this section, $N = 2$.

The circuit is partitioned into subcircuits so that each subcircuit has only a limited number of multiplication gates. The outputs of these gates can be viewed as subcircuit outputs. The client can take the subcircuit outputs that are not final circuit outputs and submit them as inputs into one of the other subcircuits, where the inputs are re-encrypted under a new secret key and possibly a new modulus.

Figure 11 gives an example of dividing a circuit into two subcircuits $C_1$ and $C_2$. The input gates are circles with an $i$ character inside. The inputs to $C_1$ are $A, B, C, D, E$, and $F$. The inputs to $C_2$ are $G, H, I, J, K$, and $L$. Note that $G$ and $H$ are outputs of multiplication gates in $C_1$, so they are re-encrypted with the new secret key for subcircuit $C_2$.

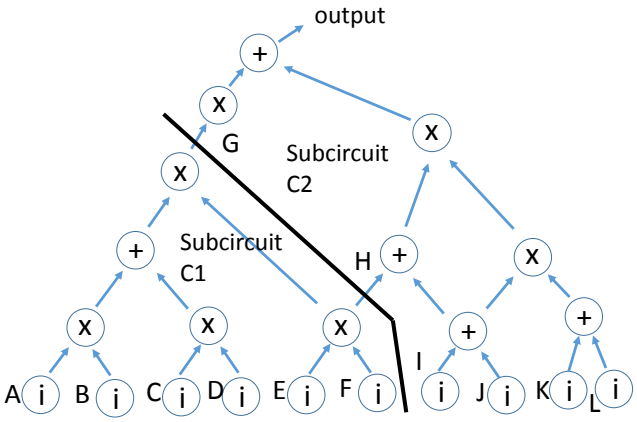



**Fig. 11.** An Example of Scaling the 2PC Protocol with Subcircuits

We now describe an extended 2PC scheme for a circuit which has been partitioned into subcircuits. All of the subcircuits will use the same permutation table and modulus $m$. Each subcircuit has its own secret key. The idea is to run the intial setup for each subcircuit; these setup exchanges can be performed as part of preprocessing (even though not all of the inputs are known) or in parallel with the subcircuit evaluation exchanges. When transitioning between subcircuits, the following subprotocols can be used:

**1 down 1 up exchange protocol:** The server sends an element that should be decrypted and replaced with an element encrypted with the secret key of the new subcircuit. The server adds a hidden subset sum that is selected to have

odd parity with 1/2 probability. The client decrypts and sends the index of the new element (index in the ciphertext vector row). The server adds an odd parity element to the returned element from the client if the server added an odd parity subset sum to the request element.

**2 down 3 up exchange protocol:** As part of a multiplication query, the server sends down an element from each multiplication pair (see Algorithm 7.) The client can then compute the parities of the remaining 2 elements. These will be encrypted with the new subcircuit key if transitioning to a new subcircuit, else the existing secret key will be used. The client only has to send the indices of the product elements in the permutation table row. Only 3 indices are needed for the first 3 products and the server can pick the last element from the row for the fourth product. (Each block of 4 multiplication products is contiguous in the permutation table).

**Acknowledgements:** We thank Ivan Damgard for suggesting CLPKE encryption [11].

## References

1. Acharya, A., Hazay, C., Poburinnaya, O., Venkitasubramaniam, M.: Best of both worlds: Revisiting the spymasters double agent problem. In: Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I. pp. 328–359. Springer-Verlag, Berlin, Heidelberg (2023)
2. Alon, B., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A., Patra, A.: On perfectly secure two-party computation for symmetric functionalities with correlated randomness. In: Theory of Cryptography: 20th International Conference, TCC 2022. Springer, Berlin, Heidelberg (2022), https://doi.org/10.1007/978-3-031-22365-5_19
3. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. Cryptology ePrint Archive, Paper 2014/094 (2014), https://eprint.iacr.org/2014/094, https://eprint.iacr.org/2014/094
4. Badrinarayanan, S., Patranabis, S., Sarkar, P.: Statistical security in two-party computation revisited. In: Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part II. pp. 181–210. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22365-5_7, https://doi.org/10.1007/978-3-031-22365-5_7
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 1–10. STOC '88, Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/62212.62213, https://doi.org/10.1145/62212.62213
6. Boneh, D., Shoup, V.: A graduate course in appllied cryptography. Version 0.6, Jan. 2023 (2023)
7. Bouvier, C., Castagnos, G., Imbert, L., Laguillaumie, F.: I want to ride my bicycl : Bicycl implements cryptography in class groups. Journal of Cryptology **36**(3) (2023)

8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Paper 2011/277 (2011), https://eprint.iacr.org/2011/277, https://eprint.iacr.org/2011/277
9. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. In: 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. pp. 97–106 (2011). https://doi.org/10.1109/FOCS.2011.12
10. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Two-party ecdsa from hash proof systems and efficient instantiations. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III LNCS vol. 11694. pp. 191–221. Springer, Berlin, Heidelberg (2019)
11. Castagnos, G., Laguillaumie, F.: Linearly homomorphic encryption from DDH. In: Topics in Cryptology – CT-RSA 2015. vol. 9048. Springer, New York, NY, USA (2015). https://doi.org/10.1145/2554797.2554799, https://doi.org/10.1145/2554797.2554799
12. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 11–19. STOC '88, Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/62212.62214, https://doi.org/10.1145/62212.62214
13. Chor, B., Kushilevitz, E.: A zero-one law for boolean privacy (extended abstract). In: Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing. pp. 62–72. STOC '89, Association for Computing Machinery, New York, NY, USA (1989)
14. Coron, J., Gini, A.: A polynomial-time algorithm for solving the hidden subset sum problem. In: Advances in Cryptology – CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II. Springer-Verlag, Berlin, Heidelberg (2020)
15. Coron, J., Gini, A.: Provably solving the hidden subset sum problem via statistical learning. Mathematical Cryptology **1**(2), 70–84 (mar 2022)
16. Cramer, R., Damgard, I.B., Nielsen, J.B.: Secure multiparty computation and secret sharing. Cambridge University Press (2015)
17. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. pp. 643–662. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
18. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Paper 2009/616 (2009), https://eprint.iacr.org/2009/616, https://eprint.iacr.org/2009/616
19. Ducas, L., Micciancio, D.: Fhew: Bootstrapping homomorphic encryption in less than a second. Cryptology ePrint Archive, Paper 2014/816 (2014), https://eprint.iacr.org/2014/816, https://eprint.iacr.org/2014/816
20. Faulhaber, J.: Academia algebrae, darinnen die miraculosische inventiones zu den hochsten cossen weiters continuirt und profitiert werden. Augspurg [sic], Germany: Johann Ulrich Schonigs (1631)
21. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. pp. 169–178. Association for Computing Machinery, New York, NY, USA (2009), https://doi.org/10.1145/1536414.1536440
22. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. Cryptology ePrint Archive, Paper 2010/520 (2010), https://eprint.iacr.org/2010/520, https://eprint.iacr.org/2010/520

23. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. Cryptology ePrint Archive, Paper 2011/566 (2011), https://eprint.iacr.org/2011/566, https://eprint.iacr.org/2011/566
24. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. pp. 218–229. STOC '87, Association for Computing Machinery, New York, NY, USA (1987). https://doi.org/10.1145/28395.28420, https://doi.org/10.1145/28395.28420
25. Halevi, S.: Homomorphic encryption. Tutorials on the Foundations of Cryptography, Dedicated to Oded Goldreich (2017)
26. Hazay, C., Lindell, Y.: Efficient secure two-party protocols, techniques and constructions. Springer-Verlag (2010)
27. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Advances in Cryptology – CRYPTO 2010. Springe-Verlag Berlin Heidelberg, Berlin, Germany (2010)
28. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology **1996**(9), 199–216 (1996)
29. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Theory of Cryptography: 11th International Conference, TCC 2013. pp. 600–620. Springer-Verlag, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_34
30. Khurana, D., Mughees, M.H.: On statistical security in two-party computation. In: Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part II. pp. 532–561. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-64378-2_19, https://doi.org/10.1007/978-3-030-64378-2_19
31. Kolesnikov, V.: personal communication
32. Kushilevitz, E.: Privacy and communication complexity. In: 30th Annual Symposium on Foundations of Computer Science. pp. 416–421. FOCS '89, IEEE Computer Society (1989)
33. Lagarias, J.C., Odlyzko, A.M.: Solving low-density subset sum problems. Journal of the ACM **32**(1), 229–246 (1985)
34. Nguyen, P.Q., Stern, J.: The hardness of the hidden subset sum problem and its cryptographic implications. In: Advances in Cryptology – CRYPTO 1999: 19th Annual International Cryptology Conference, CRYPTO 1999, Santa Barbara, CA, USA, August 15–19, 1999, Proceedings. pp. 31–46. Springer-Verlag, Berlin, Heidelberg (1999)
35. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Advances in Cryptology – CRYPTO 1992. Springer, Berlin, Heidelberg (1992)
36. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques. pp. 223–238. EUROCRYPT'99, Springer-Verlag, Berlin, Heidelberg (1999)
37. Rivest, R.L., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation. pp. 169–180 (1978)
38. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (feb 1978). https://doi.org/10.1145/359340.359342, https://doi.org/10.1145/359340.359342

39. Rothblum, R.: Homomorphic encryption: from private-key to public-key. In: Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011. pp. 219–234 (2011)
40. Yao, A.C.C.: How to generate and exchange secrets. In: Proceedings of the 27th Annual Symposium on Foundations of Computer Science. pp. 162–167. SFCS '86, IEEE Computer Society, USA (1986). https://doi.org/10.1109/SFCS.1986.25, https://doi.org/10.1109/SFCS.1986.25

## A Definition of Somewhat Additive Homomorphic Encryption

We follow [25] for these definitions.

**Definition 7.** *A homomorphic encryption scheme $\mathcal{E}$ consists of the following four algorithms:*

- $(sk, pk) \leftarrow KeyGen(1^\lambda, 1^\tau)$ *The inputs are the security parameter $\lambda$ and a functionality parameter $\tau$, and the output is the keypair.*
- $\boldsymbol{c} \leftarrow Encrypt(pk, \boldsymbol{p})$ *Given the public key and plaintext vector output the ciphertext vector.*
- $\boldsymbol{p} \leftarrow Decrypt(sk, \boldsymbol{c})$ *Given the secret key and ciphertext vector output the plaintext vector.*
- $\boldsymbol{c}' \leftarrow Evaluate(pk, C, \boldsymbol{c})$ *Given the public key, a circuit $C$, and vector of ciphertexts with one ciphertext for each circuit input output a new ciphertext vector with one component for each of the outputs of $C$.*

**Definition 8.** *(Correctness)*

- *For any plaintext vector $\boldsymbol{p}$,*

$$Pr[Decrypt(sk, \boldsymbol{c}) = \boldsymbol{p} : (sk, pk) \leftarrow KeyGen(1^\lambda, 1^\tau), \boldsymbol{c} \leftarrow Encrypt(pk, \boldsymbol{p})] = 1;$$

-

$$\begin{aligned} Pr[Decrypt(sk, \boldsymbol{c}') = C(\boldsymbol{p}) : (sk, pk) \leftarrow KeyGen(1^\lambda, 1^\tau), \\ \boldsymbol{c} \leftarrow Encrypt(pk, \boldsymbol{p}), \boldsymbol{c}' \leftarrow Evaluate(pk, C, \boldsymbol{c})] = 1. \end{aligned}$$

A homomorphic encryption scheme $\mathcal{E}$ is additively homomorphic if it is correct for all Boolean (arithmetic) circuits consisting only of XOR (addition) gates. It is *almost additively homomorphic* or *somewhat additively homomorphic* if it is correct for a family of circuits $\mathcal{C}$ containing sums with at most $2^\tau$ variables.

## B Basic SWAHE Scheme

In this section we present a somewhat additive homomorphic encryption scheme: the basic SWAHE scheme. This scheme is secret key rather than public key. We then show that it is not possible to obtain perfect security with this scheme.

### B.1 Additive Homomorphic Scheme

**Definition 9.** ***Basic Somewhat Homomorphic Additive Encryption (Basic SWAHE)*** *Given a server circuit with a addition gates.*

***Key Generation:*** *Select positive prime integer $m$ and uniform random positive integer $b$, the base, where $b < m$. The secret key is $(b, m)$. The key is used to encrypt only one plaintext vector. A new key must be selected for each new plaintext vector.*

***Encryption:*** *$N$ is a positive integer which is a power of 2. Given plaintext vector $(r_1, \ldots, r_c)$ where $0 \leq r_i < N$, $1 \leq i \leq c$. The $r_i$ are integers; for the rest of this paper we will consider $N = 2$ and the $r_i$ as bits. For all $i$, $e_i = a_i N + r_i$ where $a_i$ is random uniform, and $e_i \leq \frac{m}{a+1}$, $1 \leq i \leq c$.*

*The client ciphertext vector is obtained by selecting an integer representative $v_i$, $1 \leq i \leq c$, where $v_i \equiv be_i \bmod m$. Then $Q(I) = (v_1, \ldots, v_c)$ is the client ciphertext vector.*

***Decryption:*** *The client decryptor receives an evaluated ciphertext $x$ from the server which consists of at most $a$ additions of the ciphertexts in the ciphertext vector. Thus the client may obtain*

$$b^{-1}x \bmod m = e$$

*where $e$ is the sum of at most $a+1$ of the $e_i's$. Since $e_i \leq m/(a+1)$ for $1 \leq i \leq c$, it follows that $e$ is the integer sum of the $e_i's$. In other words,*

$$e \leq (a+1) \max e_i \leq \frac{(a+1)m}{(a+1)} = m$$

*so the residue modulo $m$ is the number itself.*

*Thus the parity of the integer $e$ is the modulo 2 sum of the $r_i's$.*

***Evaluate:*** *Evaluate takes the ciphertext tuple from Encrypt and the circuit and outputs another vector of ciphertexts corresponding to the circuit evaluation. Each XOR gate is processed by adding the two inputs as integers.*

The client sends the client ciphertext request to the server which uses the input elements as inputs to the circuit. The server returns the output elements to the client.

We will not prove any security properties for the basic scheme; we will take the basic scheme and add conditions to obtain the scheme in Theorem 1 and Definition 5 for which we will prove security properties.

We now give an example to show there are some permutation tables that have limited entropy as $m$ grows for a fixed $c$.

**Proposition 2.** *Given any $m \geq 7$ where $m - 1$ is divisible by 5 in the basic SWAHE scheme described above. Given a circuit with a addition gates. Given*

*the permutation table $T$ that has 1, 4, and 5 in the same row. Then the set of associated binary vectors for $T$ is missing at least $2^{c-2}$ of the possible $2^c$ binary vectors.*

*Proof.* We first consider the c = 3 case. Consider the order for $T$ where the row with 1, 4, and 5 is the start row ($T$ has the same vectors regardless of which vector is the start vector in the table.) Before the 5 column 1st flip, we have 2 associated binary strings. We may gain at most 2 more distinct binary strings after the 5 flip, before the 4 flip. Then we can pick up 2 more binary strings after the 4 flip prior to the next 5 flip, for a total of 6. When we come to the 2nd 5 flip, we are at row $2(m-1)/5+1$. $2(m-1)/5+1 > 3/8(m-1)+1 = (m-1)/4+(m-1)/8+1$. Thus we are more than half way through the 2nd permutation run for the 4 column and the elements are too large to be less than $\alpha = (m-1)/(a+1)$. After the next 4 flip, we are at row $(m-1)/2+1 > \alpha$. Thus no more associated binary vectors can occur. The general $c$ case follows except that the number of missing binary vectors is at least $2^{c-2}$. ∎

The upshot of the proposition is that it is possible in the basic SWAHE scheme to have less than perfect security.