Perfect 2-Party Computation from Additive Somewhat Homomorphic Encryption

Jonathan Trostle

Independent Consultant

Abstract. Two-party computation has been an active area of research since Yao's breakthrough results on garbled circuits. We present secret key somewhat additive homomorphic schemes where the client has perfect privacy (server can be computationally unbounded). Our basic scheme is somewhat additive homomorphic and we extend it to support multiplication. The server handles circuit multiplication gates by returning the multiplicands to the client which updates the decryption key so that the original ciphertext vector includes the encrypted multiplication gate outputs. We give a 2-party protocol that also incorporates server inputs where the client has perfect privacy. Server privacy holds against a computationally bounded adversary since it depends on the hardness of the subset sum problem. Correctness of the server in the malicious model can be verified by a 3rd party where the client and server privacy are protected from the verifier.

Keywords: Somewhat additive homomorphic encryption · perfect security.

1 Introduction

Two-party computation protocols include garbled circuit based protocols which is a technique first presented by Yao [30]. Goldreich Micali Wigderson [17], [5] apply to n parties. These protocols allow for the secure computation of arbitrary functions keeping the inputs of each party private except for any leakage associated with the function output.

Homomorphic encryption schemes with respect to a single operation (addition or multiplication) include Goldwasser-Micali, Paillier [25],and textbook RSA [27]. Rivest et al. posed the question of homomorphic encryption [26] in 1978. Gentry's breakthrough work [14] 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 [12], [6], [7], [2], [13] 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 homomorphic additive encryption schemes (any circuit can be handled by setting scheme parameters to be large enough for the circuit). Our schemes are secret key based and provide security against a computationally unbounded attacker. These schemes are the basis for our 2-party computation (2PC) protocols that provide perfect security for the client.

Our basic scheme includes a modulus m, base b where m is prime and random exponents e_i corresponding to each client ciphertext 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 security. Our main results for somewhat homomorphic additive encryption are Theorem 1 and Theorem 4 which prove that our Perfect Somewhat Homomorphic Additive Encryption (P-SWHAE) scheme has perfect security.

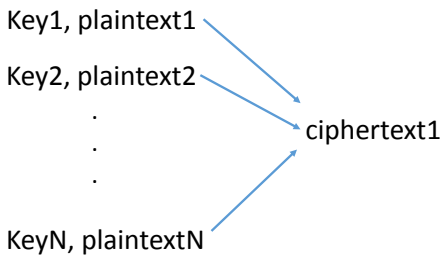

Fig. 1. Distinct Plaintext Key Pairs Map to Same Ciphertext

We extend P-SWHAE to handle multiplication gates (Section 2.3). In P-SWHAE with multiplication, the server returns the multiplicands to the client 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-SWHAE with multiplication protocol provides perfect security 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.

We give a 2-party MPC (2PC) scheme where client privacy is perfect and server privacy is based on the hardness of the modular subset sum problem. Initially, in a preprocessing step, the client sends two additional sets of elements to the server; one set has odd parity exponent elements and the other set has even parity exponent elements. When the server has two multiplicands to return to the client, it actually returns four elements where the additional two elements have opposite parity exponents which are obtained by adding odd parity subset sums to the original multiplicands. Then the client will compute all four products and return four new ciphertext elements to the server. The server discards the three incorrect elements and keeps the fourth element. Thus the server evaluates the circuit obliviously except for knowledge of addition gates, multiplication gates, and its own inputs. Our security argument for server privacy requires that the client is semi-honest.

There exist 2-party schemes [22], [3] where one party enjoys statistical privacy and the other party is protected from a computationally bounded adversary. Our 2-party scheme provides perfect privacy for one party and computational security for the other party. The schemes in [22] are in the malicious security model where our 2PC protocol client is assumed to be semi-honest.

In our protocol, receiver (client) privacy is protected unconditionally including if the sender (server) is malicious provided the client does not share the output with the server. Sender correctness can be established by a verifier. The client and server privacy is protected unconditionally from the verifier, provided that the verifier does not have access to the client secret key, client plaintext inputs, or server plaintext inputs.

1.1 An Example

We now give an example to motivate and briefly illustrate the ideas in our work.

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 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 1) 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 $is_j \bmod m$ where $s_1 = 1$, $s_2 = 4$, and $s_3 = 16$; we take representatives to be integers between 1 and 96. There are other possible choices for the s_i (the column start elements) but one property is that s_i/s_{i+1} which results in the table having some short vectors (components sum to less

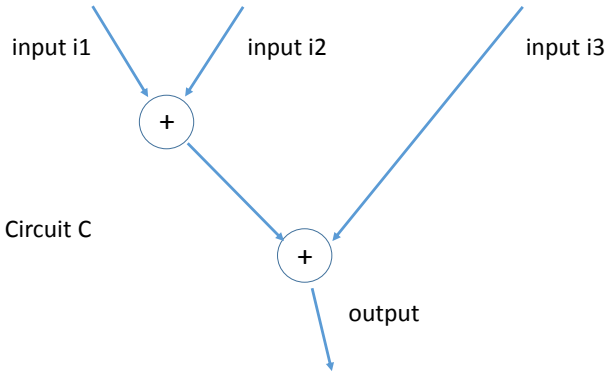

Fig. 2. Circuit C with Two Addition Gates

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 2). 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 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.

Table 1. Permutation Table T for $m = 97$, $c = 3$, $s_1 = 1$, $s_2 = 4$, $s_3 = 16$; 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

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.

1.2 Our Contributions

We have the following results:

1. We give a secret key additive homomorphic encryption algorithm (P-SWHAE) that provides perfect client privacy, and extend this to include homomorphic multiplication. The resulting protocol protects the client privacy from a computationally unbounded server. To the best of our knowledge, our additive somewhat homomorphic encryption is the first such algorithm that provides perfect privacy.
2. We show that our P-SWHAE scheme has perfect security in Theorem 1, Theorem 4, and Theorem 6.

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. We show that this protocol provides perfect client privacy (Theorems 7 and 8.) Client privacy is protected from a malicious server if the server does not receive output (Theorem 7).
4. We show the 2PC protocol protects server privacy assuming hardness of the subset sum assumption given a semi-honest client.
5. The 2PC protocol malicious server's correctness can be verified in the protocol by a verifier entity where the verifier has access to the protocol transcript, the server computations, and the circuit specification. The client and server privacy is information-theoretic secure from the verifier provided the verifier does not have access to the client secret key, client plaintext inputs, or server plaintext inputs.

1.3 Related Work

There has been extensive work in homomorphic encryption since the breakthrough work of Gentry [14] (e.g., [15], [16], [12], [6], [7], [2], [13]. Our schemes provide perfect privacy for the client.

Foundational work in multiparty computation includes [30], [17], [5], [9].

It is not possible to protect both parties with statistical or perfect security in a 2-party secure computation protocol.

Dakshita and Mughees [22] give 2-party secure protocols where one party is statistically secure and the other party is computationally secure. Their protocols use garbled circuits [30] in combination with Oblivious Transfer to obtain a 5 round protocol. Their protocols are secure against a malicious adversary whereas we assume our client, or receiving party, is in the semi honest model (but our receiving party's privacy is protected even if the sender is malicious as long as only the receiving party gets output).

Badrinarayanan [3] allows the results of [22] to be based on additional assumptions such as CDH. Acharya [1] generalizes [22] 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.

Koleskinov [24] gives a 2-party secure protocol (GESS) where the secrets are assigned to wires. Their protocol can be viewed as a generalization of Yao's circuit garbling [30]. It has information theoretic security if the underlying oblivious transfer (OT) protocol does. OT protocols exist that provide information theoretic security for either the sender or the receiver, but not both [8], [11]. Crepeau et. al. [11] leverages a noisy channel for unconditional OT. The GESS protocol's applicability is for shallow circuits. Additional work on information-theoretic secure computation includes [4], [10], [21], [23], and [29]. The scheme in [20] provides perfect security. This scheme is less scaleable than [24].

Rothblum [28] 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 a good security bound, 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.

1.4 Terminology and Background

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 $b\boldsymbol{x} = (bx_1, \ldots, bx_c) \bmod m$.

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 $[1, m-1]$.

Server privacy depends on the modular subset sum problem. More precisely, we will assume the following: Given uniform random $e_1, \ldots, e_n$ where $e_i \leq m/2n$, $1 \leq i \leq n$. Let $\boldsymbol{a} = (a_1, \ldots, a_n)$ where $a_i = be_i \bmod m$, $1 \leq i \leq n$. Define $f_{\boldsymbol{a}}(\boldsymbol{s}) = (\boldsymbol{a}, \boldsymbol{s} \cdot \boldsymbol{a})$ for uniform random $\boldsymbol{s} \in \{0,1\}^n$, where $\cdot$ denotes vector inner product. Then $f_{\boldsymbol{a}}(\boldsymbol{s})$ is a $1/m$ universal hash function. In other words,

$$Pr(f_{\boldsymbol{a}}(\boldsymbol{v}) = f_{\boldsymbol{a}}(\boldsymbol{w})) \leq 1/m$$

for every pair $\boldsymbol{v}, \boldsymbol{w} \in \{0,1\}^n$ where the probability is over the random choice of $\boldsymbol{a}$. We require $\log(m) \leq cn$ where $c < 1$.

Following [19], we define a probability distribution D on $\{0,1\}^m$ to be quasi-random within ϵ if $\forall X \subset \{0,1\}^m$ we have that $|Pr_D[X] - |X|/2^m| < \epsilon$, where $Pr_D[X]$ is the probability that an element chosen according to D is in X. We use Proposition 1.1(2) from [19]:

Proposition 1. *Let $\log(m) \leq cn$ for $c < 1$. For all but an exponentially small fraction of $\boldsymbol{a} = (a_1, \ldots, a_n)$, the distribution given by $f_{\boldsymbol{a}}(\boldsymbol{s})$ for a randomly chosen $\boldsymbol{s}$ is quasi-random within an exponentially small amount.*

For circuits, we let the parameter a be the number of addition gates 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.

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 A there exists a negligible function μ 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 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 3. 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 [18]. 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 Π'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 Π'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.

2 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 $\mathbb{Z}_m^c$.

We then present a somewhat homomorphic additive encryption scheme with perfect security (Definition 3), and we prove our somewhat homomorphic additive encryption (P-SWHAE) 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 2.) We prove correctness in Theorem 5 and perfect security in Theorem 6. Our schemes are secret key rather than public key.

2.1 Permutation Tables

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

Definition 1. *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 $(e_1, \ldots, e_c)$ in $\mathbb{Z}_m^c$ generates a* ***permutation table*** *T with $m-1$ rows and c columns. The ith row of the table, $1 \le i \le m-1$, is the vector $(ie_1 \bmod m, \ldots, ie_c \bmod m)$ 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 $\boldsymbol{v}$ in T a vector $\boldsymbol{r}$ with c components: $r_i = v_i \bmod N$, $1 \le i \le c$. A permutation table with no zeroes (****no zeroes or perfect 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 4 gives another example of a perfect permutation table.

Table 4. 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

Fact 1: If $s \leq (m-1)/2$, 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 $s \leq (m-1)/2$, then run period(s) $= (m-1)/s$, where s is the column start element.
Fact 3: For each column in a permutation table where $s \leq (m-1)/2$, 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 $s \leq (m-1)/2$.

2.2 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 row vector $(d_1, d_2, \ldots, d_c)$ where d_i/d_{i+1}, $1 \leq i \leq c-1$ and $d_1 = 1$. Then the d_{i+1} column flips when the d_i column does.*

Proof. We have $kd_i = qm + r$, $kd_i + d_i = qm + r + d_i$ 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 2. *Let T be a permutation table with row vector $(d_1, d_2, \ldots, d_c)$ where d_i/d_{i+1}, $1 \le i \le c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \le i \le c$, and $Nd_c \le \alpha$. Given column j in T where $2 \le j \le c$. 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 jth component, $2 \le j \le c$, occurs in a row prior to the Nth flip of the jth column that occurs after some left of j row, the 1st column component is bounded by α, and the cth component occurs in a row between zero and $N-1$ rows after a flip of the cth column.*

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

Example: Row 13 is a left of j row for column 3 in Table 4.

Lemma 2. *Let T be a permutation table with row vector $(d_1, d_2, \ldots, d_c)$ where d_i/d_{i+1}, $1 \le i \le c-1$ and $d_1 = 1$. Let $d_i > N(a+1)d_{i-1}$, $2 \le i \le c$, and $Nd_c \le \alpha$. Then $\lfloor N\frac{m-1}{d_i} \rfloor + 1$ is an upper bound for the sum of the lengths of the first N permutation runs for the column with start element d_i, where $d_i > 1$.*

Proof. The element d_i is the start element of the first permutation run in the sequence of N permutation runs in column i.

$$d_i + wd_i \le m - 1 + (N-1)(m) = Nm - 1$$

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

$$w + 1 \le \frac{Nm-1}{d_i}$$

$$w + 1 \le \left\lfloor \frac{Nm-1}{d_i} \right\rfloor \le \left\lfloor \frac{Nm}{d_i} \right\rfloor \le \left\lfloor \frac{N(m-1)}{d_i} \right\rfloor + 1$$

since $d_i \ge d_2 > N(a+1) > N$. ∎

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 $0 \le k_i \le d_i - 1$. If $1 \le x \le k_i$, 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

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

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 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 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 $Nd_c \leq \alpha$. (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

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

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

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

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

From above we have

$$\begin{aligned}\alpha &\geq \frac{m-1}{d_i}Nd_{i-1}+\alpha\frac{d_{i-1}}{d_i} \geq \frac{m-1}{d_i}Nd_{i-1}+Nd_{i-1} \\ &= \left(\frac{m-1}{d_i}+1\right)Nd_{i-1} \geq \left(\frac{Nm+d_i-2}{d_i}\right)d_{i-1}. \end{aligned} \tag{3}$$

due to (2) 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

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

where w is the number of d_i values added to d_i 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 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 $Nd_c \leq \alpha$. 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 α 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 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 $Nd_c \leq \alpha$. 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 2) 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.) 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 3 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 1. Let $\alpha = (m-1)/(a+1)$. Given a permutation table T 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 $Nd_c \leq \alpha$. 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 $\mathbb{Z}_N^c$. ∎

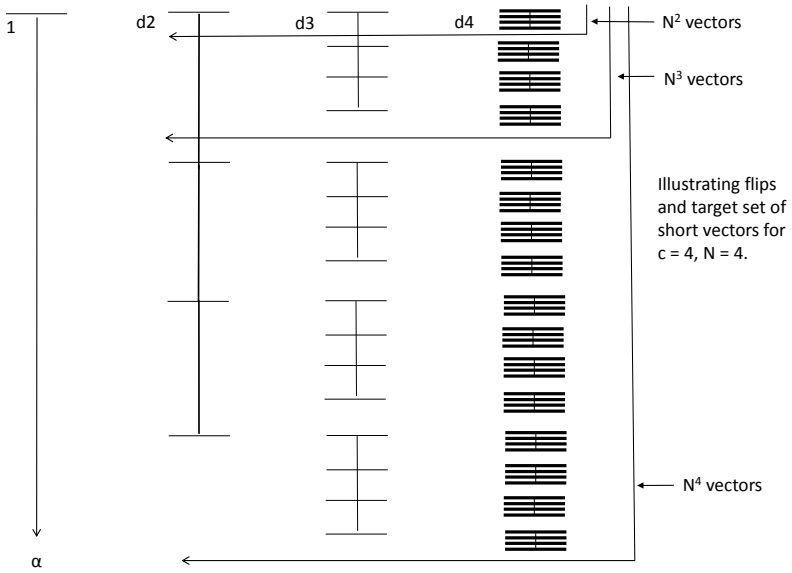

Fig. 3. Target Set of Short Vectors in Permutation Table, $c = 4, N = 4$. Short Horizontal Lines Indicate Start of New Permutation Runs (flips). Each Thick Line in Last Column Represents Last Column Components of N vectors in Target Set

Definition 3. ***Perfect Somewhat Homomorphic Additive Encryption (P-SWHAE)*** *Given a server circuit with a addition gates. N is a positive integer which is a power of 2.*

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 $\boldsymbol{v} = (v_1, \ldots, v_c)$:

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

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

Algorithm 1 Encryption Steps for P-SWHAE

1: **while** $\boldsymbol{w} \neq \boldsymbol{r}$ **do**
  i is index of the first component disagreement between $\boldsymbol{w}$ and $\boldsymbol{r}$.
  Let $f = (a(w_i - r_i)) \bmod N$.
  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 $w_1 = r_1, \ldots, w_i = r_i$.
  $w_i = \bar{w_i} \bmod N$ for $1 \leq i \leq c$, where $\bar{\boldsymbol{w}} = (\bar{w_1}. \ldots, \bar{w_c})$.
  $\boldsymbol{w_{saved}} = \bar{\boldsymbol{w}}$
endwhile

Let $\boldsymbol{e} = \bar{\boldsymbol{w}}$. The components of $\boldsymbol{e}$ are called exponents.

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.

Theorem 2. *Given the P-SWHAE encrypt algorithm which creates the vector* $\boldsymbol{e} = (e_1, \ldots, e_c) \in \mathbb{Z}_m^c$. *We add the restriction that the integers* $d_1, \ldots, d_c$ *are odd.* $\boldsymbol{e}$ *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. Initially we assume $d_c/m - 1$. After the initial steps of the encrypt algorithm that compute vector $\boldsymbol{v}$, we have vector $\bar{\boldsymbol{v}}$ in the permutation table where $v_1 = r_1$. Let $i \geq 2$ and $v_1 = r_1, \ldots, v_{i-1} = r_{i-1}, v_i \neq r_i$. Let $xd_i < m$ and $xd_i + d_i > m$. Let $xd_i \cong l_i \bmod N$. Then

$$(x+1)d_i \bmod m \bmod N = ((x+1)d_i - m) \bmod N = \\ l_i \bmod N + d_i \bmod N - m \bmod N$$

Thus each flip of the ith column results in subtracting $s = m \bmod N$ from the next element in the cyclic progression of the group $\mathbb{Z}_N$ (d_i odd implies that the entire group $\mathbb{Z}_N$ is cycled.)

Denote the permutation run in the ith column containing vectors with associated vectors having prefix $r_1, \ldots, r_i$ as z. Partition z into sequences of consecutive N vectors. Since there are no flips of any permutation in columns j where $j < i$, or in z during the run of z, then each sequence of N vectors has a vector with associated vector having prefix $r_1, \ldots, r_i$. Thus the prefix $r_1, \ldots, r_i$ is associated with a vector in the first N vectors of the permutation run of column i containing vectors that have this associated prefix.

Let f be the number of permutation runs of column i prior to the permutation run with vectors associated with prefix $r_1, \ldots, r_i$. Then

$$f(-s) + \bar{\boldsymbol{w}}_{\boldsymbol{i}} = \bar{\boldsymbol{r}}_{\boldsymbol{i}} \bmod N$$

Since $\bar{\boldsymbol{w}}_{\boldsymbol{i}} \bmod N = w_i$, and $\bar{\boldsymbol{r}}_{\boldsymbol{i}} \bmod N = r_i$, we have

$$fs \bmod N = (w_i - r_i) \bmod N,$$

$$f = s^{-1}(w_i - r_i) \bmod N.$$

(Since m is odd and N is a power of 2, $gcd(m, N) = 1$ so m has an inverse modulo N.)

Thus

$$\bar{\boldsymbol{v}}_{\boldsymbol{j}} = \left\lfloor \frac{m-1}{d_i} f \right\rfloor d_j \bmod m, 1 \leq j \leq c$$

calculates the permutation table vector in the last row prior to a flip in column i such that the sum $\bar{\boldsymbol{v}} + \boldsymbol{w}_{\boldsymbol{saved}} \bmod m$ gives a vector within the first N rows after the column i flip, where the first N rows contain a vector with associated vector prefix $r_1, \ldots, r_i$. Then the advance up or down the rows of T finds the vector

with prefix $r_1, \ldots, r_i$, (at most N rows up or down). Note that since $f < N$, the update $\bar{\boldsymbol{v}} + \boldsymbol{w_{saved}} \bmod m$ does not cause any flips in columns j where $j < i$ so the prefix $r_1, \ldots, r_{i-1}$ is preserved.

Each calculation of prefix $r_1, \ldots, r_i$ given prefix $r_1, \ldots, r_{i-1}$ selects the highest row in the permutation table that has associated vector with prefix $r_1, \ldots, r_i$. The associated prefix $r_1, \ldots, r_{i-1}$ in the first $i-1$ columns is within N vectors (rows) of the $i-1$ column flip so the prefix $r_1, \ldots, r_i$ will be prior to N flips of column i after the left of j row. Finally, the associated vector $r_1, \ldots, r_c$ will be within the first N vectors after a flip of column c. Thus the selected vector is in the target set of short vectors.

At the end, we obtain the vector $\boldsymbol{e}$ in the target set of short vectors such that $e_i \bmod N = r_i$, $1 \leq i \leq c$.

When $d_c \nmid m-1$, the same argument applies except vector row positions may be off by 1 row.

∎

Theorem 3. *Given m prime, circuit C with a addition gates, N a power of 2, and a permutation table T with start vector $d_1, \ldots, d_c$ as in the setup for P-SWHAE encryption (d_i/d_{i+1}, $1 \leq i \leq c$, $d_i > N(a+1)d_{i-1}$, $2 \leq i \leq c$, and $Nd_c \leq \alpha$.) We also assume d_i is odd, $1 \leq i \leq c$. Given x as input to P-SWHAE 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-SWHAE 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-SWHAE 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 $b_1^{-1}ye_1 = e_1$ where e_1 is the first coordinate of $\boldsymbol{e}$. Since $e_1 \in]1, m-1]$, we have $b_1^{-1}y = 1$ so $b_1 = y$. Thus

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

∎

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

2.3 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 $e \bmod N = r_1r_2 \bmod N$. [1]

P-SWHAE with Multiplication Scheme To preserve perfect security, the client is assumed to know the number of multiplication gates in the circuit, and we will use the P-SWHAE scheme described above. Each multiplication gate will add 1 to the c parameter.

At a high level, P-SWHAE 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

[1] 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.

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

Algorithm 2 P-SWHAE with Multiplication

1: **Initialize** with P-SWHAE 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. Create $\boldsymbol{e^0}$ using $\boldsymbol{r}$ as plaintext input to P-SWHAE encryption. b is secret key. Client sends $b\boldsymbol{e^0} \bmod m$ 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 decrypts by multiplying by $(bb_1 \cdots b_{i-1})^{-1} bw_j^i \bmod m$, $j = 1, 2$
to obtain $t_j \bmod m$, $j = 1, 2$. Let $r_j = t_j \bmod N$, $j = 1, 2$.
The client computes $s_i = r_1 r_2 \bmod N$; 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-SWHAE 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 $b_f^{-1}(o) \bmod m \bmod N$.

P-SWHAE with multiplication works as follows. Initially, the client executes P-SWHAE 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 2 for details. For P-SWHAE 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-SWHAE with Multiplication (Algorithm 2) 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.*

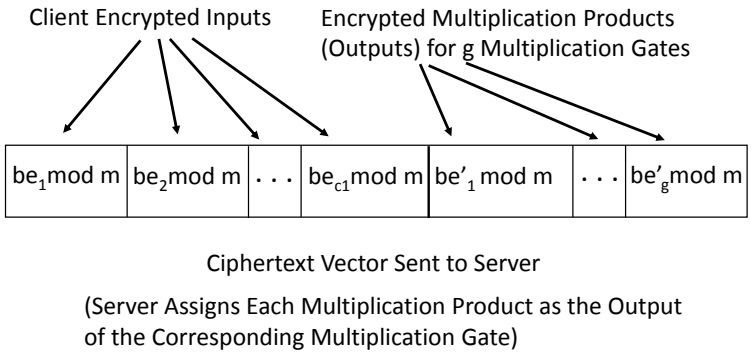

Fig. 4. Structure of Ciphertext Vector for P-SWHAE with Multiplication

Let o be the output obtained by the client using P-SWHAE with Multiplication. Then $o_1 = o$.

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 $\boldsymbol{e}^j_i \bmod N = r_i$, $1 \leq i \leq c_1$ and $\boldsymbol{e}^j_i \bmod N = s_i$, $c_1 + 1 \leq i \leq c_1 + j$. 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-SWHAE encryption to get $\boldsymbol{e}^{j+1}$ where $\boldsymbol{e}^{j+1}_i \bmod N = r_i$, $1 \leq i \leq c$, and $\boldsymbol{e}^{j+1}_i \bmod N = s_i$, $c_1+1 \leq i \leq c_1+j+1$. $b_{j+1}\boldsymbol{e}^{j+1} = \boldsymbol{e}^j$ 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 : $\boldsymbol{e^g} = (e^g_1, \ldots, e^g_c)$ where $\boldsymbol{e}^g_i \bmod N = r_i$ if $1 \leq i \leq c_1$, and $\boldsymbol{e}^g_{c_1+i} \bmod N = s_i$ 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-SWHAE with Multiplication (Algorithm 2) has perfect security.*

Proof. The only information sent to the server is a ciphertext vector from the P-SWHAE encryption algorithm in Algorithm 1. By Theorem 4, P-SWHAE 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 2, 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 3. Thus the multiplicands can be viewed as an encoding of the circuit. The proofs of correctness and security are similar to the proofs for Algorithm 2.

Algorithm 3 P-SWHAE with Multiplication: Subsequent Request

1: **Initialize** with P-SWHAE 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-SWHAE 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 $x = e_1^{-1} f_1 \bmod m$.
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 $r_j = t_j \bmod N$, $j = 1, 2$.
$s_i = r_1 r_2 \bmod N$; 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-SWHAE 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 $b_f^{-1}(o) \bmod m \bmod N$.

---

3 Two-Party Computation (2PC)

In this section, we consider the case where the server also has inputs for the circuit. As in Section 2, 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 8).

We support boolean ($N = 2$) circuits with both addition (XOR) and multiplication (AND) 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 7.) (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.) If the client receives notification from a verifier that verifies the correctness of server's processing after the output is delivered to the client, then the client can deliver an output to the server and be assured that client privacy is not impacted by a malicious server.

Server privacy depends on the hardness of the subset sum problem and our assumption that the client is semi-honest. Correctness also assumes the client is semi-honest. A verifier can inspect the protocol transcript along with the specification of the circuit to verify that the server's actions are correct. If the secret key and parities of the server inputs are not shared with the verifier, then client and server privacy is information-theoretically protected from the verifier.

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

Our protocol includes a preprocessing step that occurs prior to client and server inputs and subsequent processing. The client creates elements of the form $be \bmod m$ following the P-SWHAE encrypt algorithm as described above. This set of elements is then partitioned into client input elements, server input elements, multiplication gate elements, and noise generator elements. The client knows the parities of the elements before creating them; half of the noise generator element parities are even and the multiplication gate parities are 3 to 1 even to odd or vice versa depending on whether the circuit has AND or NAND gates. The client sends these elements to the server along with the parities of the exponents for the server input elements and the noise elements. The server input elements include an even and an odd parity exponent element for each server input.

For processing a multiplication gate, the server computes random subset sums of the noise elements and adds them to each multiplicand. Four elements are sent to the client.[2] Each multiplicand is added to both an odd sum and an even 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 exponents is

[2] For optimization, 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.

even (odd). Each pair of elements is randomly ordered prior to being sent to the client. The client is able to obtain the least significant bit after multiplying by $b^{-1} \bmod m$ for all four of the elements and return four products. 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 5).

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

1: The client selects the c parameter: $c = c_c + 2c_s + c_1 + 4g$ where g is the number of multiplication gates in the circuit, c_c is the number of client inputs, c_s is the number of server inputs, and c_1 is a small integer (e.g., $c_1 = 8$) for the number of noise generator elements used to create noise elements. Note that the client knows the parities of the c elements (half of server and noise generator elements are even parity) so it creates the $\boldsymbol{e}$ vector per the P-SWHAE key-generation and encryption algorithms ($N = 2$.) We replace α in P-SWHAE with $\alpha_2 = (m-1)/(2(a+1))$ to accomodate the subset sums that the server will add to points prior to sending to the client.

2: The client creates uniform random noise elements from the c_1 elements discarding any duplicates until the client has n new elements. n is larger than $\log(m)$. The random noise element exponents have the form $a_1t_1 + a_2t_2$ where a_1 and a_2 are integers and t_1, t_2 are relatively prime integers in the set of c_1 elements. Let $h_1, \ldots, h_n$ be the set of noise element exponents. The client ensures that these values are positive and $\sum_i h_i < m/2$:
$i = 0$
while $i < n$ **do**
  $h_i \leftarrow a_{i1}t_1 + a_{i2}t_2$; $a_{i1}, a_{i2} \leftarrow \mathbb{Z}$, t_1, t_2 are noise generator elements.
  if $h_i = h_j$ for some $j < i$ or $h_i > m/2n$ or h_i equals a client input or multiplication gate element **then**
    discard h_i
  else
    store h_i, parity(h_i)
    i = i + 1
  endif
endwhile

3: The client sends the noise elements to the server along with information to identify the parity of the exponents for each element. Noise elements are defined to have even (odd) parity if and only if the noise element exponents have even (odd) parity. A sum of noise elements is defined to be a subset sum with even (odd) parity if and only if the sum of noise element parities is even (odd).
send $z_i = bh_i \bmod m$, parity(h_i) to server, $1 \le i \le n$.
server stores z_i, parity(h_i), $1 \le i \le n$.
send m to the server (for modular subset sum)

Algorithm 5 Input Processing Steps for 2-Party Protocol with Perfect Security for Client

1: The client creates $2c_s$ server input elements each of the form $y_i = be_i \bmod m$ where these $2c_s$ elements are part of the larger set of c elements. Half of the server input elements have exponents with even parity; the other half has exponents with odd parity. Each server input exponent e_i is taken from the $\boldsymbol{e}$ vector created during the P-SWHAE encryption algorithm.

2: Client randomizes the order of $y_i, 1 \leq i \leq c_s$, and sends the y_i to the server. It sends a separate message with y_i exponent parities (the parities of e_i in $y_i = be_i \bmod m$) to the server. (The exponent parities will not be shared with a verifier.)

3: The client creates c_c input elements each of the form $w_i = be_i \bmod m$ where these c_c elements are part of the larger set of c elements. Each client input element e_i value is taken from the $\boldsymbol{e}$ vector created during the P-SWHAE encryption algorithm.

4: Client sends w_i to server, $1 \leq i \leq c_c$.

5: The server can now begin processing through the circuit given the client and server input elements.

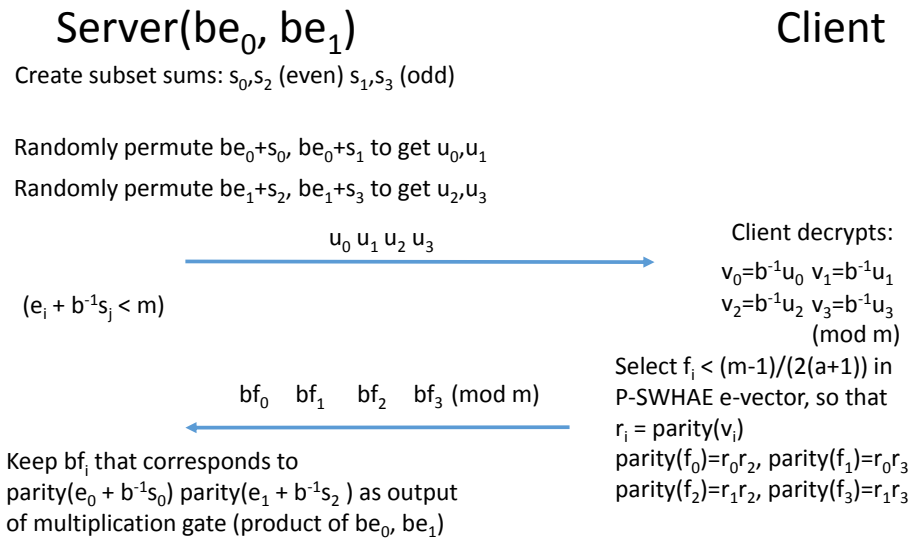

Fig. 5. Multiplication Gate Protocol for 2PC Scheme

Algorithm 6 Steps for 2-Party Protocol with Perfect Security for Client: Multiplication, Addition Gates and Output

1: When the server needs to multiply be_1 and be_2, it creates four elements (in $\mathbb{Z}_m$) to send to the client: be_1 plus an even parity subset sum element (using the n z_i noise elements), be_1 plus an odd parity subset sum element, be_2 plus plus an even parity subset sum element, and be_2 plus an odd parity subset sum element. The 1st and 2nd elements are randomly ordered and the 3rd and 4th elements are randomly ordered:
   for $i = 0$ to 3 **do**
   $\boldsymbol{z} = (z_1, \ldots, z_n)$
   $\boldsymbol{s} \leftarrow \{0,1\}^n$
   $sum_i \leftarrow \boldsymbol{s} \cdot \boldsymbol{z} \bmod m$
   define parity$(sum_i) = \sum_{s_j=1} parity(h_j)$
   do while $parity(sum_i) \neq i \bmod 2$
   select random j where $s_j = 1, sum_i = sum_i - z_j$
   select random j where $s_j = 0, sum_i = sum_i + z_j$
   end do while
   end for
   $x_0 = be_1 + sum_0 \bmod m.$
   $x_1 = be_1 + sum_1 \bmod m.$
   $x_2 = be_2 + sum_2 \bmod m.$
   $x_3 = be_2 + sum_3 \bmod m.$
2: x_0 and x_1 are randomly ordered and sent to the client.
3: x_2 and x_3 are randomly ordered and sent to the client.
4: Relabel x_0, x_1, x_2, x_3 as u_0, u_1, u_2, u_3 where u_0, u_1, u_2, u_3 is the receiving order.
5: Client computes:
   for $i = 0$ to 3 **do**
   $v_i = u_i b^{-1} \bmod m$
   if $parity(v_i) = 1$ **then**
   $r_i = 1$
   else
   $r_i = 0$
   endif
   end for
6: Client selects f_0, f_1, f_2, f_3 from $\boldsymbol{e}$ that was created during P-SWHAE encrypt step where $parity(f_0) = r_0 r_2$, $parity(f_1) = r_0 r_3$, $parity(f_2) = r_1 r_2$, and $parity(f_3) = r_1 r_3$.
7: The client computes $bf_0 \bmod m$, $bf_1 \bmod m$, $bf_2 \bmod m$, and $bf_3 \bmod m$, and sends these elements to the server in this order.
8: The server keeps the element corresponding the correct product and discards the other 3 elements. The kept element is the output of the multiplication gate.
9: Addition gates are processed by the server without client interaction; the two inputs are added as integers to get the output.
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.

3.1 Proof of Security

Theorem 7. *Given the 2-party protocol of Algorithms 4, 5, and 6 where the client is semi-honest and does not share the output with the server. The server may be computationally unbounded and malicious.*[3] *Then the protocol has server privacy given the hardness of the modular subset sum problem and perfect client privacy.*

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 Π'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, output, and the security parameter m. It selects uniform random b such that $b < m$ (m prime) and follows P-SWHAE encrypt steps. It creates the noise elements as described in Algorithm 4.

It then selects c_s server input elements of the same form $be \bmod m$ 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.

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 subset sum element. Otherwise, the output is summed with an odd parity subset sum element. Thus we ensure that the client obtains the same output in both the simulation and the real protocol.

Proposition 1.1 (2) of [19] shows that when $\log(m) < n$ the sums of noise elements have a probability distribution that is exponentially close to uniform random for all but an exponentially small number of sums. 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 candidate bits,

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

noise generator bits, and multiplication element bits generate components of $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$. The 2PC protocol messages from the client to the server consist of components from the ciphertext vector $\boldsymbol{w}$. Thus we have

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

∎

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

Theorem 8. *Given the 2-party protocol of Algorithms 4, 5, and 6 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 the modular subset sum problem and perfect client privacy.*

Proof. The proof for server privacy is the same as the proof of server privacy in Theorem 7.

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 Π'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 P-SWHAE encrypt steps. It creates noise generator elements as described in Algorithm 4. The client plaintext bits are random. The probability of selecting any set of bits for the noise generator, multiplication, and server inputs between the real protocol and the simulator is the same. The difference is the client input 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 components of the ciphertext vector $\boldsymbol{w}$. The same is true for $\mathcal{S}_2$. 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. ∎

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. 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
3. 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
4. Beaver, D.: Minimal-latency secure function evaluation. In: Proceedings EUROCRYPT 2000. Lecture Notes in Computer Science, vol. 1807, pp. 335–350. Springer-Verlag, Berlin, Heidelberg (2000)
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. 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
7. 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
8. Branco, P., Döttling, N., Srinivasan, A.: A framework for statistically sender private ot with optimal rate. 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. 548–576. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38557-5_18, https://doi.org/10.1007/978-3-031-38557-5_18
9. 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
10. Cleve, R.: Towards optimal simulations of formulas by bounded-width programs. In: STOC 90: Proceedings of the 22nd ACM Symposium on Theory of Computing. pp. 271–277. ACM Press, New York, NY (1990)
11. Crepeau, C., Morozov, K., Wolf, S.: Efficient unconditional oblivious transfer from almost any noisy channel. In: Security in Communication Networks 2004. pp. 47–59 (2004)
12. 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
13. 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
14. 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

15. 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
16. 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
17. 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
18. Hazay, C., Lindell, Y.: Efficient secure two-party protocols, techniques and constructions. Springer-Verlag (2010)
19. Impagliazzo, R., Naor, M.: Efficient cryptographic schemes provably as secure as subset sum. Journal of Cryptology **1996**(9), 199–216 (1996)
20. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: ICALP 2002. pp. 244–256 (2000)
21. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: Proceedings 41th IEEE Symposium on Foundations of Computer Science. IEEE Computer Society (2000)
22. 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
23. Kilian, J.: Founding crytpography on oblivious transfer. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 20–31. STOC '88, Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/62212.62215, https://doi.org/10.1145/62212.62215
24. Kolesnikov, V.: Gate evaluation secret sharing and secure one-round two-party computation. In: Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3788, pp. 136–155. Springer (2005). https://doi.org/10.1007/11593447_8, https://iacr.org/archive/asiacrypt2005/134/134.pdf
25. 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)
26. Rivest, R.L., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation. pp. 169–180 (1978)
27. 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
28. 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)
29. Sander, T., Young, A., Yung, M.: Non-interactive cryptocomputing for nc1. In: Proceedings 40th IEEE Symposium on Foundations of Computer Science. pp. 554–566. IEEE Computer Society, New York, NY (1999)

30. 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 Basic SWHAE Scheme

In this section we present a somewhat homomorphic additive encryption scheme: the basic SWHAE 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.

A.1 Additive Homomorphic Scheme

Definition 4. ***Basic Somewhat Homomorphic Additive Encryption (Basic SWHAE)*** *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 3 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 SWHAE 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 SWHAE scheme to have less than perfect security.

B Integrating a Verifier

A verifier without access to the client secret key and parities of the client, server inputs but with access to the rest of the server's data and calculations, client's protocol messages, and circuit specification can verify that the server's computations are correct:

Theorem 9. *A verifier with access to a run of the 2-party protocol of Algorithms 4, 5, and 6 can verify that the server's computations are correct. Access to the run is defined as:*

1. *access to the server's internal data and computations other than the server input parity information,*
2. *access to the client's protocol record of messages sent and received by the client, and*

3. *access to the circuit specification.*

If the verifier does not have access to additional data (client secret key, input parities, and server's input parities), then client and server privacy is protected information-theoretically from a computationally unbounded verifier. If the verifier is correct then a malicious unbounded server that deviates from the protocol will be detected with high probability assuming the client is semi-honest.

Proof. Based on the server encrypted input elements and client encrypted input elements, the verifier can calculate an expected encrypted output element v_{out}, where $v_{out} + subsum_{final} = out$ where out is the encrypted output element sent to the client. At every multiplication gate, the server shows $v_1 + subsum_1 = c_1$ and $v_2 + subsum_2 = c_2$ where c_i are from the client transcript, $subsum_i$ are verifiable subset sums and v_i are the expected encrypted inputs for the gate, $i = 1, 2$. Thus verifier is able to select the correct response from the 4 elements in the client transcript. To verify successfully, the final server encrypted element s satisfies $s = v_{out}$ and $v_{out} + subsum_{final} = out$ where $subsum_{final}$ has even parity.