Information-Theoretic 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 additive somewhat homomorphic schemes where the client has perfect privacy (server can be computationally unbounded). Our basic scheme is additive somewhat homomorphic and we give protocols to handle addition and multiplication. In one scheme, the server handles circuit multiplication gates by returning the multiplicands to the client which does the multiplication and sends back the encrypted product. We give a 2-party protocol that also incorporates server inputs where the client has perfect privacy. Server privacy is not information-theoretic, but rather depends on hardness of the subset sum problem. Correctness for the server in the malicious model can be verified by a 3rd party with high probability where the client and server privacy are information-theoretically protected from the verifier. Scaling the 2PC protocol via separate encryption parameters for smaller subcircuits allows the ciphertext size to remain constant as circuit size grows.

Keywords: Additive somewhat homomorphic encryption · information-theoretic.

1 Introduction

Two-party computation protocols include garbled circuit based protocols which is a technique first presented by Yao [28]. Goldreich Micali Wigderson [20], [4] 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 [24],and textbook RSA [26]. Rivest et al. posed the question of homomorphic encryption [25] in 1978. Gentry's breakthrough work [17] 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 [15], [6], [7], [10], [9], [2], [16] 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 information-theoretic (including perfect) security for the client.

Our basic scheme includes a modulus m, base b where $gcd(b, m) = 1$, 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 the pair (b, m).)

The 2nd property gives the information-theoretic security. Our main results for somewhat homomorphic additive encryption are Theorem 1 and Theorem 3 which prove that our No Zeroes Somewhat Homomorphic Additive Encryption (NZ SWHAE) scheme has perfect security.

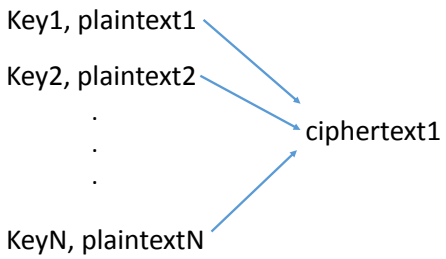

Fig. 1. Distinct Plaintext Key Pairs Map to Same Ciphertext

In our first multiplication scheme, the server returns the multiplicands to the client which decrypts and performs the multiplication. The client sends a new ciphertext element back to the server encrypting the product of the plaintexts. All additions and multiplications are mod 2. (In the general case we replace 2 with N which is power of 2 so addition and multiplication are mod N.)

A second scheme with multiplication includes additional client elements with both odd and even parities. The server computes 2^g results where g is number of multiplication gates. The server assigns one of the additional client elements as the output of each multiplication gate. Thus this scheme can only be practical for a circuit with a small number of multiplication gates.

We give a 2-party MPC (2PC) scheme where client privacy is perfect (the server is computationally unbounded and leakage is 0) and server privacy is based on the hardness of the 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 elements 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 security 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. To the best of our knowledge, our 2-party scheme is the first one to provide 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 and 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

Consider an example with parameters $m = 31$, $b = 17$, and $c = 2$. We set $N = 2$ (bit encryption) and $e_1 = 12$, $e_2 = 15$.

The client sends the pair $be_1 \bmod m, be_2 \bmod m$ to the server. This pair encrypts the client input pair $(0, 1)$. This bit input pair is obtained from the parity of e_1 and e_2.

$be_1 \bmod m = 18, be_2 \bmod m = 7$.

Table 1 shows the full set of multiples for $m = 31, c = 2, e_1 = 12, e_2 = 15$.

Given the pair $(18, 7)$, the server cannot deduce the client input pair. For example, if we take $d = 16$, $f_1 = 5$, $f_2 = 14$, then $df_1 = 18$, $df_2 = 7$. Let $d = 7, f_1 = 7, f_2 = 1$. Again $df_1 = 18$, $df_2 = 7$. Thus the pair $(18, 7)$ is consistent with multiple different input plaintext pairs; in this case both $(1, 0)$ and $(1, 1)$.

For this example, these exponent pairs all satisfy the relation that the sum of the exponents is less than m. Thus these inputs to the server can be used to evaluate a circuit with one addition gate. The client can decrypt the returned

Table 1. Multiples Table for $m = 31$, $c = 2$, $e_1 = 12$, $e_2 = 15$.

Multiple	*Pair*	*Multiple*	*Pair*
1	(12, 15)	16	(6, 23)
2	(24, 30)	17	(18, 7)
3	(5, 14)	18	(30, 22)
4	(17, 29)	19	(11, 6)
5	(29. 13)	20	(23, 21)
6	(10, 28)	21	(4, 5)
7	(22, 12)	22	(16, 20)
8	(3, 27)	23	(28, 4)
9	(15, 11)	24	(9, 19)
10	(27, 26)	25	(21, 3)
11	(8, 10)	26	(2, 18)
12	(20, 25)	27	(14,2)
13	(1, 9)	28	(26, 17)
14	(13, 24)	29	(7, 1)
15	(25, 8)	30	(19, 16)

value from the server by multiplying by $b^{-1} \bmod m$ and computing the parity of the result.

Table 2 shows the various possible exponent pairs that are consistent with each bit input pair where the sum of the exponents is less than m. Thus there is a small amount of leakage since the different input pairs have different frequencies given the ciphertext pair. The entropy for this example is $-\sum p_i \log(p_i) \approx 1.89$ which is 0.11 bits less than perfect security (2 bits).

Table 2. Client Exponent Pairs Consistent with Server Received Pair $(18, 7)$ for $m = 31$, $c = 2$.

Bit input pair (0.1)	*Pair* $(1,0)$	*Pair* $(1,1)$	*Pair* $(0,0)$
(12, 15)	(5, 14)	(3, 27)	(8, 10)
(6, 23)	(11, 6))	(15, 11)	(2, 18)
(18, 7)		(1, 9)	(14, 2)
(4, 5)		(9, 19)	
		(21, 3)	
		(7, 1)	

1.2 Our Contributions

We have the following results:

1. We give an additive homomorphic encryption algorithm that provides perfect client privacy, and extend this to include multiplication both via a non-interactive protocol and also a more scaleable protocol that relies on client

assistance for multiplication operations. The resulting protocols are secret key somewhat homomorphic that protect the client privacy from a computationally unbounded server. To the best of our knowledge, our additive homomorphic encryption is the first such algorithm that is information-theoretic secure and more specifically, provides perfect privacy.
2. We show that our No Zeroes SWHAE scheme has perfect security in Theorem 1 and Theorem 3.
3. Based on the somewhat homomorphic protocol, 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 as in Theorem 3, from the unbounded server. The server is in the malicious model.
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 with high probability, 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.
6. The 2PC protocol can be scaled by dividing the circuit into smaller subcircuits each with separate encryption parameters. This scaling allows the ciphertext size to remain constant as circuit size grows.

1.3 Related Work

There has been extensive work in homomorphic encryption since the breakthrough work of Gentry [17] (e.g., [18], [8], [5], [19], [15], [6], [7], [10], [9], [2], [16]. These schemes require less rounds than our main scheme with requires interaction with the client for every multiplication gate. Our schemes provide perfect privacy for the client.

Foundational work in multiparty computation includes [28], [20], [4], [12].

It is not possible to protect both parties with statistical 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 [28] 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).

[3] allows the results of [22] to be based on additional assumptions such as CDH. [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 [23] 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 [28]. 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 [11], [14]. Crepeau et. al. [14] leverages a noisy channel for unconditional OT.

We give a very rough estimate of communication complexity for GESS [23] and our 2PC protocol based on the NZ SWHAE encryption (see Section 3) for AES-128 encryption. The details for our scheme are explained in the following sections. The server's input is the AES encryption key and the client's input is the plaintext to encrypt. Only the client receives output. The AES circuit from https://csrc.nist.gov/Projects/circuit-complexity has 6400 AND gates, 22200 other gates (XNOR and XOR) and depth 326. The GESS communication complexity is $2^d d^2 \log d$ where d is the depth of the circuit. 2^d can be replaced with the number of input wires which we take as 256. Our calculation gives 216.6 Mb for GESS. Our scheme scales best by subdividing the circuit (see Section 4). A more detailed analysis of how to subdivide the AES circuit is beyond the scope of this paper. We approximate using 800 subcircuits with 8 AND gates and 28 non-AND gates each. Our parameter c is calculated as $c = 4g + 4 + (2 \cdot 10)$ where g is the number of AND gates, 4 is for noise generator elements, and 20 is for re-encryption of subcircuit outputs from XOR gates when passing into another subcircuit (one element sent from server and two elements sent back from client). The $4g$ term includes re-encryption of subcircuit outputs from AND gates. Thus $c = 56$. We take $m \approx c(2a+2)^{c-1}(a+1) + 1 \approx 2^{333}$. Then communication per subcircuit includes 6 elements per AND gate (6 times 8) and 3 elements per XOR gate transition (3 times 10). The total is $48 + 30 = 78$. We also have $n = \log(m^{1/3})$ noise elements for a total of $78 + 111 = 189$ total elements communicated per subcircuit. The total is $189 \cdot 333 \cdot 800 = 50.35$ Mb. Security for both GESS and our scheme allow the server to be computationally unbounded and our scheme also ensures the privacy of the client if the server is malicious (rather than semi-honest).

Rothblum [27] 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 Theorem 1). Thus the Rothblum constructions do not apply to our scheme.

Our additively homomorphic encryption can be applied to give a PIR protocol. Information-theoretic PIR protocols cannot be more efficient than the trivial PIR protocol [13] and the resulting PIR protocol is less efficient than the trivial protocol.

1.4 Terminology and Background

A vector $\mathbf{x}$ 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.

We will leverage the subset sum problem: given integers $\mathbf{a} = (a_1 \ldots, a_n) \in \mathbb{Z}^n$ and a uniform random vector $\mathbf{s}$ where $s_i \in \{0,1\}$. The adversary is given $(T = \mathbf{a} \cdot \mathbf{s}, \mathbf{a})$ and must find $\mathbf{s}$, where "$\cdot$" denotes the inner product. The subset sum problem is considered hard for a computationally bounded adversary. The best known algorithms are exponential in the smaller of n and T.

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.

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

We follow [21] for our definition of simulation privacy in the semi-honest model, assuming a deterministic functionality f. Our 2PC protocol has server privacy if there is a probabilistic polynomial-time algorithm $\mathcal{S}$ such that

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

where $VIEW_c$ 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.

2 Somewhat Additive Homomorphic Protocols

We first present a somewhat homomorphic additive encryption scheme and then schemes with both addition and multiplication. Our schemes are secret key rather than public key.

2.1 Additive Homomorphic Scheme

Definition 1. ***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 $0 \le r_i < N$, $1 \le i \le c$. 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 $e_i \le \frac{m}{a+1}$, $1 \le i \le c$.*

The client ciphertext vector is obtained by selecting an integer representative v_i, $1 \le i \le c$, 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.

The vector $\boldsymbol{e} = (e_1, \ldots, e_c)$ from the above definition generates a cyclic vector subspace L of $\mathbb{Z}_m^c$ over the field $\mathbb{Z}_m$. We are interested in these subspaces and now give some definitions. Unless stated otherwise, vector components are assumed to be nonnegative.

Definition 2. *A vector $(e_1, \ldots, e_c)$ in $\mathbb{Z}_m^c$ generates a **permutation table** with $m-1$ rows and c columns. The ith row of the table is the vector $(ie_1 \bmod m, \ldots, ie_c \bmod m)$ in $\mathbb{Z}_m^c$. A **short vector** in the table is a vector $(v_1, \ldots, v_c)$ such that $v_i \leq (m-1)/(a+1) = \alpha$ for $1 \leq i \leq c$. We associate with each short vector a binary vector $\boldsymbol{r}$ with c components: $r_i = e_i \bmod 2$, $1 \leq i \leq c$. A permutation table with no zeroes (**no zeroes table**) is a table where for each of the possible 2^c binary vectors there is some short vector in the table that has that binary vector as its associated binary vector. A **perfect table** is a no zeroes table where each binary vector is associated an equal number of times.*

*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 average length*

of a permutation run is the ***permutation 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****.*

The entropy of a permutation table is the entropy of the binary vectors associated with the short vectors. The probability of a binary vector is the number of occurences of short vectors which are associated with the binary vector divided by the total number of short vectors in the table.

Example: In Table 1, the first column first permutation run has two elements, 12 and 24. The first column second permutation run has 3 elements: 5,17, and 29.

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 $\lfloor (m-1)/s \rfloor + 1$.

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

Proposition 1. *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.

2.2 Additive Homomorphic Scheme with Perfect Security

We will prove the following theorem:

Theorem 1. *Given a circuit with a addition gates and parameters m and c as described in the basic SWHAE scheme above. Let $\alpha = (m-1)/(a+1)$. Suppose there exists 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 $d_i \geq (a+2)d_{i-1}$, $2 \leq i \leq c$, and $cd_c \leq \alpha$. Then T is a no zeroes table.*

Proof. We use induction on c with the $c = 3$ case first. In this case, $2^c = 2^3 = 8$ so we will show there exist 8 distinct associated binary vectors. These vectors occur in the first two rows of the table, the two rows after the d_3 permutation has its first flip, the two rows after the d_2 permutation has its first flip, and the first two rows of the next d_3 permutation flip. First we show that these vectors are all short. Since $2d_3 \leq \alpha$, and the 2nd row is $(2, 2d_2, 2d_3)$, we see that the first two row vectors are short.

After a d_3 column flip, the 3rd column value is at most $d_3 - 1$. So the values in the d_3 column first two rows after a flip are always short (less than or equal to α.) Since d_2/d_3, it follows that the d_3 column flips whenever the d_2 column flips. So all of the 8 vector 3rd column values are short. We now examine the 2nd (d_2) column. $2d_2 < 2d_3 \leq \alpha$ so the values in the first two rows of the d_2 column are always short after a d_2 column flip.

Now consider the d_2 column vector values after the d_3 column flips. The 2nd row entry after the first d_3 column flip is $(\lfloor (m-1)/d_3 \rfloor + 1)d_2 + d_2$.

We will show that

$$\left(\left\lfloor \frac{m-1}{d_3} \right\rfloor + 2 \right) d_2 + d_2 \leq \alpha$$

$$\left(\left\lfloor \frac{m-1}{d_3} \right\rfloor + 2 \right) d_2 + d_2 \leq \left(\frac{m-1}{(a+2)d_2} + 2 \right) d_2 + d_2 = \tag{1}$$

$$\left(\frac{m-1}{a+2} + 3d_2 \right) \leq \frac{m-1+3d_3}{a+2} \leq \frac{m-1+\alpha}{a+2} = \tag{2}$$

$$\left(\frac{1}{a+2} \right) ((a+1)\alpha + \alpha) = \frac{(a+2)\alpha}{a+2} = \alpha \tag{3}$$

The value for the d_2 column after the d_3 column flips and the d_2 column has flipped once previously is also short since we require

$$\left(\left\lfloor \frac{m-1}{d_3} \right\rfloor + 2 \right) d_2 + d_2 - 1 \leq \alpha$$

which is implied by the previous inequality above.

We now show that the 1st column also has short values in the 8 vector rows. The largest value in the 8 short rows of the 1st column is bounded by

$$\left\lfloor \frac{m-1}{d_2} \right\rfloor + \left\lfloor \frac{m-1}{d_3} \right\rfloor + 3;$$

we must show this value is $\leq \alpha$.

$$\left\lfloor \frac{m-1}{d_2} \right\rfloor + \left\lfloor \frac{m-1}{d_3} \right\rfloor + 3 \leq \frac{m-1}{a+2} + \frac{m-1}{(a+2)^2} + 3 \leq \tag{4}$$

$$\alpha \left(\frac{a+1}{a+2} + \frac{a+1}{(a+2)^2} \right) + 3 \leq \alpha \left(\frac{a+1}{a+2} + \frac{a+1}{(a+2)^2} + \frac{3(a+1)}{m-1} \right) \leq \tag{5}$$

$$\alpha \left(\frac{a+1}{a+2} + \frac{a+1}{(a+2)^2} + \frac{1}{(a+2)^2} \right) = \tag{6}$$

$$\alpha \left(\frac{(a+2)(a+1) + (a+1) + 1}{(a+2)^2} \right) = \alpha \left(\frac{a^2+4a+4}{(a+2)^2} \right) = \alpha \tag{7}$$

Thus all of the 3 column values in the 8 identified rows are short. We now show that all 8 possible binary vectors of length 3 are associated with the 8 rows.

Table 3 below gives the parities of the elements in the 8 rows and 3 columns. Appendix A presents 4 tables with instantiation of u, v, and d_3/d_2 from Table 3 in order to more easily verify that all 8 possible short vectors are present.

The $c = 3$ proof is complete and we now prove the inductive step.

Row number	*1st Column*	*d_2 column*	*d_3 column*
1st row	1	$u = d_2 \bmod 2$	$v = d_3 \bmod 2$
2nd row	0	0	0
$\lfloor (m-1)/d_3 \rfloor + 1$ row	w	uw	1 if $v = 0$, $1 - w$ if $v = 1$
$\lfloor (m-1)/d_3 \rfloor + 2$ row	$1 - w$	$u(1-w)$	1 if $v = 0$, w if $v = 1$
$\lfloor (m-1)/d_2 \rfloor + 1$ row	x	1 if $u = 0$, $1 - x$ if $u = 1$	$d_3/d_2 \bmod 2$ if $v = 0$, $1 - x$ if $v = 1$
$\lfloor (m-1)/d_2 \rfloor + 2$ row	$1 - x$	1 if $u = 0$, x if $u = 1$	$d_3/d_2 \bmod 2$ if $v = 0$, x if $v = 1$
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor$ +1 row	y	1 if $u = 0$, $1 - y$ if $u = 1$	$1 - d_3/d_2 \bmod 2$ if $v = 0$, y if $v = 1$
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor$ +2 row	$1 - y$	1 if $u = 0$, y if $u = 1$	$1 - d_3/d_2 \bmod 2$ if $v = 0$, $1 - y$ if $v = 1$

Table 3. Parities for 8 short rows when $c = 3$, w, x, and y are either 0 or 1.

For the inductive step, we must prove three things:

1. The new short vector (see Figure 2) components are short. There are 2^{c-1} existing short vectors in the first $c - 1$ columns by the inductive hypothesis and we will show 2^{c-1} additional short vectors (the new vectors) in the c columns.
2. Each of the 2^c associated binary vectors are represented.
3. The last component (cth column) of the existing short vectors are short.

The last entries of the existing and new short vectors are short since the 1st entry after the cth column flip is $\leq d_c - 1$ and so the 2nd entry is $\leq 2d_{c-1} - 1$

and $2d_c \leq \alpha$. The cth component of all of the short vectors is in the 1st or 2nd row after a c column flip.

Now we show that the first $c-1$ entries of the new short vectors are short.

There are 2^{c-2} pairs of new vectors.

Given i where $2 \leq i \leq c-1$. We have

$$d_i\left(\left\lfloor\frac{m-1}{d_{i+1}}\right\rfloor+\left\lfloor\frac{m-1}{d_{i+2}}\right\rfloor+\ldots+\left\lfloor\frac{m-1}{d_c}\right\rfloor+c-i+1\right)+d_i\leq$$
$$d_i\left(\frac{m-1}{d_{i+1}}+\frac{m-1}{d_{i+2}}+\ldots+\frac{m-1}{d_c}+c-i+1\right)+d_i\leq$$
$$d_i\left(\frac{m-1}{d_i(a+2)}+\frac{m-1}{d_i(a+2)^2}+\ldots+\frac{m-1}{d_i(a+2)^{c-i}}+c-i+1\right)+d_i=$$
$$\frac{m-1}{a+2}+\frac{m-1}{(a+2)^2}+\ldots+\frac{m-1}{(a+2)^{c-i}}+(c-i+2)d_i\leq$$
$$\frac{(m-1)((a+2)^{c-i-1}+(a+2)^{c-i-2}+\ldots+1)+d_c(c-i+2)}{(a+2)^{c-i}}\leq$$
$$\frac{\alpha(a+1)((a+2)^{c-i-1}+(a+2)^{c-i-2}+\ldots+1)+\alpha}{(a+2)^{c-i}}=$$
$$\frac{\alpha(a+1)\left(1-(a+2)^{c-i}/(-a-1)\right)+\alpha}{(a+2)^{c-i}}=\alpha$$

We now show that the first column entries for the $c-1$ vectors are short. We require

$$\left\lfloor\frac{m-1}{d_2}\right\rfloor+\left\lfloor\frac{m-1}{d_3}\right\rfloor+\ldots+\left\lfloor\frac{m-1}{d_c}\right\rfloor+c-2+2\leq\alpha$$

$$\left\lfloor\frac{m-1}{d_2}\right\rfloor+\left\lfloor\frac{m-1}{d_3}\right\rfloor+\ldots+\left\lfloor\frac{m-1}{d_c}\right\rfloor+c\leq\frac{m-1}{d_2}+\ldots+\frac{m-1}{d_c}+c\leq$$
$$\alpha\left(\frac{a+1}{a+2}+\frac{a+1}{(a+2)^2}+\ldots+\frac{a+1}{(a+2)^{c-1}}\right)+c=$$
$$\alpha\left(\frac{a+1}{a+2}+\frac{a+1}{(a+2)^2}+\ldots+\frac{a+1}{(a+2)^{c-1}}+\frac{c(a+1)}{m-1}\right)\leq$$
$$\alpha\left(\frac{a+1}{a+2}+\frac{a+1}{(a+2)^2}+\ldots+\frac{a+1}{(a+2)^{c-1}}+\frac{1}{d_c}\right)\leq$$
$$\alpha\left(\frac{a+1}{a+2}+\frac{a+1}{(a+2)^2}+\ldots+\frac{a+1}{(a+2)^{c-1}}+\frac{1}{(a+2)^{c-1}}\right)=$$
$$\alpha\left(\frac{(a+1)((a+2)^{c-2}+\ldots+(a+2)+1)}{(a+2)^{c-1}}\right)=$$
$$\alpha\left(\frac{(a+1)((1-(a+2)^{c-1})/(-a-1))+1}{(a+2)^{c-1}}\right)=\alpha$$

Thus the first $c-1$ entries of the new short vectors are short.

We now show that each of the 2^c possible binary vectors are associated in the table. In Figure 2, the pair of short vectors denoted by 1 associate with the pair of short vectors denoted by 5. Similarly for 2 and 6, 3 and 7, and 4 and 8. If we consider only the first $c-1$ columns, then the pair of binary vectors in 1 and 5 are the same. The reason is that the odd components in the start element for a column change their parity on each row and only the even components change parity after a column flip. So the vector an even number of rows away is identical in the first $c-1$ columns. The *cth* column parity is reversed for the identical $c-1$ column vectors due to the flip in the *cth* column between 1 and 5 pairs. So the short vectors in the first 2^{c-1} columns are extended with both a zero and a one bit in the *cth* column. The reader can see Table 4. ∎

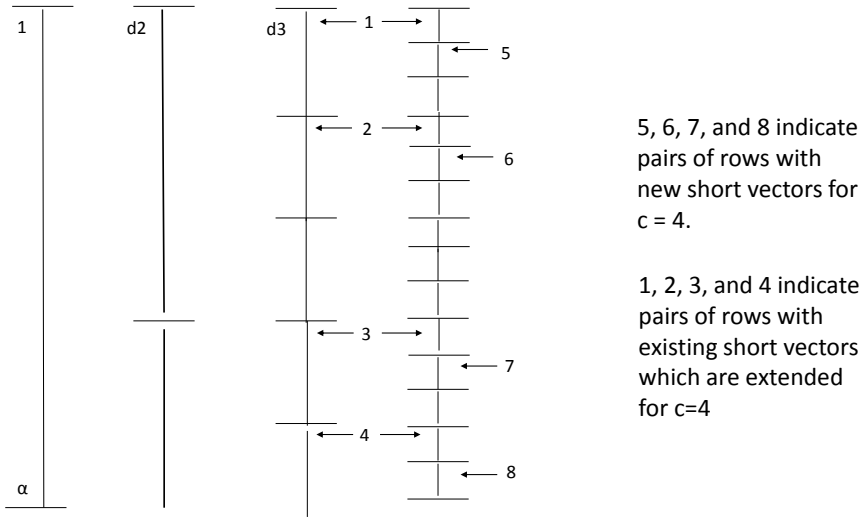

Fig. 2. Short Vectors Location in Permutation Table, $c = 4$, Horizontal Lines Indicate Start of New Permutation Runs (flips)

Definition 3. ***Perfect Somewhat Homomorphic Additive Scheme (No Zeroes 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.*

Start element parity	d_c *col. 1st row parity*	d_c *col. (1 + even) row parity*	d_c *column (1 + odd) row parity*	d_c *col. 2nd row parity*
d_c odd	odd	even (new run)	odd (new run)	even
d_c even	even	odd (new run)	odd (new run)	even

Table 4. Parities for cth column

Encryption: *N is a positive integer which is a power of 2. Given plaintext vector $\boldsymbol{r} = (r_1, \ldots, r_c)$ where $0 \leq r_i < N$, $1 \leq i \leq c$. We take $N = 2$ and the r_i as bits. 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 $d_i \geq (a+2)d_{i-1}$, $2 \leq i \leq c$, and $cd_c \leq \alpha$.*

We compute the vector $(e_1, \ldots, e_c)$ as follows: First we compute the bit vector $\boldsymbol{v} = (v_1, \ldots, v_c)$ where $v_i = d_i \bmod 2$, $1 \leq i \leq c$ if $r_1 = 1$, or $v_i = 0$, $1 \leq i \leq c$ if $r_1 = 0$. Then we perform Algorithm 1.

Algorithm 1 Encryption Steps for No Zeroes SWHAE

1: Initialize $\mathbf{w} = \mathbf{v}, f = 0$.
while $\mathbf{w} \neq \mathbf{r}$ **do**
i is the first bit position disagreement between $\mathbf{w}$ and $\mathbf{r}$.
Let $f = f + d_c/d_i$.
Set $w_i = r_i$. For $j > i$, set $n_j = d_j/d_i \bmod 2$ and $w_j = |n_j - w_j|$.
endwhile

Let

$$f_2 = f(m-1)/d_c + 1$$

if $r_1 = 1$ and $f(m-1)/d_c$ is even or $r_1 = 0$ and $f(m-1)/d_c$ is odd, or

$$f_2 = f(m-1)/d_c + 2$$

if $r_1 = 1$ and $f(m-1)/d_c$ is odd or $r_1 = 0$ and $f(m-1)/d_c$ is even.

Let $\boldsymbol{d} = (d_1, \ldots, d_c)$. Let $\boldsymbol{e} = f_2 \cdot \boldsymbol{d}$.

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 $e_i < m/(a+1)$ 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.*

Theorem 2. *Given the No Zeroes (NZ) SWHAE encrypt algorithm which outputs vector $\boldsymbol{e} = (e_1, \ldots, e_c)$. Then $e_i \bmod 2 = r_i, 1 \leq i \leq c$, where $\boldsymbol{r}$ is the plaintext input vector.*

Proof. Suppose $r_1 = 1$ and $v_i = d_i \bmod 2$, $1 \leq i \leq c$. Let $\boldsymbol{z}$ be obtained from $\boldsymbol{v}$ by changing the first bit, bit i, where $\boldsymbol{v}$ disagrees with $\boldsymbol{r}$. $\boldsymbol{r}$ and $\boldsymbol{z}$ agree on the first i bits.

$$\frac{(m-1)/d_i}{(m-1)/d_c} = d_c/d_i = f$$

is the number of flips of the d_c column that occur for the d_i column to flip once and cause the ith bit of $\boldsymbol{r}$ and $\boldsymbol{z}$ to be the same. d_j/d_i is the number of flips for column j and $d_j/d_i \bmod 2$ determines whether column j bit is the same or reversed. If reversed, then $w_j = |1 - v_j|$ and $w_j = |0 - v_j|$ otherwise. So Algorithm 1 finds the first vector $\boldsymbol{w}$ that is associated with f flips of column d_c in the permutation table. $\boldsymbol{w}$ agrees with $\boldsymbol{r}$ on the first i bits. Note that $\boldsymbol{w}$ is one vector of a pair of vectors and is located one or two rows after the last flip.

Applying the same argument inductively, we obtain the vector in the permutation table that occurs after f flips of column d_c and is associated with $\boldsymbol{r}$. $f(m-1)/d_c$ is the last row before the last flip of column d_c. If $f(m-1)/d_c$ is even, then vector $\boldsymbol{e}$ associated with $\boldsymbol{r}$ is located at row $f(m-1)/d_c + 1$ since $\boldsymbol{r}$ must be associated with an odd row vector in the permutation table. Otherwise $\boldsymbol{e}$ is located at row $f(m-1)/d_c + 2$. A similar argument applies for the case where $r_1 = 0$. ∎

Theorem 3. *The No Zeroes SWHAE scheme has perfect security and the adversary's advantage in the CPA security game is 0.*

Proof. In the CPA security game, the adversary makes encryption oracle queries prior to and after submitting the two challenge plaintexts P_1 and P_2 of its choice. Each encryption query uses a fresh uniform random key b. Each such key is chosen independently from all other keys. Thus the encryption queries yield no information on the key b that is used to create the challenge ciphertext. The ciphertext C is returned to the adversary.

We have

$$Pr[C|P_1] = Pr[C \text{ and } P_1]/Pr[P_1] = Pr[C]$$

and

$$Pr[C] = Pr[C \text{ and } P_2]/Pr[P_2] = Pr[C|P_2] = 1/(m-1)$$

since events C and P_1 are independent as are C and P_2. In other words, P_1 and P_2 result in unique e_1 and e_2 vectors in the No Zeroes SWAHE scheme and the possible ciphertexts for each vector are the same $m-1$ ciphertexts corresponding to the $m-1$ choices for b. This last statement holds since we are working in the same no zeroes permutation table. Thus the adversary advantage cannot exceed $1/2$ even though the adversary is unbounded. ∎

Remark 1. For No Zeroes SWHAE we have $m \geq c(a+2)^{c-1}(a+1)+1$.

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 $e \leq m/(a+1)$ and $e \bmod N = r_1 r_2 \bmod N$. [1]

Non-Interactive Scheme Our first scheme requires the client to send N elements $be_1 \bmod m, \ldots, be_N \bmod m$ for each multiplication gate where $e_i \bmod N = r_i$ and $r_1, \ldots, r_N$ are the N possible client plaintexts. The server computes all N^g possible results where g is the number of multiplication gates. The server selects one possible multiplication result, $be_i \bmod m$, for each multiplication gate. For each of the N^g possible results, the server returns the 2 multiplicand elements and the selected multiplication result for each gate, plus the output. Thus the server returns $3g+1$ elements for each of the N^g selection tuples. Note that the server does not know which of the N^g results contains correct multiplications. Only one of the results has all correct multiplications.

The client, upon receiving the server results, performs the following steps:

1. The client loops through all N^g results. Each result is a tuple of g triples:

$$(x_1, y_1, z_1), \ldots, (x_g, y_g, z_g)$$

2. For each tuple, the client computes $r_1 = b^{-1}x_1 \bmod m \bmod N$ and $r_2 = b^{-1}y_1 \bmod m \bmod N$ and checks if $r_1 r_2 \bmod N = b^{-1}z_1 \bmod m \bmod N$. If not, then the tuple is discarded.
3. Only 1 tuple will not be discarded and the output is the output element from this tuple.

This scheme can only be practical for a circuit with a small number of multiplication gates since the work scales exponentially with the number of multiplication gates.

[1] For $N = 2$, (a boolean circuit), a multiplication gate is an AND gate. Alternatively, we could replace the AND gate with a NAND gate in our descriptions below.

Interactive Scheme To preserve information-theoretic security, the client is assumed to know the number of multiplication gates in the circuit, and we will use the No Zeroes SWHAE scheme described above. Each multiplication gate adds up to 2 to the c parameter in the information-theoretic security analysis above.

The interactive scheme works as follows. Initially, the client executes No Zeroes SWHAE (NZ SWHAE) key generation and encryption with $c = c_1 + 2g$ where c_1 is the number of client inputs and g is the number of multiplication gates in the circuit. Each multiplication gate has a 1 bit and a 0 bit in $\boldsymbol{e}$. It sends the c_1 client inputs $be_1 \bmod m, \ldots, be_{c_1} \bmod m$ to the server, one element for each of the client's inputs to the circuit. The server can process addition gates on its own. The server leverages the client to process multiplication gates:

1. When the server has two multiplicands for a multiplication gate, it returns both elements, $be_1 \bmod m$ and $be_2 \bmod m$ to the client.
2. The client decrypts by multiplying by $b^{-1} \bmod m$ to obtain $e_1 \bmod m$ and $e_2 \bmod m$. The client then computes $r_1 = e_1 \bmod m \bmod N$ and $r_2 = e_2 \bmod m \bmod N$.
3. The client computes $r = r_1 r_2 \bmod N$; we take $r < N$. Assuming $N = 2$, the client selects an unused e component from $\boldsymbol{e}$ such that $e \bmod 2 = r$. We recall that $\boldsymbol{e}$ was computed during the encryption step of NZ SWHAE.
4. The client sends $be \bmod m$ to the server.
5. The server lets $be \bmod m$ be the output of the multiplication gate.

The server returns the final output to the client.

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. Optionally, the client may share the decrypted output with the server.

For multiplication, we assume the interactive scheme is being used. The client has perfect privacy per the results in Section 2; client privacy holds even if the server is fully malicious and computationally unbounded, except possibly if the client shares the decrypted output with the server. (A malicious server could learn additional information about client input values if the client returns the decrypted output to the server 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 (see Section 3 for details), 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.

Our security proof assumes $N = 2$.

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 NZSWHAE encrypt algorithm as described above. This set of elements is then partitioned into client input elements, server input elements, multiplication gate elements, and noise elements. The client knows the parities of the elements before creating them; half of the noise 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 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 3).

[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.

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

2-Party Protocol with Information-Theoretic Security for Client: Preprocessing

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 base elements used to create noise elements. The client selects m such that m and c satisfy the bound from the remark after Theorem 3. 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 NZ SWHAE encryption algorithm. We replace α in NZ 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: $b \leftarrow \mathbb{Z}_m$ where $gcd(b, m) = 1$. The secret key is (b, m). The key is used only once as described above.

3: The client creates uniform random noise elements from the c_1 elements discarding any duplicates until the client has n new elements. n is approximately $\log(m^{1/3})$. The random noise element exponents have the form $a_1e_1 + a_2e_2$ where a_1 and a_2 are integers and e_1, e_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**
  $h_i \leftarrow a_{i1}e_1 + a_{i2}e_2$; $a_{i1}, a_{i2} \leftarrow Z$, $e_1, e_2 \in C$.
  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

4: The client sends the noise elements to the server along with information to identify the parity of the exponents for each element:
send $z_i = bh_i \bmod m$, parity(h_i) to server, $1 \le i \le n$.
server stores z_i, parity(h_i), $1 \le i \le n$.

Algorithm 3 Input Processing Steps for 2-Party Protocol with Information-Theoretic Security for Client

2-Party Protocol with Information-Theoretic Privacy for Client: Processing Inputs Through Circuit

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. Each server input element e_i value is taken from the $\boldsymbol{e}$ vector created during the NZ SWHAE encryption algorithm.

2: Client randomizes the order of $y_i, 1 \leq i \leq c_s$, stores the y_i exponent parities separately, sends the y_i to the server, and sends a separate message with y_i exponent parities 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 NZ 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.

Algorithm 4 Steps for 2-Party Protocol with Information-Theoretic Security for Client: Multiplication, Addition Gates and Output

2-Party Protocol with Information-Theoretic Privacy for Client: Processing Multiplication, Addition, 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**
   $\mathbf{z} = (z_1, \ldots, z_n)$
   $\mathbf{s} \leftarrow \{0,1\}^n$
   $sum_i \leftarrow \mathbf{s} \cdot \mathbf{z} \bmod m$
   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 NZ 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 integer inputs are added to get the output.
10: An output o is summed with an even parity subset sum element s to get $s + o$ which is sent to the client. The client computes the plaintext output as $parity(b^{-1}(s + o) \bmod m)$.

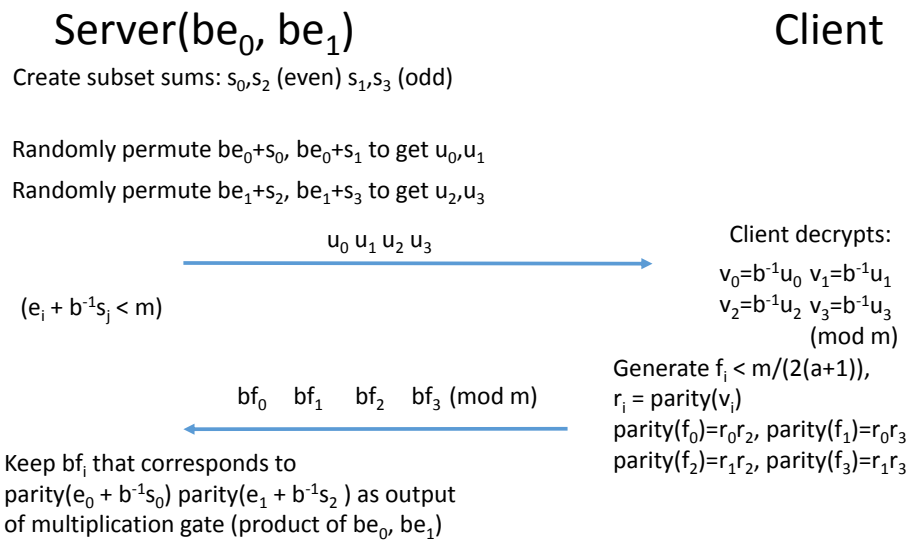

Fig. 3. Multiplication Gate Protocol

3.1 Proof of Security

Definition 4. *(Linear Subset Sum Problem): Given a circuit with a addition gates. For server security, we depend on the subset sum problem: we have a uniform random vector* $\boldsymbol{s}$ *where* $s_i \in \{0,1\}$, $T = \boldsymbol{s} \cdot b\boldsymbol{e} \bmod m$, $\boldsymbol{e} = (e_1, \ldots, e_n)$, $2 \leq e_i \leq m/(2n)$, $1 \leq i \leq n$. *The adversary is given* T, m, $\boldsymbol{e}$, *and* b, *and must find* $\boldsymbol{s}$.

Theorem 4. *Suppose the adversary has a polynomial time algorithm for the linear subset sum problem defined above. Then the adversary can use this algorithm as a subroutine to solve an integer subset sum problem in polynomial time.*

Proof. We are given integers $a_1, \ldots, a_n$, and $T = \sum_{i=1}^{n} s_i a_i$ where $\mathbf{s} = (s_1, \ldots, s_n)$, $s_i \in \{0,1\}$, $1 \leq i \leq n$. The adversary constructs the linear subset sum problem as follows: We select m such that $a_1, \ldots, a_n < m/(2n)$, $b \in \mathbb{Z}_m$ such that $gcd(b, m) = 1$. Then let $T_2 = bT \bmod m$. Thus T_2, $(a_1, \ldots, a_n)$, b is an instance of a linear subset sum problem and the adversary can obtain $\mathbf{s}$ in polynomial time. ∎

Theorem 5. *The 2-party protocol of Algorithms 2, 3, and 4 has server privacy given the hardness of the subset sum problem and perfect client privacy where server privacy also assumes the client is semi-honest.*

Proof. Our proof is one-sided simulation for the client; the client inputs are indistinguishable to the server by Theorem 3. Theorem 3 establishes perfect privacy

for the client given a malicious model server that does not receive decrypted output from the client. If the malicious server is to receive decrypted output from the client, the client first checks with the verifier (see below) to confirm that the server's protocol actions are correct. If so, then the client can send the decrypted output to the server.

Given that the client decrypts elements it receives from the server, collisions do not affect server privacy. The client ensures that both its client input elements and multiplication query response elements do not intersect with either the server input elements or the noise generator and noise elements. Theorem 3 establishes no leakage of the client parities (the server's knowledge of its own element parities does not affect the possible parities for the client elements).

We use a simulator argument for server privacy. The simulator is given the client inputs, output, and the security parameter m. It selects uniform random b such that $gcd(b, m) = 1$ and follows NZ SWHAE encrypt steps. It creates the noise elements as described in Algorithm 2.

It then selects c_s server input elements of the same form $be \bmod m$ as described in Algorithm 3. The (trial) server plaintext bits are random.

The simulator processes the plaintext client and trial server inputs through the circuit and obtains the trial client output.

If the trial client output is not equal to the actual client output, the simulator will change one or more of the random trial server inputs and recheck the resulting trial client output as follows: The simulator can mark all of the inputs to the gates in the circuit as not modifiable, for the inputs that derive solely from the client inputs. The simulator then works backwards through the circuit starting with flipping one of the bits that is an input to the output gate. This forces changes to each of the gate inputs on a path from the output gate to an input gate. If the path dead ends prior to reaching an input gate, the simulator has to backtrack and push the changes up another path to an input gate of the circuit. The process completes when the simulator reaches an input gate and flips a server input bit. The resulting modified server input bits are now used as the trial server input bits. The simulator can now encrypt these bits as in Algorithm 3 and recheck and repeat the steps until the correct client output is obtained. Scaling to larger circuits is accomplished by dividing into subcircuits to ensure the simulator is polytime (see Section 4.) In other words, we limit the size of the circuit that we apply our encryption scheme to.

∎

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 6. *A verifier with access to a run of the 2-party protocol of Algorithms 2, 3, and 4 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.

Remark 2. The server has a small probability chance of creating a collision between an element of its choosing and the correct sum element for a multiplication query to the client such that detection by the verifier fails. If the malicious server is lucky, then the multiplication gate will be incorrectly evaluated. The size of m is large compared to c and therefore the possible set of sums is highly unlikely to intersect with one of the multiplicands plus a subset sum.

4 Scaling the Additive Homomorphic Protocol and 2PC Schemes

For the 2PC scheme, the client will need to create $4g$ additional elements of the form $be \bmod m$, where g is the number of multiplication gates in the circuit. We recall $c = c_c + c_s + c_1 + 4g$. Thus m grows exponentially as the number of multiplication gates grows.

In order to limit the growth of m, the circuit can be partitioned into subcircuits so that each subcircuit has only a small 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 4 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. More precisely, all four of the client responses are returned to the server and the server selects the correct product for each of the gates. The correct products are the inputs G and H to subcircuit C_2.

Definition 5. *(Subcircuits): Given boolean circuit C. A subcircuit C_1 is a connected subgraph of C such that the output of every gate in C_1 can be computed using only the wires and gates of C_1. A terminal gate of C_1 is a gate G such that all of the output wires of G lead to gates outside of C_1, and G is not an output gate of C. These outputs are the terminal outputs of C_1 and are $C_1's$ inputs to other subcircuits of C. If subcircuit C_1 uses secret key b and modulus m,*

then the terminal outputs are encrypted in keys that correspond to the subcircuit for which they are inputs. The other subcircuits have separately generated secret keys. The other subcircuits also use separate sets of noise elements (see Algorithm 2). The non-terminal output inputs to the other subcircuits are computed as in Algorithm 3.

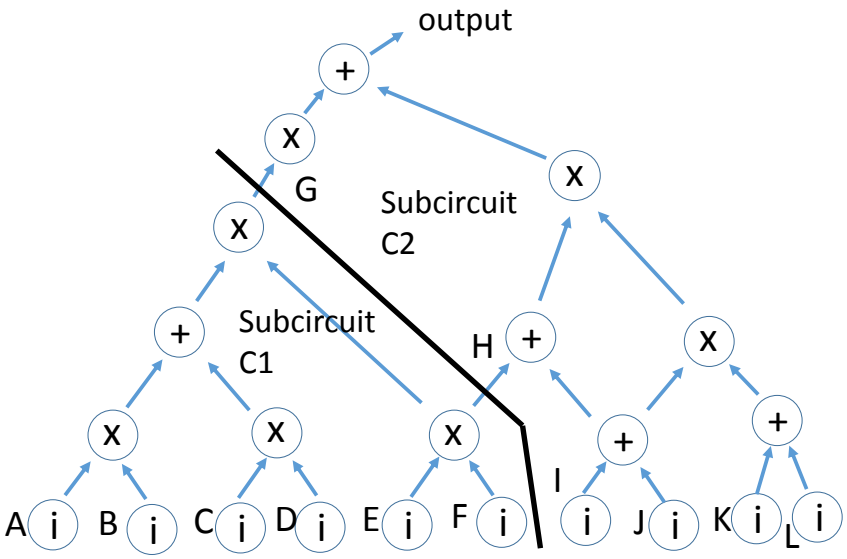

Fig. 4. An Example of Scaling the 2PC Protocol with Subcircuits

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

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 20âĂŞ24, 2023, Proceedings, Part I. pp. 328–359. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-38557-5_11, https://doi.org/10.1007/978-3-031-38557-5_11
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. 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
5. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. Cryptology ePrint Archive, Paper 2012/078 (2012), https://eprint.iacr.org/2012/078, https://eprint.iacr.org/2012/078
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. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Paper 2011/344 (2011), https://eprint.iacr.org/2011/344, https://eprint.iacr.org/2011/344
9. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. pp. 505–524. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
10. Brakerski, Z., Vaikuntanathan, V.: Lattice-based fhe as secure as pke. In: Proceedings of the 5th Conference on Innovations in Theoretical Computer Science. pp. 1–12. ITCS '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2554797.2554799, https://doi.org/10.1145/2554797.2554799
11. 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
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., Goldreich, O., Kushilevitz, E., M., S.: Private information retrieval. In: Proceedings 36th IEEE Symposium on Foundations of Computer Science. pp. 41–51 (1995)
14. 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)
15. 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

16. 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
17. 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
18. 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
19. 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
20. 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
21. Goldreich, O.: Foundations of cryptography volume ii basic applications. Cambridge University Press (2004)
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. 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
24. 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)
25. Rivest, R.L., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation. pp. 169–180 (1978)
26. 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
27. 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)
28. 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 Parity tables for $c = 3$

Row number	*1st Column*	d_2 *column*	d_3 *column*
1st row	1	1	1
2nd row	0	0	0
$\lfloor (m-1)/d_3 \rfloor + 1$ row	w	w	$1-w$
$\lfloor (m-1)/d_3 \rfloor + 2$ row	$1-w$	$1-w$	w
$\lfloor (m-1)/d_2 \rfloor + 1$ row	x	$1-x$	$1-x$
$\lfloor (m-1)/d_2 \rfloor + 2$ row	$1-x$	x	x
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 1$ row	y	$1-y$	y
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 2$ row	$1-y$	y	$1-y$

Table 5. Parities for 8 short rows when $c = 3$, $u = 1$, $v = 1$, w, x, and y are either 0 or 1.

Row number	*1st Column*	d_2 *column*	d_3 *column*
1st row	1	1	0
2nd row	0	0	0
$\lfloor (m-1)/d_3 \rfloor + 1$ row	w	w	1
$\lfloor (m-1)/d_3 \rfloor + 2$ row	$1-w$	$1-w$	1
$\lfloor (m-1)/d_2 \rfloor + 1$ row	x	$1-x$	0
$\lfloor (m-1)/d_2 \rfloor + 2$ row	$1-x$	x	0
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 1$ row	y	$1-y$	1
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 2$ row	$1-y$	y	1

Table 6. Parities for 8 short rows when $c = 3$, $u = 1$, $v = 0$, w, x, and y are either 0 or 1.

Row number	*1st Column*	d_2 *column*	d_3 *column*
1st row	1	0	0
2nd row	0	0	0
$\lfloor (m-1)/d_3 \rfloor + 1$ row	w	0	1
$\lfloor (m-1)/d_3 \rfloor + 2$ row	$1-w$	0	1
$\lfloor (m-1)/d_2 \rfloor + 1$ row	x	1	0
$\lfloor (m-1)/d_2 \rfloor + 2$ row	$1-x$	1	0
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 1$ row	y	1	1
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 2$ row	$1-y$	1	1

Table 7. Parities for 8 short rows when $c = 3$, $u = 0$, $v = 0$, $d_3/d_2 \bmod 2 = 0$, w, x, and y are either 0 or 1.

Row number	*1st Column*	d_2 *column*	d_3 *column*
1st row	1	0	0
2nd row	0	0	0
$\lfloor (m-1)/d_3 \rfloor + 1$ row	w	0	1
$\lfloor (m-1)/d_3 \rfloor + 2$ row	$1-w$	0	1
$\lfloor (m-1)/d_2 \rfloor + 1$ row	x	1	1
$\lfloor (m-1)/d_2 \rfloor + 2$ row	$1-x$	1	1
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 1$ row	y	1	0
$\lfloor (m-1)/d_2 + (m-1)/d_3 \rfloor + 2$ row	$1-y$	1	0

Table 8. Parities for 8 short rows when $c = 3$, $u = 0$, $v = 0$, $d_3/d_2 \bmod 2 = 1$, w, x, and y are either 0 or 1.