A Novel Secure Bit Decomposition Protocol

Author: Jay Patel
Mentor: Anish Mathuria

May 2023

Contents

1 Abstract

Many secure data analysis tasks, such as private comparison, secure clustering and classification, require efficient mechanisms to convert the intermediate encrypted integers into the corresponding encryptions of bits. Secure Bit Decomposition(SBD) is a cryptographic protocol that allows two parties to decompose an encrypted value into encryption of its individual bits without revealing the value itself.

$$\textbf{SBD: } E(x) \rightarrow (E(x_0), E(x_1), ..., E(x_{m-1}))$$

Where $E(x)$ is the encryption of integer x and $(E(x_0), E(x_1), ..., E(x_{m-1}))$ are encrypted values of individual bits of x. Here m, x_0 and x_m correspondingly represent the bit-length, LSB and MSB of x.

In our study we have analysed an existing secure bit-decomposition protocol SBDp[1], which can be used to get encryption of individual bits of x if we have been provided with $E(x)$, where x is encrypted using public key additive homomorphic encryption schemes. This protocol guarantees security as per the semi-honest security definition of secure multi-party computation (MPC) and is also very efficient. This protocol always returns the correct result, however, it is probabilistic in the sense that the correct result can be generated in the first run itself with very high probability. The computation time of the proposed protocol grows linearly with the input domain size in bits. But in the problems like private comparison where we only need a single encryption (encryption of MSB) this can be computationally quite expensive. That's why we have proposed another protocol which gives the encryption of MSB in constant time. We theoretically analyze the complexity of the proposed protocol with the existing method in detail.

2 Private Comparison

Private Sorting is a cryptographic protocol in which we have been given with an array of $A = [E(X_1), E(X_2), ..., E(X_n)]$ and we want to reorder the elements of the array in such a way that the decryption array D that we get from decrypting individual elements of A, is in ascending order.

Private Comparison is a cryptographic protocol that can be used as a primitive in Private Sorting protocol. It allows two parties to get the comparison results of two encrypted values. For different privacy requirements there can be different variants Private Comparison protocol. Below we have stated our privacy requirements for the Private Comparison protocol and how SBD can be used as a building block in this private comparison protocol.

2.1 Problem Statement

There are 2 parties Alice and Bob. Alice has public key p_k and secret key s_k. Bob has $p_k, E(z)$ and $E(y)$. Bob wants to know $E(R) = E(z \geq y)$. Which implies

that R is 1 if $z \geq y$ and R is 0 if $z < y$. While in this process, Bob should not know any information related to z, y and R. Given $m = \max(\text{bitlen}(z), \text{bitlen}(y)) + 1$, if we can compute $x = 2^{m-1} + z - y$. Then $\text{MSB}(m^{th} \text{ bit} = x_{m-1})$ of x will give the comparison result R. If given $E(z)$ and $E(y)$, if we can compute $E(x)$, then an SBD protocol such as SBDp can produce $E(x_{m-1})$. That will be the result of private comparison. But the correctness of SBDp requires us to compute all the encrypted bits, which is computationally expensive. That's why we will be proposing an alternate SBD protocol which can give the comparison result $R = x_{m-1}(\text{MSB of } x)$ in constant amount of computational cost.

3 Preliminaries

3.1 Notations

Here are some notations that are going to be used through out this paper.

$E(x)$	Encryption of x
$D(x)$	Decryption of x
m	Bit length of x
K	Encryption key size in bits
p_k	public key
s_k	private key
x_0	Least Significant Bit(LSB)
x_{m-1}	Most Significant Bit(MSB)

3.2 Security Definition

This paper adopts the well-known security definition of semi-honest (also referred to as honest-but-curious) model from the field of secure multi-party computation (MPC). In a semi-honest protocol, the parties involved in the computation are assumed to follow the protocol correctly, but may attempt to learn as much as possible about the other parties' private inputs by analyzing the messages they receive during the execution of the protocol. In other words, semi-honest parties are honest in their actions, but curious about the other parties' data.

3.3 Paillier Cryptosystem

The Paillier cryptosystem[2] is an additive homomorphic and probabilistic asymmetric encryption scheme. The problem of computing n-th residue classes is believed to be computationally difficult. The decisional composite residuosity assumption is the intractability hypothesis upon which this cryptosystem is based. The scheme is an additive homomorphic cryptosystem; this means that, given only the public key and the encryption of m_1 and m_2, one can compute the encryption of $m_1 + m_2$.

3.3.1 Key generation

1. Choose two large prime numbers p and q randomly and independently of each other such that $\gcd(pq, (p-1)(q-1)) = 1$. This property is assured if both primes are of equal bit-length.
2. Compute $N = pq$ and $\lambda = \text{lcm}(p-1, q-1)$.
3. Select random integer g where $g \in \mathbb{Z}^*_{N^2}$.
4. Ensure N divides the order of g by checking the existence of the following modular multiplicative inverse: $\mu = (L(g^\lambda \mod N^2))^{-1} \mod n$, where L is defined as $L(x) = \frac{x-1}{N}$. (Note: The notation a/b denotes the quotient of a divided by b.)

- Public key: (N, g)
- Private key: (λ, μ)

By using p, q of equivalent bit-length, a simpler variant of the above key generation steps would be to set

1. $g = N + 1, \lambda = \phi(N)$
2. $\mu = \phi(N)^{-1} \mod N$, where $\phi(N) = (p-1)(q-1)$

3.3.2 Encryption

1. Let m be a message to be encrypted where, $0 \leq m < N$
2. Select random r where, $0 < r < Nn$ and $\gcd(r, N) = 1$
3. Compute ciphertext as: $c = g^m \cdot r^N \mod N^2$

3.3.3 Decryption

1. Let c be the ciphertext to decrypt, where $c \in \mathbb{Z}^*_{N^2}$
2. Compute plaintext as: $m = L(c^\lambda \mod N^2) \cdot \mu \mod N$

Let E be the encryption function with public key p_k given by (N, g) and D be the decryption function with secret key s_k given by λ (that is, the knowledge of the factors of N). Here, $N = p * q$ of bit length K (Paillier encryption key size); where p and q are large primes of similar bit length, and generator $g \in \mathbb{Z}^*_{N^2}$. For any given $y, z \in \mathbb{Z}_N$, the Paillier cryptosystem exhibits the following properties:

a. Homomorphic Addition: $E(y + z) \leftarrow E(y) * E(z) \bmod N^2$;

b. Homomorphic Multiplication: $E(z * y) \leftarrow E(y)^z \bmod N^2$;

c. Semantic Security: The encryption scheme is semantically secure. Means given a set of ciphertexts, an adversary cannot deduce any additional information about the plaintext.

4 SBDp

In this section, we present probabilistic SBDp protocol. In addition, we analyze its security, correctness and how it's correctness requires us to compute encryption of all the bits of x instead of the encryption of MSB($E(x_{m-1})$) only.

We assume that Bob holds a Paillier encryption of integer x i.e., $E(x)$ such that x is not known to both Alice and Bob, where $0 \leq x < 2^m$. The goal of SBD_p is to compute $E(x_0), \ldots, E(x_{m-1})$ without revealing either the value of x or any of x_i 's to Alice and Bob. At the end of the SBD_p protocol, only Bob will be knowing the encryptions of x_i 's, for $0 \leq i \leq m-1$. The proposed SBD_p protocol mainly involves the following two stages:

- **Stage 1** - Computing Encrypted Bits of x : In this Stage, Alice and Bob jointly compute encryptions of the individual bits of x in an iterative fashion. At the end of this stage, only Bob knows $E(x_i)$'s, for $0 \leq i \leq m-1$.
- **Stage 2** - Secure Verification of Result: Following from Stage 1, Alice and Bob securely verify whether the result from Stage 1 is correct or not. Only the output of the verification stage is revealed to Alice and Bob. If the result from Stage 1 is correct, then both parties terminate the protocol. Otherwise, the Stage 1 is repeated.

Algorithm 1 SBDp, Algorithm 1

Require: Bob has Paillier encrypted value $E(x)$, where x is not known to both parties and $0 \leq x < 2^m$; (Note: The public key (g, N) is known to both Alice and Bob whereas the secret key s_k is known only to Alice)

1: $l \leftarrow 2^{-1} \bmod N$
2: $T \leftarrow E(x)$
3: **for** $i = 0 \rightarrow m-1$ **do**
4: $\quad E(x_i) \leftarrow \text{Encrypted}_{\text{LSB}}(T, i)$
5: $\quad Z \leftarrow T * E(x_i)^{N-1} \bmod N^2$
6: $\quad$ ▷ update T with encrypted value of q_i
7: $\quad T \leftarrow Z^l \bmod N^2$
8: **end for**
9: $\gamma \leftarrow \text{SVR}(E(x), \langle E(x_0), E(x_1), ..., E(x_{m-1}) \rangle)$
10: **if** $\gamma = 1$ **then**
11: $\quad$ return
12: **else**
13: $\quad$ go to Step 2
14: **end if**

The overall steps involved in SBD_p are highlighted in Algorithm 1. Now, we explain the steps involved in each stage in detail.

Stage 1 - Computing Encrypted Bits of x : Initially, Bob computes l as multiplicative inverse of 2 modulo N and assigns $E(x)$ to T. Then, in an iterative fashion, Bob and Alice jointly compute the encrypted value $E(x_i)$, for $0 \leq i \leq m-1$, by updating T. Briefly, in the first iteration, Bob computes $E(x_0)$ (the encryption of least-significant bit of x) using T. Then he updates T with the encryption of quotient q_0 (which is equivalent to $\lfloor \frac{x}{2} \rfloor$). This process is continued for m iterations. Before explaining the steps of Stage 1, we first discuss its basic idea which follows from Observations 1 and 2.

Observation 1: For any given x, let $y = x + r \bmod N$, where r is a random number from Z_N. Here the relation between y and r depends on whether $x + r \bmod N$ leads to an overflow (i.e., $x + r$ is greater than N) or not.

$$y = \begin{cases} x + r & \text{if there is no overflow} \\ x + r - N & \text{otherwise} \end{cases}$$

Observation 2: For any given $y = x + r \bmod N$, where N is odd, the following property regarding the LSB of x (i.e., x_0) will always hold:

$$x_0 = \begin{cases} \lambda_1 \oplus \lambda_2 & \text{if } r \text{ is even} \\ 1 - (\lambda_1 \oplus \lambda_2) & \text{otherwise} \end{cases}$$

$$\lambda_1 = \begin{cases} 1 & \text{overflow} \\ 0 & \text{No overflow} \end{cases}$$

$$\lambda_2 = \begin{cases} 1 & y \text{ is odd} \\ 0 & y \text{ is even} \end{cases}$$

Here λ_1 denotes whether an overflow occurs or not, and λ_2 denotes whether y is odd or not. Observe that $1 - (\lambda_1 \oplus \lambda_2)$ is the negation of bit $\lambda_1 \oplus \lambda_2$. Also note that N is always odd in the Paillier cryptosystem.

EXAMPLE 1. Consider an even integer x, i.e., $x_0 = 0$. Without loss of generality, suppose r be an odd integer such that $x + r \bmod N$ causes an overflow. Then, $y = x + r \bmod N$ is always an even integer. That is $\lambda_2 = 0$. Also, $\lambda_1 = 1$ since there is an overflow. As r is odd, following from Observation 2, we have $1 - (\lambda_1 \oplus \lambda_2) = 1 - (1 \oplus 0) = 0 = x_0$.

By using the properties of Observation 2, Stage 1 computes the encryption of least significant bit of T by iteratively updating T to the encryption of corresponding quotient. The main steps involved in Stage 1 are shown as steps 2 to 8 in Algorithm 1. Stage 1 utilizes a sub-protocol (denoted by $\text{Encrypted}_{\text{LSB}}$) to compute the encryption of LSB from current T. The overall steps involved in the $\text{Encrypted}_{\text{LSB}}$ protocol are shown in Algorithm 2.

Algorithm 2 SBDp, Algorithm 2

Require: Bob has T from current iteration i

1: **Bob:**
   (a). $Y \leftarrow T * E(r) \bmod N^2$, where r is random in $\mathbb{Z}_N$
   (b). Send Y to Alice
2: **Alice:**
   (a). Receive Y from Bob
   (b). $y \leftarrow D(Y)$
   (c). **if** y is even **then** $\alpha \leftarrow E(0)$
   else $\alpha \leftarrow E(1)$
   (d). Send α to Bob
3: **Bob:**
   (a). Receive α from Alice
   (b). **if** r is even **then** $E(x_i) \leftarrow \alpha$
   else $E(x_i) \leftarrow E(1) * \alpha^{N-1} \bmod N^2$
   (c). return $E(x_i)$

To start with, in the first iteration, Bob randomizes the value of T (note that initially $T = E(x)$) by computing $Y = T * E(r) \bmod N^2$, and sends Y to Alice [1]. Upon receiving, Alice decrypts it to get $y = D(Y)$, where $y = x + r \bmod N$. If y is odd, Alice computes $\alpha = E(1)$, else she computes $\alpha = E(0)$, and sends it to Bob. Observe that $\alpha = E(\lambda_2)$. Although it is possible to compute $E(\lambda_1)$, but it would be a computationally expansive task. Therefore, in our SBD_p protocol we decided to use a probabilistic assumption. That is, Bob simply assumes $\lambda_1 = 0$ (no overflow). Though Bob assumes no overflow, $y = x + r \bmod N$ can still have an overflow which depends on the actual values of x and r. In the later parts of this section, we show that for many practical applications, the above probabilistic assumption is reasonable.

Once Bob receives α from Alice, he computes $E(x_i)$ depending on whether r is even or odd as below.

- If r is even, then $E(x_0) = \alpha = E(\lambda_2)$.
- Else $E(x_0) = E(1) * \alpha^{N-1} \bmod N^2 = E(1 - \lambda_2)$

Since λ_1 is assumed to be 0 , following from Observation 2, we have $\lambda_1 \oplus \lambda_2 = \lambda_2$. Also, note that $N-1$ is equivalent to -1 under $\mathbb{Z}_N$. After this, Bob updates T to $E(q_0) = E\left(\lfloor \frac{x}{2} \rfloor\right)$ by performing following homomorphic additions:

- $Z = T * E(x_0)^{N-1} \bmod N^2 = E(x - x_0)$
- $T = Z^l \bmod N^2 = E\left((x - x_0) * 2^{-1}\right) = E(q_0)$

where $l = 2^{-1} \bmod N$. The main observation is that $x - x_0$ is always a multiple of 2 ; therefore, $(x - x_0) * 2^{-1}$ always gives the correct quotient under

[1] In Paillier's cryptosystem, ciphertext multiplications are followed by modulo N^2 operation so that the resulting ciphertext is in $\mathbb{Z}_{N^2}$.

$\mathbb{Z}_N$. The above process is continued iteratively such that in iteration i, Bob computes $E(x_i)$ and updates T to $E(q_i)$, for $0 \leq i \leq m-1$

Stage 2 - Secure Verification of Result: Since Bob assumes $\lambda_1 = 0$ during each iteration of Stage 1 , there is a possibility for the wrong outcome. Though this probability is very low for many practical applications, as will be shown later, it is better to provide a mechanism to mitigate this effect. Along this direction, Stage 2 securely verifies whether the result of Stage 1 is correct and takes appropriate steps to compute the correct result (only in the case of wrong result from Stage 1).

The overall steps involved in Stage 2 of SBD_p are highlighted as steps 9 to 14 in Algorithm 1. As shown in step 9 of Algorithm 1, Stage 2 utilizes a sub-protocol that performs the secure verification of the result (denoted by SVR). The main steps involved in SVR are given in Algorithm 3.

Algorithm 3 SBDp, Algorithm 3

Require: Bob has T from current iteration i

1: **Bob:**
   (a). $U \leftarrow \prod_{i=0}^{m-1} (E(x_i))^{2^i} \bmod N^2$
   (b). $V \leftarrow U * E(x)^{N-1} \bmod N^2$
   (c). $W \leftarrow V^{r'} \bmod N^2$, where r' is random in $\mathbb{Z}_N$
   (d). Send W to Alice

2: **Alice:**
   (a). Receive W from Bob
   (b). **if** $D(W) = 0$ **then** $\gamma \leftarrow 1$
   else $\gamma \leftarrow 0$
   (c). Send γ to Bob

Initially, Bob computes the encrypted integer corresponding to the encrypted bits computed in Stage 1 and performs the following homomorphic operations.

- $U = E\left(x_0 + \ldots + 2^{m-1} * x_{m-1}\right)$
- $V = E\left(x_0 + \ldots + 2^{m-1} * x_{m-1} - x\right)$
- $W = V^{r'} = E\left(r' * \left(x_0 + \ldots + 2^{m-1} * x_{m-1} - x\right)\right)$

After this, Bob sends W to Alice. Upon receiving W, Alice decrypts it and sets γ to 1 (denoting the result from Stage 1 is correct) if $D(W) = 0$, else γ is set to 0 (denoting the result is incorrect), and sends γ to Bob. Observe that $\gamma = 0$ iff $x = x_0 + \ldots + 2^{m-1} * x_{m-1}$, i.e., result of Stage 1 is correct. Note that $D(W)$ is either 0 or random; therefore, the above process does not reveal any information about x and x_i 's to Alice. Finally, if $\gamma = 1$, Bob terminates the SBD_p protocol implying the result of Stage 1 is correct. Else, Bob initiates Stage 1 of SBD_p (i.e., go to Step 2 of Algorithm 2) and the above process is

repeated until the result is correct.

EXAMPLE 2. Suppose $x = 5$ and $m = 3$. We show various intermediate results during the execution of Stage 1 in the proposed protocol. Initially T is set to $E(5)$. We assume that r is even and there is no overflow. Note that r is different in each iteration.

Iteration 0:

$$y = 5 + r \bmod N = \text{ an odd integer}$$
$$E(x_0) = \alpha = E(1)$$
$$T = E\left((5-1) * 2^{-1}\right) = E(2)$$

Iteration 1:

$$y = 2 + r \bmod N = \text{ an even integer}$$
$$E(x_1) = \alpha = E(0)$$
$$T = E\left((2-0) * 2^{-1}\right) = E(1)$$

Iteration 2:

$$y = 1 + r \quad \bmod N = \text{ an oddinteger}$$
$$E(x_2) = \alpha = E(1)$$
$$T = E\left((1-1) * 2^{-1}\right) = E(0)$$

The output of Stage 1 is $\langle E(1), E(0), E(1)\rangle$.

4.1 Correctness

We emphasize that the final encrypted bits computed from the SBD_p protocol are always correct. This is because of the secure verification step (i.e., Stage 2) in SBD_p which makes sure that the protocol terminates only if the output of Stage 1 is correct. It is possible for the SBD_p protocol to generate the correct output after many runs of Stage 1 & 2 (which depends on x and random r chosen in each iteration of Stage 1). However, the proposed protocol produces the correct output in the first run of Stage 1 and 2 for many practical applications. We theoretically prove this below.

In general, many secure data analysis tasks such as secure clustering and classification has m much smaller than K. During Stage 1 , in each iteration, r can take any value in $\mathbb{Z}_N$. We observe that if $r \in [N - 2^m, N)$, only then the corresponding computed (encrypted) bit can be wrong (due to overflow). That is, the number of possible values of r that can give rise to error are 2^m. Since we have N number of possible values for r, the probability for producing wrong bit is $\frac{2^m}{N} \approx \frac{1}{2^{K-m}}$. Therefore, the probability for computing the encryption of x_i correctly is approximately $1 - \frac{1}{2^{K-m}}$. This probability remains same for all the bits since r is chosen independently in each iteration. Hence, the probability for Stage 1 to compute the correct output is given by:

$$\left(1 - \frac{1}{2^{K-m}}\right)^m \approx e^{-\frac{m}{2^{K-m}}}$$

In general, for many real-world applications, m can be at most 100. Therefore, for 1024-bit key size (i.e., $K = 1024$), the probability for SBD_p to produce correct output in the first run is approximately $e^{-\frac{100}{2^{924}}} \approx 1$. Hence, for practical domain values of x, with a probability of almost 1 , the SBD_p protocol gives the correct output in the first run itself. Based on the above analysis, it is clear that Stage 2 is not required in practice.

4.2 Security Analysis

In the SBD_p protocol, the communication between Alice and Bob occurs both in Stage 1 and 2 . During Stage 1 , in each i^{th} iteration, Bob randomizes current T (from step 1(a) of Algorithm 2) and sends it to Alice. Therefore, the decryption of Y by Alice (from step 2(b) of Algorithm 2) yields a random value y which is uniformly distributed in $\mathbb{Z}_N$. Thus, no information about x is revealed to Alice. Also, depending on whether y is even or odd, Alice sends $\alpha = E(0)$ or $E(1)$ to Bob. Here, the ciphertext α is uniformly random in $\mathbb{Z}_{N^2}$ and does not provide any information regarding the corresponding plaintext to Bob due to the semantic security of Paillier cryptosystem. Note that, after computing $E\left(x_{i-1}\right)$ in the i^{th} iteration, the process of updating T is done locally by Bob, for $1 \leq i \leq m$. Hence, it is clear that no information related to x is revealed to Alice and Bob during Stage 1 of SBD_p.

During Stage 2, Bob sends $W = V^{r'} \bmod N^2$ to Alice (step 1(d) of Algorithm 3), where r' is a random number in $\mathbb{Z}_N$ known only to Bob. Since the decryption of W by Alice gives either 0 or a uniformly random value in $\mathbb{Z}_N$, no information is revealed to Alice and Bob except whether the output of Stage 1 is correct or not. Therefore, Stage 2 is secure under the assumption (which is also practical) that the output γ can be revealed to both parties. We emphasize that revealing γ does not leak any information regarding x to Alice and Bob.

4.3 Complexity Analysis

For the SBD_p protocol, Stage 1 involves m number of iterations, where each iteration requires one round of communication between Alice and Bob. Whereas, Stage 2 requires one round of communication between Alice and Bob. As shown earlier, for many practical applications, the probability of getting the correct result in the first instantiation of the SBD_p protocol is very high. Therefore, we assume that Stage 1 and 2 are run only once. Thus, the round complexity of SBD_p is bounded by $O(\text{ m})$.

During Stage 1 of SBD_p, in each iteration, Bob has to randomize current T which requires one encryption and one homomorphic addition operations. Alice further performs one decryption and one encryption operations (at step 2(b) and 2(c) of Algorithm 2). Also, if r is odd, Bob has to perform one exponentiation

and one homomorphic addition operations (at step 3(b) of Algorithm 2). In addition, Bob has to perform two exponentiation and one homomorphic addition operations while updating T in each iteration. We emphasize that the values of $l = 2^{-1} \bmod N$ and $E(1)$ (step 3(b) of Algorithm 2) are pre-computed and can be re-used in each iteration. Therefore, Stage 1 requires $3m$ encryptions (assuming that encryption and decryption under Paillier take almost similar time), $3m$ exponentiation, and $3m$ homomorphic addition operations. In general, the cost of $3m$ homomorphic additions is negligible compared to the cost of $3m$ exponentiations. Therefore, the computation cost of Stage 1 is bounded by $3m$ encryptions and $3m$ exponentiations.

Furthermore, Stage 2 of SBD_p requires one decryption, $m+2$ exponentiation, and m homomorphic addition operations. Hence, the total computation cost of SBD_p (one instantiation) is roughly $3m + 1$ encryptions and $4m + 2$ exponentiations.

The communication complexity of our protocol entirely depends on Stage 1. During Stage 1, in each iteration, Bob sends encrypted value Y to Alice. Also, Alice sends an encrypted value α to Bob. Since we have m iterations, the total communication complexity of SBD_p is bounded by $O(m * K)$ bits.

5 SBDv_0

In this section, we present our modified probabilistic SBDv_0 protocol which is a variant of original SBDp. In addition, we analyze its security, correctness and how by doing some more modifications in it we can get SBDv_1 which allows us to compute MSB of x without the need of computing all the encrypted bits of x.

We assume that Bob holds a Paillier encryption of integer x i.e., $E(x)$ such that x is not known to both Alice and Bob, where $0 \leq x < 2^m$. The goal of SBDv_0 is to compute $E(x_{m-1}), \ldots, E(x_0)$ without revealing either the value of x or any of x_i 's to Alice and Bob. At the end of the SBDv_0 protocol, only Bob will be knowing the encryptions of x_i 's, for $0 \leq i \leq m-1$. The proposed SBDv_0 protocol mainly involves the following three phases:

- **Phase 1** - Setting the x_0 to 1 : In this phase, Alice and Bob jointly set the LSB of x to 1 and update the value of $E(x)$ accordingly.
- **Phase 2** - Computing Encrypted Bits of x : In this phase, Alice and Bob jointly compute encryptions of the individual bits of x in an iterative fashion. At the end of this stage, only Bob knows $E(x_i)$'s, for $0 \leq i \leq m-1$.
- **Phase 3** - Secure Verification of Result: Following from phase 2, Alice and Bob securely verify whether the result from phase 2 is correct or not. Only the output of the verification stage is revealed to Alice and Bob. If the result from phase 2 is correct, then both parties terminate the protocol. Otherwise, the phase 2 and 3 are repeated.

Now, we explain the steps involved in each phase in detail.

Phase 1 - Setting the x_0 to 1 : Overall steps of the phase 1 are shown in the algorithm A.

Algorithm 4 SBDv_0, Algorithm A

Require: Bob has Paillier encrypted value $E(x)$, where x is not known to both parties and $0 \leq x < 2^m$; (Note: The public key (g, N) is known to both Alice and Bob whereas the secret key s_k is known only to Alice)

1: **Bob:**
$r = \text{random}(1, N - 2^m)$
$E(y) = E(r) * E(x) \bmod N$
Send $Y = E(y)$ to Alice

2: **Alice:**
Receives Y from Bob
$y = D(Y)$
if $y\%2 == 0$ **then** $\lambda_2 = 0$
else $\lambda_2 = 1$
return $E(\lambda_2)$

3: **Bob:**
Receives $E(\lambda_2)$ from Alice
if $r\%2 == 0$ **then** $E(\lambda) = E(1 - \lambda_2)$
else $E(\lambda) = E(\lambda_2)$
$E(x) = E(x + \lambda)$

Initially, Bob randomizes the value x that is being sent to Alice by adding a random value $r \in (1, N - 2^m)$, here r is chosen such that there is no overflow(if $y = x + r$ then $y < N$).

Observation 3: When $x + r < N$ for any $y = x + r \bmod N$, the following property regarding $\lambda = 1 - x_0$ always hold:

$$\lambda = \begin{cases} 1 - \lambda_2 & \text{if } r \text{ is even} \\ \lambda_2 & \text{otherwise} \end{cases}$$

$$\lambda_2 = \begin{cases} 1 & y \text{ is odd} \\ 0 & y \text{ is even} \end{cases}$$

Here λ_2 denotes whether y is odd or not. Observe that assigning $E(x + \lambda) = E(x + 1 - x_0)$ to $E(x)$ will set the LSB of original value x to 1.

EXAMPLE 3. Consider an even integer x, i.e., $x_0 = 0$. Without loss of generality, suppose r be an even integer such that $x + r < N$. Then, $y = x + r \bmod N$ will be an even integer. That is $\lambda_2 = 0$. As r is even, following from Observation 3, we have $\lambda = 1 - \lambda_2 = 1 - 0 = 1 - x_0$. By adding λ to x we

would set it's LSB to 1.

Phase 2 - Computing Encrypted Bits of x : Throughout phase 2, we would be assuming that there is no overflow in the calculations(summations, multiplications, subtractions of encrypted values). Overall steps of the phase 2 are shown in the algorithm B.

Algorithm 5 SBDv_0, Algorithm B

1: $l \leftarrow 2^m \bmod N$
2: **for** $i = m-1 \rightarrow 1$ **do**
3: **Bob:**
$l \leftarrow l/2$
$r_1 \leftarrow \text{random}(1, N-1), r_2 \leftarrow \text{random}(1, N-1)$
$E(T_1) \leftarrow E((x + r_1) * r_2)$, $E(T_2) \leftarrow E((l + r_1) * r_2)$
$r_3 \leftarrow \text{random}(1, r_2 - 1)$, $E(T_2) \leftarrow E(T_2 + r_3)$
B $\leftarrow$ random(0, 1)
if B == 1 : Send $E(T_1), E(T_2)$ to Alice
else : Send $E(T_2), E(T_1)$ to Alice
4: **Alice:**
Receives $E(Z_1), E(Z_2)$ from Bob
$z_1 \leftarrow D(Z_1), z_2 \leftarrow D(Z_2)$
if $z_1 > z_2 : \gamma = 1$
else : $\gamma = 0$
return $E(\gamma)$
5: **Bob:**
Receives $E(\gamma)$ from Alice
if B == 1 : $E(x_i) \leftarrow E(\gamma)$
else : $E(x_i) \leftarrow E(1 - \gamma)$
$E(x) \leftarrow E(x - x_i * l)$
6: **end for**
7: $E(x_0) \leftarrow E(x - \lambda)$

Initially, Bob computes l as 2^m modulo N. Then, in an iterative fashion, Bob and Alice jointly compute the encrypted value $E(x_i)$, for $1 \leq i \leq m-1$, by updating $E(x)$. Through phase 1 Bob has set the LSB of x to 1(i.e. $x_0 = 1$). From this we can come to below observation for the first iteration of for loop, where $i = m - 1$:

Observation 4: Here $l = 2^{m-1}$, $r_1$and r_2 are random numbers, r_3 is a random number in range $(1, r_2 - 1)$, $T_1 = ((x + r_1) * r_2), T_2 = ((l + r_1) * r_2) + r_3$.

$$\textbf{if } x_{m-1} == 1:$$
$$x \geq l + 1$$
$$((x + r_1) * r_2) \geq ((l + r_1) * r_2) + r_2$$
$$((x + r_1) * r_2) > ((l + r_1) * r_2) + r_3$$
$$T_1 > T_2$$
$$\textbf{if } x_{m-1} == 0:$$
$$x < l$$
$$((x + r_1) * r_2) < ((l + r_1) * r_2)$$
$$((x + r_1) * r_2) < ((l + r_1) * r_2) + r_3$$
$$T_1 < T_2$$

Comparison of T_1 and T_2 would give the value of x_{m-1}(i.e. if $T_1 > T_2$ then $x_{m-1} = 1$ else $x_{m-1} = 0$). If B = 1, Bob will send $E(T_1), E(T_2)$, else send $E(T_2), E(T_1)$, this swapping is done to mask(hide information) the output, so that Alice would not be able find the value of x_{m-1}. In the first iteration of for loop, Bob computes $E(x_{m-1})$ (the encryption of MSB of x) using $E(x)$. After computing $E(x_{m-1})$ he updates $E(x)$ with $E(x - x_{m-1} * 2^{m-1})$, basically Bob sets the MSB of x to 0 and updates $l \leftarrow l/2$ in the next iteration. From this updated values of $E(x)$ and l, Bob will be able to compute x_{m-2}. In similar way he will compute x_i's for $i = (m-1)$ to 1.

After all the iterations, all the bits of x would be set to 0,except the LSB of x. In phase 1 we had updated x by setting the LSB of x to 1 by adding λ to it. Subtracting λ from the x will give us the LSB(x_0) of original value x.

As we have mentioned before that in calculations of phase 2 we have assumed that there would be no overflow. But in actual calculations overflow might occur, in those situation comparison result of T_1 and T_2 might not give us correct result for MSB of x or any of the intermediate bits. Phase 3 checks if the computed encryption bits are correct or not. Phase 3 is just the stage 2(Algorithm 3) of previously mentioned SBDp algorithm.

5.1 Why we need Phase 1 ?

Without Phase 1 our algorithm will give wrong results of x_i's for all the values of x when x is even. We can understand it from the below example.

Example 4:(Without stage 1)

1) $x = 4, N = 713 = 23 * 31$,Binary representation of $x = 100, x_2 = 1, x_1 = 0, x_0 = 0, m = 3$.

I1: $x = 4, r_1 = 3, r_2 = 9, r_3 = 5, l = 2^{3-1} = 4, B = 1$

$$
\begin{aligned}
T_1 &= (4+3) * 9 \bmod 713 \\
T_1 &= 63 \\
T_2 &= (4+3) * 9 \bmod 713 \\
T_2 &= 63 + 5 \bmod 713 \\
T_2 &= 68 \\
T_1 &< T_2 \\
\gamma &= 0 \\
x_2 &= 0
\end{aligned}
$$

In the 1^{st} iteration we get wrong result for x_2 therefore our implementation gives wrong answer for MSB.

Example 4:(With Phase 1)

P1: $x = 4, N = 713 = 23 * 31, r = 42$ Binary representation of $x = 100$

$$
\begin{aligned}
y &= 42 + 4 \\
y &= 46 \\
\lambda_2 &= 0 \\
\lambda &= 1 - \lambda_2 \\
\lambda &= 1 \\
x &= x + \lambda \\
x &= 4 + 1 \\
x &= 5
\end{aligned}
$$

P2: $x = 5, N = 713 = 23 * 31$,Binary representation of $x = 100, x_2 = 1, x_1 = 0, x_0 = 0, m = 3$.

I1: $x = 4, r_1 = 3, r_2 = 9, r_3 = 5, l = 2^{3-1} = 4, B = 1$

$$
\begin{aligned}
T_1 &= (5 + 3) * 9 \bmod 713 \\
T_1 &= 72 \\
T_2 &= (4 + 3) * 9 \bmod 713 \\
T_2 &= 63 + 5 \bmod 713 \\
T_2 &= 68 \\
T1 &> T_2 \\
\gamma &= 1 \\
x_2 &= 1 \\
x &= 5 - (4) * 1 \\
x &= 1
\end{aligned}
$$

I2: $x = 1, r_1 = 5, r_2 = 6, r_3 = 3, l = 2^{2-1} = 2, B = 0$

$$
\begin{aligned}
T_1 &= (1+5) * 6 \bmod 713 \\
T_1 &= 36 \\
T_2 &= (2+5) * 6 \bmod 713 \\
T_2 &= 36 + 3 \bmod 713 \\
T_2 &= 39 \\
T_1 &< T_2 \\
\gamma &= 1 \\
x_1 &= 1 - \gamma \\
x_1 &= 0 \\
x &= 1 - (2) * 0 \\
x &= 1
\end{aligned}
$$

End of for loop

$$
\begin{aligned}
x_0 &= x - \lambda \\
x_0 &= 1 - 1 \\
x_0 &= 0
\end{aligned}
$$

Therefore at the end of SBD we will get,

$$
\begin{aligned}
E(x_2) &= E(1) \\
E(x_1) &= E(0) \\
E(x_0) &= E(0)
\end{aligned}
$$

Example 5:

P1: $x = 4, N = 713 = 23 * 31, r = 30$ Binary representation of $x = 100$

$$
\begin{aligned}
y &= 30 + 4 \\
y &= 34 \\
\lambda_2 &= 0 \\
\lambda &= 1 - \lambda_2 \\
\lambda &= 1 \\
x &= x + \lambda \\
x &= 4 + 1 \\
x &= 5
\end{aligned}
$$

P2: $x = 5, N = 713 = 23 * 31$,Binary representation of $x = 100, x_2 = 1, x_1 = 0, x_0 = 0, m = 3$.

I1: $x = 4, r_1 = 3, r_2 = 90, r_3 = 5, l = 2^{3-1} = 4, B = 1$

$$
\begin{aligned}
T_1 &= (5+3) * 90 \bmod 713 \\
T_1 &= 720 \bmod 713 \\
&\quad \text{Overflow happens in the value of } T_1 \\
T_1 &= 7 \\
T_2 &= (4+3) * 90 \bmod 713 \\
T_2 &= 630 + 5 \bmod 713 \\
T_2 &= 635 \\
T1 &< T_2 \\
\gamma &= 0 \\
x_2 &= 0
\end{aligned}
$$

Due to the overflow in value of T_1 we get a wrong result for x_2. This error will be found in phase 3 and will start the protocol from phase 2.

5.2 Correctness

We emphasize that the final encrypted bits computed from the SBDv_0 protocol are always correct. This is because of the secure verification step (i.e., Phase 3) in SBDv_0 which makes sure that the protocol terminates only if the output of Phase 2 is correct. It is possible for the SBDv_0 protocol to generate the correct output after many runs of Phase 2 & 3 (which depends on x and random numbers r_1, r_2, r_3 chosen in each iteration of Phase 2).

5.3 Security Analysis

In the SBDv_0 protocol, the communication between Alice and Bob occurs in every Phase. We would individually explain the security of each phase:

Phase 1: In Phase 1, Bob randomizes the value of x by adding r. This will hide the value of x from Alice. Alice would return the encrypted value of λ_2 which will not reveal any information of x to Bob. Therefore Phase 1 is secure.

Phase 2: SBDp requires only a single random number r to randomize the value of x in Stage 1. During Phase 2, in each iteration, Bob randomizes current value of x using random numbers r_1, r_2 and r_3. Why does Bob needs three random numbers ?

Suppose $D = x - l$, the difference between x and l, where $l = 2^{m-1}$. Due to Phase 1, $D > 0$ when $x_{m-1} = 1$, $D < 0$ when $x_{m-1} = 0$. Bob needs to know if $x > l$ or $x < l$, to get the MSB of x. Directly sending $V_0 = (x, l)$ would reveal the value of x. He can hide the value x by adding r_1 to x and sending $V_1 = (x + r_1, l)$, but this would alter D, which would give wrong result for MSB.

Therefore Bob would need to add the same r_1 to l and send $V_2 = (x+r_1, l+r_1)$. We are assuming that Alice knows the bit-length of x (i.e m) and the protocol used by Bob. Therefore he would be knowing the value of l and by the difference of the two values of V_2 he would get $D_2 = D$. l and D combined will reveal the value of x to Alice. That's why we are multiplying both the terms of V_2 by r_2, which will make their difference to be $D_2 = D * r_2$. Factorization of D_2 can reveal the D to Alice. That is why Bob will be adding r_3 to second value of V_2 which will make the difference to be $D_3 = D * r_2 - r_3$, for the tuple $V_3 = ((x + r_1) * r_2, (l + r_1) * r_2 + r_3) = (T_1, T_2)$. From D_3 Alice would not be able to find the value of x. From the Observation 4 we know that the MSB of $x = 1$ when $T_1 > T_2$ and MSB of $x = 0$ when $T_1 < T_2$. Bob would send T_1 andT_2 in random order such that Alice would not know which value is T_1 and which is T_2. Therefore Phase 2 will be secure.

Phase 3: Phase 3 is the Stage 2 of SBDp protocol. By the security analysis of SBDp we know that Stage 2 is secure. Therefore Phase 3 is also secure.

5.4 Complexity Analysis

For the SBDv_0 protocol, Phase 2 involves $m - 1$ number of iterations, where each iteration requires one round of communication between Alice and Bob. Whereas, Phase 1 and 3 require one round of communication between Alice and Bob. For many practical applications, the probability of getting the correct result in the first instantiation of the SBDv_0 protocol is very high. Therefore, we assume that Phase 2 and 3 are run only once. Thus, the round complexity of SBDv_0 is bounded by $O(m)$.

In Phase 1 of SBDv_0, Bob performs 1 encryption to get $E(r)$ and 1 homomorphic addition to get $E(y)$. Then he sends $E(y)$ to Alice. Alice decrypts it and sends $E(\lambda_2)$ to Bob, which requires 1 decryption and 1 encryption. If we assume that r is even then Bob requires to compute $E(1 - \lambda_2)$ which requires 1 exponentiation and 1 homomorphic addition. Then 1 homomorphic addition to compute $E(x + \lambda)$. Overall Phase 1 requires 3 encryptions, 1 exponentiation and 3 additions. We have assumed that encryption and decryption has same computational costs.

In Phase 2 of SBDv_0, in each iteration Bob firstly computes $E(T_1) = E((x + r_1) * r_2)$ requires 1 homomorphic addition followed by 1 exponentiation. Computing $E(T_2) = E((l + r_1) * r_2 + r_3)$ requires 1 encryption since Bob will compute $(l+r_1)*r_2+r_3$ and encrypt it. Alice will need 2 decryptions followed by 1 encryption to send $E(\gamma)$ to Bob. If we assume that T_1 and T_2 has been swapped then Bob will have to perform 1 exponentiation and 1 homomorphic addition to find $E(1-\gamma)$ from $E(\gamma)$. To update the value of x Bob will need 1 exponentiation and 1 homomorphic addition. So each iteration will require 4 encryptions, 3 exponentiations and 3 additions. Overall it would be $4(m - 1)$ encryptions, $3(m - 1)$ exponentiations and $3(m - 1)$ additions. To find $E(x_0) = E(x - \lambda)$ Bob will need to do 1 exponentiation and 1 homomorphic addition.

Phase 3 of SBDv_0 requires 1 decryption, $m + 2$ exponentiation, and m homomorphic addition operations. Hence, the total computation cost of SBDv_0

(one instantiation) is roughly $4m$ encryptions, $4m+1$ exponentiations and $4m+1$ additions.

The communication complexity of our protocol mostly depends on Phase 2. During Phase 2, in each iteration, Bob sends encrypted value T_1, T_2 to Alice. Also, Alice sends an encrypted value γ to Bob. Since we have $m-1$ iterations, the total communication complexity of SBDv_0 is bounded by $O(m * K)$ bits.

To find all the encrypted bits, SBDv_0 has nearly same computational and communicational costs as the SBDp protocol. But if we are only interested in finding the MSB of x then we can do some modifications in the SBDv_0 and get the MSB of x in constant computational and communicational time.

6 SBDv_1

SBDv_0 is probabilistic in the sense that computing T_1 or T_2 might lead to an overflow which will result in wrong result for the computed encyrpted bits. But we can remove it's probabilistic nature by adding criteria on the values of r_1 and r_2. If n_1 and n_2 are bit-lengths of r_1 and r_2 then choosing r_1 and r_2 such that $n_1 + n_2 + m + 1 < K$ will always result in the case of no overflow. Explanation of this is given below.

Statement: If T_1 has bit-length l_1 and T_2 has bit-length l_2 then $T_1 * T_2$ will have bit-length $\leq l_1 + l_2$. Proof of this is given below.

$$\begin{aligned} T_1 &< 2^{l_1} \\ T_2 &< 2^{l_2} \\ T_1 * T_2 &< 2^{l_1+l_2} \end{aligned}$$

Now we would explain that $n_1 + n_2 + m + 1 < K$ will always result in the case of no overflow.

$$\begin{aligned} &\textbf{if } m < n_1 : \\ \text{bitlen}(x + r_1) &\leq n_1 + 1 \\ \text{bitlen}((x + r_1) * r_2) &\leq n_1 + n_2 + 1 \\ \text{bitlen}((x + r_1) * r_2) &< n_1 + n_2 + m + 1 \\ \text{bitlen}((x + r_1) * r_2) &< K \\ \text{bitlen}(T_1) &< K \end{aligned}$$

$$\begin{aligned} \text{bitlen}(l + 1 + r_1) &\leq n_1 + 1 \\ \text{bitlen}((l + r_1) * r_2 + r_2) &\leq n_1 + n_2 + 1 \\ \text{bitlen}((l + r_1) * r_2 + r_3) &\leq n_1 + n_2 + 1 \\ \text{bitlen}((l + r_1) * r_2 + r_3) &< n_1 + n_2 + m + 1 \\ \text{bitlen}((l + r_1) * r_2 + r_3) &< K \\ \text{bitlen}(T_2) &< K \end{aligned}$$

$$
\begin{aligned}
&\textbf{else } m \geq n_1 : \\
&\text{bitlen}(x + r_1) \leq m + 1 \\
&\text{bitlen}((x + r_1) * r_2) \leq m + n_2 + 1 \\
&\text{bitlen}((x + r_1) * r_2) < n_1 + n_2 + m + 1 \\
&\text{bitlen}((x + r_1) * r_2) < K \\
&\text{bitlen}(T_1) < K
\end{aligned}
$$

$$
\begin{aligned}
&\text{bitlen}(l + 1 + r_1) \leq m + 1 \\
&\text{bitlen}((l + r_1) * r_2 + r_2) \leq m + n_2 + 1 \\
&\text{bitlen}((l + r_1) * r_2 + r_3) \leq m + n_2 + 1 \\
&\text{bitlen}((l + r_1) * r_2 + r_3) < n_1 + n_2 + m + 1 \\
&\text{bitlen}((l + r_1) * r_2 + r_3) < K \\
&\text{bitlen}(T_2) < K
\end{aligned}
$$

In real world calculations the bit-length of message x(i.e. m) is generally very small as compared to the key size K. Therefore we can easily choose values of r_1 and r_2 such that $n_1 + n_2 + m + 1 < K$. Which will make our SBDv_0 a deterministic protocol instead of a probabilistic one. We will name this protocol SBDv_1. Security analysis of this protocol will be similar to the SBDv_0 protocol. We have already explained the correctness of this protocol. Now we will see its Complexity Analysis.

6.1 Complexity Analysis

This protocol will require only Phase 1 and 1 iteration of the for loop of Phase 2 of SBDv_0 protocol to compute the MSB of x.

It will require 1 round of communication in Phase 1 and 1 round of communication in Phase 2. Overall it will require 2 round of communications.

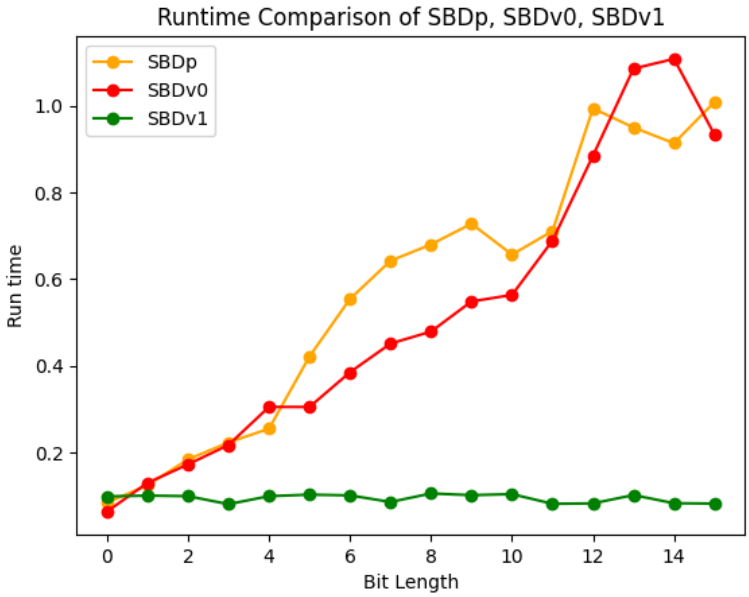
It will require 3 encryptions, 1 exponentiation and 3 homomorphic additions in Phase 1. In Phase 2 there will be 4 encryptions, 2 exponentiation and 2 homomorphic additions (we have removed the cost of updating x in Phase 2). Overall it will require 7 encryptions, 3 exponentiations and 5 homomorphic additions.

The communication complexity would be 2 * K. Cause there are only 2 communication rounds.

Complexity comparison table for computing MSB of x for all the 3 protocols is given below. Here En, Ex and A are abbriviations for Encryption, Exponentiation and homomorphic Addition.

	Rounds	Computations			Communications
		En	Ex	A	
SBDp	$O(m)$	$3m + 1$	$4m + 2$	$3m$	$O(m * K)$
SBDv_0	$O(m)$	$4m$	$4m + 1$	$4m + 1$	$O(m * K)$
SBDv_1	2	7	3	5	$2 * K$

We have implemented SBDp, SBDv_0 and SBDv_1 algorithms to find the MSB for different bit-length values in python language. *phe* library has been used for the implementation of Paillier encryption scheme.



In above image we have shown the run-time for different bit-length values for SBDp, SBDv_0 and SBDv_1. From the image we can infer that computation time for finding MSB increases linearly for SBDp and SBDv_0, and is constant for SBDv_1.

References

[1] Bharath KK Samanthula, Hu Chun, and Wei Jiang. An efficient and probabilistic secure bit-decomposition. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 541–546, 2013.

[2] Nina Pettersen. Applications of paillier's cryptosystem. Master's thesis, NTNU, 2016.