

Abstract—We propose a solution to the secure integer comparison problem by reducing it to the secure set membership problem on a set where the elements occur in consecutive order. Our protocol provides flexibility due to this reduction such that it can easily be modified to support more complex boolean functions other than the function evaluating the 'Less Than' predicate (LT). In the paper we show one such boolean function, 'IsInInterval', which checks if an integer is in an interval (3I). We also show that our protocols can be easily extended to support shared outputs so that each party would hold a share of the result. Our generic solution can be applied to many e-commerce and data-mining applications requiring one or more such boolean functions to be computed securely. Our protocol not only is adaptable to different boolean functions but also is experimentally more efficient than the state-of-the-art integer comparison protocols supporting shared outputs.

I. Introduction

Secure multi-party computation (SMC) was first suggested by Yao[1] as the millionaire’s problem, in which two millionaires want to learn who is richer without revealing their wealth to each other. The problem with its solution gave rise to the more general problem, where multiple parties try to compute some function securely given each party contributes some secret input.

With increasing need to do computation securely on data that is distributed among multiple parties that do not necessarily trust each other, SMC techniques have become a turning point due to their strong security guarantees. However, they often lack desired efficiency, limiting their adoption in e-commerce applications[2]. In this paper we present a technique that not only provides the strong security features of cryptography but also is more efficient than the state-of-the-art approaches to the problem. Our technique can be used for the secure computation of many boolean functions; we specifically detail secure comparison and secure interval test.

Secure integer comparison (SC) protocols (given two private integer inputs a and b, output true if \( b \leq a \), false otherwise) have been widely studied since originally proposed with Yao’s millionaire’s problem. It is a particularly interesting function, as the state of the art [3], [4], [5], [6], [7] has not been able to achieve dramatic performance improvements over the generic circuit evaluation approach. As many applications such as privacy-preserving on-line auction, benchmarking, and data mining require multiple uses of SC, this is a significant factor in the aforementioned performance limitations on adoption of the technology. While perhaps not sufficient to overcome all performance concerns, we present a protocol that can support shared outputs and demonstrate significant performance improvement over the current leader (5I).

In addition, at relatively small additional cost our approach can extend to additional functions; in this paper we show a generalization to the ‘IsInInterval’ function (3I). Given two integer inputs \( a_1, a_2 \) from one party and \( b \) from a second party, 3I outputs true if \( a_1 \leq b \leq a_2 \), or false otherwise. Secure evaluation of 3I has various applications including anonymous authentication, electronic voting, range queries over outsourced data, and anonymization techniques such as δ-presence [8].

Briefly, the idea behind our protocol is that we specially form a binary tree such that any interval, and any single value, can be represented using a (fixed size) set of nodes in that tree. Intersection of the two nodes sets reveals if the value is or is not in the interval. (Setting one value of the range to an endpoint of the space gives Secure Comparison.) We show that using secure polynomial evaluation to perform the intersection test, this approach gives a significant improvement in performance over other protocols with equivalent security guarantees.

A. Our Contributions

In this paper, we propose a new secure multi-party computation protocol that can be used for various problems similar to secure comparison problem. We apply our protocol to secure comparison of two integers (SC) and checking some integer over an interval securely (3I). We believe that our generic protocol can also address many other similar problems.

In summary, our protocol uses a perfect binary tree (PBT), in which the leaf level contains all possible integers, 0 through \( n-1 \), that can be given as input in SC or 3I. A set that contains elements from non leaf nodes and/or leaf nodes is used to represent a set of consecutive nodes in the leaf level. By using the elements in this set as input, a secure set intersection (SSI) protocol is called as many times as the height of the PBT in a single run of our protocol. The input sets for each SSI consists of 1 element for SC and 2 elements for 3I. Since the cardinality of each input set is at most 2, each SSI runs in constant time regardless of the problem (SC or 3I). Since PBT has a height of
log \( n \) and for each level the number of modular multiplications is \( O(\log n + \ell) \) where \( \ell \) is the bit-length of a security parameter discussed later, the computation complexity of our protocol is \( O(\log n (\log n + \ell)) \). To the best of our knowledge, our protocol is not only the first approach based on secure set intersection which also supports shared outputs, but also the most flexible one in terms of fitting into more complex boolean functions.

Our protocol uses the SSI protocol in [9] without the hash bucket improvement since the polynomials evaluated have at most degree 2. The SSI protocol uses the semantically secure, additively homomorphic RSA encryption from [10] which exploits small subgroups of \( \mathbb{Z}_n^* \) of hidden order. As a result of using a \( k \)-bit RSA modulus, the communication complexity of our protocol is \( O(k \log n) \). To be precise, in SC version, the party creating the polynomials send \( 2k \log n \) bits to the party that is going to evaluate it. The other party sends back only the encrypted results which are totally \( k \log n \) bits long. In the 3I version, the only change is in the side of the party creating the polynomials, who now sends \( 3k \log n \) bits in this case. So the total bits that are sent are \( 3k \log n \) in SC and \( 4k \log n \) in 3I.

The rest of the paper is outlined as follows. The details about the encryption scheme and secure set intersection protocol is given in Section 2. Section 3 gives a formal definition of SC and 3I, and explains our protocol with respect to SC and 3I. Section 4 shows the experimental results of our protocol with a comparison with the secure integer comparison protocol of [5].

B. Related Work

Secure comparison protocols along with the secure multiparty computation problems have been widely studied after suggestion of Yao’s millionaires problem [1]. There are many solutions to the secure comparison problem, one of which is applying generic protocols which can securely evaluate an arbitrary function. The most efficient solution to SMC related problems is using secure circuit evaluation purposed in [1], [11]. Although such protocols provide generality, they cannot offer the same performance as more specialized protocols.

Secure integer comparison (SC) is the starting point of SMC protocols since Yao’s millionaires problem is in fact an SC problem. There are many specialized solutions to the problem which provides efficiency with respect to generic methods (e.g. [3], [4], [5], [6], [7]). Most of these solutions are based on doing calculations on the bits of integers by using homomorphic encryption or encrypting bits as quadratic residues and non-residues modulo an RSA modulus. To our knowledge, the most efficient one is proposed by Damgård et. al [5]. For a secure comparison of \( \ell \)-bit numbers, The protocol by Damgård et. al only needs to encrypt values of less than \( \ell + 2 \). They also proposed a new additively homomorphic RSA encryption to exploit the fact that small size messages needs to be encrypted. The idea comes from the encryption scheme of Groth’s in [10] which exploits subgroups of \( \mathbb{Z}_N^* \) for an RSA modulus \( N \) making the cryptosystem efficient for encryption. In their protocol, the communication cost for both parties are \( O(\ell k) \) where \( k \) is the key size which is typically 1024 for an RSA encryption scheme. The computational cost is \( O(\ell (t + \log \ell)) \) multiplications mod \( N \) where \( t \) is the ‘information theoretic’ security parameter of the encryption system that is used in the protocol which is 320bits based on [12].

As for the 3I protocol, there is no specialized protocol that has been proposed except the one in [13] to the best of our knowledge. Nishide et.al. simplifies bit-decomposition protocol which is proposed in [14] in order to construct more efficient protocols for SC and 3I (and also equality test) than the original protocol. Their protocol naturally support shared inputs and output due to bit-decomposition.

Furthermore, an SC protocol can be modified to adapt 3I (by running the protocol for \( a_1 \leq b \) and \( b \leq a_2 \) separately) as long as the protocol provides shared outputs. Otherwise, intermediate results leak information not implied by the output (i.e. \( a_1 \leq b \leq a_2 \) being false does not imply \( a_1 \leq b \) being false). Damgård et. al states their protocol can be modified for shared output but at the cost of doubling the computation time. Thus, a secure 3I protocol requires four times the execution time of the original protocol by Damgård et. al.

As far as we know the nearest idea to ours is [15]. They also provide a solution to secure integer comparison based on secure set intersection and homomorphic encryption. However their protocol does not support shared outputs inherently as a result their protocol fails to extend to a secure interval check protocol.

Although there is not many ad hoc SMC for 3I function, there are numerous solutions to secure range proofs [16], [17], [18], [19]. However the applications of secure range proofs are different than that of 3I. Secure range proof is intended when it is the case that a commitment of an integer is proved to be in some particular integer range whereas 3I is applicable to two parties one holding the interval and the other holding the integer, in which they want to calculate the function \( a_1 \leq b \leq a_2 \) without revealing their inputs.

II. B

In this section, we explain basic blocks of our protocol.

A. Homomorphic Encryption

We require an additively homomorphic and semantically secure encryption scheme with plaintext space \( \mathbb{Z}_p \), for some prime \( p \):

\[
E(m_0) \cdot E(m_1) = E(m_0 + m_1 \mod p)
\]

\[
E(m)^c = E(c \cdot m \mod p)
\]

The one proposed in [10] is quite efficient for our purposes. It exploits small subgroups of \( \mathbb{Z}_n^* \) of hidden order thus provides fast encryption. The cryptosystem is secure under a strong RSA assumption [10]. Encryption of \( m \in \mathbb{Z}_n \) is as follows:

\[
E(m) = g^m h^\prime \mod n
\]
where \( n = pq = (2p^r r_p + 1)(2q^r r_q + 1) \) in which \( p' \) and \( q' \) are primes and \( r_p, r_q \) consists of distinct odd prime factors smaller than some low bound \( B \), e.g., \( 2^{65} \). \( g \) has order \( p'q' r_p r_q \), \( h \) has order \( p'q' \) and \( r_p r_q \). And the decryption is as follows:

\[
D(E(m)) = (g^m h^r)^{p'q'} = g^{p'q'r} h^{p'q'} r = (g^{p'q'})^m \mod n
\]

where \( g^{p'q'} \) has order \( r_p \) since \( g \) has order \( p'q' r_p r_q \). Although \( r_p \) is a small prime which makes decryption possible, the decryption operation is slow. Fortunately, our protocol never actually decrypts an encrypted message. It only checks whether a cipher-text is an encryption of 0 or not.

We choose the bit-lengths of the parameters as in [10] except that of \( r \). \( p' \) and \( q' \) are 100-bit prime numbers, \( r \) is a 250-bit integer since the multiplicative order of \( h \) is \( p'q' \) which is 200-bit. Damgard et al. states that size of \( r \) should be larger than the size of the order of \( h \) [12]. They have \( 2 \cdot t \) as the size of \( h \)'s order and \( 2.5 \cdot t \) as the the size of the random \( r \). For fairness, we follow the same reasoning and set \( r \) to be 250-bit long. As for the other parameters in the cryptosystem, \( B \) is equal to \( 2^{65} \) and \( r_p \) is chosen based on the upper-bound bit length of the integers that are given as input to our protocol.

### B. Secure Set Intersection

The secure set intersection scheme with arguments \((S, b)\) is a two-party protocol between Alice and Bob. Alice has the set of elements \( S = \{a_1, \ldots, a_n\} \), Bob has a single input \( b \). At the end of the protocol Alice learns if \( b \in S \).

The SSI protocol we use in this paper is based on evaluating polynomials by using homomorphic encryption [9]. Alice creates the following polynomial for her inputs:

\[
P(x) = (x - a_1)(x - a_2) \ldots (x - a_n) = \sum_{k=1}^{n} c_k x^k
\]

Alice sends the homomorphic encryptions of the coefficients (e.g., \( c_k \)) of this polynomial. Bob calculates \( c = E(P(b)) \) where \( r \) is a random number that randomizes the output if it is not 0. Then Bob sends \( c \) to Alice for decryption. If Alice notices that \( c \) is an encryption of 0, then \( b \) is in the set \( S \).

Although we use Horner's rule of evaluating polynomials as suggested in [9], we do not use the technique of hashing for bucket allocation since we have at most 3 coefficients in one polynomial for each level of the PBT. In other words, the structure in our protocol allocates polynomials to each level of the PBT and as an outcome the bucket allocation is naturally satisfied without hashing.

### III. O P

#### A. Secure Integer Comparison

In this section, we formally define the secure integer comparison problem and present an efficient protocol that is based on the secure set intersection problem.

![3 Perfect Binary Tree](image)

**Fig. 1.** 3 Perfect Binary Tree

1) **Problem Definition:**

**Definition 1 (Secure Integer Comparison (SC)):** The secure integer comparison scheme with arguments \((a, b)\) is a two-party protocol between Alice and Bob. Alice and Bob have \( \ell_n \) bit inputs \( a \) and \( b \) respectively. At the end of the protocol, Alice learns if \( b \leq a \).

Obviously, an SC protocol with arguments \((a, b)\) reduces to an SSI protocol with arguments \( \bigcup_{x \in \mathbb{Z}} \{s, (b)\} \) where the size of the first set can be as large as \( n = 2^{65} \). As noted before, the number of exponentiations required by such a secure SSI protocol is in the order of \( O(n) \). This is certainly not acceptable since there already exists other integer comparison protocols with complexities that are linear in \( \ell_n \). In the next section, we deal with this problem.

2) **Binary Tree Index:**

**Definition 2 (Tree node):** A tree node is a data structure that consists of a unique label \((h, o)\) and possibly two pointers \( left \) and \( right \) to other tree nodes. The label has two components; the height \( h \) and the order \( o \). A leaf node is a tree node with a label \((0, o)\) and no pointers. We use \( . \) operator to demonstrate the components of a tree node \( v \) (e.g., \( v.o, v.left \), \( \cdot \)).

**Definition 3 (\( \ell_n \) Perfect Binary Tree (\( \ell_n \) PBT)):** An \( \ell_n \) perfect binary tree is a binary tree with a set of leaf nodes \( L = \{(0, 0), (0, 1), \ldots, (n, n - 1)\} \) and a set of non leaf nodes \( NL \) such that \( \ell_n \in NL \) and for all \((h, o)\in NL \), there exists \( (h - 1, 2o), (h - 1, 2o + 1) \in (L \cup NL) \) with \((h, o).left = (h - 1, 2o) \) and \((h, o).right = (h - 1, 2o + 1) \). In an \( \ell_n \) PBT, the root node is the node \((\ell_n, 0)\).

In Figure 1, we show a 3 Perfect Binary Tree.

**Definition 4 (Coverage):** Given an \( \ell_n \) PBT, we say a tree node \((h_1, o_1)\) covers a leaf node \((0, o_2)\) if there exist a path from \((h_1, o_1)\) to \((0, o_2)\) in the tree (e.g., if \( o_1 \cdot 2^{h_1} \leq o_2 < (o_1 + 1) \cdot 2^{h_1} \)). The covering set of a given leaf node \( v \) is the set of all nodes in the PBT that cover \( v \) (e.g., all nodes on the path from \( v \) to the root). The coverage of a tree node \( v \) is the set of all leaf nodes covered by \( v \). \( v.leftLeaf[v.rightLeaf] \) returns the left [right] most leaf node in the coverage of \( v \) (\( v.leftLeaf = (0, o_1 \cdot 2^{h_1}), v.rightLeaf = (0, (o_1 + 1) \cdot 2^{h_1} - 1) \)).

In Figure 1, \((2, 1)\) covers \((0, 6)\). Covering set of the leaf node \((0, 6)\) is \( \{(0, 6), (1, 3), (2, 1), (3, 0)\} \). The coverage of \((2, 1)\) is \( \{(0, 4), (0, 5), (0, 6), (0, 7)\} \). \((2, 1)\).leftLeaf is \((0, 4)\).
\textbf{Definition 5 (Representer Set):} Given an $\ell_n$ PBT, a representer of a set of leaf nodes $L' \subseteq L$ is a set of nodes $R \subseteq (L \cup NL)$ such that
\begin{itemize}
  \item for all nodes $v \in L'$, there exists a node in $R$ that covers $v$, and
  \item for all nodes $v \in R$, there is no leaf node $v' \notin L'$ that is covered by $v$.
\end{itemize}
A representer $R$ for the set of leaf nodes $L'$ is minimal, if there is no other representer $R'$ of $L'$ with $|R'| < |R|$.

\textbf{Lemma 1:} Let $L$ be the set of leaves of an $\ell_n$ PBT. Let $L_0, L_1 \subseteq L$ be sets of leaf nodes, also suppose there exists a node $v$ in the PBT such that all nodes of $L_0$ are covered by $v.lef t$ and all nodes of $L_1$ are covered by $v.ri ght$ and there exists at least one other leaf node $v_i \notin (L_0 \cup L_1)$ covered by $v$. Then $R_{L_0} \cup R_{L_1}$ is a minimal representer of $L_0 \cup L_1$.

\textbf{Proof:} Clearly $R_{L_0} \cup R_{L_1}$ is a representer of $L_0 \cup L_1$. Since $R_{L_0}$ and $R_{L_1}$ are disjoint, $|R_{L_0} \cup R_{L_1}| = |R_{L_0}| + |R_{L_1}|$. Suppose there is a minimal representer $R$ of $L_0 \cup L_1$, such that $|R| < |R_{L_0} \cup R_{L_1}|$.

Let $R_0$ be all nodes of $R$ which covers nodes from $L_0$, and define $R_1$ similarly. Since $L_0$ and $L_1$ are in different sub-trees, $R_0$ and $R_1$ must be disjoint, so $|R| = |R_0| + |R_1|$. Since $|R|$ is smaller than $|R_{L_0}| + |R_{L_1}|$, either $|R_0| < |R_{L_0}|$ or $|R_1| < |R_{L_1}|$. But this contradicts the minimality of the representers $R_0$ and $R_1$.

In Figure 1, $(1,1)$ is a minimal representer for $(0,2), (0,3)$, and $(0,4)$ is a minimal representer for $(0,4)$. Then $(1,1)$, $(0,4)$ is a minimal representer for $(0,2), (0,3), (0,4)$.

\textbf{Lemma 2:} If the coverage of a node $v$ is a set of nodes $L'$, then $\{v\}$ is a minimal representer for $L'$.

\textbf{Proof:} By definition 5, $\{v\}$ is a representer for $L'$. A minimal representer for $L'$ cannot be an empty set as long as $L'$ is not an empty set. Thus, $\{v\}$ is also minimal.

Given an $\ell_n$ PBT and an upper limit $a$, we are interested in finding a minimal representer for the set of nodes $(0,0), \cdots, (0,a)$. The following lemmata states that the size of such a minimal representer is at most $\ell_n$.

\textbf{Lemma 3:} Let $R$ be a minimal representer for $L' = \{(0,0), \cdots, (0,a)\}$. For each level $0 \leq i \leq \ell_n$, there can be at most one node $v \in R$ such that $v.h = i$.

\textbf{Proof:} Suppose that $v, v' \in R$ both have the same height.
Without loss of generality let $v.o < v'.o$. Also suppose $\tilde{v}$ is the parent of $v$. Then $v.lef t.Leaf.o < v'.ri ght.Leaf.o \leq a$, and so the set $\{(0,v.lef t.Leaf), \cdots, (0,v'.ri ght.Leaf)\}$ is a subset of $L'$ (and so are the leaves covered by all siblings between $v$ and $v'$). This implies that the nodes covered by the other child node $v_i$ of $\tilde{v}$ should also be in $L'$. Then $\tilde{v}$ only covers leaves in $L'$, and so $v$ and $v_i$ can be substituted by $\tilde{v}$, thus reducing the size of $R$ by one. But this contradicts that $R$ is minimal, so no two nodes can be on the same height.

\textbf{Lemma 4:} For $\ell_n \geq 1$ and for any $0 \leq a < 2^{\ell_n}$, the size of any minimal representer $R$ for the set of leaf nodes $L_a = \{(0,0), \cdots, (0,a)\}$ is at most $\ell_n$.

\textbf{Proof:} Refer to Appendix Section A1.

In Algorithm 1, we present an algorithm that returns a minimal representer.

\begin{algorithm}
\caption{Returns a minimal representer for the set of leaf nodes \{node.leftLeaf, \cdots, (0,a)\}.}
\begin{algorithmic}
  \Procedure{RepNodes}{node, $a$}
    \If{node.rightLeaf does not cover (0, $a$)}
      \State return \Procedure{RepNodes}{node.left, $a$}
    \Else
      \State return \Procedure{RepNodes}{node.left, $a$} \cup \Procedure{RepNodes}{node.right, $a$}
    \EndIf
  \EndProcedure
\end{algorithmic}
\end{algorithm}

To get a representer for $(0,0), \cdots, (0,5)$, we call \Procedure{RepNodes}{(3,0), \cdots, (0,5)}, $(3,0).rightLeaf = (0,7) \neq (0,5)$ and $node.right = (2,1)$ covers $(0,5)$, the algorithm returns $\{2,0\} \cup \Procedure{RepNodes}{(2,1), \cdots, (0,5)}$. Similarly, $\Procedure{RepNodes}{(2,1), \cdots, (0,5)}$ returns $\Procedure{RepNodes}{(1,2), \cdots, (0,5)}$. As $(1,2).rightLeaf = (0,5)$, $\Procedure{RepNodes}{(1,2), \cdots, (0,5)}$ returns $\{(1,2)\}$. Thus the representer returned by the algorithm is $(2,0), (1,2)$. We now prove that the algorithm always returns a representer of size at most $\ell_n$ for the input set of leaf nodes.

\textbf{Theorem 1:} Algorithm \Procedure{RepNodes}{node, $a$} returns a minimal representer for the set of leaf nodes \{node.leftLeaf, \cdots, (0,a)\}.

\textbf{Proof:} Refer to Appendix Section A2.

\textbf{Theorem 2:} Given an $\ell_n$ PBT with a root node root, algorithm \Procedure{RepNodes}{root, $a$} returns a representer set $R$ for the set of leaf nodes $(0,0), \cdots, (0,a)$ such that $|R| \leq \ell_n$ for $\ell_n \geq 1$.

\textbf{Proof:} By Definition 4, \Procedure{root.leftLeaf}{(0,0)} = (0,0). Thus, by Theorem 1, \Procedure{RepNodes}{root, $a$} returns a minimal representer $R$ for the set of leaf nodes $(0,0), \cdots, (0,a)$. By Lemma 4, $|R| \leq \ell_n$.

Algorithm 1 can be rewritten to return a representer of size at most $\ell_n$ also for $(0,0), \cdots, node.rightLeaf].$ Due to the symmetric behavior, in the original algorithm, it suffices to swap ‘left’ and ‘right’ to get such a representer. Theorem 2 still applies. Such an algorithm, when given a set of leaf nodes, say $(0,2), \cdots, (0,7)$ would return $\{(1,1), (2,1)\}$ as a representer.

3) SC Protocol: In Algorithms 2 and 3, we present a secure two party protocol for secure integer comparison.

In Algorithm 2, Alice creates a representer set for the leaf nodes $(0,0), \cdots, (0,a)$ by calling Algorithm 1. For each level $i$ in the PBT, Alice creates a polynomial $P_i$ whose root is the order of the representer node with height $i$. Alice uses an additively homomorphic public key encryption scheme, $E$, to encrypt the coefficients and sends the encrypted polynomials to Bob. Bob calculates the covering set $B$ of the node $(0, b)$.

For each node $v$ in $B$, he securely evaluates polynomial $P_{r,v}$ on $v.o$ with help of the homomorphic property of the encryption. He multiplies the results with positive random numbers, and sends the shuffled results back to Alice. Alice returns true (e.g., $b \leq a$) if any of the results decrypts to 0.
Algorithm 2 Secure integer comparison for Alice who holds the interval

procedure compareInt(a)

Require: Let root be the root node of \( \ell_b \) PBT. \( E \) is an additively homomorphic semantically secure encryption function. \( \text{pk} \) is the public key of Alice.

\( R = \text{repNodes(root, a)} \)

for all height \( h \) of PBT do

if \( \exists r \in R \mid r.h = h \) then

\[ P[h] = x \cdot r \cdot o \]

\[ a_h = -r \cdot o \]

else

\[ P[h] = x + 1 \]

\[ a_h = 1 \]

\( [\text{Alice encrypts the coefficients of her polynomial}] \)

\( EP[h] = E_{pk}(P[h]) = x + E_{pk}(a_h) \)

end

send \( EP \) to Bob [for evaluation of polynomials]

wait for \( EPR \) from Bob [\( EPR \) holds the encrypted result of each polynomial against Bob’s input]

result = false

for all \( i \in EPR \) do

if \( i \) is an encryption of 0 then

result = true

send result to Bob

return result

Algorithm 3 Secure integer comparison for Bob who holds the single integer

procedure checkInt(b)

Require: Same as in Algorithm 2.

Let \( B \) be the covering set for the leaf node \((0, b)\) (e.g., \( B = (0, b), (0, b).parent, \ldots , \text{root} \)).

wait for \( EP \) from Alice

for all \( i \) from 0 to \( \ell_b \) do

[evaluation of polynomial is done homomorphically using Horner’s rule]

\( EPR[i] = (EP[i](B[i].o))' = E_{pk}(0) \) where \( r \) is a fresh random [this randomization is to hide \( B[i].o \) if it is not 0]

shuffle \( EPR \) and send it to Alice [so that Alice doesn’t learn which polynomial evaluates to 0]

wait for result from Alice

return result

As an example, suppose \( a = 5, b = 2 \). For the set of leaf nodes \( \{(0,0), \ldots , (0,5)\} \), Alice creates the representer \( \{(1,2), (2,0)\} \). She sends \( E_{pk}(1), E_{pk}(-2), E_{pk}(0) \) to Bob in order. Bob finds the covering of \((0,2); \{(0,2,1,1,2,0)\}\) and calculates \( E_{pk}(2+1), E_{pk}(1+(-2)), E_{pk}(0+0) \) and sends back to Alice in random order. Alice sees one of the outputs decrypts to 0, she concludes \( b \leq a \). In practice, Alice and Bob do not need to send and evaluate the root node in a separate polynomial. The root and its children cannot appear together in the representer, thus the three nodes can be treated as being on the same level of the tree. Thus, it would be sufficient for Alice to send two encryptions in this example.

We now prove that the SC protocol is sound; it outputs true if and only if \( b \leq a \).

Lemma 5: Given an \( \ell_b \) PBT, let \( R_a \) be a representative set for the set of leaf nodes \( A = \{(0,0), \ldots , (0,a)\} \) and let \( B \) be the covering set for the leaf node \((0, b)\). Then \( b \leq a \) if and only if \( R_a \cap B \neq \emptyset \).

Proof: \((\Rightarrow)\) if \( R_a \cap B \neq \emptyset \), then \((0, b)\) is covered by a tree node in \( R_a \). By Definition 5, any node in \( R_a \) can only cover a leaf node in \( A \). Thus, \((0, b)\) \( \in A \) which implies \( b \leq a \).

\((\Leftarrow)\) if \( b \leq a \), then \((0, b)\) \( \in A \). By Definition 5 there exists a node \( v \in R_a \) that covers \((0, b)\). Since \( B \) contains all nodes that covers \((0, b), v \in B \) as well.

Theorem 3: The SC protocol given in Algorithms 2 and 3, returns true if and only if \( b \leq a \).

Proof: For each level \( i \), if there exists \( v_i \in R_a \) and \( v_i.h = i \), Alice sends a polynomial with root \( v_i.o \) otherwise with root -1 which does not appear as a label in the tree. Bob evaluates the polynomial on \( v_i.o \) where \( v_i \in B \) and \( v_i.h = i \). If the protocol returns true, at least one of the polynomials evaluates to 0, then there exists \( v_i.o \in R_a \) and \( v_i.h \in B \) with \( v_i.o = v_i'.o, v_i.h = v_i'.h \). Since labels are unique in the PBT, this implies \( v = v' \in R_a \cap B \). By Lemma 5, \( b \leq a \). Similarly, if the protocol returns false, \( R_a \cap B = \emptyset \). Again by Lemma 5, \( b > a \). Intuitively the comparison protocol is secure against curious Bob because of the semantic security of the encryption scheme. It is secure against curious Alice because she receives a number of random encryptions of random numbers, where only one of the encrypted numbers can be zero (when the statement is true). Before proving security formally, we state a few helping lemmata.

Lemma 6: Let \( R \) be a minimal representer of the set of leaf nodes \( L \). For each node \( n \in R \), no other node \( n' \in R \) is a descendant of \( n \).

Proof: Let a minimal representer \( R \) of \( L \), and element \( n \in R \) be given. For contradiction, suppose that \( n' \in R \) is a descendant of \( n \). Since \( n \) covers all leaves which \( n' \) covers, the set \( R \setminus \{n'\} \) is a representer of \( L \), contradicting that \( R \) is a minimal representer. So no \( n' \in R \) is a descendant of \( n \).

Lemma 7: For any \( a, b \in [0, \ldots , 2^a - 1] \), the intersection between a minimal representer of \( [0,0], \ldots , (0,a) \) and the covering of \((0, b)\) contains at most one element.

Proof: It follows directly from Lemma 5 that if \( b > a \), the intersection is empty. Now assume that the intersection contains two distinct nodes \( n_0 \) and \( n_1 \). Since a covering contains all nodes on a path from a leaf to the root, we can assume without loss of generality that \( n_0 \) is a descendant of \( n_1 \). However, this contradicts Lemma 6, so at most one node can exist in the intersection.

Theorem 4: Let \( (G, E, D) \) be an additively homomorphic, semantically secure public key encryption scheme with plain text space \( \mathbb{Z}_p \), for some prime \( p \). The SC protocol given in Algorithms 2 and 3 is a secure comparison protocol in the semi-honest model. The protocol is secure against computationally bounded Bob, and information theoretically secure.
against Alice.

Proof: In the first round of the protocol Bob receives the encrypted coefficients of the \( \ell_n \) first degree polynomials. Since the encryption scheme is semantically secure, these ciphertexts are computationally indistinguishable from encryptions of random numbers.

Our claim is that the result received by Alice is a list of random encryptions of random numbers from \( \mathbb{Z}_p \), with one encryption of 0 if and only if the result of the comparison is true.

Let \( c_i = EPR[i] = r * E[i] \) be the polynomial evaluations, for \( i = 1, \ldots, \ell_n \), \( \ell_n \) be the root representer, then \( rP(B_i, o) = E_\mathbb{Z}(rP(B_i, o)) \), for \( i = 1, \ldots, \ell_n \), be the root representer. It follows from Lemma 3 that the minimal representers of \( (0, a_1), \ldots, (0, a_2) \) are not covered (0, 1) \( \wedge \text{node.rightLeaf} = (0, a_2) \) then return (node)

\textbf{B. Secure Interval Check}

Definition 6 (Secure IsInInterval (3I)): Secure IsInInterval with arguments \((a_1, a_2, b)\) is a two-party protocol between Alice and Bob. Bob has an \( \ell_n \) bit input \( b \) and Alice has \( \ell_n \) bit inputs \( a_1, a_2 \) with \( a_1 \leq a_2 \). The end of the protocol, Alice learns if \( a_1 \leq b \leq a_2 \).

The protocols in Algorithms 2 and 3 can be modified for 3I. Instead of computing a minimal representer for \( (0, 0), \ldots, (0, a_2) \), Alice computes a minimal representer for \( (0, a_1), \ldots, (0, a_2) \). The size of the minimal representer for intervals are bounded as well:

Lemma 8: Let \( R \) be a minimal representer for \( L' = \{ (0, a_1), \ldots, (0, a_2) \} \). For each level \( 0 \leq i \leq \ell_n \), there can be at most two nodes with height \( i \).

Proof: (By induction in \( \ell_n \)). Base case: The lemma is trivial for \( \ell_n = 1 \).

Inductive step: There are two cases: 1) \( L' \) is entirely contained in the set of leaves covered by either \( \text{root.leftLeaf} \) or \( \text{root.rightLeaf} \), or 2) \( L' \) contains leaf nodes from both subtrees. If \( L' \) is entirely contained in one subtree, the lemma follows from the induction hypothesis. If \( L' \) contains leaf nodes from both subtrees, we define \( L'_\text{left} = \{ (0, a_0), \ldots, (0, 2^{\ell_n} - 1) \} \) and \( L'_\text{right} = \{ \text{leftLeaf}, \ldots, (0, a_1) \} \). It then follows from Lemma 3 that the minimal representers of \( L'_\text{right} \) and \( L'_\text{left} \) have only one node per level. It follows from Lemma 1 that the minimal representer of \( L' \) is the union of the minimal representers of \( L'_\text{right} \) and \( L'_\text{left} \), which can then have at most two nodes per level (one in each subtree).

Lemma 9: Given an \( \ell_n \) PBT, \( \ell_n > 1 \), and a set covering an interval \( S = \{ (0, a_1), \ldots, (0, a_2) \} \), the minimal representer contains at most \( 2(\ell_n - 1) \) nodes.

Proof: (By induction in \( \ell_n \)) Base case: For \( \ell_n = 2 \) the representer sets of each of the possible sets are easily seen to contain at most two nodes.

Inductive step: For a tree of height \( \ell_n > 2 \) both the right and left sub-trees of the root may have elements. If \( S \) is the set of all leaves, the root node is a minimal representer. If one of the sub-trees contains all the elements of \( S \) then, by induction hypothesis, the minimal representer contains at most \( 2(\ell_n - 2) < 2(\ell_n - 1) \) nodes. If both subtrees contain elements from \( S \), the minimal representer is the union of the representer sets of \( (0, a_1), \ldots, (0, 2^{\ell_n}/2 - 1) \) in the left sub-tree and \( (0, 2^{\ell_n}/2), \ldots, (0, 2^{\ell_n}) \) in the right sub-tree. By Lemma 4, the representer for the right sub-tree contains at most \( \ell_n - 1 \) nodes. By symmetry the representer of the left sub-tree also contains at most \( \ell_n - 1 \) nodes. In total the representer of the union contains at most \( 2(\ell_n - 1) \) nodes.

We now show how such a representer of bounded size can be found.

Algorithm 4 Returns a representer for the set of leaf nodes \( \{ (0, a_1), \ldots, (0, a_2) \} \).

\textbf{procedure} \textit{RepNodesI(node, a_1, a_2)}

\textbf{Require:} node in an \( \ell_n \) PBT, a lower limit \( a_1 \) and an upper limit \( a_2 \) of \( \ell_n \) bits, node covers \( (0, a_1) \) and \( (0, a_2) \).

let root be the root of the PBT.

if node.leftLeaf = (0, a_1) \&\& node.rightLeaf = (0, a_2) then return (node)

if node.right does not cover \( (0, a_2) \) then

return \textit{RepNodesI(node.left, a_1, a_2)}

if node.left does not cover \( (0, a_1) \) then

return \textit{RepNodesI(node.right, a_1, a_2)}

\textbf{return} \textit{RepNodesI(node.left, a_1)} \cup \textit{RepNodesI(node.right, a_2)}

To get a representer for \( \{ (0, 1), \ldots, (0, 3) \} \), we call \textit{RepNodesI((3, 0), (1, 3))}. As \( (3, 0).\text{leftLeaf} = 0 \neq 1 \) and \( (3, 0).\text{right} = (2, 1) \) does not covers \( (0, 3) \), the algorithm returns \textit{RepNodesI((2, 0), (1, 3)). (2, 0).\text{leftLeaf} = 0 \neq 1 \) and \( (2, 0).\text{right} = (1, 0) \) covers \( (0, 1) \) and \( (0, 3) \). Thus, \textit{RepNodesI((2, 0), (1, 3))}.

Theorem 5: Algorithm \textit{RepNodesI(node, a_1, a_2)} returns a minimal representer for the set of leaf nodes \( \{ (0, a_1), \ldots, (0, a_2) \} \) covered by node.

Proof: Refer to Appendix Section A3.

Theorem 6: Given an \( \ell_n \) PBT with a root node \( \text{root} \), algorithm \textit{RepNodesI(root, a_1, a_2)} returns a representer set \( R \) for the set of leaf nodes \( \{ (0, a_1), \ldots, (0, a_2) \} \) such that \( |R| \leq 2^{\ell_n - 2} \) for \( \ell_n \geq 2 \) and there can be at most two nodes \( v \in R \) such that \( v.\text{height} = i \).

Proof: Follows from Lemma 9 and 8.

As mentioned before, the protocol for 3I is very similar to that of SC given in Algorithms 2 and 3. The major difference is that Alice creates a representer set for the leaf nodes
\{(0, a_1), \cdots, (0, a_2)\} by calling Algorithm 4. For each level \(i\) in the PBT, Alice creates a polynomial \(P_i\) (of order 2 rather than 1) whose roots are the orders of the two representer nodes with height \(i\) (if less than 2 representers exists, Alice sets -1 for each of the roots). Bob proceeds exactly the same way, evaluating the polynomials of order 2 instead of 1. Again Alice returns true (e.g., \(a_1 \leq b \leq a_2\)) if any of the results decryts to 0.

As an example, suppose \(a_1 = 1, a_2 = 5, b = 2\). For the set of leaf nodes \(\{(0, 1), \cdots, (0, 5)\}\), Alice creates the minimal representer \(\{(0, 1), (1, 1), (1, 2)\}\). She encrypts the coefficients of \((x-1)\cdot(x+1) = x^2 - 1, (x-1)(x-2) = x^2 - 3x + 2\) thus sends the encryptions \(E_{pk}(0), E_{pk}(-1), E_{pk}(-3), E_{pk}(2)\) to Bob. Bob again finds the covering of \((0, 2); \{(0, 2), (1, 1)\}\) and calculates \(E_{pk}(2^2 - 1), E_{pk}(1^2 - 3 \cdot 1 + 2)\) and sends the results to Alice in random order. Alice sees one of the outputs decrypts to 0, she concludes \(a_1 \leq b \leq a_2\).

C. Shared Output

The protocols given in this section can be modified to generate shared outputs. In other words, we can modify the protocols such that instead of both parties learning the comparison result \(o\); Alice learns a random bit \(o_1\) and Bob learns \(o_2\) where \(o_1 \oplus o_2 = o\). Note that a shared output comparison protocol is crucial for secure distributed protocols using the SC or 3I as a subprocedure since no intermediate result that cannot be derived from output should be revealed.

To get a shared output SC protocol, Alice and Bob switch their roles. Alice creates an encrypted polynomial whose roots are the covering set of her private node and sends the polynomial to Bob. Bob flips a coin and get a random bit \(o_2\). If \(o_2\) is 0, Bob evaluates the polynomial on the minimal representer of the set \(\{(0, b), \cdots, (0, n)\}\). If \(o_2\) is 1, Bob evaluates the polynomial on the minimal representer of the set \(\{(0, 0), \cdots, (0, b - 1)\}\). Bob sends the resulting encryptions to Alice. Alice’s private bit \(o_1\) will be 1 if and only if one of the ciphertexts is an encryption of 0. Note that Alice learns the result of the boolean predicate \(b < a\) if \(o_2 = 0\) and learns the result of the predicate \(b > a\) if \(o_2 = 1\). In both cases, \(o_1 \oplus o_2 = 1\) if and only if \(b < a\). The shared output protocol for 3I is similar. Bob takes the sender role, calculates the covering set of \((0, b)\), sends the encrypted polynomial with roots achieved from the covering set to Alice. If her random bit \(o_2\) is 0, Alice evaluates the polynomial on the minimal representer of the set \(\{(0, a_1), \cdots, (0, a_2)\}\), else on the union of representers for \(\{(0, 0), \cdots, (0, a_1 - 1)\}\) and \(\{(0, a_2 + 1), \cdots, (0, n)\}\).

As an example, suppose \(a = 5, b = 2\). Alice creates the covering set \(\{(0, 5), (1, 2), (2, 1)\}\). She sends \(E_{pk}(-5), E_{pk}(-2), E_{pk}(-1)\) to Bob in order. Let’s say Bob’s random bit \(o_2\) is 1, then Bob finds the minimal representer of set \(\{(0, 0), (0, 1)\}\) which is \(\{(1, 0)\}\) and calculates \(E_{pk}(-1 - 5), E_{pk}(0 - 2), E_{pk}(-1 - 1)\) and sends back to Alice. Alice sees that none of the outputs decrypts to 0, thus she sets \(o_1 = 0\). Note that \(o_1 \oplus o_2 = 1\) and \(b < a\) is true.

Shared output version of our 3I protocol is a bit less efficient than our original 3I protocol since in the original 3I, Alice is

the party that holds the minimal representer set and creates the polynomials of degree 2 (and Bob evaluates each polynomial once) whereas in the shared output version of 3I, Alice holds the covering set for her private input and creates polynomials of degree 1 (and Bob evaluated each polynomial twice since Bob has two nodes for each level of the tree).

IV. E

In this section we compare our protocol with the protocol by Damgard et.al [5], in which the corrections in [12] are also applied, in the setting that both inputs are private and shared outputs are used. We compare the protocols based on only computation complexities since the communication time is negligible with respect to the time taken for computation. We show the computational performance of each protocol for both SC and 3I. However, their protocol needs to be modified to support 3I first. Arranging \(a_1 \leq b \leq a_2\) as \(a_1 \leq b \leq a_2\) is the only way we could find to adapt their protocol to 3I. As it is seen, if \(a_1 \leq b \leq a_2\) is false, either \(a_2 \leq b \leq a_2\) is false. Knowing one of these predicates would leak unintended information however since we use the shared (secret is used in the original paper) output version of [5] neither of the results of these predicates would be known by any party. However as stated in [5], the modified protocol in the setting of shared outputs has roughly twice the cost of the original protocol.

The modification to the protocol of Damgard et.al is simple. Assume that Alice holds \([a_1, a_2]\) and Bob holds \(b\). For inputs \(a_1, b; b > a_1 - 1\) is true indicates that \(a_1 \leq b\) is true. For inputs \(a_2, b; b > a_2\) is false indicates that \(a_2 \geq b\) is true. So if the protocol outputs true on inputs \(a_1 - 1, b\) and outputs false on inputs \(a_2, b\) then \(a_1 \leq b\) and \(b \leq a_2\) is true. In this arrangement, it seems like the protocol is run twice independently however in fact the input \(b\) is encrypted only once for the two runs thus the resulting modification has less overhead than running the protocol twice.

We use the same security parameters that is used in both [5] and [12] for Damgard et.al’s protocol. The security parameters
of the cryptosystem that we use are chosen based on the parameters in [10]. The reader may refer to Section 2 to see these values. Lastly, the bit-length of RSA modulus is 1024 for both protocols. The implementation of our protocol was done in Java 1.6. We used a machine with a dual core 2.26GHz processor and 4GB RAM for both protocols.

The comparison for secure integer comparison protocols is done by drawing 8 pair of random integers (i.e. \((a, b)\)) from interval \([2^{i-1}, 2^i - 1]\) for each bit-length, \(i\), in \(B = \{8, 16, 24, 32, 40, 48, 56, 64\}\). As for the secure interval check protocols, 8 triple of random integers (i.e. \((a_1, a_2, b)\)) are drawn from interval \([2^{i-1}, 2^i - 1]\) for each bit-length, \(i\), in \(B\) with the guarantee that \(a_2 \geq a_1\). Last but not least, implementation of finding representer nodes for both our secure integer comparison and interval check is different from the Algorithm 1 and Algorithm 4 which are just illustrative, making our protocols easier to read and understand. The same results can be achieved by using only invariants of our PBT (e.g. a node \((h, o)\) is the parent of \((h-1, 2 \ast o)\) and \((h-1, 2 \ast o + 1)\) rather than storing all nodes in memory.

Figure 2 shows the comparison of our SC protocol with theirs. Our protocol is approximately 4 times faster than theirs. Secure comparison of two 32-bit integers takes 0.78 seconds in their setting whereas our protocol compares the same integers in 0.19 seconds. In Figure 3, the comparison of our 3I protocol with their modified protocol is shown. In this case our protocol is approximately 3 times faster. Secure 3I takes 1.02 seconds in their protocol whereas it takes 0.31 seconds in our protocol.

V. C

In this paper, we presented a SMC protocol for secure integer comparison and secure interval check problems by reducing the problems to secure set intersection problem. While providing a range of solutions to these problems, we showed that our protocol is faster than the state-of-art protocol in term of computations done in the protocols.

As a future work we will try to integrate secure evaluation of more complex boolean functions into our protocol. Specifically, boolean functions involving comparison predicate as well as other predicates can be evaluated securely at little extra cost. We also intend to exploit more of the properties of minimal representatives to reduce the number of polynomial evaluations.

R

A. Proofs

1) Proof of Lemma 4: By induction in \( \ell_n \):

**Base case:** If \( \ell_n = 1 \), then we only have two leaves \((0,0),(0,1)\) with a root \((1,0)\). If \( a = 0 \), the minimal representer \( R \) is \( \{ \} \). If \( a = 1 \), \( R = \{(1,0)\} \). Thus \( |R| \leq 1 \).

**Inductive step:** Suppose for an \( \ell_n - 1 \) PBT, \(|R| \leq \ell_n - 1 \). For an \( \ell_n \) PBT, we now prove \(|R| \leq \ell_n \) for all \( 0 \leq a < 2^{\ell_n} \). Let \( \text{root} \) be the root node. Note that \( \text{root.left} \) and \( \text{root.right} \) are the roots of two disjoint subtrees each being structurally equivalent to an \( \ell_n - 1 \) PBT (e.g., only node labels of the right tree is different than the PBT). Let \( T_l \) and \( T_r \) be the sets of nodes in each of these subtrees. Now we consider three cases:

- **Case 1:** \( a < 2^{\ell_n - 1} \): As no node from \( T_l \) covers any node from \( L_a \), there can be no node from \( T_r \) in \( R \). Also the root cannot be in the representative since it covers nodes not in \( L_a \). Thus a minimal representer should be a subset of \( T_l \) which forms an \( \ell_n - 1 \) PBT. Thus by induction \(|R| \leq \ell_n - 1 \).

- **Case 2:** \( 2^{\ell_n - 1} \leq a < 2^\ell_n - 1 \): Let \( L_l = \{(0,0),\cdots,(0,2^{\ell_n - 1} - 1)\} \) and \( L_r = \{(0,2^{\ell_n - 1}),\cdots,(0,a)\} \). Note that \( L_a = L_l \cup L_r \). Lemma 1 holds for \( L_l \) and \( L_r \) as they share a parent node \( \text{root} \) and \((0,2^{\ell_n - 1}) \notin L_a \) is covered by the root. \( L_l \) is the coverage of \( \text{root.left} \), so by Lemma 2, the minimal representer \( R_l \) for \( L_l \) is \( \{ \text{root.left} \} \). By Lemma 1, \(|R| = |R_l| + |R_r| \). By induction \(|R_l| \leq \ell_n - 1 \). Thus, \(|R| \leq \ell_n \). **Case 3:** \( a = 2^\ell_n - 1 \): In this case, \( L \) is the coverage of \( \text{root} \) and by Lemma 2, \( R = \{ \text{root} \} \). Thus \(|R| \leq \ell_n \).

2) Proof of Theorem 1: By induction:

**Base case:** If \( \text{node.rightLeaf} = (0,a) \), then the coverage of node, by Definition 4, is \( \{ \text{node.leftLeaf},\cdots,\text{node.rightLeaf} \} \) which is the set of leaf nodes input to the algorithm. Thus the algorithm returns \( \{ \text{node} \} \) which is, by Lemma 2, a minimal representer set. Note that this case also covers the case when \( \text{node} \) is a leaf node with \( \text{node} = (0,a) \).

**Inductive step:** If \( \text{node.right} \) does not cover any of the input leaf nodes, none of the tree nodes in the subtree rooted by \( \text{node.right} \) can be in the minimal representer set. In this case \( \text{RepNodes}(\text{node},a) = \text{RepNodes}(\text{node.left},a) \). Note that \( \text{node.left.leftLeaf} = \text{node.leftLeaf} \). If \( \text{RepNodes}(\text{node.left},a) \) returns a minimal representer for \( \{ \text{node.left.leftLeaf},\cdots,(0,a) \} \), then \( \text{RepNodes}(\text{node},a) \) returns a representer for \( \{ \text{node.left.leftLeaf},\cdots,(0,a) \} \).

If \( \text{node.right} \) covers \((0,a)\), the algorithm returns \( \text{node.left} \cup \text{RepNodes}(\text{node.right},a) \). Let \( L_l = \{ \text{node.leftLeaf},\cdots,\text{node.left.rightLeaf} \} \) and \( L_r = \{ \text{node.right.leftLeaf},\cdots,(0,a) \} \). Note that \( \{ \text{node.leftLeaf},\cdots,(0,a)\} = L_l \cup L_r \). Again Lemma 1 holds for \( L_l \) and \( L_r \) as they share a parent node and \( \text{node.rightLeaf} \notin (L_l \cup L_r) \) is covered by the node. By Lemma 2, \( \{ \text{node.left} \} \) is a minimal representer for \( L_l \). By Lemma 1, if \( \text{RepNodes}(\text{node.right},a) \) is a minimal representer for \( L_r \), then \( \text{node.left} \cup \text{RepNodes}(\text{node.right},a) \) is a minimal representer for \( L_l \cup L_r \).

3) Proof of Theorem 5: By induction:

The base case is similar to the proof of Theorem 1. So we proceed with the inductive step.

If \( \text{node.right} \) does not cover any of the input leaves, no node in the subtree rooted by \( \text{node.right} \) can be in the representer. Thus \( \text{RepNodes}(\text{node},a_1,a_2) = \text{RepNodes}(\text{node.left},a_1,a_2) \). If the latter returns a representer, so does the former. Same goes for the case in which \( \text{node.right} \) does not cover any of the input leaves.

If both \( \text{node.left} \) and \( \text{node.right} \) cover some of the input leaves, then we can write \( \{(0,a_1),\cdots,(0,a_2)\} = \{(0,a_1),\cdots,\text{node.left.rightLeaf}\} \cup \{\text{node.right.leftLeaf},\cdots,(0,a_2)\} \).

So by Lemma 1, the minimal representer is the union of representers of the two subtrees and by Theorem 1, \( \text{RepNodes}(\text{node},a_1,a_2) = \text{RepNodes}(\text{node.left},a_1) \cup \text{RepNodes}(\text{node.right},a_2) \).