IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 29, 2022, accepted May 19, 2022, date of publication May 25, 2022, date of current version June 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3177841

Datalog Static Analysis in Secrecy

MOJGAN KOUHOUNESTANI AND WOOSUK LEE
Department of Computer Science and Engineering, Major in Bio Artificial Intelligence, Hanyang University, Ansan 15588, South Korea
Corresponding author: Woosuk Lee (woosuk@hanyang.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant 2020R1C1C1014518 and Grant NRF-2021R1A5A1021944, in part by the Institute for Information & Communications Technology Promotion (IITP) funded by the Korea Government (MSIT) under Grant 2021-0-00758, and in part by the Research Fund of Hanyang University under Grant HY-2020-2474.

ABSTRACT We present a secure static-analysis-as-a-service (SaaaS) system where a client may outsource static analysis to the cloud. To address copyright concerns associated with SaaaS, clients are allowed to encrypt the source code of a target program and upload it to the cloud. Our goal is to secure the privacy of the design and implementation of static analysis as well as the source code of the target program. Considering a family of static analyses written in Datalog, we propose a generic protocol that combines homomorphic encryption (HE) with secure two-party computation to manage the huge cost of HE operations. The server occasionally delegates sub-parts of analysis which are costly in the cipher-world to the client without exposing the design of analysis. During server-client interactions, the information of both sides (client and server) is not leaked to the opposite. We evaluated our system on two static analyses in Datalog in secrecy, which have not been feasible using the previous techniques. For example, Andersen pointer analysis is completed in an average of 45 mins for 14 C programs comprising up to 1.6 KLoC.

INDEX TERMS Datalog, homomorphic encryption, privacy-preserving software-as-a-service, static analysis, secure two-party computation.

I. INTRODUCTION

There have been two ways for a user to adopt a static analysis tool in software development to identify potential bugs for improving software quality. One way is to run the static analyzer in the user's self-hosted environment. In this way, the user downloads an open-source tool, or purchases/subscribes to a license to have free access to an on-premise version of a commercial tool. The other way is to use software-as-a-service (SaaS) applications [1]–[4] where the user uploads the target program to the cloud and gets the analysis result back.

From a perspective of two-party computation where two parties (here, the user and the analysis provider) jointly compute a result on their inputs, either of the two parties inevitably reveals its secret to the opposing party. On one hand, the design and implementation of a static analysis may be revealed to the user when the user has free access to the analyzer. On the other hand, the source code of the target program to be analyzed may be exposed to the analysis provider when the static analysis is outsourced to the cloud.

The associate editor coordinating the review of this manuscript and approving it for publication was Pedro R. M. InáCio.

If static analysis can be performed without sharing any information between the two (called *static analysis in secrecy* [5]), the usability of static analysis will be improved in the following two aspects.

- **No copy-right concerns**: The user can upload the target program to a static-analysis-as-a-service (SaaaS) system without any copy-right concerns. This gives better flexibility to the user who can promote the cost savings by the cloud (e.g., payment per Line of Code (LoC) [4]) rather than on-premise systems.
- **Security enhancement**: Static analysis-based malware detection mechanisms need to be unknown to malware developers as much as possible to prevent *evasive malware* [6], [7]. Attackers have a vested interest in crafting their code to evade the detection of analysis tools through mechanisms such as code obfuscation, while remaining effective at exploiting benign users. Maintaining security of detection mechanisms can hamper such attempts.

One immediate method for achieving this goal is to use Fully Homomorphic Encryption (FHE) [8]. FHE allows for computation on encrypted data without requiring the decryption key. Using FHE, the program owner can encrypt

and upload the program to the static analysis service while the service provider can still analyze the encrypted program without decryption. Only the owner of the decryption key (the program owner) can decrypt the encrypted analysis result. It is theoretically guaranteed that no information is shared between the two.

However, this method is not immediately applicable in practice due to the high complexity of HE operations. Practical deployments of HE applications require application-specific optimizations. In a wide range of application domains such as image recognition [9], statistical analysis [10], bioinformatics [11], and database [12], application-specific techniques have been proposed to make FHE applications practical.

A. EXISTING APPROACH

The first proof-of-concept study of static analysis in secrecy has been done by Lee *et al.* [5]. They proposed a method for *pointer analysis in secrecy*. In their approach, a pointer analysis algorithm is represented as a sequence of additions and multiplications over integers (i.e., arithmetic circuit), and a HE scheme is used to evaluate the arithmetic operations in the cipher-world.

The decisive performance bottleneck in homomorphic computation over arithmetic circuits is the nested depth of multiplications [5], which is simply the maximum number of sequential multiplications required to perform the computation. For example, the arithmetic circuit $c(x_1, x_2, x_3, x_4, x_5) = (((x_1x_2)x_3)x_4) + x_5$ has multiplicative depth 3. The lower the multiplicative depth is, the more efficiently a circuit can be evaluated. For example, we can optimize c by replacing it with $c'(x_1, x_2, x_3, x_4, x_5) = (x_1x_2)(x_3x_4) + x_5$ that has depth 2.

Despite application-specific techniques specialized for pointer analysis, they proposed to reduce the maximum multiplicative depth, analyzing even tiny programs (∼300 LoC) takes a huge amount of time to the high complexity of HE operations. The main reason for the high complexity is as follows. Static analysis performs a fixed point computation. Since the server cannot know whether a fixpoint is reached during the fixpoint iterations, it should repeat the iterations as if in the worst case. This worst-case fixpoint iteration significantly degrades the overall performance of homomorphic evaluations by necessitating a large multiplicative depth.

B. OUR APPROACH

Contrast to the prior work, we propose an efficient method for a broader class of static analyses in secrecy. We target static analyses written in a subset of Datalog. Thanks to its concise and declarative nature and powerful fixed point frameworks [13], [14], Datalog has been popularly used to write sophisticated static analyses for highly precise pointer analysis [15], detecting common security vulnerabilities [16] and malware [17] among many others.

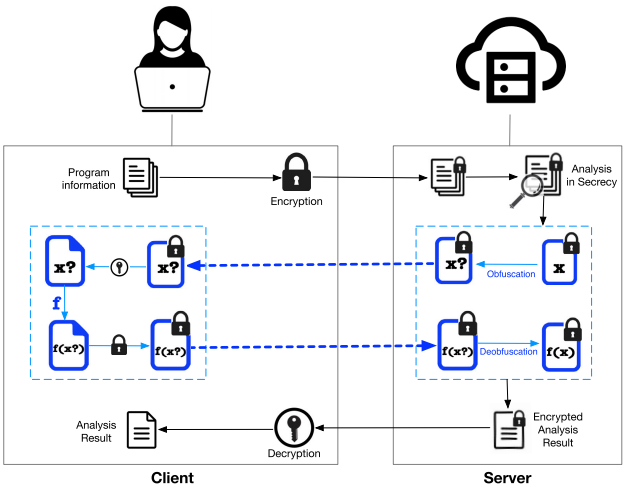


FIGURE 1. Overview of our protocol.

In particular, we consider a subset of Datalog where each recursive rule is of the form (known as *tail-recursive* or *linear recursive*)

$$R_1(u_1) \leftarrow R_2(u_2), R_1(u_3).$$

where the R_i are relation names (i.e., predicates) and the u_i are argument tuples. And there are no constraints over non-recursive rules. This family includes useful standard static analyses such as pointer analysis and escape analysis which will be detailed in Section II-C.

Our key insight for efficiency is to reduce the cost of homomorphic evaluation of a linear recursive rule by delegating a sub-part of computation which is costly in cipher-world to the client. The client can easily compute the sub-part, and send the result to the server, followed by HE operations to complete the rule evaluation on the server-side. During the interaction, the server and client never reveal their secret to the opposing party.

Figure 1 depicts our secure protocol. First, the client provides the program information to the server after encryption. Then, the server performs a static analysis on it by evaluating Datalog rules in cipher-world. During the evaluation in secrecy, the server occasionally delegates a sub-part of computation, namely a function f, which is costly in cipher-world to the client. This interaction is depicted in Figure 1 with the dotted blue strokes. Suppose the server wants to apply the f function to the ciphertext of x, which is an intermediate analysis result. A simple method for the delegation is to send the ciphertext of x to the client, who decrypts it, applies f, and sends back the ciphertext of the result to the server. However, x may contain partial information from which the client may infer the design of the analysis. To prevent such an information leak, the server homomorphically obfuscates x with a random one-time pad and obtains ciphertext of $x?$ before the delegation. After decryption, the client can infer no information about the analysis design from $x?$, which looks different from x. The

client applies the expensive function f, obtains $f(x?)$, and then sends back ciphertext of it to the server. The server homomorphically deobfuscates it to obtain the ciphertext of $f(x)$, which is the desired result. This process is repeated until the server obtains a final analysis result. After the analysis is complete, the server returns the encrypted analysis result to the client, who can inspect the final results after decryption.

We evaluate our method by performing two standard Datalog analyses in secrecy: escape analysis for Java being used in Petablox [18], a program analysis platform, and Andersen pointer analysis for C. Our method shows viable performance. For example, for 10 Java programs ($\sim$ 700 LoC), escape analysis in secrecy is completed in an average of 75 mins. For 14 C programs ($\sim$ 1.6 KLoC), our Andersen pointer analysis in secrecy is completed in an average of 45 mins. On the other hand, even for the smallest one (290 LoC) among the 14 programs, the previous method for Andersen pointer analysis in secrecy takes over 6 hours.

C. CONTRIBUTIONS

- A novel secure protocol for efficient evaluation of Datalog rules in secrecy: we combine a novel secure two-party computation protocol with homomorphic encryption. Our secure two-party computation protocol requires the client and the server to collaboratively perform computation. At the cost of only slight overhead on the client-side, our protocol is much more efficient than the previous approach solely based on FHE.
- Confirming the method's effectiveness: in the experiments with five Datalog analyses, the method shows viable performance which could not be achieved by the previous technique.

D. OUTLINE

Section II defines the problem of datalog static analysis in secrecy with the background concepts of homomorphic encryption and Datalog. Section III describes a basic protocol for datalog static analysis in secrecy, which is simple but inefficient. Section IV presents an advanced protocol that is much more efficient than the basic protocol in Section III. Section V presents the experimental studies. Section VI discusses related work. Section VII discusses how to further improve scalability and security of our approach and Section VIII concludes the paper.

II. PROBLEM DEFINITION

In this section, we define the problem of static analysis in secrecy. We first provide background on homomorphic encryption (Section II-A) and Datalog (Section II-B). In Section II-C, the problem is formally stated.

A. HOMOMORPHIC ENCRYPTION

Homomorphic encryption (HE) enables arbitrary computation over encrypted data. This enables secure outsourcing of computation where an untrusted third party, such as a cloud provider, performs computation over a client's private data without obtaining any (even partial) information of it.

In this paper, we consider HE schemes that operate over integers and are homomorphic concerning addition and multiplication. A fully homomorphic encryption (FHE) scheme with integer plaintext space $\mathbb{Z}_w = \{0, 1, \cdots, w-1\}$ (where w is a positive integer) can be described by the interface:

$$\mathsf{Enc}_{pk} : \mathbb{Z}_w \to \Omega \quad \mathsf{Dec}_{sk} : \Omega \to \mathbb{Z}_w$$
$$\mathsf{Add}_{pk} : \Omega \times \Omega \to \Omega \quad \mathsf{Mul}_{pk} : \Omega \times \Omega \to \Omega$$

where pk is a public key, sk is a secret key, Ω is a ciphertext space (e.g. large cardinality set such as $\mathbb{Z}_t$, i.e., integers modulo an integer t). For all plaintexts $m_1, m_2 \in \mathbb{Z}_w$, the interface should satisfy the following algebraic properties:

$$\mathsf{Dec}_{sk}(\mathsf{Add}_{pk}(\mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_2))) = m_1 + m_2,$$
$$\mathsf{Dec}_{sk}(\mathsf{Mul}_{pk}(\mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_2))) = m_1 \times m_2.$$

Example 1: As an example, let us consider a simple symmetric version (where only a secret key is used for both encryption and decryption) of the HE scheme [19] based on approximate common divisor problems [20]:

- *The secret key (sk) is a random integer p.*
- *For a plaintext m, $\mathsf{Enc}(m)$ outputs $pq + wr + m$, where q and r are randomly chosen integers such that $|r| \ll |p|$. r is a noise for ensuring semantic security [21].*
- *For a ciphertext $\bar{c}$, $\mathsf{Dec}(\bar{c})$ outputs $((\bar{c} \bmod p) \bmod w)$.*
- *For ciphertexts $\bar{c}_1$ and $\bar{c}_2$, $\bar{c}_1 \bar{+} \bar{c}_2$ outputs $\bar{c}_1 + \bar{c}_2$.*
- *For ciphertexts $\bar{c}_1$ and $\bar{c}_2$, $\bar{c}_1 \bar{\times} \bar{c}_2$ outputs $\bar{c}_1 \times \bar{c}_2$.*

For ciphertexts $\bar{c}_1 \leftarrow \mathsf{Enc}(m_1)$ and $\bar{c}_2 \leftarrow \mathsf{Enc}(m_2)$, we know each $\bar{c}_i$ is of the form $\bar{c}_i = pq_i + wr_i + m_i$ for some integer q_i and noise r_i. Hence $\mathsf{Dec}(\bar{c}_i) = (\bar{c}_i \bmod p) \bmod w) = m_i$, if $|wr_i + m_i| < p$. Then, the following equations hold:

$$\bar{c}_1 + \bar{c}_2 = p(q_1 + q_2) + \underbrace{w(r_1 + r_2) + m_1 + m_2}_{noise_{\mathsf{Add}}}$$
$$\bar{c}_1 \times \bar{c}_2 = p(pq_1q_2 + \cdots) + \underbrace{w(wr_1r_2 + r_1m_2 + r_2m_1) + m_1m_2}_{noise_{\mathsf{Mult}}}$$

Based on these properties, we can show that

$$\mathsf{Dec}(\bar{c}_1 + \bar{c}_2) = m_1 + m_2 \text{ and } \mathsf{Dec}(\bar{c}_1 \times \bar{c}_2) = m_1 \cdot m_2$$

if the absolute values of $noise_{\mathsf{Add}}$ and $noise_{\mathsf{Mult}}$ are less than p.

The random noise injected with ciphertexts to ensure security should be properly managed to certify the correctness of homomorphic evaluation. In the above example scheme, the noise in the resulting ciphertext increases during homomorphic addition and multiplication (twice and quadratically as much noise as before respectively). If the noise becomes larger than p, the decryption may return a random garbage value. To prevent such situations, one may set parameters (e.g., p in the example scheme) of HE schemes large enough. However, larger parameter values increase the

memory footprint of ciphertexts and require more compute resources to operate over large-sized ciphertexts.

Because multiplication dominates noise growth, the performance of homomorphic evaluation is often measured by the maximum *multiplicative depth* of evaluated arithmetic circuits. The maximum multiplicative depth influences the parameters of a HE scheme. Minimizing the multiplicative depth results in not only smaller ciphertexts but also less overall execution time. For example, if we support a few numbers of consecutive multiplications, the secret key p can be small in the above example HE scheme, leading to less overall computational costs. Many FHE schemes are *leveled* (also called *somewhat homomorphic*) in that, for fixed encryption parameters they only support computation of a particular depth.[1]

Note that any arithmetic circuit that performs up to n consecutive multiplications on ciphertexts can be evaluated by HE schemes supporting $O(\log n)$ multiplicative depth. For example, suppose we aim to acquire the product of an array of n ciphertexts. We can recursively cut the array in half and multiply the two halves, resulting in $O(\log n)$ depth.

B. DATALOG

Let us assume there are fixed sets of variables, constants, and relation symbols (i.e., predicates) each of which is associated with an arity. For brevity, we assume every constant is an integer.[2] An atom is an application of a relation symbol to a vector of variables and constants, e.g., $r(x, y)$ for a relation r with arity 2. A ground atom is an application of a relation symbol to constants. A Datalog rule R is an expression of the form:

$$A \text{ :- } B_1, B_2, \cdots, B_n.$$

where $A, B_1, \cdots, B_n$ are atoms. The literal to the left of the ":-" operator is called the *head*. On the other hand, the literals on the right hand side form the rule *body*. A Datalog rule can be interpreted as a logical implication where all variables are universally quantified: "For all variable valuations, if $B_1, \cdots, B_n$ are true, so is A". A Datalog program is a finite set of rules $P = \{R_1, R_2, \cdots, R_n\}$, and we will denote $P(r)$ as a set of rules of which head is of relation symbol r. There are two kinds of relation symbols: the input relations $\mathcal{I}$ whose contents are given, and the output reations $\mathcal{O}$ whose contents are derived from the input relations by evaluation of the program P. Only output relations can appear in the head of a rule. We use I to denote the set of facts (ground atoms) in the input relations. The Herbrand Base $\mathcal{B}$ denotes all possible applications of the output relations to vectors of constants in I. A Datalog program is recursive if a relation symbol appears in both the head and the body of a rule.

[1]A leveled scheme may be turned into a fully homomorphic one by introducing a bootstrapping operation [8], which is computationally heavy. Thus, we only consider somewhat HE schemes in this paper.

[2]We assume constants of other types can be transformed into integers using a numbering scheme (e.g., hashing).

Evaluating P on I yields a minimal Herbrand model, which is the smallest ground atoms satisfying the rules in P and input I. Informally, one can obtain such a model by starting with the input tuples I and repeatedly instantiating the variables of each rule to derive new output tuples, until no further conclusions can be added. We will write $P(I)$ for the set of output tuples (i.e., a minimal Herbrand model) produced by a Datalog program operating on a set of input tuples I.

C. PROBLEM FORMULATION

1) A SUBSET OF DATALOG OF INTEREST

In this work, we are interested in a subset of Datalog where each rule is either of the following forms:

$$r(\vec{x}_0) \text{ :- } r_1(\vec{x}_1), \ldots, r_n(\vec{x}_n). \tag{1}$$

where the $\vec{x}_i$ are vectors of variables of arity 2, or

$$r(x, y) \text{ :- } p(x, z), r(z, y). \tag{2}$$

In other words, every relation is binary without negation,[3] and every recursive rule is linear in that the head relation of a rule appears only once in the body of the rule. This subset of Datalog is called *linear Datalog* [22].

2) PROBLEM

We assume that a program owner (i.e., client) and an analysis provider (i.e., server) are *semi-honest*. In this security model, the client and the server run the protocol exactly as specified, but may try to learn as much as possible about the analysis design and the program information, respectively.

We aim to create a generic protocol that allows both parties to jointly perform the analysis on the target program with the following security requirements.

- The analysis provider can learn no information about the target program other than the program size (more precisely, the size of the Herbrand Base $\mathcal{B}$).
- The client can learn no information of the analysis other than input/output relation symbols (i.e., $\mathcal{I}$ and $\mathcal{O}$) and their associated arities (i.e., relational schema).

III. BASIC METHOD

In this section, we present a basic method for Datalog evaluation in secrecy which the prior work [5] is based on. In this method, an integer arithmetic circuit of a Datalog program is designed only using integer additions and multiplications. The program owner encrypts some numbers representing her program under a HE scheme. On the encrypted data, the server performs a series of corresponding homomorphic operations referring to the arithmetic circuit and outputs encrypted pointer analysis results. This basic approach is simple but very costly.

[3]Handling n-ary relations where $n > 2$ is left for future work. About negation, our approach is applicable to Datalog programs with *stratified negation*, as negated atoms $\neg r(x, y)$ can be encoded as $r'(x, y)$ such that $\forall x, y.\ \neg r(x, y) \iff r'(x, y)$ for a new relation symbol r'.

In the next section, a more efficient method improved upon the basic one will be described.

A. A LINEAR ALGEBRAIC APPROACH TO DATALOG

The basic method is based on a linear algebraic approach that encodes relations as matrices and obtains the least solution of a Datalog program P given input I by performing operations over the matrices. We begin with some notations.

1) NOTATIONS

We first introduce a matrix encoding of relations. From now on, matrices are denoted by boldface uppercase letters like "$\mathbf{A}$". In particular, $\mathbf{I}$ is an identity matrix, and $\mathbf{0}$ is the zero matrix. For a matrix $A = [a_{ij}]$, we write $(A)_{ij} = a_{ij}$ to denote the element of A at i-th row and j-th column. We introduce an operator norm for matrices, $|A|_\infty = \max_i \sum_j |a_{ij}|$, which means the maximum sum of entries in a row. In addition, we use a non-linear function notZero(x) defined by notZero$(x) = \begin{cases} 1 & x \neq 0 \\ x & \text{(o.w.)} \end{cases}$ For a matrix $A = [a_{ij}]$, we denote notZero(A) as the result of component-wise application of the notZero function into the entries, i.e., [notZero(a_{ij})]. Let N be the number of all constants considered in the Datalog program P. We introduce $N \times N$ matrices $\mathbf{R}_r \in \{0,1\}^{N\times N}$ for each relation symbol r to encode binary relations of form $r(\cdot,\cdot)$ where

$$(\mathbf{R}_r)_{ij} = \begin{cases} 1 & r(i,j) \in P(I) \\ 0 & \text{(otherwise.)} \end{cases}$$

In other words, if the entry of matrix $\mathbf{R}_r$ at i-th row and j-th column is not zero, we conclude $r(i,j)$ can be derived from the initial input I by evaluating the Datalog program P.

With these concepts, we are ready to formulate evaluation of Datalog programs as consecutive matrix operations. For rules of type (1), we introduce matrices $\mathbf{R}_r, \mathbf{R}_{r_1}, \cdots \mathbf{R}_{r_n}$ that encode relations of forms $r(\cdot,\cdot), r_1(\cdot,\cdot), \cdots, r_n(\cdot,\cdot)$ respectively. Then, the following equation should hold.

$$\mathbf{R}_r = \text{notZero}(\mathbf{R}^\circ_{r_1}\mathbf{R}^\circ_{r_2}\cdots\mathbf{R}^\circ_{r_n}) \tag{3}$$

where $\mathbf{R}^\circ_{r_i}$ is either $\mathbf{R}^T_{r_i}$ or $\mathbf{R}_{r_i}$.

Example 2: Consider the task of identifying sibling nodes in a directed graph. Two nodes x, y are sibling if there exist edges from x to z and y to z for some node z. The problem involves one input relation edge and one output relation sibling with the following meaning:

- *edge(x,y) : there is an edge from node x to y.*
- *sib(x,y) : node x are y are sibling.*

Suppose the user populates the input relation as follows:

$$edge(1,2), edge(2,3), edge(3,1), edge(4,1).$$

Then, $\mathbf{R}_{edge} = \begin{pmatrix} 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \\ 1&0&0&0 \end{pmatrix}$. *The Datalog rule for identifying siblings is*

$$sib(x,y) \text{ :- } edge(x,z), edge(y,z).$$

We can obtain the least solution as follows:

$$\mathbf{R}_{sib} = notZero(\mathbf{R}_{edge}\cdot\mathbf{R}^T_{edge}) = \begin{pmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&1 \\ 0&0&1&1 \end{pmatrix},$$

which means that we have the following output relations as a result.

$$sib(1,1), sib(2,2), sib(3,3), sib(4,4), sib(4,3), sib(3,4).$$

For a recursive rule of type (2), similarly we introduce matrices $\mathbf{R}_r$ and $\mathbf{R}_p$ encoding relations $r(\cdot,\cdot)$ and $p(\cdot,\cdot)$. We may well assume there is at least one non-recursive rule whose head involves r (otherwise, no ground atoms for r could be derived). Let $\mathbf{R}'_r$ denote the resulting matrix after evaluating the non-recursive rules. Then, for the recursive rule of type (2), the following equation should hold.

$$\mathbf{R}_r = \text{notZero}(\mathbf{R}'_r + \mathbf{R}_p\mathbf{R}_r) \tag{4}$$

The least solution of (4) gives $r(\cdot,\cdot)$ in the least model $P(I)$. Here, "least" means the ordering among matrices defined by for $A = [a_{ij}]$ and $B = [b_{ij}]$, $A \sqsubseteq B$ iff $\forall i,j.\ a_{ij} \leq b_{ij}$. To obtain the least solution, we define a series of monotonically increasing matrices $\{\mathbf{R}^{(k)}_r\}_{k=0,1,\cdots}$

$$\begin{aligned} \mathbf{R}^{(0)}_r &= 0 \quad \text{(matrix with every element 0)} \\ \mathbf{R}^{(k+1)}_r &= \text{notZero}(\mathbf{R}'_r + \mathbf{R}_p\mathbf{R}^{(k)}_r) \end{aligned} \tag{5}$$

$\{\mathbf{R}^{(k)}_r\}$ converges within N iterations. It is customary to prove that the limit $\lim_{k\to\infty}\{\mathbf{R}^{(k)}_r\}$ gives the least solution of (4).

Example 3: Suppose we want to compute transitive closure of the graph in Example 2. The problem involves one input relation edge described in Example 2 and one output relation path where path(x,y) means there exists a path from node x to y. The Datalog rules for computing transitive closure is

$$\begin{aligned} path(x,y) &\text{ :- } edge(x,y). \\ path(x,z) &\text{ :- } path(x,y), edge(y,z). \end{aligned}$$

Given the input relations in Example 2, we can obtain the least solution after 4 iterations (starting from $\mathbf{R}^{(0)}_{path} = 0$*). We conclude* $\{path(i,j) \mid 1 \leq i \leq 4, 1 \leq j \leq 3\}$ *is the set of final output relations.*

In summary, we can compute the least solution of P by repeatedly updating $\mathbf{R}_r$ for each output relation symbol r according to equations (3) and (5) until saturation.

B. BASIC PROTOCOL

Now we present the basic protocol for Datalog analysis in secrecy based on the linear algebra approach in the previous section.

To handle matrices, we make an additional assumption over an underlying HE scheme; it is capable of encrypting matrices and performing homomorphic operations over the matrices.

1) HOMOMORPHIC MATRIX OPERATIONS

Many modern HE schemes [23], [24] allow to efficiently encrypt matrices and perform various operations such as addition, multiplication, and transposition over the encrypted matrices. Those schemes allow a vector of plaintext integers to be encrypted into a single ciphertext with operations behaving in a SIMD manner (known as *ciphertext packing*). Therefore, one can represent a matrix as multiple vectors in either row-major order or column-major order, encrypt each vector, and efficiently perform homomorphic SIMD operations over the ciphertexts (see [25], [26] for more details).

Assuming such a HE scheme is available, from now on, we will denote an encrypted version of a matrix A as $\overline{A}$. With a slight abuse of notation, we will denote $+$ and $\cdot$ as homomorphic matrix-matrix addition and multiplication respectively to simplify the presentation. The homomorphic matrix transposition operator will be denoted as HE.MatTrans. In terms of noise growth, matrix multiplication is significant whereas the other operations are negligible.

Fig. 2 depicts the basic protocol. The server has a Datalog program P (i.e., analysis), and the client has a target program. The server and client share the information of the input and output relation symbols $\mathcal{I}$ and $\mathcal{O}$, and the number N of all constants in a target program.

The client first populates input relations from the target program, represents them as matrices, and encrypts the matrices and sends them to the server (**Program Encryption** in Fig. 2).

Next, the server performs Datalog analysis in secrecy (**Analysis in Secrecy** in Fig. 2). It initializes the matrices of output relations to be encrypted zero matrices (line 1), and repeats the main loop (lines 2 – 8) N^2 times. The main loop incrementally updates matrices of output relations by homomorphically evaluating each rule in P. If a currently considered rule whose head involves r is non-recursive (line 4), it updates $\overline{\mathbf{R}}_r$ according to the equation (3) (lines 5–6). If the rule is recursive (line 7), it updates $\overline{\mathbf{R}}_r$ according to the equation (5) (line 8). Note that the server updates $\overline{\mathbf{R}}_r$ N times because the server should evaluate the rule as if in the worst case. The main loop is repeated N^2 times because the server cannot know if a fixpoint is reached. Specifically, a single iteration of the main loop can add at least a single analysis fact (i.e., groud atom) to the analysis result computed so far, and we may have at most N^2 ground atoms for each output relation.

Lastly, the server sends the encrypted matrices for all the output relations to the client, who decrypts each matrix $\overline{\mathbf{R}}_r$ and interprets each non-zero element $\mathbf{R}_{ij}$ in the matrix as an analysis fact $r(i, j)$ (**Output Determination** in Fig. 2).

C. LIMITATION OF THE BASIC PROTOCOL

The basic protocol suffers from a scalability issue in practice due to the huge number of consecutive homomorphic multiplications. For example, if we perform Andersen's pointer analysis in secrecy using the basic protocol, we would need $O(N^2 \log N)$ multiplicative depth where N is the number of pointer variables in a target program [5]. A toy program having 10 variables requires an arithmetic circuit of depth 300, which is infeasible to be homomorphically evaluated even by the state-of-the-art HE schemes. Lee *et al.* [5] reduces the number of the main loop iterations (lines 2 – 8) from N^2 to n where n is the maximum level of pointer, which is very small (≤ 5) in usual C programs. Despite this domain-specific optimization reducing the depth from $O(N^2 \log N)$ to $O(n \log N)$, however, it takes over one day to analyze even a tiny program of 40 variables.

Basic Protocol

Server Input:
- A Datalog program P
- The sets $\mathcal{I}$ and $\mathcal{O}$ of input and output relations resp.

Client Input: Input tuples derived from a target program

Shared Information:
- $\mathcal{I}$ and $\mathcal{O}$
- N: the number of all constants

– Program Encryption (Client's work)
1) **for** $r \in \mathcal{I}$ **do** $\overline{\mathbf{R}}_r \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{R}_r)$
2) Send $\{\overline{\mathbf{R}}_r \mid r \in \mathcal{I}\}$ to the server.

– Analysis in Secrecy (Server's work)
1) **for** $r \in \mathcal{O}$ **do** $\overline{\mathbf{R}}_r \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{0})$
2) **for** $i = 1$ to N^2 **do**
3) **for** $R \in P$ **do**
4) **if** R is of form $r(\cdot,\cdot)$:- $r_1(\cdot,\cdot), \cdots, r_n(\cdot,\cdot)$ **then**
5) **for** $j = 1$ to n **do** $\overline{\mathbf{R}^T}_{r_j} \leftarrow \mathsf{HE.MatTrans}(\overline{\mathbf{R}}_{r_j})$
6) $\overline{\mathbf{R}}_r \leftarrow \overline{\mathbf{R}}_r + \overline{\mathbf{R}^\circ}_{r_1} \cdot \overline{\mathbf{R}^\circ}_{r_2} \cdots \overline{\mathbf{R}^\circ}_{r_n}$
7) **if** R is of form $r(\cdot,\cdot)$:- $p(\cdot,\cdot), r(\cdot,\cdot)$ **then**
8) **for** $j = 1$ to N **do** $\overline{\mathbf{R}}_r \leftarrow \overline{\mathbf{R}}_p + \overline{\mathbf{R}}_p \cdot \overline{\mathbf{R}}_r$
9) Send $\{\overline{\mathbf{R}}_r \mid r \in \mathcal{O}\}$ to the client

– Output Determination (Client's work)
1) **for** $r \in \mathcal{O}$ **do** $\mathbf{R}_r \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\overline{\mathbf{R}_r})$
2) Compute the analysis result $\{r(i,j) \mid r \in \mathcal{O}, (\mathbf{R}_r)_{ij} \neq 0\}$

FIGURE 2. Basic protocol for datalog analysis in secrecy.

IV. ADVANCED PROTOCOL

In this section, we present an advanced protocol that is much more efficient than the basic protocol in Section III The performance degradation of the basic protocol is caused by the following two factors that increase the overall multiplicative depth.

- *Inefficient evaluation of linear recursive rules*: For each linear recursive rule, the server should perform matrix-matrix multiplication N times consecutively.
- *Worst-case fixpoint iterations*: Since the server cannot know if a fixpoint is reached, the analysis in secrecy always assumes the worst-case regardless of actual inputs.

We will show how to improve the above two components with aid of the client. Section IV-A will describe how to more efficiently evaluate linear recursive rules.

Section IV-B will explain how to avoid the always-worst-case fixpoint iterations. Section IV-C will show a working example to facilitate understanding of our protocol. Lastly, Section IV-D will explain some optimizations additionally applied to the protocol to improve efficiency in practice.

A. EFFICIENT HANDLING OF LINEAR RECURSION

Recently, a more efficient linear algebraic approach to evaluation of linear recursive rules has been proposed [27]. Consider the following alternative to the equation (4)

$$\mathbf{R}_r = \epsilon(\mathbf{R}'_r + \mathbf{R}_p\mathbf{R}_r) \tag{6}$$

where ϵ is a small number such that $(I - \epsilon\mathbf{R}_p)^{-1}$ exists, for example $0 < \epsilon < \frac{1}{|\mathbf{R}_p|_\infty}$. The least solution of (6) is the limit of series $\{\mathbf{R}_r^k\}_{k=0,1,\cdots}$ which is defined to be

$$\mathbf{R}_r^{(0)} = 0$$
$$\mathbf{R}_r^{(k+1)} = \epsilon(\mathbf{R}'_r + \mathbf{R}_p\mathbf{R}_r^{(k)}).$$

In other words, $\mathbf{R}_r^{(\infty)} = \epsilon(\mathbf{R}'_r + \mathbf{R}_p\mathbf{R}_r^{(\infty)})$. Because $(I - \epsilon\mathbf{R}_p)^{-1}$ exists,

$$\mathbf{R}_r^{(\infty)} = (I - \epsilon\mathbf{R}_p)^{-1}\epsilon\mathbf{R}'_r. \tag{7}$$

Theorem 1 (From [27]): Suppose $0 < \epsilon \leq \frac{1}{|\mathbf{R}_p|_\infty+1}$. *The least solution of (4) is equal to the conversion of* $(I-\epsilon\mathbf{R}_p)^{-1}\epsilon\mathbf{R}'_r$ *to a 0-1 matrix by thresholding matrix entries at 0.*

Example 4: Recall the transitive closure computation in Example 3. As $|\mathbf{R}_{edge}|_\infty = max\{1, 1, 1, 1\} = 1$, *we can set* ϵ *to be* $(1 + |\mathbf{R}_{edge}|_\infty)^{-1} = \frac{1}{2}$. *We can obtain the least solution* $\mathbf{R}_{path}$ *as follows*:

$$(I - \epsilon\mathbf{R}_{edge})^{-1}\epsilon\mathbf{R}_{edge}$$
$$= \begin{pmatrix} 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ -\frac{1}{2} & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0.1428 & 0.5714 & 0.2857 & 0 \\ 0.2857 & 0.1428 & 0.5714 & 0 \\ 0.5714 & 0.2857 & 0.1428 & 0 \\ 0.5714 & 0.2857 & 0.1428 & 0 \end{pmatrix}.$$

By conversion to a 0-1 matrix, $\mathbf{R}_{path} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$.

This new method (7) for linear recursion is more efficient than the basic method (5) since the worst-case time complexity of computing (7) is $O(N^3)$, whereas that of the basic method is $O(N^4)$. Because the cost of homomorphic evaluation follows the worst-case complexity, we could take advantage of this new method to expedite our analysis in secrecy.

However, the problem here is that computing (7) involves matrix inversion (due to $(I - \epsilon\mathbf{R}_p)^{-1}$), and to the best of our knowledge, there is no known efficient method for homomorphic matrix inversion.[4]

We can solve this problem by delegating the matrix inversion to the client. To simplify notations from now on, let us introduce a function Prep which is defined as follows:

$$\text{Prep}()R \stackrel{\text{def}}{=} (I - \epsilon\mathbf{R})^{-1}.$$

Using this notation, our goal is written as to delegate computation of $\text{Prep}(\mathbf{R}_p)$ to the client. If the server gives the ciphertext of $\mathbf{R}_p$ to the client asking it for computing $\text{Prep}(\mathbf{R}_p)$ and sending back ciphertext of it, the server will be able to just multiply the result by ciphertext of $\mathbf{R}'_r$ to obtain the final evaluation result of the linear recursive rule. This trick would drastically reduce the depth by replacing the previous N matrix multiplications in (5) by single matrix multiplication.

Unfortunately, this method potentially introduces a security threat. The problem here is that $\mathbf{R}_p$ may contain partial information of the analysis design if it has been derived as an evaluation result of other Datalog rules. Thus, the client can potentially infer the analysis design by inspecting $\mathbf{R}_p$.

Our key idea to solve this security issue is to let the server obfuscate ciphertext of $\mathbf{R}_p$ before giving it to the client. The server and the client perform the following steps.

1) The server determines $\epsilon = \frac{1}{t}$ and shares it with the client. Here, t is a random large positive integer such that for every relation r, the norm $|\mathbf{R}_r|_\infty$ is smaller than t during the analysis.
2) The server generates a random non-zero matrix A such that $(I + A)^{-1}$ exists.
3) The server computes
$$\overline{\mathbf{R}}_p(\overline{I} + \overline{A}) - \frac{1}{\epsilon}\overline{A} \tag{8}$$
Let us denote the result by $\overline{T}$.
4) The server sends $\overline{T}$ to the client, who gets $\overline{T}$ and decrypts it.
5) The client computes $\text{Prep}(T)$, sends the result to the server after encryption. Let us call this encrypted matrix $\overline{K}$.
6) The server computes $(\overline{I} + \overline{A}) \cdot \overline{K}$.

Following the above steps, the server can get ciphertext of $\text{Prep}(\mathbf{R}_p)$ without any security issues for the following reason. When the client gets $\overline{T}$, due to the random matrix (i.e., one-time pad) A, it cannot infer any information of $\mathbf{R}_p$. And, what the server computes at the last step (6) gives ciphertext of $\text{Prep}(\mathbf{R}_p)$ because

$$K = (I - \epsilon T)^{-1}$$
$$= (I - \epsilon(\mathbf{R}_p + \mathbf{R}_pA - \frac{1}{\epsilon}A))^{-1}$$

[4]We are aware of a method for homomorphic computation of matrix determinant [28], but do not know of any prior work of efficient homomorphic matrix inversion.

$$= (I + A - \epsilon \mathbf{R}_p - \epsilon \mathbf{R}_p A)^{-1}$$
$$= ((I - \epsilon \mathbf{R}_p) \cdot (I + A))^{-1}$$
$$= (I + A)^{-1} \cdot (I - \epsilon \mathbf{R}_p)^{-1}$$

and

$$(I + A) \cdot K = (I + A) \cdot ((I + A)^{-1} \cdot (I - \epsilon \mathbf{R}_p)^{-1})$$
$$= (I - \epsilon \mathbf{R}_p)^{-1}$$

By multiplying it with a current $\overline{\mathbf{R}_r}$, the server can obtain a newly updated $\overline{\mathbf{R}_r}$ as a result of the homomorphic evaluation of the linear recursive rule. This method is reflected in the advanced protocol depicted in Fig. 3.

Remark: In many integer HE schemes, the plaintext space is often defined as non-negative integers modulo some integer w, i.e., $\mathbb{Z}_w$. When the server computes $\overline{T}$, the matrix may contain negative integers as $\frac{1}{\epsilon} = t$ in the expression (8) is a large integer. This issue can be easily solved by interpreting integers in the range of $[w/2, w)$ as negative ones. In other words, the plaintext can be adjusted to integers in $[-w/2, w/2)$. In addition, in the above step, K may contain real numbers since ϵ is a real number, thereby requiring homomorphic operations over real number entries on the server side. Modern integer FHE schemes can support fixed-point arithmetic with an explicit scaling factor. For instance, we can represent $\overline{3.14}$ as $\overline{314}$ with scale 100 ($\overline{n}$ denotes a ciphertext of n).[5] Lastly, care is needed when determining t. It should not be too large to prevent underflow when computing (8). If any matrix entry becomes smaller than $-\frac{w}{2}$, underflow occurs and the entry may suddenly become positive. Possible mitigation to this potential issue is to set w to be an even larger integer when determining the plaintext space so that it can be much greater than t.

B. AVOIDING THE ALWAYS-WORST-CASE ITERATIONS

We avoid the always-worst-case iterations by resorting to server-client interactions. We apply the semi-naive method [22] into our setting, but at a coarse granularity; we are at the level of relation symbols instead of ground tuples. Instead of using the full content of all the relations in each iteration as in the naive method, we only use the content of relations which may be updated in each iteration. Relation symbols of possible updates in each iteration can be easily identified from dependencies between relations. Specifically, the server first construct a *precedence graph* G_P of the Datalog program P prior to the analysis. The graph G_P visualizes dependencies between relations in P. Nodes in G_P are relation symbols, and edges are derived as follows: if A :- $\ldots B \ldots \in P$, $B \rightarrow A$ is added as an edge in G_P. By referring to the graph, the server can homomorphically evaluate each Datalog rule in order that guarantees the server does not evaluate rules that certainly will never generate new tuples (similarly to the worklist algorithm). In other words, the server visits each output relation in the graph in topological order and homomorphically evaluates rules involving the relation in turn.

Advanced Protocol

Server Input: same as Fig. 2
Client Input: same as Fig. 2
Shared Information:
- $\mathcal{I}$, $\mathcal{O}$, and N, the number of all constants
- ϵ: $\frac{1}{t}$ where t is a large positive integer

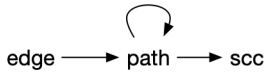
– Program Encryption (Client's work): same as Fig. 2
– Analysis in Secrecy (Server's work)
1) **for** $r \in \mathcal{O}$ **do** $\overline{\mathbf{R}}_r \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{0})$
2) **for** $r \in \mathcal{O}$ **do** $\overline{\mathbf{R}}_r^{old} \leftarrow \overline{\mathbf{R}}_r$
3) **for** $r \in \mathcal{O}$ in topological order of G_P **do**
4) **for** $R \in P(r)$ **do**
5) **if** R is of form $r(\cdot,\cdot)$:- $r_1(\cdot,\cdot), \cdots, r_n(\cdot,\cdot)$**then**
6) **for** $j = 1$ to n **do** $\overline{\mathbf{R}^T}_{r_j} \leftarrow \mathsf{HE.MatTrans}(\overline{\mathbf{R}}_{r_j})$
7) $\overline{\mathbf{R}}_r \leftarrow \overline{\mathbf{R}}_r + \overline{\mathbf{R}}^{\circ}{}_{r_1} \cdot \overline{\mathbf{R}}^{\circ}{}_{r_2} \cdots \overline{\mathbf{R}}^{\circ}{}_{r_n}$
8) **if** R is of form $r(\cdot,\cdot)$:- $p(\cdot,\cdot), r(\cdot,\cdot)$ **then**
9) $\overline{\mathbf{A}} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathbf{A})$ for a random matrix $\mathbf{A}$
10) $\overline{\mathbf{T}} \leftarrow \overline{\mathbf{R}}_p(\overline{\mathbf{I}} + \overline{\mathbf{A}}) - \frac{1}{\epsilon}\overline{\mathbf{A}}$
11) Send $\overline{\mathbf{T}}$ to the client and get $\overline{\mathbf{K}}$ from the client s.t. $\overline{\mathbf{K}} \equiv \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Prep}(\mathsf{Dec}_{\mathsf{sk}}(\overline{\mathbf{T}})))$
12) $\overline{\mathbf{R}_r} \leftarrow (\overline{\mathbf{I}} + \overline{\mathbf{A}}) \cdot \overline{\mathbf{K}} \cdot \overline{\mathbf{R}_r}$
13) Send $\mathcal{S} = \{\overline{\mathbf{A}_i} \cdot (\overline{\mathbf{R}}_r - \overline{\mathbf{R}}_r^{old}) \mid r \in \mathcal{O}, i \in [1, |\mathcal{O}|]\}$ to the client for checking termination condition ($\mathbf{A}_i$ are distinct random matrices filled with positive integers)
14) **if** the client responds "Yes" **then**
15) Sends $\{\overline{\mathbf{R}}_r \mid r \in \mathcal{O}\}$ to the client
16) **else** goto step (2)
– Checking Termination Condition (Client's work):
1) **Send** "Yes" to the server iff $\forall \overline{\mathbf{M}} \in \mathcal{S}.\ \mathsf{Dec}_{\mathsf{sk}}(\overline{\mathbf{M}}) = \mathbf{0}$
– Output Determination (Client's work): same as Fig. 2

FIGURE 3. Advanced protocol for datalog analysis in secrecy. Lines 11 and 13 in Analysis in Secrecy are parts involving the client intervention.

Example 5: *Consider the following rules for computing all strongly connected components in a directed graph.*

$$path(x, y) \text{ :- } edge(x, y).$$
$$path(x, z) \text{ :- } path(x, y), edge(y, z).$$
$$scc(x, y) \text{ :- } path(x, y), path(y, x).$$

where scc(x,y) denotes nodes x and y are strongly connected to each other – i.e., there is a path from x to y and vice versa. The precedence graph for the rules is as follows:



Sorting the relations topologically tells us to evaluate the two rules for path first, and then the rule for scc next.
After evaluating all the rules once, the server asks the client if a fixpoint is reached. Please note that the precedence graph

[5]The scaling factor may quickly grow with multiplication. The state-of-the-art integer FHE schemes such as CKKS [23] enables "rescaling" to address this issue. For example, it can convert $\overline{2000}$ at fixed-point scale 100 to $\overline{20}$ at scale 1.

can be constructed only by the server who knows about the rules and the graph is not revealed to the client.

Fig. 3 depicts the final protocol with this idea. The server derives graph G_P from P, and stores content derived so far for every relation (line 2), which will be used for checking the termination condition later. The server visits each output relation r in topological order of G_P (line 3), and homomorphically evaluates each rule whose head involves r. After evaluating all the rules, the server determines whether or not the evaluation in secrecy should continue. The server computes the set of encrypted matrices $\mathcal{S}$ as specified on line 13, and sends it to the client. If there has been no update in the content of r, $\overline{\mathbf{R}}_r - \overline{\mathbf{R}}_r^{old}$ will be an encrypted zero matrix. Otherwise, it will be an encrypted non-zero matrix, which may contain partial information of the analysis. Again, we use random one-time pad matrices (i.e., A_i in the protocol). If all the matrices from the server are zero matrices, the client responds "Yes" to the server meaning a fixpoint has been reached, which makes the protocol end. Otherwise, the server repeats the process again by going back to line (2) (line 16). The protocol continues until a fixpoint is reached. As we will show in the evaluation, the number of server-client interactions is small in practice.

The overall multiplicative depth required for the advanced protocol is $O(k\ell \log n)$, where ℓ is the length of the longest path in the precedence graph, n is the maximum length of bodies in the rules, and k is the number of iterations of the main loop (lines 2 – 16) until a fixpoint is reached.

C. WORKING EXAMPLE

We present a working example using the Andersen pointer analysis for a better understanding of our protocol.

1) DATALOG RULES

The pointer analysis involves four input relations `pt0`, `copy0`, `load`, `store`, and two output relations `pt` and `copy` with the following meanings.

- `pt0(x,y)` : there is an assignment `x := &y` in a given input program.
- `copy0(x,y)` : there is an assignment `x := y`.
- `load(x,y)` : there is a load statement `y := *x`.
- `store(x,y)` : there is a store statement `*x := y`.
- `pt(x,y)` : `x` may point to `y`.
- `copy(x,y)` : if `y` may point to some variable `v`, then `x` also may point to `v`.

For brevity, we abbreviate `copy`, `store`, and `load` to `cp`, `st`, and `ld`, respectively. The following Datalog rules specify the pointer analysis.

$$\texttt{pt(a, b)} \;\texttt{:-}\; \texttt{pt0(a, b).} \tag{9}$$
$$\texttt{pt(a, b)} \;\texttt{:-}\; \texttt{cp(a, c), pt(c, b).} \tag{10}$$
$$\texttt{cp(a, b)} \;\texttt{:-}\; \texttt{cp0(a, b).} \tag{11}$$
$$\texttt{cp(a, b)} \;\texttt{:-}\; \texttt{st(c, b), pt(c, a).} \tag{12}$$
$$\texttt{cp(a, b)} \;\texttt{:-}\; \texttt{ld(c, a), pt(c, b).} \tag{13}$$

2) PROGRAM ENCRYPTION

Suppose we aim to analyze the following simple C program.

```
int** a;   int *b, *c, *d;
a = &b;    *b = d;      c = b;      c = *d
```

and we assign an integer ID to each variable as follows: $\texttt{a} \mapsto 1, \texttt{b} \mapsto 2, \texttt{c} \mapsto 3, \texttt{d} \mapsto 4$. The client populating input relations obtains the following matrices encoding them.

$$\mathbf{R}_{\texttt{pt0}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{R}_{\texttt{cp0}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$
$$\mathbf{R}_{\texttt{ld}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad \mathbf{R}_{\texttt{st}} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The client encrypts the matrices and sends them to the server.

3) ANALYSIS IN SECRECY

The server determines ϵ to be 10^{-3} and lets the client know about it. The server initializes each of $\overline{\mathbf{R}}_{\texttt{pt}}$, $\overline{\mathbf{R}}_{\texttt{pt}}^{old}$, $\overline{\mathbf{R}}_{\texttt{cp}}$, and $\overline{\mathbf{R}}_{\texttt{cp}}^{old}$ to be an encrypted zero matrix. The server performs assignments $\overline{\mathbf{R}}_{\texttt{cp}} \leftarrow \overline{\mathbf{R}}_{\texttt{cp0}}$ and $\overline{\mathbf{R}}_{\texttt{pt}} \leftarrow \overline{\mathbf{R}}_{\texttt{pt0}}$, which are for homomorphic evaluation of the rules (9) and (11), respectively. The server generates a ciphertext of random matrix $\mathbf{A}$ such that $I + A$ is invertible and a ciphertext of $\mathbf{T}$ of which entries are as follows:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -10^3 & 0 & 0 & -10^3 \end{pmatrix}$$

The client gets the ciphertext $\overline{T}$, decrypts it, and computes $K(= \text{Prep}(T))$ which is $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 10^{-3} & 1 & 0 \\ -2^{-1} & 0 & 0 & 2^{-1} \end{pmatrix}$. The server gets a ciphertext of K and homomorphically evaluates the rule (10) by performing the following assignment.

$$\overline{\mathbf{R}}_{\texttt{pt}} \leftarrow (\overline{I} + \overline{A}) \cdot \overline{K} \cdot \overline{\mathbf{R}}_{\texttt{pt}} = \overline{\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}$$

Next, to evaluate the rules (12) and (13), the server performs assignment

$$\overline{\mathbf{R}}_{\texttt{cp}} \leftarrow \overline{\mathbf{R}}_{\texttt{cp}} + \overline{\mathbf{R}}_{\texttt{pt}}^T \cdot \overline{\mathbf{R}}_{\texttt{st}} + \overline{\mathbf{R}}_{\texttt{ld}}^T \cdot \overline{\mathbf{R}}_{\texttt{pt}}$$

and obtains $\overline{\mathbf{R}}_{cp} = \overline{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}}$. Because every Datalog rule has been visited once, now is the time for checking if

a fixpoint is reached. The server generates random matrices A_1 and A_2 with every entry positive integer and sends the following set of encrypted matrices to the client.

$$\mathcal{S} = \{\overline{A}_1 \cdot (\overline{\mathbf{R}}_{\mathtt{pt}} - \overline{\mathbf{R}}^{old}_{\mathtt{pt}}), \overline{A}_2 \cdot (\overline{\mathbf{R}}_{\mathtt{cp}} - \overline{\mathbf{R}}^{old}_{\mathtt{cp}})\}$$

The client decrypts the matrices and confirms there exist non-zero entries. Since the termination condition has not been met, the process should return to line (2) of the protocol to begin the next round.

Next, $\overline{\mathbf{R}}^{old}_{\mathtt{cp}}$ and $\overline{\mathbf{R}}^{old}_{\mathtt{pt}}$ are assigned $\overline{\mathbf{R}}_{\mathtt{cp}}$ and $\overline{\mathbf{R}}_{\mathtt{pt}}$, respectively. The server generates a one-time pad matrix $\mathbf{A}$ similarly to the previous round and ciphertext of $\mathbf{T}$ of which entries are as follows:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1000 \\ 0 & 1 & 0 & 1 \\ 0 & -1000 & 0 & -1000 \end{pmatrix}$$

The client gets $\overline{T}$, and computes $K = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 \\ 0 & 10^{-3} & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}$. The server gets $\overline{K}$, and homomorphically evaluates the rule (10) similarly to the previous round. As a result, there is no update in $\mathbf{R}_{\mathtt{pt}}$ and $\mathbf{R}_{\mathtt{cp}}$. Because $\mathbf{R}_{\mathtt{pt}}$ and $\mathbf{R}_{\mathtt{cp}}$ are equal to their old versions, the set $\mathcal{S}$ will only contain ciphertexts of zero matrices. The client checks that all the matrices in $\mathcal{S}$ are zero matrices, and transmits ''Yes'' to the server as the termination condition has been met. The server sends $\overline{\mathbf{R}}_{\mathtt{pt}}$ and $\overline{\mathbf{R}}_{\mathtt{cp}}$ to the client as the encrypted final analysis result.

D. OPTIMIZATIONS

1) DELEGATION OF COMPUTATIONS OVER INPUT RELATIONS TO THE CLIENT

To reduce the number of communication rounds required for analysis in secrecy, the client may compute $\mathrm{Prep}(\mathbf{R}_r)$ for each input relation r in advance and send them to the server. Similarly, to reduce the cost for homomorphic matrix transposition on the server-side, the client may compute $\mathbf{R}^T_r$ for every input relation r in advance and send them to the server.

2) FAKE INTERACTIONS

The number of communication rounds until a fixpoint is reached may expose partial information of the analysis design to the client. To prevent a leak of partial information, the server may perform fake interactions. Even if analysis in secrecy does not require client-server interactions, we compel it to perform meaningless ones. The time spent by both for such a fictitious interaction is negligible and has almost no effect on total processing time.

V. EVALUATION

We implemented our method as a tool named SecureDL.[6] This section evaluates our SecureDL system to answer the questions:

Q1: Given the level of overhead in HE computation, can SecureDL show viable performance for standard Datalog static analyses in secrecy?

Q2: How does SecureDL compare with the existing technique for static analysis in secrecy which does not involve any client intervention during analysis?

All of our experiments were conducted on Linux machines with 40 cores of Intel Xeon 2.6GHz CPU and 256G of memory.

A. EXPERIMENTAL SETUP

1) TARGET ANALYSES

For our evaluation, we use the following two static analyses excerpted from the benchmarks used in [29]. All of the analyses are written in the subset of Datalog described in Section II-C.

- **escape** is an escape analysis for Java. This analysis consists of 11 Datalog rules (6 input relations and 3 output relations), including recursive rules with a multiplicative depth of 5.
- **andersen** is a classic pointer analysis for C. It has overall 5 Datalog rules which contain recursive rules with a multiplicative depth of 10 (4 input relations and 2 output relations).

We chose these analyses for the following reasons. First, the analyses are standard and fundamental in that other more complex static analyses are built upon them [18]. Given the significant cost of HE, showing the viability of our method in these analyses is the first necessary step for more complex analyses in secrecy. Second, the analyses are non-trivial in terms of homomorphic evaluation in that they comprise up to 11 Datalog rules and 6 input relations with recursion (**escape**), thus good targets for testing the efficiency of our method.

2) IMPLEMENTATION

We have implemented two modules for server and client. The client module populates input relations from syntax of a given target program to be analyzed and communicates with the server module as specified in our protocol. The server module takes Datalog rules of a static analysis and performs analysis in secrecy interacting with the client module as specified in the protocol. We use the latest version of HElib [30], an open-source library that implements HE. Among the HE schemes the library provides, we use the BGV scheme [24]. To derive input relations from a target program, we use the front-end of Petablox [18] for Java and Sparrow [31] for C. Petablox and Sparrow are open-source static analysis frameworks. Our implementation runs in parallel using the multi-threading supported by HElib.

[6] A System for **Secure** Evaluation of **D**ata**L**og.

TABLE 1. Result for escape analysis for Java. The timeout is set to be 24 hours. *N* gives the number of all Datalog constants (i.e., all Java program variables in case of escape). **Client** and **Server** give the time taken by the client and server modules respectively.

Programs			Time	
Name	N	LoC	Client	Server
airlinetickets	216	100	9.8 s	30m 28s
bubblesort2	216	131	9.8 s	30m 31s
sunsaccount	221	159	11.3 s	37m 23s
bufwriter	248	251	12.9 s	30m 21s
manager	306	194	16.3 s	1h 26m 7s
account	352	168	21.8 s	1h 57m 47s
mergesort	359	377	21.1 s	2h 20m 34s
pingpong	377	281	20.8 s	1h 34m 45s
shop	401	296	23.4 s	1h 3m 44s
tsp	608	706	41.7 s	2h 25m 20s

3) TARGET PROGRAMS

Our benchmark of target programs comprises 10 Java programs and 14 C programs. The details can be found in Table 2 and 1. The Java programs are from the evaluation benchmarks [32] used along with Petablox. They originated from programming assignments in a software course. The C programs are from the benchmark by Zitser *et al.* [33] that include 14 source code examples containing serious buffer overflow vulnerabilities found in three security-sensitive applications. Though the programs are small, they exhibit diverse language features common in practice.

4) BASELINE

We compare SecureDL to the work by Lee *et al.* [5], which is the only prior work of static analysis in secrecy to the best of our knowledge. The approach is similar to the basic protocol described in Figure 2 along with optimizations specialized for the pointer analysis. We conduct an experimental comparison only for the pointer analysis for C since the prior work only targets the pointer analysis. To compare SecureDL against the previous approach, we run the authors' implementation on our benchmark programs. The implementation of the prior work obtained from the authors also uses the same version of HElib with the BGV scheme, thus the comparison is on equal footing.

B. PERFORMANCE OF SecureDL

We evaluate SecureDL on the five static analyses. For each analysis of each target program, we measure the time taken by the client and server modules. The timeout limit is set to 24 hours. In all the analyses, we set the security parameter 72, which means a ciphertext can be broken in a worst-case time proportional to 2^{72}. For each analysis, we checked the correctness by comparison to the result of plaintext analysis using a Datalog solver [13].

Table. 1 shows the results for the four Java analyses, and Table 2 summarizes the result for Andersen pointer analysis for C. The column **Client** gives the time for all the parts entitled "Client's work" in the protocol depicted in Fig. 3 along with the computation of $\overline{K}$ on line (11). The column (**Server**) gives the time taken for all the parts entitled "Server's work". All of the analyses could be completed within a small number of communication rounds (≤ 3).

TABLE 2. Result for Andersen pointer analysis for C. *N* gives the number of all Datalog constants (i.e., pointer variables). The timeout is set to be 24 hours.

Programs			SECUREDL		Lee et al. [5]	
Name	LoC	N	Client	Server	Client	Server
bind-1	1300	92	14.2s	2h 8m 22s	**timeout**	
bind-2	1634	104	16.1s	1h 8m 53s	**timeout**	
bind-3	344	26	4.2s	5 m	**timeout**	
bind-4	555	55	8.7s	47m 39s	**timeout**	
sm-1	884	56	8.5s	50m 22s	**timeout**	
sm-2	570	39	6.1s	36m 45s	**timeout**	
sm-3	592	52	8.2s	1h 56m 15s	**timeout**	
sm-4	620	34	5.4s	15m 29s	**timeout**	
sm-5	691	43	6.7s	57m 31s	**timeout**	
sm-6	290	13	2s	25s	46s	6h 21m
sm-7	1206	96	14.9s	1h 33m 1s	**timeout**	
ftp-1	336	21	3.2s	1m 9s	**timeout**	
ftp-2	996	28	4.3s	1m 51s	**timeout**	
ftp-3	642	39	6.1s	12m 22s	**timeout**	

SecureDL is able to analyze all the programs in all the analyses within the timeout limit. As far as the time taken by the client is concerned, it is always negligible compared to the time taken by the server. The time taken by the server is also roughly proportional to N, but not always. This variance is due to "ciphertext packing" supported by the underlying HE scheme described in Section III-B. How many plaintext messages can be "packed" into a single ciphertext is called the number of *slots* (namely l), and it affects the overall performance of homomorphic matrix operations. The performance variance is due to that it is non-trivial to set l to a number we want.

In summary, SecureDL could perform all the analyses on every target program within the timeout limit, which shows the viability of our method.

C. COMPARISON TO THE BASELINE

As can be seen in Table 2, our method significantly outperforms the previous approach [5] by reducing the required multiplicative depth thanks to the client intervention during analysis. Only one out of 14 programs could be analyzed by Lee *et al.*, whereas our method could analyze all the programs within the timeout limit. The important advantage of SecureDL against the prior work is that the depth in our protocol is determinedly independently of the number of constants N, i.e., the number of pointer variables in each target program. In all the programs, our Andersen analysis in secrecy could be performed with a depth 10. On the other hand, the depth required for Lee *et al.* [5] is proportional to N. For example, the smallest program sm-6 requires a depth of 43. The programs bind-1, bind-2, sm-3, and sm-7 require depth of 61, and the other remaining programs require depth of 55. Because ciphertext sizes are non-linearly proportional to depth in general [34],

the significant differences in the depths lead to the remarkable performance gap between the two methods.

VI. RELATED WORK

A. STATIC ANALYSIS IN SECRECY

To the best of our knowledge, the work by Lee *et al.* [5] is the only prior work of static analysis in secrecy. As a first step, they present a HE algorithm for the simple Andersen pointer analysis. To expedite the analysis, they propose a way to perform pointer analysis level-by-level, reducing the overall multiplicative depth of the analysis as already described in Section III-C. Our method is more generally applicable to static analyses written in the subset of Datalog described in Section II-C, and even without any domain-specific optimization techniques, it outperforms the previous method as shown in the evaluation. However, our protocol may require more communication cost than the prior work by delegating sub-parts of analysis to the client.

B. COMBINING HOMOMORPHIC ENCRYPTION AND SECURE MULTI-PARTY Computation (MPC)

In various application domains, there have been attempts to combine homomorphic encryption and secure MPC in order to reduce the number of homomorphic operations solely performed by the server. Gazelle [35], SecureML [36], and MiniONN [37] are secure neural network inference system that combines homomorphic encryption with secure two-party computation techniques. Another line of work employed HE for arithmetic operations and resort to Yao's Garbled Circuit (GC) for the other operations that can be represented as Boolean circuits. Nikolaenko *et al.* [38] propose a secure matrix factorization method based on this approach. Blanton *et al.* [39] also follow this approach and propose a secure protocol for biometric identification of which goal is to enable secure biometric data matching performed by two distrusted parties, one of which holds one biometric image while the other owns a possibly large biometric collection.

Although combining homomorphic encryption with secure MPC has been explored in various domains, to the best of our knowledge, we are the first to adopt this approach to static analysis in secrecy.

C. COMPUTATION ON ENCRYPTED DATA

DFAuth [40] minimizes potential attack surfaces when employing a secure enclave by operating over data encrypted with partially homomorphic encryption (PHE) schemes on an untrusted server. In the trusted module, switching to different PHE schemes is done to perform diverse operations on the server. While promising in terms of performance, partial information of target programs may be leaked because an attacker would be able to observe the control flow of the analysis algorithm that runs on the server.

Our new approach ensures that no information is leaked to opposing parties.

VII. DISCUSSION

A. FOR BETTER SCALABILITY

While we believe the experimental result is indicative of the viability of our idea, there is still a long way to go toward practical use. We discuss possible ways to scale to analyses at a larger scale. First, clients can help to improve the scalability by encrypting only sensitive sub-parts of their target programs. The other parts are provided without encryption. In this case, analysis operations with the mixture of ciphertexts and plaintexts can be used (e.g., homomorphically multiply a ciphertext by a plaintext integer). These kinds of operations are much more efficient than operations between ciphertexts incurring smaller noise increases. Second, we may improve the overall performance by taking advantage of the latest results for more efficient homomorphic matrix multiplication [41], [42]. Currently, we are using a standard method [25] also used by Lee *et al.* [5]. Integrating the state-of-the-art HE algorithms for matrix multiplication is left for future work. Lastly, constant developments and advances in FHE and much room for parallelization are other opportunities. Because each homomorphic matrix operation used in our algorithm is an embarrassingly parallel workload (as shown in the experiment, we have used 40 cores to run the protocol in parallel), we can easily boost efficiency further by using more threads.

B. PREVENTING EVASIVE MALWARE

Static analysis in secrecy can be potentially used for detecting malware without revealing the detection mechanism to malware developers. Only input and output of a Datalog-based static analysis are revealed to the user so that the user cannot infer any partial information of the entire Datalog rules. Learning Datalog programs from input-output samples (a process known as *inductive logic programming*) is a difficult endeavor; even approximate learning is difficult [43]. Therefore, the encryption procedure protects the analysis algorithm against attackers to maintain the algorithm's privacy.

C. ABOUT THE CLIENT INTERVENTION

The client's operations for the server do not impose significant overhead on the client since the operations are rather simple (matrix inversion and checking if a zero matrix is sent from the server). In addition, we have shown the number of communication rounds is small in practice, so that the number of client operations can be also small. If the client is not available during analysis, the server can fall back to the basic protocol described in Figure 2. However, in that case, the analysis would be much more costly than that with the advanced protocol currently adopted by SecureDL.

D. ABOUT SECURITY

In our protocol, the client can learn an upper bound of the size of the server's Datalog rules. The overall multiplicative depth

should be determined by the server before analysis and known by the client for encryption of the target program. Since the depth is $O(k\ell \log n)$ with the number k of communication rounds, the length ℓ of the longest path in the precedence graph, and the maximum length n of bodies of the rules, the client can learn about an upper bound of the size of the server's Datalog rules. However, this security threat can be easily mitigated by setting the depth larger than actually necessary along with fake interactions already described in Section IV-D.

On the other hand, the server can learn an upper bound of the size of the target program (more precisely, the number N of Datalog constants, e.g., Java program variables in case of escape analysis). This security threat can be also easily mitigated by the client's setting N larger than actually necessary.

Other than that, there is no other information revealed since all messages are encrypted under the BGV-type cryptosystem which is secure under the hardness of the ring learning with errors (RLWE) problem (see [24] for the details).

VIII. CONCLUSION

We have shown a secure static-analysis-as-a-service (SaaaS) system where a client may outsource static analysis to the cloud without privacy concerns. Our method ensures privacy of the design and implementation of static analysis as well as the source code of the target program. Considering a family of static analyses written in Datalog, we propose a generic protocol that combines homomorphic encryption (HE) with secure two-party computation to manage the huge cost of HE operations. We have demonstrated the viability of our method by evaluating our system using Andersen pointer analysis for 14 C programs and escape analysis for 10 Java programs.

REFERENCES

[1] *Sparrow Cloud*. Accessed: Apr. 22, 2022. [Online]. Available: https://cloud.sparrowfasoo.com/

[2] *DeepScan*. Accessed: Apr. 22, 2022. [Online]. Available: https://deepscan.io

[3] *Coverity Scan*. Accessed: Apr. 22, 2022. [Online]. Available: https://scan.coverity.com

[4] *Sonar Cloud*. Accessed: Apr. 22, 2022. [Online]. Available: https://sonarcloud.io

[5] W. Lee, H. Hong, K. Yi, and J. H. Cheon, "Static analysis with set-closure in secrecy," in *Proc. Int. Static Anal. Symp.* Berlin, Germany: Springer, 2015, pp. 18–35.

[6] D. Kirat and G. Vigna, "MalGene: Automatic extraction of malware analysis evasion signature," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 769–780, doi: 10.1145/2810103.2813642.

[7] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *Proc. 22nd USENIX Secur. Symp. (USENIX Security)*. Washington, DC, USA: USENIX Association, Aug. 2013, pp. 637–652. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/kapravelos

[8] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. (STOC)*, New York, NY, USA, 2009, pp. 169–178, doi: 10.1145/1536414.1536440.

[9] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, New York, NY, USA, Jun. 2019, pp. 142–156, doi: 10.1145/3314221.3314628.

[10] W. Lu, S. Kawasaki, and J. Sakuma, "Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, San Diego, CA, USA, Feb. 2017, doi: 10.14722/ndss.2017.23119.

[11] J. H. Cheon, M. Kim, and K. Lauter, "Homomorphic computation of edit distance," in *Financial Cryptography and Data Security*, M. Brenner, N. Christin, B. Johnson, and K. Rohloff, Eds. Berlin, Germany: Springer, 2015, pp. 194–212.

[12] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," in *Applied Cryptography and Network Security*, M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds. Berlin, Germany: Springer, 2013, pp. 102–118.

[13] H. Jordan, B. Scholz, and P. Subotić, "Soufflé: On synthesis of program analyzers," in *Computer Aided Verification*, S. Chaudhuri and A. Farzan, Eds. Cham, Switzerland: Springer, 2016, pp. 422–430.

[14] M. Madsen, M.-H. Yee, and O. Lhoták, "From datalog to flix: A declarative language for fixed points on lattices," in *Proc. PLDI*, 2016, pp. 194–208.

[15] Y. Smaragdakis and M. Bravenboer, "Using datalog for fast and easy program analysis," in *Datalog Reloaded*, O. de Moor, G. Gottlob, T. Furche, and A. Sellers, Eds. Berlin, Germany: Springer, 2011, pp. 245–251.

[16] D. Avots, M. Dalton, V. Benjamin, and M. S. Lam, "Improving software security with a c pointer analysis," in *Proc. 27th Int. Conf. Softw. Eng. (ICSE)*, 2005, pp. 332–341.

[17] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of Android malware through static analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 576–587, doi: 10.1145/2635868.2635869.

[18] *Petablox*. Accessed: Apr. 22, 2022. [Online]. Available: https://github.com/petablox/petablox

[19] M. V. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Proc. EUROCRYPT*, 2010, pp. 24–43.

[20] N. Howgrave-Graham, "Approximate integer common divisors," in *Proc. CaLC*, 2001, pp. 51–66.

[21] G. Oded, *Foundations of Cryptography: Basic Applications*, vol. 2, 1st ed. New York, NY, USA: Cambridge Univ. Press, 2009.

[22] T. J. Green, S. S. Huang, B. T. Loo, and W. Zhou, *Datalog and Recursive Query Processing*. Boston, MA, USA: Now, 2013.

[23] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology*, T. Takagi and T. Peyrin, Eds. Cham, Switzerland: Springer, 2017, pp. 409–437.

[24] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd Innov. Theor. Comput. Sci. Conf. (ITCS)*, New York, NY, USA, 2012, pp. 309–325, doi: 10.1145/2090236.2090262.

[25] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology—CRYPTO*, J. A. Garay and R. Gennaro, Eds. Berlin, Germany: Springer, 2014, pp. 554–571.

[26] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Des., Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[27] T. Sato, "A linear algebraic approach to datalog evaluation," *Theory Pract. Log. Program.*, vol. 17, no. 3, pp. 244–265, May 2017.

[28] H. Zong, H. Huang, and S. Wang, "Secure outsourced computation of matrix determinant based on fully homomorphic encryption," *IEEE Access*, vol. 9, pp. 22651–22661, 2021.

[29] X. Si, W. Lee, R. Zhang, A. Albarghouthi, P. Koutris, and M. Naik, "Syntax-guided synthesis of datalog programs," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*. New York, NY, USA: Association for Computing Machinery, Oct. 2018, p. 515, doi: 10.1145/3236024.3236034.

[30] *HElib*. Accessed: Apr. 22, 2022. [Online]. Available: https://github.com/homenc/HElib

[31] *Sparrow*. Accessed: Apr. 22, 2022. [Online]. Available: https://github.com/ropas/sparrow

[32] *The Petablox Benchmark*. Accessed: Apr. 22, 2022. [Online]. Available: https://github.com/petablox/petablox-bench

[33] M. Zitser, R. Lippmann, and T. Leek, "Testing static analysis tools using exploitable buffer overflows from open source code," in *Proc. 12th ACM SIGSOFT 12th Int. Symp. Found. Softw. Eng. (SIGSOFT/FSE)*. New York, NY, USA: Association for Computing Machinery, 2004, p. 97, doi: 10.1145/1029894.1029911.
[34] D. Lee, W. Lee, H. Oh, and K. Yi, "Optimizing homomorphic evaluation circuits by program synthesis and term rewriting," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 503–518, doi: 10.1145/3385412.3385996.
[35] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 1651–1669.
[36] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.
[37] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 619–631.
[38] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 801–812.
[39] M. Blanton and P. Gasti, "Secure and efficient protocols for iris and fingerprint identification," in *Proc. Eur. Symp. Res. Comput. Secur.* Berlin, Germany: Springer, 2011, pp. 190–209.
[40] A. Fischer, B. Fuhry, F. Kerschbaum, and E. Bodden, "Computation on encrypted data using dataflow authentication," *Proc. Privacy Enhancing Technol.*, vol. 2020, no. 1, pp. 5–25, Jan. 2020.
[41] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.
[42] P. Mishra, D. Rathee, D. Duong, and M. Yasuda, "Fast secure matrix multiplications over ring-based homomorphic encryption," *Inf. Secur. J., Global Perspective*, vol. 30, no. 4, pp. 219–234, 2021, doi: 10.1080/19393555.2020.1836288.
[43] W. W. Cohen, "Pac-learning non-recursive prolog clauses," *Artif. Intell.*, vol. 79, no. 1, pp. 1–38, Nov. 1995.

MOJGAN KOUHOUNESTANI received the B.S. degree in computer science from the University of Isfahan, Isfahan, Iran, in 2015, and the M.S. degree in computer science from Yazd University, Yazd, Iran, in 2018. Since 2020, she has been doing research with the Programming Systems Laboratory, Hanyang University, Ansan, South Korea, under the supervision of Prof. Woosuk Lee.

WOOSUK LEE received the B.S. and Ph.D. degrees in computer science from Seoul National University. He worked as a Postdoctoral Researcher with the University of Pennsylvania, USA, before joining Hanyang University, Ansan, South Korea, where he is currently an Assistant Professor with the College of Computing. His research interests include, but not limited to, static program analysis, and program synthesis.