Privacy-Preserving Alpha Algorithm for Software Analysis

Gamze Tillem Zekeriya Erkin Reginald L. Lagendijk
Delft University of Technology
Department of Intelligent Systems, Cyber Security Group
Delft, The Netherlands
G.Tillem@tudelft.nl Z.Erkin@tudelft.nl R.L.Lagendijk@tudelft.nl

Abstract

Validation in a big software system can be managed by analysis of its behaviour through occasionally collected event logs. Process mining is a technique to perform software validation by discovering process models from event logs or by checking the conformance of the logs to a process model. A well-known algorithm in process mining to discover process models is alpha algorithm. However, while utilising alpha algorithm is useful for software validation, the existence of some sensitive information in the log files may become a threat for the privacy of users. In this work, we propose a protocol for privacy-preserving alpha algorithm on encrypted data. Our protocol aims to generate process models for a software without leaking any information about its users. It achieves same computational complexity with the original algorithm despite the additional computation overhead.

1 Introduction

Software systems have an evolving nature which enables them to grow continuously with new updates and interactions. While growth of software systems is beneficial for its functionality, conversely, it complicates managing its validation. In the traditional approach software validation is maintained by analysing the conformance of pre-defined cases in the design time. However, for complex software systems which has interactions with several external tools, a priori prediction of cases is challenging. This challenge introduces a new approach for software validation which shifts the validation procedure to online phase, namely, analysis of software in the run time. The event logs that are generated during the execution of software, enable observation of behaviour and checking the conformance of design requirements.

A non-trivial technique to monitor software behaviour for validation is process mining. As a field between data mining and process modeling, the aim of process mining is to discover, monitor and improve the real processes by extracting information from the event logs [13]. Process mining utilises log information in three categories. The first category is process discovery which generates a process model from log data. The second category is conformance checking whose purpose is to indicate that the real behaviour of the system conforms to the model by comparing event log of a process with an existing process model. Finally, the third category is enhancement where an existing process model is improved by comparing it with event logs.

As the core component of process mining, log data has a crucial role to determine in which way the software behaviour is modelled. Software log can contain information about user, system settings (e.g. type of operating system, number of cores, memory usage), interactions with other components and the date or duration of execution. Aforementioned information is valuable for process miner to obtain knowledge about the software behaviour. On the other hand, the content of information is vulnerable
against privacy threats since it may contain sensitive information about user or system. An example of such a threat is recently experienced by GHTorrent platform [6]. Aiming to monitor GitHub events to simplify searching on them, GHTorrent does not consider removal of personal data from events. However, it appears that some users of the platform abused the personal data to send survey invitations to data owners [5]. Receiving hundreds of e-mails from external parties, the data owners has started complaining about their privacy in collected log, which, in the end, required the platform developers to revise their privacy policy [6]. The case of GHTorrent shows that as log based software analysis gets popular, the importance of privacy in log files becomes prominent.

In our work, we aim to design a privacy-preserving process discovery protocol to generate process models from event logs while guaranteeing the privacy of event logs. As an initial step, alpha algorithm [14] is selected for process discovery since it clearly shows the steps for discovery of process models. Our protocol utilises encryption to guarantee the confidentiality of logs. To overcome difficulty of retrieving information from encrypted data, we use homomorphic encryption schemes which enable us to perform operations on ciphertext without using decryption mechanism. These schemes are useful especially in multiparty settings which requires prevention of information leakage to other parties while performing computations on encrypted data.

Privacy in software is investigated from different aspects by research community. Several works focus on providing privacy in released test data through anonymization techniques [7, 10] or machine learning techniques [8]. Some other works, e.g. [2, 3], are interested in controlling crash report generation to eliminate sensitive information in reports. Furthermore, preventing the leakage of sensitive data from running software is another concern in software privacy which is achieved by utilising information flow mechanisms in [4] and [15]. However, to the best of our knowledge, none of the existing works deals with the privacy of software validation under process mining. Our protocol is the first attempt to operate process discovery algorithms on software in a privacy-preserving manner.

In the rest of the paper, first we provide some preliminary knowledge (Section 2). Then, we introduce the protocol for privacy preserving alpha algorithm in Section 3 and continue with the complexity analysis in Section 4. Finally, in Section 5, we conclude our paper and explain the directions of future research.

2 Preliminaries

Prior to explain the protocol for privacy-preserving alpha algorithm, we provide some preliminary knowledge about alpha algorithm and cryptographic tools in this section.

2.1 Alpha Algorithm

Alpha algorithm is one of the first process discovery algorithms to discover process models from event logs. Since it covers basic steps of discovery, it is favourable as a starting point for process discovery. It takes an event log \( L \) as input and outputs a process model. The process model is represented as a Petri net, which is a modelling language used in process mining [14]. \( L \) is a set of traces and each trace is a set of activities. Formally, \( L = [\sigma_1, \sigma_2, ..., \sigma_x] \) where \( \sigma_i \) is a trace and \( x \in \mathbb{Z}^+ \). For each \( \sigma_i = \langle t_1, ..., t_j \rangle \), \( t_j \) is an activity where \( 1 \leq j \leq K \) and \( K \) is the maximum number of activities. Then, \( L = [\langle t_{11}, ..., t_{j_1} \rangle, \langle t_{12}, ..., t_{j_2} \rangle, ..., \langle t_{1x}, ..., t_{j_x} \rangle] \).

Alpha algorithm runs in 8 steps to generate a process model. In this section, the steps of alpha algorithm is explained through an example. Assuming that following event log \( L \) is collected from a software

*https://github.com/ghtorrent/ghtorrent.org/issues/32
the alpha algorithm proceeds through the following steps:

**Step 1:** Discovers distinct set of activities \((T_L)\) in \(L \implies T_L = \{a, b, c, d, e, f\}\).

**Step 2:** Discovers initial activities \((T_I)\) in each \(\sigma_i \implies T_I = \{a\}\) and assigns an initial place \(i_L\).

**Step 3:** Discovers final activities \((T_O)\) in each \(\sigma_i \implies T_O = \{f\}\) and assigns a final place \(o_L\).

**Step 4:** Groups activities using ordering relations (direct succession \((\to\))", causalit

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
<td>#</td>
</tr>
</tbody>
</table>

Figure 1: Footstep matrix and process model as Petri net for \(E(L)\).
YASHE: While Paillier cryptosystem is constructed on integers, YASHE scheme is constructed on ideal lattices. The security of the scheme is based on Ring Learning with Errors assumption [1]. Because of page limitation, we refer readers to [1] for more details. Here we only summarise homomorphic properties of YASHE scheme.

We are given two ciphertexts $c_1$ and $c_2$ which are encryptions of $m_1$ and $m_2$ and $[\cdot]_a$ refers to reduction to modulus $a$. Then, homomorphic addition is achieved by adding $c_1$ and $c_2$ as $c = [c_1 + c_2]_q$ which is equal to the encryption of $[m_1 + m_2]_q$. On the other hand, homomorphic multiplication is performed in two phases. In the first phase an intermediate ciphertext $\hat{c} = [\lceil t/q \cdot c_1 \cdot c_2 \rceil]_q$ is computed. Since this operation increases noise which prevents a correct decryption of ciphertext [1], in the second phase a Key Switching mechanism is applied to $\hat{c}$ to transform it into a decryptable ciphertext $c$.

3 Privacy-Preserving Alpha Algorithm

We now describe our protocol for privacy-preserving alpha algorithm on encrypted data. The protocol is based on semi-honest model with three entities which are User, Log Repository and Process Miner. User is the end user of a software who generates event logs and sends them to Log Repository in encrypted form. Log Repository is a semi-honest storage unit which is responsible for collecting and storing encrypted event logs. It can be either specific to a certain software product or a common repository which manages logs from different software products. Since Log Repository is not fully trusted, in our setting it is not allowed to see order relations between any two encrypted activities of event log $L$. Finally, Process Miner is a semi-honest third party which has capabilities to generate process models from encrypted event logs. To be able to generate process models, Process Miner has to learn the order relations in event logs. However, it cannot learn the content of log files.

The protocol is based on three main phases which are Set Up, Relation Discovery and Model Discovery which are demonstrated in Figure 2. As it is explained later in this section, Relation Discovery phase requires utilisation of secure equality tests to discover relations between encrypted activities. Thus, Secure Equality Check subcomponent is integrated to that phase. We show two efficient Secure Equality Check mechanisms [12, 9] here, but, other efficient mechanisms can also be adapted to the protocol. Rest of this section explains each phase of privacy-preserving alpha algorithm protocol in detail.

![Figure 2: Overview of Privacy-Preserving Alpha Algorithm](image)

3.1 Set Up

In Set Up, initially cryptographic keys are generated by a trusted third party and distributed to related entities. Since user is only responsible for generation of encrypted event logs, he is provided public key $pk$. Log Repository and Process Miner are given their secret shares $sk_{LR}$ and $sk_{PM}$, respectively.

In the second part of Set Up phase, according to user’s interaction with software an event log $L$ is generated, as explained in Section 2. After generation of $L$, user
encrypts it under selected encryption scheme (Paillier or YASHE) using \( pk \) and outsources encrypted log \( E(L) \) to Log Repository. Finally, Log Repository shares \( E(L) \) with Process Miner which is going to discover process model in encrypted log data. The format of data that Process Miner retrieves is:

\[
E(L) = [(E(t_{i_1}), ..., E(t_{j_1})), (E(t_{i_2}), ..., E(t_{j_2})), ..., (E(t_{i_z}), ..., E(t_{j_z}))].
\]

### 3.2 Relation Discovery

The core of our protocol is to securely detect distinct activities, address initial and last activities in each trace and identify the relations between them. To that end, we construct a relation table \( RT \) whose indices correspond to encrypted activities. For each index, \( RT \) shows whether the activity is initial (Init) or last (Last) in its trace and it stores the list of direct successors (Direct Successor) for the activity. When an encrypted activity \( E(t_{y_i}) \) where \( y \in [1, j] \) is retrieved, \( RT \) is searched to find a match for \( E(t_{y_i}) \) using secure equality checks. If there is no match for \( E(t_{y_i}) \), then it is inserted into \( RT \) as a new index. Figure 3 demonstrates the procedure of Relation Discovery phase.

**Figure 3: Overview of Relation Discovery phase**

To clarify the procedure, we can construct \( RT \) by using the example log data in Section 2. Initially, Process Miner has the following encrypted log data:

\[
E(L) = [(E(a), E(b), E(c), E(f)), (E(a), E(b), E(e), E(c), E(d), E(b), E(f)),
\]

\[
(E(a), E(b), E(c), E(e), E(d), E(b), E(f)), (E(a), E(b), E(c), E(d), E(e), E(b), E(f)),
\]

\[
(E(a), E(c), E(b), E(e), E(d), E(b), E(f))].
\]

Starting from the first activity \( E(a) \) in trace \( \sigma_1 = (E(a), E(b), E(e), E(f)) \), Process Miner scans \( RT \) to find a match for the current activity. Since initially the table is empty, \( E(a) \) is directly added to \( RT \) (Table 1). For second activity, \( E(b) \), one equality check should be performed to compare it with \( E(a) \). Since \( E(b) \neq E(a) \), \( E(b) \) is inserted into \( RT \) as a new index (Table 2). Furthermore, since \( E(b) \) directly follows \( E(a) \), it is added into Direct Successor list of \( E(a) \). When the same operations are applied for each encrypted activity, the relation table \( RT \) is completed as shown in Table 3.

### 3.2.1 Secure Equality Check for Relation Discovery

Construction of \( RT \) requires comparison of encrypted activities which has to be managed by secure equality check (SEC) mechanisms. Since proposing an equality check mechanism is not our main concern, we adapted two existing mechanisms to our protocol [9][12]. Below, we briefly describe these mechanisms and refer the readers to [9][12] for their detailed description.
Table 1: \( RT \) with one element

<table>
<thead>
<tr>
<th>Index</th>
<th>Init</th>
<th>Last</th>
<th>Direct Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E(a) )</td>
<td>0</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 2: Inserting \( E(b) \) into \( RT \)

<table>
<thead>
<tr>
<th>Index</th>
<th>Init</th>
<th>Last</th>
<th>Direct Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E(a) )</td>
<td>0</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>( E(b) )</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Complete version of relation table \( RT \)

<table>
<thead>
<tr>
<th>Index</th>
<th>Init</th>
<th>Last</th>
<th>Direct Successor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E(a) )</td>
<td>0</td>
<td>1, 2</td>
<td></td>
</tr>
<tr>
<td>( E(b) )</td>
<td>1</td>
<td>2, 3, 4</td>
<td></td>
</tr>
<tr>
<td>( E(c) )</td>
<td>2</td>
<td>-</td>
<td>1, 3, 4, 5</td>
</tr>
<tr>
<td>( E(f) )</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( E(e) )</td>
<td>4</td>
<td>-</td>
<td>2, 5</td>
</tr>
<tr>
<td>( E(d) )</td>
<td>5</td>
<td>-</td>
<td>1, 2</td>
</tr>
</tbody>
</table>

**SEC by Toft [12]:** Toft [12] proposes a SEC protocol by employing Jacobi symbol. The protocol requires a virtual trusted third party which is Arithmetic Black Box (ABB) to provide secure storage and to perform arithmetic computations. The equality of two values is tested by testing whether their difference \( d \) is equal to 0. For an encryption modulus \( M \), if \( d = 0 \), then Jacobi symbol for \( d + r^2 \) where \( r \in R \) \( M \) is \( J_{d+r^2} = \left( \frac{d + r^2}{M} \right) = 1 \). Otherwise, if \( d \neq 0 \), \( J_{d+r^2} = -1 \). Although, Toft’s scheme is efficient, the result is correct with 1/2 probability due to probabilistic nature of Jacobi symbol. Thus, reducing the probability to a negligible degree requires the repetition of protocol \( \kappa \) times with the same input.

**SEC by Lipmaa & Toft [9]:** Different from [12], Lipmaa and Toft [9] introduce a SEC protocol which utilises Hamming distance. Similar to [12], the protocol is based on zero check for difference \( d \) and ABB is responsible for secure storage and arithmetic computations. Hamming distance is computed between a random \( r \) and \( m = r + d \). To reduce the complexity of operations in Hamming distance computation, an offline preprocessing phase to compute random values, random inverses and random exponents is proposed. Furthermore, for online phase Lagrange interpolation is used. Although the result of the protocol is deterministic, it has drawback of computational complexity which is bounded by the bit length of encryption modulus \( M \).

### 3.3 Model Discovery

After discovery of the order relations between encrypted activities, the final phase of our protocol generates the process model using the information in \( RT \). In the original algorithm, a footstep matrix is constructed to demonstrate casual, parallel and choice relations between activities based on direct successions (see Section 2). In the same manner, our protocol constructs the footstep matrix using Direct Successor lists in \( RT \). Finally, the process model as a Petri net is generated using the ordering relations in footstep matrix as it is showed in Figure 4.

### 4 Complexity Analysis

Utilising encryption is advantageous to maintain the confidentiality of log files. However, a qualified scheme for software analysis should also consider the efficiency of computations for practicability. Thus, in this section we analyse the complexity of our protocol.

To evaluate performance of our protocol, initially we have to investigate the complexity of original alpha algorithm. Since in the original algorithm computations are
handled by one party (Process Miner), the complexity can be analysed only in terms of computational cost. Step 4 and 5 of the algorithm dominate computational complexity. Construction of footstep matrix in 4th Step requires $O(xK^2)$ comparisons to find order relations where $x$ is total number of traces and $K$ is maximum number of activities in one trace. In Step 5, $O(K^2)$ comparisons are performed to find maximal relation sets. Consequently, the overall computational complexity of original algorithm is $O(xK^2)$.

In our protocol, the computational complexity is dominated by construction of relation table RT. Similar to the original alpha algorithm, this process necessitates $O(xK^2)$ comparisons in the worst case. However, each comparison is performed by running a secure equality check protocol rather than integer or string comparisons as in the scheme with plaintext. Therefore, despite in theoretical bounds our protocol has the same complexity with the original scheme, it is useful to analyse the cost of one equality check protocol to understand additional cost of encryption in the protocol. Table 4 overviews the complexity of computations for SEC protocols [9, 12] which are used in Relation Discovery phase. To comply with notations in ABB schemes, the complexity as the number of ABB operations is also provided.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paillier based implementation</td>
<td>$O(\ell)$</td>
<td>$O(\kappa)$</td>
</tr>
<tr>
<td>(num of multiplications and exponentiations)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YASHE based implementation</td>
<td>$O(\ell)$</td>
<td>$O(\kappa)$</td>
</tr>
<tr>
<td>(num of additions and multiplications)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Overview of complexity in SEC protocols

The analysis results shows that the cost of computation is bounded by bit size in [9] where $\ell = \lceil \log_2 M \rceil$. The operations in the preprocessing phase dominates the complexity of protocol. On the other hand, in [12] computation cost is determined by correctness parameter $\kappa$. Although, one run of protocol is handled by constant number of operations, since the result is probabilistic, it requires $\kappa$ repetitions. Finally, both additive and somewhat homomorphic setting have same theoretical bounds, but in reality the number of operations for somewhat homomorphic based implementation is less than the number of operations in additive homomorphic implementation. However, it does not necessarily imply that somewhat homomorphic setting is more efficient than additive homomorphic since the bit size of modulus and the complexity of operations differ in two settings.

5 Conclusion and Future Work

In this work we have addressed for the first time the privacy in software analysis under process mining techniques. Specifically, we presented a naive protocol for privacy-preserving alpha algorithm to generate process models from encrypted event logs. Our protocol achieves same theoretical bounds with the original algorithm. However, it requires usage of secure comparison protocols which imposes additional computations
with larger bit sizes. In the future, we continue exploring privacy issues in different algorithms of process mining for software analysis. Furthermore, we extend our research with implementation of algorithms and utilisation of other privacy enhancing technologies.

References


