Mining Encrypted Software Logs using Alpha Algorithm

Gamze Tillem, Zekeriya Erkin and Reginald L. Lagendijk
Cyber Security Group, Department of Intelligent Systems, Delft University of Technology, The Netherlands

Keywords: Software Privacy, Homomorphic Encryption, Applied Cryptography, Software Process Mining.

Abstract: The growing complexity of software with respect to technological advances encourages model-based analysis of software systems for validation and verification. Process mining is one recently investigated technique for such analysis which enables the discovery of process models from event logs collected during software execution. However, the usage of logs in process mining can be harmful to the privacy of data owners. While for a software user the existence of sensitive information in logs can be a concern, for a software company, the intellectual property of their product and confidential company information within logs can pose a threat to company's privacy. In this paper, we propose a privacy-preserving protocol for the discovery of process models for software analysis that assures the privacy of users and companies. For this purpose, our proposal uses encrypted logs and processes them using cryptographic protocols in a two-party setting. Furthermore, our proposal applies data packing on the cryptographic protocols to optimize computations by reducing the number of repetitive operations. The experiments show that using data packing the performance of our protocol is promising for privacy-preserving software analysis. To the best of our knowledge, our protocol is the first of its kind for the software analysis which relies on processing of encrypted logs using process mining techniques.

1 INTRODUCTION

Software systems have an evolving nature which enables them to respond to the needs of technological advances continuously (van der Aalst, 2015). While this evolution is advantageous to improve service quality for users, the drawback is growing complexity which complicates the management of software systems (Rubin et al., 2007). The complication occurs especially in the verification and validation of the system properties. Considering that current systems can reach up to billions of lines of code (Levenberg, 2016), the classical analysis of software becomes impractical (van der Aalst, 2015). Overcoming the difficulties of classical approach is possible using model-based analysis techniques. In these techniques, a formal model of a system is generated and the conformance of properties are checked by automated tools to address defects in the design (Gluch et al., 2002).

A common approach in model-based analysis is modeling the system behavior through event logs that contain information about software execution (Pecchia and Cinque, 2013). A promising technique for such an analysis is process mining that aims to *discover*, *monitor* and *enhance* processes using the information in event logs (van der Aalst, 2016). The discovery, i.e. *process discovery*, aims to generate a process model from the logs to observe system behavior. Monitoring, or *conformance checking*, compares an existing model with real logs of the same process to conform the real behavior to the expected behavior. Finally, enhancement, i.e. *process enhancement*, improves an existing model with the real event logs, to replay the reality on the existing model.

In every category of process mining, the content of event logs are crucial in the system analysis. The logs may contain information about users (e.g. user id or e-mail), duration of execution, system properties (e.g. memory usage, OS type) or component interactions. Although this information is useful in modelling the behavior, the content might leak sensitive information of owners; user and software company. For a user, sharing sensitive data with third parties may pose a privacy threat. A recent discussion about GHTorrent (Gousios, 2013), a platform to monitor and publish GitHub events as dataset, exemplifies such a threat in shared logs. In the dataset user e-mails used to be published since they are already public on GitHub (Gousios, 2016). However, this situation initiated a displeasure when the dataset is used by third companies to send survey e-mails to data owners (Gousios, 2016). The discussion ended

Tillem, G., Erkin, Z. and Lagendijk, R.
Mining Encrypted Software Logs using Alpha Algorithm.
DOI: 10.5220/0006408602670274
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 267-274
ISBN: 978-989-758-259-2

by removing personal data from the dataset (Gousios, 2016). Sharing logs is also arguable for software companies regarding the intellectual property and confidential information in logs. (Leemans and van der Aalst, 2015) show it is possible to reverse engineer software logs with process mining. Considering the risk of piracy through reverse engineering (Naumovich and Memon, 2003), the companies are not willing to share information with external parties.

The existing literature on software analysis for security and privacy approaches the problem from several aspects. The studies for the protection of the intellectual property are mostly focus on cryptographic solutions such as code obfuscation (Collberg et al., 1997), watermarking (Collberg and Thomborson, 1999) and tamper-proofing (Aucsmith, 1996). For the protection of user privacy, some studies approach the problem as the privacy of data in testing applications (Grechanik et al., 2010; Lucia et al., 2012) and provide solutions by applying anonymization. Several studies attempt to protect user privacy during log generation by reducing the sensitive information in log reports (Castro et al., 2008; Broadwell et al., 2003). Furthermore, the control of information flow between software components is also a concern. (Enck et al., 2014) and (Zhu et al., 2011) address the problem of controlling sensitive information flow using taint tracking and analysis mechanisms.

While there are many efforts for securing log-based software analysis in the literature, no studies have focused on privacy issues in software analysis with process mining. In this paper, we propose a protocol for privacy-preserving process discovery for software analysis, namely AlphaSec. Thus, we select the alpha algorithm (van der Aalst et al., 2004) which is a favorable algorithm in understanding the mechanism of discovery with a relatively simple structure.

Our scenario has three parties namely, users, software company (SC) and process miner (PM). The users send the event logs to SC and are not active in the rest of the protocol. PM executes the process discovery protocol on the logs under the supervision of SC. We assume a semi-honest setting where PM and SC do not collude. In order to achieve privacy, we encrypt the logs under a homomorphic cryptosystem. To identify the items in the logs and the relations between them, we use several cryptographic protocols as secure equality checking, secure multiplication and bit decomposition. Furthermore, we use data packing to eliminate the repetition of same operations and to exploit encryption modulus optimally. During the protocol execution, PM and SC are not allowed to directly decrypt the logs. Moreover, the decryptions on intermediate values are secured. In this setting, our protocol guarantees the privacy of data owners. To the best of our knowledge, our paper presents the first protocol for privacy-preserving software analysis with process mining which assures both user and software privacy. Our protocol does not change the original structure of alpha algorithm and it can be adapted to other discovery algorithms with slight modifications. While our proposal adopts well-known cryptographic protocols, it reduces the cost of those protocols significantly by using data packing. We provide computational and communication complexity analysis along with experiments to show the improvement of our protocol.

2 PRELIMINARIES

In this section we summarize the alpha algorithm and and introduce the cryptographic tools used in our protocol. Table 1 summarizes the notation.

Table 1: Explanation of the notation.

Symbol	Explanation
T	Set of activities t_i s.t. $T = \{t_1, t_2, \cdots, t_\Delta\}$
σ_i	A trace with ω_i events s.t. $\sigma_i = \langle e_{1\sigma_i}, \cdots, e_{\omega_i \sigma_i} \rangle$
$e_{j\sigma_i}$	j^{th} event of σ_i, where $1 \leq j \leq \omega_i$ and $1 \leq i \leq \tau$
L	Event log with τ traces, s.t. $L = \{\sigma_0, \cdots, \sigma_\tau\}$
$\otimes$	Secure multiplication operator
$\oplus$	Homomorphic addition operator
$M_{x \times y}$	A matrix M of size $x \times y$
$M_{x,y}$	Index of matrix M in row x and column y
$M_{*,y}$	y^{th} column of matrix M
θ	Compartment size for data packing
N	Plaintext modulus for Paillier cryptosystem
$\mu_{\mathbf{X}}$	Number of packs for the packed array **X**

2.1 The Alpha Algorithm

The alpha algorithm takes an event log $L = \{\sigma_0, \cdots, \sigma_\tau\}$ as input, where L is a set of traces σ_i such that every σ_i is composed of events $e_{j\sigma_i}$, scans it to find patterns and outputs the result as a Petri net[1] (van der Aalst et al., 2004). Moreover, every $e_{j\sigma_i}$ contains several attributes, such as activity, timestamp or resource which determine the perspective of process discovery. Following the common approach in process mining, in this work we assume that activity attribute is used for process discovery, so every $e_{j\sigma_i}$ has only one attribute which is activity.

The algorithm runs in 8 steps (van der Aalst, 2016). In **Steps 1-3**, the set of activities appeared in L, $T_L \subset T$, and the sets of the first ($T_I \subset T$) and last ($T_O \subset T$) activities are discovered. **Step 4** aims to discover the ordering relations between activities. The

[1] A modeling language used in process mining. See (van der Aalst et al., 2004) for details.

ordering is based on *direct succession*, $t_b > t_c$, which means t_c directly follows t_b in σ_i. The direct successions are used to define 3 ordering relations which are 1. *Causality* ($t_b \rightarrow t_c$ or $t_c \leftarrow t_b$): $t_b > t_c$, but not $t_c > t_b$, 2. *Parallel* ($t_b \mid\mid t_c$): both $t_b > t_c$ and $t_c > t_b$, and 3. *Choice* ($t_b \# t_c$) : neither $t_b > t_c$ nor $t_c > t_b$. The result of orderings is represented as a footprint matrix. Once the footprint matrix is created, the pairs with causality relation are collected in X_L and in **Step 5** the maximal pairs of X_L are assigned to Y_L. In **Steps 6-7** the set of places P_L and the set of arches, F_L, which connects the elements of P_L are determined. Finally, **Step 8** returns the result $\alpha(L)$ as (P_L, T_L, F_L).

To illustrate how the alpha algorithm works, we provide a toy example in the following. Let $L = \{\langle a,b,e,f\rangle, \langle a,b,e,c,d,b,f\rangle, \langle a,b,c,e,d,b,f\rangle, \langle a,b,c,d,e,b,f\rangle, \langle a,e,b,c,d,b,f\rangle\}$ be an event log. The 8 steps of alpha algorithm for L is:

- $T_L = \{a,b,c,d,e,f\}$, $T_I = \{a\}$, $T_O = \{f\}$.
- $X_L = \{(\{a\},\{b\}), (\{a\},\{e\}), (\{b\},\{c\}), (\{b\},\{f\}), (\{c\},\{d\}), (\{d\},\{b\}), (\{e\},\{f\}), (\{a,d\},\{b\}), (\{b\}, \{c,f\}\}$. See the footprint matrix in Table 2 for orderings.

Table 2: Footprint matrix for L.

	a	b	c	d	e	f
a	#	→	#	#	→	#
b	←	#	→	←	||	→
c	#	←	#	→	||	#
d	#	→	←	#	||	#
e	←	||	||	||	#	→
f	#	←	#	#	←	#

- $Y_L = \{(\{a\},\{e\}), (\{c\},\{d\}), (\{e\},\{f\}), (\{a,d\},\{b\}), (\{b\}, \{c,f\}\}$.
- $P_L = \{i_L, o_L, p_{(\{a\},\{e\})}, p_{(\{c\},\{d\})}, p_{(\{e\},\{f\})}, p_{(\{a,d\},\{b\})}, p_{(\{b\},\{c,f\}}\}$.
- $F_L = \{(i_L,a), (f,o_L), (a, p_{(\{a\},\{e\})}), (p_{(\{a\},\{e\})}, e), (c, p_{(\{c\},\{d\})}), \cdots, (p_{(\{b\},\{c,f\})}, c), (p_{(\{b\},\{c,f\})}, f)\}$.
- The output $\alpha(L) = (P_L, T_L, F_L)$ as in Figure 1.

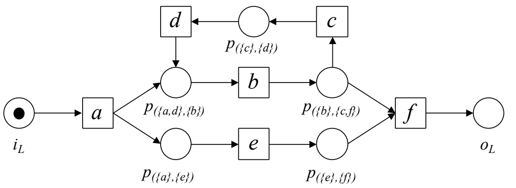


Figure 1: The output of the alpha algorithm for the example L as Petri net.

The output of the alpha algorithm is used in conformance checking and process enhancement, to observe the system behavior and to detect the deviations.

2.2 Paillier Cryptosystem

For our protocol we select Paillier cryptosystem (Paillier, 1999) for the encryption of L due to its homomorphic property. In Paillier, encryption of a message m modulus $N = p \cdot q$ is performed as $E(m) = g^m \cdot r^N \bmod N^2$, where p,q are large primes, $g = N + 1$ and $r \in_R \mathbb{Z}_N^*$. We refer readers to (Paillier, 1999) for details of decryption scheme. Paillier cryptosystem enables to perform homomorphic addition on ciphertexts as $E(m_1) \times E(m_2) = E(m_1 + m_2)$. In the rest of the paper, we represent a Paillier ciphertext by $[\cdot]$ and a homomorphic addition by $\oplus$, for the sake of simplicity.

2.3 Data Packing

In our protocol to eliminate the cost of repeated operations, we use data packing as in (Erkin et al., 2012). The bit size of inputs in plaintext, determines the compartment size, θ, in packed ciphertext. The number of items in one pack is computed as $\rho = \lfloor \log_2 N/\theta \rfloor$ where $\log_2 N$ is the length of plaintext modulus. Let $[W] = \{[w_0], \cdots, [w_{s-1}]\}$ be an encrypted array of s elements, w_i, we pack $[W]$ into $\mu = \lceil s/\rho \rceil$ ciphertexts such that $[W_{pack}] = \{[W_{pack_0}], \cdots, [W_{pack_{\mu-1}}]\}$ where data packing for every $[W_{pack_t}]$ is performed as $[W_{pack_t}] = \sum_{j=0}^{\rho-1} [w_j] \cdot (2^\theta)^j$, s.t. $0 \leq t \leq \mu - 1$. Using $[W_{pack}]$, we can simultaneously employ homomorphic addition and also reduce the total cost of decryption. In the rest of the paper, we represent data packing as $pack([W], \theta, N)$.

2.4 Homomorphic Protocols

For encrypted data processing, we use secure equality check (Nateghizad et al., 2016), secure multiplication (Erkin et al., 2012) and bit decomposition (Lazzeretti, 2012) protocols.

2.4.1 Secure Equality Check (SEQ)

The common approach to securely check whether $[x] = [y]$ is to check if $[q] = [x - y]$ is 0. One way to test if $[q] = 0$ is to use Hamming distance as in (Lipmaa and Toft, 2013). In our work, we use NEL-I SEQ protocol from (Nateghizad et al., 2016) that is an efficient version of (Lipmaa and Toft, 2013). We refer reader to (Lipmaa and Toft, 2013) and (Nateghizad et al., 2016) for the details.

2.4.2 Secure Multiplication Protocol (SMP)

(Erkin et al., 2012) presents an SMP protocol where Alice has $[a]$ and $[b]$ and Bob holds the secret key as follows. Alice selects randoms r_a, $r_b \in_R \mathbb{Z}_N$, blinds the inputs as $[a'] = [a] \cdot [-r_a]$, $[b'] = [b] \cdot [-r_b]$ and sends $[a']$, $[b']$ to Bob. After decryption, Bob computes $a' \cdot b'$, and sends $[a' \cdot b']$ to Alice. Computing $[a \cdot b] = [a' \cdot b'] \cdot [b]^{r_a} \cdot [a]^{r_b} \cdot [-r_a \cdot r_b]$, Alice gets the encrypted multiplication.

2.4.3 Bit Decomposition (BD)

Using BD protocol in (Lazzeretti, 2012), Alice and Bob can compute the encrypted bits of an ℓ-bit x as follows. Assume Alice has $[x]$, and Bob holds the secret key. Alice blinds $[x]$ as $[z] = [x - r]$, where $r \in_R \{0,1\}^{\ell+\kappa}$, and sends $[z]$ to Bob. After decryption, Bob sends the least significant ℓ bits of z to Alice in encrypted form. Using $[c_i] = [z_i]^{r_i} \cdot [c_{i-1}]^{r_i} \cdot [z_i \cdot c_{i-1}]$, $[x_i] = [z_i] \cdot [r_i] \cdot [c_{i-1}] \cdot [c_i]^{-2}$, Alice computes the set $\{[x_0], [x_1], \cdots, [x_{\ell-1}]\}$ which is BD of $[x]$.

3 ALPHASEC: SECURE ALPHA ALGORITHM

In this section we introduce the privacy-preserving alpha algorithm protocol, namely AlphaSec.

3.1 Scenario

Our scenario has three parties: 1. ***Software Company (SC)*** is the owner of the software product who holds public and private keys (pk, sk) and stores the encrypted logs. 2. ***Users*** are the users of the software who send the encrypted logs to SC and are not active in the rest. 3. ***Process Miner (PM)*** is a service provider for SC who models the software. PM has the knowledge and resources to perform process mining techniques, thus, SC needs PM's expertize to analyze the software.

Our goal is to minimize the information leakage for users and SC during the protocol execution. Thus, PM must not access the content of encrypted logs and his statistical observations should be restricted. He should not learn the frequencies, but can only observe the ordering relation between two encrypted activities. For instance, for activities a and b, PM can see that $[a] > [b]$ without knowing the values of $[a]$ and $[b]$ and the frequencies of $[a]$, $[b]$ and $[a] > [b]$. On the other hand, SC is only allowed to decrypt the intermediate blinded values and the output of the protocol which contains his own information. In this setting, our protocol is based on semi-honest security model where PM and SC are non-colluding.

3.2 Setup

In the setup phase, SC generates (pk, sk) and shares pk with PM and users. We assume that SC shares T with PM as $[T] = \{[t_1], \cdots, [t_\Delta]\}$. Furthermore, SC collects $[L] = \{\langle [e_{1\sigma_1}], \cdots [e_{\omega_1\sigma_1}] \rangle, \cdots, \langle [e_{1\sigma_\tau}], \cdots [e_{\omega_\tau\sigma_\tau}] \rangle\}$ from users and shares it with PM to run AlphaSec.

3.3 Process Model Discovery

AlphaSec protocol focuses on the first 4 steps of the original alpha algorithm, since the sensitive data is processed in these steps. Accordingly, the first task is the discovery of activities T_L, T_I and T_F in encrypted domain, i.e. **Steps 1-3**. The second task is to find the ordering relations, i.e. **Step 4**. Afterwards, a footprint matrix is constructed and **Steps 5-8** of the original algorithm are operated in plaintext. Thus, our protocol is based on 3 subprotocols which are 1. **Secure Activity Discovery**, where the activities are discovered, 2. **Secure Direct Succession Discovery** where the orderings are determined and 3. **Secure Modeling** where the eventual process model is generated.

Protocol 1 shows how AlphaSec works. When SC requests a process model, in Step 1, PM creates 3 matrices, namely $R_{\Delta\times\Delta}$, $ID_{\Delta\times 1}$ and $FD_{\Delta\times 1}$. While R is used to store direct successions and discovered activities, ID and FD are used to store the initial and final activities. Between Steps 2-5, for each $[\sigma_i]$ of $[L]$, **Secure Activity Discovery** and **Secure Direct Succession Discovery** subprotocols are operated subsequently. After all $[\sigma_i]$s are scanned, a Petri net is generated in Step 6, by **Secure Modelling** subprotocol.

Protocol 1 AlphaSec

```
Input: $[L]$ , $[T]$
 1: $R, ID, FD$
 2: for all $[\sigma_i] \in [L]$ do
 3:     $(AD^{\sigma_i}, ID, FD) =$SecureActivityDiscovery$([\sigma_i])$
 4:     $R =$ SecureDirectSuccessionDiscovery$(AD^{\sigma_i})$
 5: end for
 6: $\alpha([L]) =$ SecureModelling$(R, ID, FD)$
Output: $\alpha([L])$
```

3.3.1 Secure Activity Discovery

The first subprotocol aims to securely discover T_L, T_I and T_O as shown in Subprotocol 1. Accordingly, PM collaborates with SC to compare every $[e_{j\sigma_i}]$ with every $[t_m]$ using SEQ and the result is stored in $AD^{\sigma_i}_{\Delta\times\omega_i}$. As showed in Step 3, if $[e_{j\sigma_i}] = [t_m]$, $AD^{\sigma_i}_{m,j}$ is set to $[1]$, else to $[0]$. Finally, in Step 6, ID and FD are updated with $AD^{\sigma_i}_{*,1}$ and $AD^{\sigma_i}_{*,\omega_i}$, respectively. In Figure 2(a), we illustrate the procedure for the sample $[L]$.

Subprotocol 1 Secure Activity Discovery

```
Input: $[\sigma_i]$ , $ID$, $FD$
for all $[e_{j\sigma_i}] \in [\sigma_i]$ where $1 \leq j \leq \omega_i$ do
    for all $[t_m] \in [T]$ where $1 \leq m \leq \Delta$ do
        $AD^{\sigma_i}_{m,j} = ([e_{j\sigma_i}] \overset{?}{=} [t_m])\ ?\ [1]\ :\ [0]$
    end for
end for
$ID = ID \oplus AD^{\sigma_i}_{*,1}$ , $FD = FD \oplus AD^{\sigma_i}_{*,\omega_i}$
Output: $AD^{\sigma_i}, ID, FD$
```

	[a]	[b]	[e]	[f]
[a]	[1]	[0]	[0]	[0]
[b]	[0]	[1]	[0]	[0]
[c]	[0]	[0]	[0]	[0]
[d]	[0]	[0]	[0]	[0]
[e]	[0]	[0]	[1]	[0]
[f]	[0]	[0]	[0]	[1]

(a) AD^{σ_1} for σ_1 of L.

	[a]	[b]	[c]	[d]	[e]	[f]
[a]	[0]	[4]	[0]	[0]	[1]	[0]
[b]	[0]	[0]	[3]	[0]	[2]	[4]
[c]	[0]	[0]	[0]	[3]	[1]	[0]
[d]	[0]	[3]	[0]	[0]	[1]	[0]
[e]	[0]	[2]	[1]	[1]	[0]	[1]
[f]	[0]	[0]	[0]	[0]	[0]	[0]

(b) Final R matrix.

	[a]	[b]	[c]	[d]	[e]	[f]
[a]	0	1	0	0	1	0
[b]	0	0	1	0	1	1
[c]	0	0	0	1	1	0
[d]	0	1	0	0	1	0
[e]	0	1	1	1	0	1
[f]	0	0	0	0	0	0

(c) Result of zero-check.

	[a]	[b]	[c]	[d]	[e]	[f]
[a]	#	→	#	#	→	#
[b]	←	#	→	←	||	→
[c]	#	←	#	→	||	#
[d]	#	→	←	#	||	#
[e]	←	||	||	||	#	→
[f]	#	←	#	#	←	#

(d) Footprint matrix.

Figure 2: Illustrating AlphaSec protocol on the sample log.

Since SEQ is an expensive protocol that has to be repeated $\Delta \cdot \omega_i$ times for each σ_i, we use data packing in our protocol. Notice that only a number of intermediate steps of the adopted SEQ protocol (Nateghizad et al., 2016) can be modified for data packing. We use $pack([e_{j\sigma_i} - t_m], \theta, N)$ as packing function where $\theta = (\lceil \log_2 \Delta \rceil + \kappa)$, $\mu = \Delta \cdot \omega_i / \rho$ and $\rho = \lfloor \log_2 N / \theta \rfloor$.

3.3.2 Secure Direct Succession Discovery

The next step in AlphaSec is to identify direct successions between activities. To detect subsequent events in $[\sigma_i]$, we merge two subsequent columns of AD^{σ_i} by SMP. Thus, every element in the former column, $AD^{\sigma_i}_{*,j}$ is securely multiplied with every element in the transpose of latter column $(AD^{\sigma_i}_{*,j+1})^T$. Then, the result is added to corresponding index of R.

This subprotocol has two bottlenecks in terms of efficiency. First, the inputs of SMP are encrypted bits, so the plaintext space is not optimally used. Second, for every σ_i SMP protocol runs $\Delta^2 \cdot (\omega_i - 1)$ times. These bottlenecks require us to use data packing. Accordingly, we pack the column $AD^{\sigma_i}_{*,j+1}$ as $pack(AD^{\sigma_i}_{*,j+1}, \theta, N)$ where $\theta = \lceil \log_2 \Gamma \rceil$ and the column $AD^{\sigma_i}_{*,j}$ as $pack(AD^{\sigma_i}_{*,j}, \theta, N)$ where $\theta = \lceil \log_2 \Gamma \rceil \cdot \Delta$ and Γ is the number of events in L. Since, the protocol requires to add the result to R, we select a larger compartment size, which is the total number of events in the worst case. The result of SMP is a packed ciphertext with $\theta = \lceil \log_2 \Gamma \rceil \cdot \Delta$. The number of compartments in one pack and the number of packs are $\rho_1 = \lfloor \log_2 N / \lceil \log_2 \Gamma \rceil \cdot \Delta \rfloor$, $\mu_1 = \Delta \cdot \omega_i / \rho_1$ and $\rho_2 = \lfloor \log_2 N / \lceil \log_2 \Gamma \rceil \rfloor$, $\mu_2 = \Delta \cdot \omega_i / \rho_2$, respectively. In this setting, SMP runs $\mu_1 \cdot \mu_2 \cdot (\omega_i - 1)$ times for every σ_i. In Subprotocol 2, we show how to perform secure direct succession discovery with packing. The result of SMP, $mult$, is stored in R_{pack}, whose size is $\mu_1 \cdot \mu_2$.

Subprotocol 2 Secure Direct Succession Discovery

```
Input: $AD^{\sigma_i}$
for $1 \leq j \leq \omega_i - 1$ do
    $AD^p_1 = pack(AD^{\sigma_i}_{*,j}, \theta, N), AD^p_2 = (AD^{\sigma_i}_{*,j+1}, \theta, N)$
    for $1 \leq k \leq \mu_1$ do
        for $1 \leq m \leq \mu_2$ do
            $mult = AD^p_{1_k} \otimes AD^p_{2_m}$
            $R_{\text{pack}_{k,m}} = R_{\text{pack}_{k,m}} \oplus mult$
        end for
    end for
end for
Output: $R_{\text{pack}}$
```

After the execution of subprotocol, the result R_{pack} is unpacked using BD to create R. It is important to mention that BD outputs individual bits, but every index of R is a $\lceil \log_2 \Gamma \rceil$-bit integer. Thus, after BD, we perform data packing for every $\lceil \log_2 \Gamma \rceil$ bits to create R. Figure 2(b) shows R matrix for the sample L.

3.3.3 Secure Modelling

In the last step of AlphaSec, the output $\alpha([L])$ is generated using R, ID, FD. Here PM needs to know which activity pairs have an ordering relation, but the frequency of the relation should be hidden from him. Thus, we perform a zero-check function on the inputs to observe whether two encrypted activities has an ordering relation, also, whether an activity is first or last activity. For zero-check, PM blinds $R_{i,j}$ with $r \in_R \mathbb{Z}_N$ as $[R'_{i,j}] = [R_{i,j}]^r$ where $1 \leq i, j \leq \Delta$ and sends $[R'_{i,j}]$ to SC for a secure decryption. If the result of the decryption is non-zero, which means the activity pairs have a direct succession relation, then SC sends 1 and otherwise sends 0 to PM. Hence, PM can only observe the relation between two encrypted activities, but nothing else. Using the result of zero-check, the footprint matrix can be constructed and then the output is generated as in the original alpha algorithm. The only difference is that activities are encrypted and only SC can decrypt them. In Figure 2(c)-2(d), we illustrate the result of zero-check on R and the footprint matrix, respectively.

4 PROTOCOL ANALYSIS

In this section, we first provide a security analysis for our protocol, then analyze its computational and communicational complexity and show experimental results. In Table 3, we summarize the notation.

Table 3: Summary of the notation for complexity analysis.

Notation	Explanation
Γ	Total number of events in L, s.t. $\Gamma = \sum_{i=1}^{\tau} \omega_i$
HAD	Homomorphic addition
HSM	Homomorphic scalar multiplication
ZCF	Zero check function
SEQ	Secure Equality Check
SMP	Secure Multiplication
BD	Bit Decomposition
SAD	Secure Activity Discovery
SDS	Secure Direct Succession Discovery
MD	Secure Modelling

4.1 Security Analysis

The privacy considerations in our protocol are twofold: user privacy and software company privacy. On one hand, users want to protect their sensitive information from PM and SC. On the other hand, SC wants to protect the intellectual property of his product from PM. In the following, we analyze how these concerns are overcome against each party.

Users are not active during protocol execution. They only take part in generation of $[L]$, so they do not have an active adversarial role in our setting.

PM has access to $[L]$ and the results of SEQ, SMP and HAD. The cryptographic protocols are proven to be secure, thus, we assume that PM cannot infer any additional information. Furthermore, to prevent statistical inferences, we hide the frequencies from PM by zero-check. PM can only observe the ordering between two encrypted activities. However, it is not an advantage for PM since the real values are unknown.

SC holds *sk* and collaborates with PM to operate SEQ and SMP protocols. As the owner of *sk*, he does not have direct access to $[L]$ to assure user privacy. During SMP, decryption result is blinded, thus, SC cannot infer the original values. For SEQ, we rely on the security of the underlying protocol.

4.2 Computational Analysis

Prior to the analysis of AlphaSec, we analyze the computational complexity of the original alpha algorithm. The operations in the original algorithm are mostly integer or string comparisons which detect distinct activities and the orderings. Thus, T_L, T_I and T_O can be discovered in Γ comparisons. For the discovery of direct successions, every $e_{j\sigma_i}$ can be paired with its successor in Γ operations. Then, the footprint matrix can be generated with at most Δ^2 comparisons.

For the analysis of AlphaSec, we count the number of operations in every subprotocol and illustrate them in Table 4 without packing (w/o Packing) and with packing (w/ Packing). Apart from the operations in Table 4, Γ and Δ encryptions are performed to encrypt L and T in setup. In AlphaSec, SDS dominates the computations by the quadratic complexity of SMP and HAD. Using data packing, the number of SMP reduces from Δ^2 to $\lceil(\Delta/\rho_1)\rceil \cdot \lceil(\Delta/\rho_2)\rceil$, where $\rho = \lfloor \log_2 N/(\kappa + \lceil \log_2 \Delta \rceil) \rfloor$, $\rho_1 = \lfloor (\log_2 N - \kappa)/(\lceil \log_2 \Gamma \rceil \Delta) \rfloor$ and $\rho_2 = \lfloor (\log_2 N - \kappa)/(\lceil \log_2 \Gamma \rceil) \rfloor$.

Table 4: The number of operations performed in AlphaSec.

		w/o Packing	**w/ Packing**
SAD	SEQ	$\Delta\Gamma$	$\lceil \Delta\Gamma/\rho \rceil$
	HAD	$2 \cdot (\tau - 1)\Delta$	–
SDS	SMP	$\Delta^2(\omega_i - 1)\tau$	$\lceil(\Delta/\rho_1)\rceil \lceil(\Delta/\rho_2)\rceil (\omega_i - 1)\tau$
	HAD	$\Delta^2(\omega_i - 1)\tau$	$\lceil(\Delta/\rho_1)\rceil \lceil(\Delta/\rho_2)\rceil (\omega_i - 1)\tau$
	BD	–	$\lceil(\Delta/\rho_1)\rceil \lceil(\Delta/\rho_2)\rceil$
SM	HSM	Δ^2	–
	ZCF	Δ^2	–

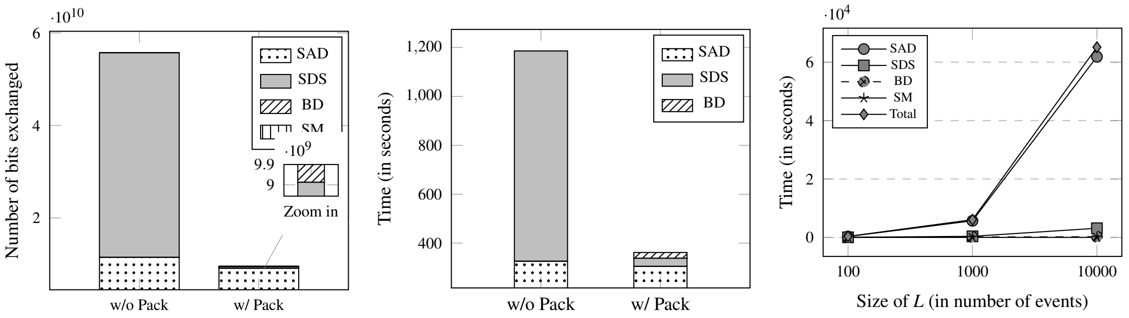
4.3 Communicational Analysis

In Table 5, we summarize the communication complexity of AlphaSec in terms of the number of ciphertexts exchanged both for packed and unpacked version. The numbers show that data packing cannot reduce the bandwidth usage for SEQ proportional to the number of packed ciphertext but it reduces the bandwidth usage in intermediate steps. On the other hand, for SMP, the reduction in bandwidth usage is directly proportional to the number of packs.

Table 5: Bandwidth usage of AlphaSec in terms of the number of exchanged ciphertexts, where $\chi = (\log_2 \log_2 \Delta)$.

	w/o Packing	**w/ Packing**
SEQ	$\Delta\Gamma(3 + \lceil \log_2 \Delta \rceil + 2 \lceil \chi \rceil)$	$3\Delta\Gamma/\rho + \Delta\Gamma(\lceil \log_2 \Delta \rceil + 2 \lceil \chi \rceil)$
SMP	$3\Delta^2(\omega_i - 1) \cdot \tau$	$3 \lceil \Delta/\rho_1 \rceil \lceil \Delta/\rho_2 \rceil (\omega_i - 1)\tau$
BD	–	$(3(\log_2 N - \kappa) - 1) \lceil \Delta/\rho_1 \rceil \lceil \Delta/\rho_2 \rceil$
ZCF	Δ^2	–

For numerical analysis, we measure the bandwidth usage for a dataset with $\Gamma = 10000$ events, $\Delta = 20$ activities, $\tau = 1000$ traces and $w_i = 10$ with and without packing, where ciphertext size 4096 bits. The comparison results in Figure 3(a) show that data packing can reduce the communication cost significantly. The total improvement in communication cost is 83%, which is mainly based on SDS, where the bandwidth usage of SMP is reduced by a factor of 133. We provide a zoom in to show the communication cost of SDS and BD for w/ Pack, but SM is not visible due to its insignificant cost.



(a) Bandwidth usage of AlphaSec with and without packing.

(b) Performance of AlphaSec in seconds without and with data packing.

(c) Execution time of AlphaSec on different datasets.

Figure 3: Evaluating the performance of AlphaSec protocol.

4.4 Experiments

To measure the real time performance of AlphaSec, we implemented it in C++ with GMP-6.1.2 library. The machine we use runs OSX El Capitan with Intel Core i5 2.7 GHz processor. We choose $\log_2 N = 2048$ for Paillier and $\kappa = 80$ as security parameter. As dataset, we select 3 synthetic datasets (D_1, D_2, D_3) from the event log dataset of IEEE TF on Process Mining[2], where for D_1 $\Gamma = 109$, $\tau = 13$ and $\Delta = 10$, for D_2 $\Gamma = 1,226$, $\tau = 100$ and $\Delta = 16$, and for D_3 $\Gamma = 10696$, $\tau = 1000$ and $\Delta = 20$.

As the first experiment, we measure the effect of packing on performance. Thus, we run AlphaSec on D_1 to compare the timing for SAD, SDS and BD on packed and unpacked inputs. Since BD is only used when data is packed, we separate it from SDS. Furthermore, we do not include SM in results, since it is same for packed and unpacked data. As the results in Figure 3(b) show applying packing in SDS reduces the computation time significantly. The improvement in the computation of SDS is 96% while the total improvement is 71% approximately. On the other hand, SAD is not affected significantly by packing, since it cannot be fully adapted to SEQ.

In the second experiment, we observe the performance on different dataset sizes. Thus we compare the timing of AlphaSec on D_1, D_2, D_3. We run this experiment only on the packed version and measure the time required for SAD, SDS, BD, SM and the total time as illustrated in Figure 3(c). For D_3 it takes 65133 seconds to run AlphaSec, of which 61885 seconds are spent for SAD, i.e. SEQ. However, performing SDS requires 3135 seconds including BD which takes around 210 seconds. Finally, SM can be performed approximately in 3 seconds.

5 CONCLUSION

In this paper, we present the first privacy-preserving protocol in process mining for model-based software analysis with the alpha algorithm. The output of our protocol can be used as an input for other process mining techniques such as conformance checking or process enhancement under a privacy-preserving setting. As a first attempt to provide dual privacy for users and SC, we propose a solution based on cryptographic primitives, which provides provable security and privacy. To achieve our goal we use homomorphic encryption along with two-party cryptographic protocols. To reduce the number of operations, we applied data packing on our computations. The performance analyses show that the employment of cryptographic techniques on log analysis provides encouraging results. Furthermore, applying data packing improves the performance significantly.

Although the state-of-the-art process mining techniques are efficient in plaintext domain, our protocol proposes a way to protect sensitive data with additional computational overhead which is promising for the future of this research line. The research challenge is to improve the efficiency of our protocol further by designing custom-tailored cryptographic protocols to replace costly operations such as SEQ and deploying our ideas on more complex process discovery algorithms. With our proposal, we aim to attract the attention of the research community to the privacy aspects of model-based software analysis, which is a distinct and important topic that deserves to be investigated.

REFERENCES

Aucsmith, D. (1996). Tamper resistant software: An implementation. In *Information Hiding, First International Workshop, Cambridge, U.K., May 30 - June 1, 1996, Proceedings*, pages 317–333.

[2] http://data.4tu.nl/repository/collection:event_logs

Broadwell, P., Harren, M., and Sastry, N. (2003). Scrash: A system for generating secure crash information. In *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*.

Castro, M., Costa, M., and Martin, J. (2008). Better bug reporting with better privacy. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2008, Seattle, WA, USA, March 1-5, 2008*, pages 319–328.

Collberg, C., Thomborson, C., and Low, D. (1997). A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, New Zealand.

Collberg, C. S. and Thomborson, C. D. (1999). Software watermarking: Models and dynamic embeddings. In *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 311–324.

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2014). Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans. Comput. Syst.*, 32(2):5:1–5:29.

Erkin, Z., Veugen, T., Toft, T., and Lagendijk, R. L. (2012). Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE Trans. Information Forensics and Security*, 7(3):1053–1066.

Gluch, D., Cornella-Dorda, S., Hudak, J. J., Lewis, G. A., Walker, J., Weinstock, C. B., and Zubrow, D. (2002). Model-based verification: An engineering practice. Technical Report CMU/SEI-2002-TR-021, Carnegie Mellon University, PA.

Gousios, G. (2013). The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 233–236, Piscataway, NJ, USA. IEEE Press.

Gousios, G. (2016). The issue 32 incident _ an update. Accessed May 3, 2016.

Grechanik, M., Csallner, C., Fu, C., and Xie, Q. (2010). Is data privacy always good for software testing? In *IEEE 21st International Symposium on Software Reliability Engineering, ISSRE 2010, San Jose, CA, USA, 1-4 November 2010*, pages 368–377.

Lazzeretti, R. (2012). *Privacy preserving processing of biomedical signals with application to remote healthcare systems*. PhD thesis, Ph. D. thesis, PhD school of the University of Siena, Information Engineering and Mathematical Science Department.

Leemans, M. and van der Aalst, W. M. P. (2015). Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, pages 44–53.

Levenberg, J. (2016). Why Google stores billions of lines of code in a single repository. *Commun. ACM*, 59(7):78–87.

Lipmaa, H. and Toft, T. (2013). Secure equality and greater-than tests with sublinear online complexity. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 645–656.

Lucia, Lo, D., Jiang, L., and Budi, A. (2012). kbe-anonymity: test data anonymization for evolving programs. In *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 262–265.

Nateghizad, M., Erkin, Z., and Lagendijk, R. L. (2016). Efficient and secure equality tests. In *IEEE International Workshop on Information Forensics and Security, WIFS 2016, Abu Dhabi, United Arab Emirates, December 4-7, 2016*, pages 1–6.

Naumovich, G. and Memon, N. D. (2003). Preventing piracy, reverse engineering, and tampering. *IEEE Computer*, 36(7):64–71.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, pages 223–238.

Pecchia, A. and Cinque, M. (2013). *Log-Based Failure Analysis of Complex Systems: Methodology and Relevant Applications*, pages 203–215. Springer Milan, Milano.

Rubin, V. A., Günther, C. W., van der Aalst, W. M. P., Kindler, E., van Dongen, B. F., and Schäfer, W. (2007). Process mining framework for software processes. In *Software Process Dynamics and Agility, International Conference on Software Process, ICSP 2007, Minneapolis, MN, USA, May 19-20, 2007, Proceedings*, pages 169–181.

van der Aalst, W. M. P. (2015). Big software on the run: in vivo software analytics based on process mining (keynote). In *Proceedings of the 2015 International Conference on Software and System Process, ICSSP 2015, Tallinn, Estonia, August 24 - 26, 2015*, pages 1–5.

van der Aalst, W. M. P. (2016). *Process Mining - Data Science in Action, Second Edition*. Springer.

van der Aalst, W. M. P., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142.

Zhu, D. Y., Jung, J., Song, D., Kohno, T., and Wetherall, D. (2011). Tainteraser: protecting sensitive data leaks using application-level taint tracking. *Operating Systems Review*, 45(1):142–154.