PP–GSM: Privacy-preserving graphical security model for security assessment as a service

Dongwon Lee, Yongwoo Oh, Jin B. Hong, Hyoungshick Kim, Dan Dongseong Kim

Article history:
Received 16 August 2022
Received in revised form 1 December 2022
Accepted 30 December 2022
Available online 16 January 2023

Keywords:
Graphical security models
Homomorphic encryption
Graph obfuscation
Security assessment
Privacy-preserving computing

ABSTRACT

Security Assessment-as-a-Service (SAaaS) allows users to outsource security assessments of their systems and networks from the cloud, reducing the burden on users who do not have sufficient resources to carry out security assessments. SAaaS can be implemented using Graphical Security Models (GSMs), such as Attack Graphs and Attack Trees, that are widely used for security assessments. However, this approach exposes users’ sensitive data (e.g., network topology, host vulnerabilities) in the cloud, which would not be acceptable in private systems such as government and/or corporation networks. This paper proposes a framework named privacy-preserving GSM (PP–GSM) for SAaaS. PP–GSM is built with (1) homomorphic encryption (HE) for protecting users’ sensitive data by performing security assessment computations on the encrypted network models, and (2) graph obfuscation techniques to confuse attackers trying to reveal users’ sensitive data. Moreover, we develop new algorithms to speed up HE by reducing the number of multiplications, which are computationally expensive arithmetic operations in HE schemes. Our experimental results using various realistic scenarios show that PP–GSM can be generated on average in 1,078 s for networks with 60 nodes (and the time taken is linearly proportional to the number of nodes). For evaluations, the time taken can be as short as on average 30 s for computing the cumulative attack success probability. Therefore, PP–GSM is a promising solution for the SAaaS to be used in practice.

1. Introduction

Security Assessment-as-a-Service (SAaaS) is a newly emerging security service often implemented in the cloud where users can outsource security assessments of their systems and networks. Graphical Security Model (GSM) (e.g., Attack Graphs or Attack Trees) can be adopted as a primary tool [1–6] for SAaaS to users who are resource-limited and have limited knowledge on GSM and/or security assessments. However, to use a GSM (or any other security assessment tools) for SAaaS, users have to send their private information such as host (e.g., host vulnerabilities information) and network data (e.g., network topology/reachability) to a cloud server. Inherently, the uploaded user’s data could be intercepted by an attacker or misused by a malicious cloud-based SAaaS provider. For example, a recent data breach incident on “Capital One” showed that an unauthorized insider attacker could access users’ data on the cloud server [7]. Protecting personal information on the server side would be an important issue in the industry [8]. As a result, SAaaS users must ensure that their sensitive data is not accessed by unauthorized entities who can leak.sell this information. In practice, some organizations and companies do not often trust even the cloud service providers (CSPs) and thus are inclined not to share their system and network information with the cloud server.

To the best of our knowledge, there has been no prior work on designing and implementing GSMs with “privacy-preserving” properties and methods. In this paper, we propose a novel approach named privacy-preserving GSM (PP–GSM) using Homomorphic Encryption (HE) [9] and Graph Obfuscation techniques. To compute security metrics in a privacy-preserving manner, we encrypt the user’s system data and upload the encrypted data to the SAaaS cloud server. The cloud server constructs a GSM model, performs a security assessment with the encrypted system data, and returns the security assessment results back to the user, which is still encrypted. The framework ensures that only authorized users (i.e., those holding the secret key for HE) can access the security assessment results.

To implement such a framework, we need to address two challenging issues. First, many multiplications and divisions for floating-point numbers are natively needed to compute security metrics for security assessments. However, existing HE libraries such as HElib [10] and Simple Encrypted Arithmetic Encryption (SEAL) [11] have limited capabilities to support a large number of multiplications and divisions for floating-point numbers.

© 2023 Elsevier B.V. All rights reserved.
of multiplications and divisions for floating-point numbers, and TFHE [12] only supports binary operations. Even though several recent works such as HEAAN [13] and Lattigo [14] introduced bootstrapping algorithms\(^1\) to increase the number of multiplications for floating-point numbers, the computational overhead of bootstrapping is still significant. For example, in HEAAN, it takes more than 20 s to perform one bootstrapping operation when the ciphertext has sufficient slots [14,15]. Therefore, it is impractical (in their current states) to calculate security metrics requiring hundreds or thousands of bootstrapping operations (even for small-sized network systems). Therefore, avoiding as many bootstrapping operations as possible (i.e., by reducing the number of multiplications needed) is essential.

Second, it is computationally infeasible to hide the whole network structure using HE. To hide the whole network structure, we need to convert a network topology into the complete graph by adding dummy edges for all the node pairs that are not connected, incurring significant storage and time overheads. It would be unacceptable to compute security metrics for a complete graph even with a small number of nodes because the number of possible attack paths between nodes in the graph grows exponentially. To minimize the computational cost, we only encrypt hosts’ attribute values. However, this design allows attackers to learn the network topology information that may be used maliciously. Therefore, we introduce four novel graph obfuscation techniques using dummy elements, which are fake (i.e., non-existing) nodes and edges added to the encrypted network topology data before uploading it to the cloud. Thus, even if the data is hijacked, it is difficult for the attacker to deduce the original network topology information.

PP–GSM is designed to address the aforementioned issues to implement privacy-preserving SAaaS. Our key contributions are summarized as follows.

- We develop a privacy-preserving GSM (PP–GSM) framework using HE and graph obfuscation techniques to provide a privacy-preserving SAaaS in the cloud (see Section 3);
- We design and implement algorithms to compute security metrics with minimal multiplications to reduce the computational overheads with HE (see Section 4);
- We propose novel graph obfuscation techniques to confuse attacks against the PP–GSM framework and analyze their effectiveness (see Section 5);
- We conduct experiments using four different network topologies to demonstrate the efficiency and security of the PP–GSM framework under various conditions (see Section 7);
- We release our source code and dataset publicly available from https://github.com/PPGSM/PPGSM for further studies in privacy-preserving techniques for GSM and SAaaS.

The rest of the paper is organized as follows. Section 2 presents the background of security assessment and HE. Section 3 presents the overview of PP–GSM. Section 4 presents the security metrics used to evaluate our proposed framework. Section 5 presents our obfuscation techniques to prevent leakage of network topology information. Section 6 provides a theoretical security analysis of PP–GSM. Section 7 presents the performance of PP–GSM and the effects of the obfuscation techniques. Section 8 discusses PP–GSM applications, limitations, and future directions. Related work is presented in Section 9, and finally, we conclude this paper in Section 10.

1 Bootstrapping is a generic technique that allows refreshing homomorphically encrypted ciphertexts. Bootstrapping is necessary for HE to control the noise propagation in ciphertexts, which are inherently caused by arithmetic operations such as multiplications.

2. Background

2.1. Security assessment

The security assessment is a widely adopted practice to understand the security posture of one’s systems and networks. GSMs are widely used for the security assessment [1,2,6], with many public and commercial tools available. Usually, a GSM tool is deployed locally within the network, and it collects input data such as reachability and vulnerability, and security assessment can be performed via the GSM producing various security metrics. Recently, the security assessment has been provided as a cloud service called SAaaS [17,18]. This allows users who do not have sufficient capabilities to perform security assessment themselves to outsource the service. However, in SAaaS settings, the user’s data inherently needs to be sent to the cloud server. Consequently, the use of SAaaS has several limitations; not all systems for users are built in the cloud, and the security assessment techniques do not inherit privacy-preserving properties. So, there is still a significant gap in providing sufficient techniques to outsource security assessment for users to evaluate the security posture of their systems and networks.

2.2. Homomorphic encryption (HE)

HE is an encryption scheme that enables ones to perform operations over encrypted data without decrypting them. In HE, functions are evaluated on the top of the encrypted data, such that Equation \(E(F(E(m_1), E(m_2))) = E(F'(m_1, m_2))\) is satisfied, where \(m_1\) and \(m_2\) are messages, \(F\) and \(F'\) are computationally feasible functions for the message, and \(E\) is the encryption function. We can apply a function \(F\) on encrypted data \(E(m_1)\) and \(E(m_2)\) such that the result is equivalent to applying another function \(F'\) on the original data \(m_1\) and \(m_2\) and then encrypting the result. In PP–GSM, we use a HE scheme with two keys: secret key and evaluation key. The secret key is private and provides decryption abilities. The evaluation key (or public key) can be exported to the cloud, enabling encryption or operations over encrypted data. Below summarizes the most widely used HE libraries:

- **TFHE**: TFHE [12] is a FHE library based on the Gentry–Sahai–Waters (GSW) scheme [19,20]. Currently, TFHE supports all binary gate operations (including MUX gates) on encrypted data. TFHE also can be extended to support additions and multiplications for floating-point values [21].
- **HElib**: HElib [10] also supports arithmetic operations for floating-point numbers. There are two available schemes: Brakerski–Gentry–Vaikuntanathan (BGV) scheme [22] and Cheon–Kim–Kim–Song (CKKS) scheme [13]. Both schemes’ security is based on the underlying hardness of the Ring Learning With Errors (RLWE) problem. The BGV scheme only supports arithmetic operations for integer numbers whereas the CKKS scheme can support arithmetic operations on floating-point numbers.
- **SEAL**: SEAL [11] supports additions and multiplications for floating-point numbers. It has two different encryption schemes, CKKS and Brakerski/Fan–Vercauteren (BFV) [23,24]. As described in HElib, CKKS supports additions and multiplications for floating-point and complex numbers, while BFV allows computations only for integer numbers. However, in the SEAL library, the number of multiplication operations is limited since bootstrapping is not supported. In HE schemes, the noise of the calculation increases with operations, and bootstrapping is the process to reduce this noise when the noise grows to affect the calculation value.

Among those libraries, we used SEAL’s implementation of CKKS to implement PP–GSM framework, as it was the most efficient when performing encryption on real and complex numbers, a desired property computing security metrics.
3. Overview of PP–GSM

In SAaaS, there are two directly involved entities: the user (client) and the cloud service provider (CSP). A user would like to carry out a security assessment on her system. Here, we assume that the user does not have sufficient resources or security knowledge to perform a security assessment by herself. In contrast, the cloud server can carry out various security assessments using evaluation tools and methods. Therefore, the user may wish to outsource the security assessment for her system to the cloud server. However, the user’s system data can often be sensitive and private. As a result, the user may decide not to outsource her system data in plaintext form to the cloud server, ultimately not being able to outsource the security assessment. To address the user’s privacy concern, we propose PP–GSM, and the overall process of the PP–GSM framework is shown in Fig. 1.

We assume that both entities share the same HE library (i.e., SEAL) utilized in PP–GSM. The PP–GSM framework consists of five main steps as follows (see Figs. 1 and 2). 1. Graph obfuscation: the user’s network topology is obfuscated by strategically adding dummy elements. 2. Host data encryption: the user’s host data are encrypted using her evaluation key and sent to the cloud server with the secret key. 3. GSM generation: the cloud server generates the GSM using the user’s encrypted host data and network topology information (containing dummy elements). 4. Security analysis: the cloud server analyzes the security using the GSM with the evaluation key, where the security analysis results remain encrypted. 5. Security analysis results decryption: the user decrypts the received encrypted results from the cloud server using the user’s secret key. Fig. 2 shows this procedure sequentially.

3.1. Graph obfuscation

To reduce the computational complexity, we do not hide the existence of each edge in a user’s system because it requires the addition of dummy edges for all the node pairs that are not connected. If an attacker can learn the important network structural information, the attacker may misuse this information to deduce
**topologically** critical nodes in the network and discover effective attack paths to reach target nodes. To address this problem, we propose to use proper graph obfuscation techniques by adding some *dummy* elements, particularly fake nodes and edges, instead of adding dummy edges for all non-existing edges (details are provided in Section 5).

### 3.2. Host data encryption

A user’s sensitive host data should be encrypted before being uploaded to the cloud server to protect the user’s sensitive data. Each user generates his/her own secret key and evaluation key pair based on parameters (e.g., ring dimension, ciphertext modulus, or key distribution) defined in an HE scheme. When the generated keys are configured for PP–GSM, the user (client) encrypts the host data using the secret key. We can categorize the user’s data to be encrypted into two types: (1) each host’s data, such as its operating system, version, and installed applications information, represented as integer numbers, and (2) binary variables to represent dummy nodes and edges. Thus, for host data encryption, \((n + d) \cdot k\) encryption operations are needed in total, where \(n\) is the number of nodes, \(d\) is the number of dummy elements, and \(k\) is the number of attributes for each host.

### 3.3. GSM generation

The cloud server uses the encrypted host data to collect relevant security attributes to generate the GSM [1,2]. Usually, the GSM can be generated using two types of information: (a) network topology and (b) each host’s data (e.g., OS, installed applications, etc.).

(a) **Network topology:** As the network topology information is not hidden, the GSM can directly process the network topology given by the user. However, the received network topology is actually different from the user’s original network topology because dummy nodes and edges have already been added.

(b) **Host data:** The host’s data is used to determine fundamental security attribute values, such as attack cost, impact, ASP, and cost to update. That is, when a user provides her hosts’ data, such as OS and installed applications, the CSP retrieves each host’s security attribute values associated with the provided OS version and applications from the CSP’s vulnerability database.

In our PP–GSM implementation, as shown in Fig. 3, the user downloads the list of OSs supported by the CSP’s security assessment service and selects the OS version that is running on the user’s host. If the \(i\)th OS in the list is running, the user sets \(S_i = 1\); the user sets \(S_i = 0\) for the other OSes that are not used. The user uploads these encrypted \(S\) values and an evaluation key to the CSP. The CSP then computes the host’s security attribute values (e.g., attack cost) with the encrypted \(S\) values and the CSP’s security vulnerability database using the evaluation key. For example, in Fig. 3, we assume that “Windows 10” is running on the host. Therefore, the user sets \(S_2 = 1\). The attack cost of the host’s OS can be computed by the sum of \(m\) values, which are products of \(S_i\) and attack cost of \(i\)th OS. However, during this process, the CSP cannot learn any information about the host’s OS because the user’s host data is encrypted, and all computations are performed on the encrypted data. We note that encrypted zero is (computationally) indistinguishable from encrypted one because HE is a randomized encryption scheme. For the installed applications, the same procedure can be applied to compute security attribute values.

### 3.4. Security analysis

Using the generated GSM, the PP–GSM can now conduct a security assessment with no knowledge of the user’s host data (i.e., they are encrypted) and even security analysis results. To demonstrate the feasibility of the PP–GSM, we implemented a few widely used security metrics (see the details of those security metrics in Section 4). Computing security metrics, such as the attack cost, the results are the same whether the dummy elements are used or not because they do not contribute to the attack cost computation due to their values set to zero. The same concept applies to other security metrics, where the dummy elements do not contribute to the computation of the chosen security metric. On the other hand, the cloud server cannot learn any information about the presence of dummy hosts because the evaluation results are still encrypted.

### 3.5. Security analysis results decryption

The computed result(s), still encrypted, is sent back to the user, which is decrypted using the user’s secret key to retrieve the assessment results (i.e., security attributes and security metrics).

### 4. Computing security metrics for PP–GSM

Among various security metrics available using GSMS [1,2,25, 26], we select a few widely used security metrics: independent ASP, risk, ROSI, attack cost, and cumulative ASP, to show the feasibility of PP–GSM. Table 1 summarizes the notations that are used in the equations in this section. We also denote an attack path as \(ap\).
Each node with the system information about the node (e.g., OS, also notes the independent ASP from approximation computation for the independent ASP. efficiently computed using SEAL). Since the logarithm function operation. When the graph grows larger, it requires multiple numbers (HEaaN) [13], it takes about 16.6 s for one bootstrapping SEAL. Therefore, it is not appropriate in a real-world implemented because of bootstrapping with existing HE libraries, including SEAL. Therefore, it is not appropriate in a real-world implementation. For example, in HE for arithmetic of approximate numbers (HEaaN) [13], it takes about 16.6 s for one bootstrapping operation. When the graph grows larger, it requires multiple times of bootstrapping, which is impractical. We solve this issue by transforming the multiplication into addition by applying a logarithm function to the probability values (consequently, it retains the ranks of path probability values while being able to efficiently computed using SEAL). Since the logarithm function cannot directly be computed using HE, we develop the following approximation computation for the independent ASP.

Given a pair of two nodes s and e, we use prob_{s,e} to represent the independent ASP from s to e. For a given attack path ap, we also denote path_{s,e} and prob_{ap} to represent a set of all attack paths from s to e and the ASP of the path ap, respectively. For the path ap, the ASP can be calculated as prob_{ap} = \prod_{i \in ap} prob_{i} where i is a node on the path ap and prob_{i} is ASP of node i. Each node i’s attack success probability is computed on the user’s machine with the system information about the node (e.g., OS, installed applications) and is processed in encrypted form to the cloud server, as illustrated in Fig. 3. We (approximately) calculate prob_{s,e} as follows:

\[ prob_{s,e} = 1 - \prod_{ap \in path_{s,e}} (1 - prob_{ap}) \] (1)

First, we apply the logarithm function to both sides of Eq. (1).

\[ \log(1 - prob_{s,e}) = \sum_{ap \in path_{s,e}} \log(1 - prob_{ap}) \] (2)

Then we can approximate \( \log(1 - prob_{ap}) \) using Taylor expansion [27] as follows:

\[ \log(1 - prob_{ap}) \approx -prob_{ap} - \frac{(prob_{ap})^2}{2} - \frac{(prob_{ap})^3}{3} \ldots \] (3)

Taylor expansion can be applied only if \( x \) is less than 1 in \( \log(1 - x) \). Because \( prob_{ap} \) is a probability less than 1, we can use Taylor expansion. Finally, we can approximate prob_{ap} as follows:

\[ prob_{ap} \approx 1 + \log(prob_{ap}) + \frac{\log(prob_{ap})^2}{2!} + \frac{\log(prob_{ap})^3}{3!} \ldots \] (4)

In Eq. (4), \( \log(prob_{ap}) \) can be calculated as \( \sum_{i \in ap} \log(prob_{i}) \) where i is a node on the path ap and log(prob_{i}) is one of the node i’s security attribute values. We note that the node i’s security attribute value can be retrieved from the CSP’s database as described in Section 3.3. Thus, we can approximate prob_{s,e} between s and e.

In this paper, we compute a summation of terms until the 10th term only for Eqs. (3) and (4). The accuracy of this approximation will be discussed in Section 7.2.

4.2. Risk

The risk describes the security risk associated with all the network components and their associated vulnerabilities (i.e., threat to hosts and the impact of such attacks). The risk can be quantified by taking into account the impact associated with a threat imposed on the network multiplied by the ASP.

To compute the risk of a path, therefore, we need to perform a series of multiplication operations (i.e., impact × ASP of each node i on the path), and then sum all the values consecutively to calculate the risk associated with an attack path which we denote as path risk (i.e., Eqs. (5) and (6)). Then, the sum of all path risk represents the risk of the network. In Eqs. (5) and (6), Impact_{i}, Risk_{i}, Risk_{ap} denotes the impact of attack on node i, the risk of node i, and the risk of path ap, respectively. The CSP can retrieve Impact_{i} and Risk_{i} for node i from the CSP’s database.

\[ Risk_{i} = \text{Impact}_{i} \times \text{prob}_{i} \] (5)

\[ Risk_{ap} = \sum_{i \in ap} Risk_{i} \] (6)

We improve the performance of computing Eq. (6) using the lazy relinearization and lazy rescaling strategy [28–30]. When multiplication is performed on ciphertexts in HE, relinearization is needed to unify the ciphertext form; rescaling is needed to reduce the noise incurred by the multiplication. Therefore, in principle, we need to perform the number of relinearization and rescaling operations proportional to the path length of ap in Eq. (6). In our PP–GSM implementation, we use the lazy relinearization and lazy rescaling strategy to perform relinearization and rescale only once after adding all Risk_{i}’s in Eq. (6), resulting in reducing the computational overhead of relinearization and rescaling operations significantly.

4.3. Return on security investment (ROSI)

ROSI is one of the widely used security metrics to evaluate the gain from a security investment (in the form of security actions) [2]. For example, a security action is to secure a node by removing all known security vulnerabilities from the node through patching. Given a pair of two nodes (s, e) where s is the start node and e is the end node, we use ROSI(s, e, t) to quantify the costs and benefits of security investments of securing node t (e.g., installing security patches for t) on the attack paths from the start node s to the end node e. C_{t} represents the cost of patching node t. The benefits of securing node t are computed as the difference between the risks before and after securing node t, which can be computed as \( \sum_{ap \in path_{s,e}} Risk_{ap} \) where path_{s,e} represents all the attack paths from the start node s to the end node e passing through node t. Finally, the formula for computing
ROSI\((s, e, t)\) is shown in Eq. (7). The CSP can retrieve \(C_t\) and the inverse of \(C_t\) for node \(t\) from the CSP’s database.

\[
\text{ROSI}(s,e,t) = \frac{\sum_{p \in \text{path}(s,t)} \text{Risk}_{ap} - C_i}{C_t}
\]  

\(7\)

4.4. Attack cost (AC)

Given an attack path \(ap\), the attack cost of \(ap\) means the total cost to compromise all the nodes on the path \(ap\). To compute the attack cost of an attack path \(ap\), a series of addition operations should be computed to sum the individual attack cost for each node on the path (i.e., in Eq. (8)). Here, \(AC_i\), \(AC_{ap}\) denotes the attack cost of node \(i\) and the attack cost of path \(ap\), respectively. The CSP can retrieve \(AC_i\) for node \(i\) from the CSP’s database.

\[
AC_{ap} = \sum_{i \in ap} AC_i
\]

\(8\)

4.5. Cumulative attack success probability (cumulative ASP)

In real attacks, exploitation of hosts may not be independent. Therefore, we also need to consider computing cumulative ASP taking into account the dependency between hosts (i.e., ASP changes depending on the attack path taken) [2]. The cumulative ASP for a node \(i\) is computed as the multiplication of \(\text{prob}(i|W)\) and \(\prod_{j \in W} \text{prob}_j\) where \(\text{prob}(i|W)\) is the conditional probability of compromising node \(i\) for given \(W\) and \(W\) is the set of nodes reachable to node \(i\) (see Eq. (9)). To compute \(\text{prob}(i|W)\), we replace Eq. (9) with Eq. (10) where \(\text{prob}(i)\) is the ASP of node \(i\) from CSP’s vulnerability database. In Eq. (10), \(\text{prob}\) is the node \(j\)’s cumulative ASP which is already analyzed. Therefore, the value is different from ASP assigned in the GSM generation step. We found that the cumulative ASP computation requires many multiplications that would not be acceptable with existing HE libraries such as SEAL [11].

Therefore, we propose an approach to compute each node’s cumulative ASP with the client’s assistance. Whenever the number of multiplications reaches the maximum number of multiplications allowed, the CSP sends an interim result (still encrypted) to the client, and the client decrypts and re-encrypts the interim result and sends it back to the CSP.

\[
\text{prob}_i = \text{prob}(i|W) \times \prod_{j \in W} \text{prob}_j
\]  

\(9\)

\[
\text{prob} = \text{prob}(i) \times (1 - \prod_{j \in W} (1 - \text{prob}_j))
\]  

\(10\)

5.5. Threat model to topology information in GSMS

We take into account the honest-but-curious service provider model as the main adversary, which is commonly used in analyzing the privacy properties of a system (e.g., as in searchable encryption scenarios [31]). We believe the honest-but-curious model is a reasonable and practical assumption — most CSPs have to provide correct security assessment services to keep their business reputation, but they could still attempt to learn about their customers’ sensitive information secretly. For instance, intelligence agencies may want to collect the data on cloud services covertly. That is, an attacker can be a CSP itself or a third party entity who might be willing to obtain the user’s sensitive information from the user data uploaded on the cloud. Other types of attacks that CSP can carry out are also discussed in Section 6.

We assume the private key is safely kept. According to Section 3, we also assume the HE algorithms are secure, so the attacker cannot learn any information about any encrypted data (e.g., individual hosts). Instead, attackers may attempt to obtain information about the *obfuscated* network topology (i.e., including dummy nodes and edges). We consider the following attack scenarios to evaluate the effectiveness of graph obfuscation techniques. We assume that an attacker has initially compromised one host in the user’s network system and knows the location of the compromised host on the network topology. In this situation, the attacker’s goal is to compromise a target host in the user’s network system, which is different from the initially compromised host. We consider two attacker types based on the attacker’s knowledge under the above-mentioned scenarios: (A1) the attacker does not know the location of a target host on the network topology due to encryption. Although the attacker tries to infer the target node’s location, there are multiple nodes in the graph with the same graph properties in general, such as degree, which limits the estimation of the exact location if it is not known; and (A2) the attacker knows the location of a target host on the network topology (e.g., an attacker with access to the network information). In the case of A1, the attacker’s strategy is to search the hidden target host with a graph traversal algorithm (such as the depth-first-search or breadth-first-search) because any node can be a target host. In this case, we specifically use the breadth-first-search because both graph traversal algorithms produce similar search costs with our datasets in the evaluation. In the case of A2, the attacker’s strategy is to enumerate all paths between the compromised node and the target node and sequentially search each candidate path in a non-decreasing order of path length until the target host is found in a greedy manner.

5.2. Proposed graph obfuscation strategies

Even though the use of dummy nodes could effectively hide the network topology information, we should also minimize the number of dummy nodes to add, as adding dummy nodes increases the performance overhead significantly. To find more effective graph obfuscation strategies, we consider the following strategies:

- **Random addition** is a baseline strategy that selects nodes to be connected with dummy nodes randomly. That is, we create a dummy node with \(k\) edges to randomly select \(d\) different nodes where \(k\) is an average degree of the graph.
- **High-centrality addition** uses weighted random sampling to select nodes with high betweenness centrality to be connected with dummy nodes from a graph \(G\). The betweenness centrality of a node measures the node’s influence over the flow of information in a graph by computing the proportion of shortest paths between all node pairs in the graph that
pass through the node. That is, we create a dummy node with \( k \) edges to \( d \) different nodes that are selected with the probability proportional to their betweenness centrality.

- **Low-centrality addition** selects nodes to be connected with dummy nodes from a graph \( G \) according to the inverse of their betweenness centrality using weighted random sampling. That is, we create a dummy node with \( k \) edges to \( d \) different nodes that are selected with the probability inversely proportional to their betweenness centrality.

- **Hybrid addition** is a hybrid strategy that takes advantage of both high-centrality addition and low-centrality addition. That is, we connect a dummy node to the \( k \) different nodes selected by one of high-centrality addition and low-centrality addition strategies, alternatively.

Against (A1) attacker, we presume that it would be better to connect dummy nodes to the nodes which are centrally located at the network. In general, betweenness centrality is widely used to measure the importance of nodes in terms of network connectivity. Therefore, we consider the “high-centrality addition” strategy for adding dummy nodes and edges to increase the probability of encountering dummy nodes for these attackers.

Against (A2) attacker, we presume that it would be better to connect dummy nodes to actual nodes such that it creates new shortest paths as many as possible in the given network topology. Therefore, we consider the “hybrid-addition” strategies for adding dummy nodes and edges to increase the probability of creating new shortest paths containing a dummy node in the network by creating shortcuts from low-centrality nodes to high-centrality nodes.

### 5.3. Canceling the side effects of dummy element addition to security metrics

When we add dummy elements (i.e., nodes and edges in the graph) as our graph obfuscation, the network topology is changed, consequently affecting the security assessment results. In our approach, a simple approach to avoid the side effects of dummy elements on the security assessment results is that we set their metric values to null. That is, their values are set to 0 (cumulative) or 1 (multiplicative), so when the cloud provider enumerates dummy elements to compute the security metric, the input and output results are the same. We explain how to cancel the side effects of dummy elements added to each security metric in detail as follows.

#### 5.3.1. Independent ASP

In Eq. (2), we can see that newly created paths using dummy nodes could affect the computation of independent ASP. We call those paths *dummy paths*. Therefore, to obtain the security assessment result for independent ASP to be the same as the original network topology, we need to ignore dummy paths in computing independent ASP.

\( d_{ap} \) specifies whether the path \( ap \) is a dummy path (0 for a dummy path, and 1 for a real path) if it contains at least one dummy node. To simplify Eq. (4), we use \( r_1 \) representing \( \sum_{i=0}^{n} \log(\text{prob}_i) \), and \( r_2 \) representing a root for Eq. (4). We note that the root of Eq. (4) would be a complex number. However, we can still encrypt it since the CKKS scheme supports encryption for complex numbers. With \( r_1 \) and \( r_2 \), \( \text{prob}_{ap} \) in Eq. (2) can be expressed as \( d_{ap} \times r_1 + (1 - d_{ap}) \times r_2 \). In this equation, when the path \( ap \) is a real path, we can obtain the original independent ASP result with \( d_{ap} = 1 \). When the path \( ap \) is a dummy path, the computation result becomes \( r_2 \) by setting \( d_{ap} = 0 \). Because \( r_2 \) is a root for Eq. (4), \( \text{prob}_{ap} \) finally becomes 0 in this equation, which does not affect the independent ASP result.

To set \( d_{ap} \) properly, we use a binary field specifying whether each node is a dummy node or not (1 for a dummy node, and 0 for a real node). By calculating the total sum of all field values for a given path \( ap \), we can get 0 for a real path and a natural number (i.e., the number of dummy nodes on the path) for a dummy path. Because our goal is to make \( d_{ap} = 0 \) when \( ap \) is a dummy path; otherwise, \( d_{ap} = 1 \). Therefore, we have to convert 0 to 1 and a natural number to 0. To achieve this goal, we use two additional functions, \( S(n) = \frac{\sin(n\pi)}{n\pi} \) and \( C(n) = 2\cos(n\pi) \), which are the following where \( n \) is either 0 or a natural number. We note that \( S(n) \) returns 1 when \( n \) is near 0 and 0 for a natural number because the encryption of 0 is not exactly the same as 0 in an encrypted form since noise is added for encryption in the CKKS scheme.

We use new Taylor expansions again to approximately compute \( S(n) \) and \( C(n) \) because we cannot directly compute \( S(n) \) and \( C(n) \) in the CKKS scheme. In particular, since the approximation only shows high accuracy when \( n \) is near 0, we diminish the range of \( n \) using the following relations.

\[
S(2n) = \frac{\sin(2n\pi)}{2n\pi} = \frac{2(\sin(n\pi) \cos(n\pi))}{2n\pi} = \frac{S(n)C(n)}{2} 
\]

\[
C(2n) = 2\cos(2n\pi) = (2\cos(n\pi))^2 - 2 = C(n)^2 - 2 
\]

By iterating Eq. (11) and (12), we can approximately compute \( d_{ap} \). To consider the tradeoff between the computation overhead and computation accuracy, we attempted to allocate proper levels of the CKKS scheme to compute \( S(n) \) and \( C(n) \), and the \( \log \) operations, respectively.

#### 5.3.2. Risk & AC

Given a path \( ap \), we can simply check whether \( ap \) is a dummy path or not by calculating the total sum of field values (specifying dummy node status) of all nodes in the path \( ap \) - we can get 0 for a real path and a natural number for a dummy path. Consequently, the user can ignore the results for dummy paths.

#### 5.3.3. ROSE

In Eq. (7), the server calculates a node’s ROSE by multiplying \( 1/C \). Since we set \( 1/C \) to 0 for dummy nodes, the ROSE values for dummy nodes become 0.

#### 5.3.4. Cumulative ASP

The cumulative ASP results of dummy nodes are 0 because the independent ASP value of any dummy node is 0, multiplied to compute its cumulative ASP. In addition, the addition of dummy nodes does not affect the cumulative ASP results of real nodes because, as shown in Eq. (10), “1 - (the ASP of a node)” is multiplied to compute the cumulative ASP results of the corresponding real node. Because the ASP of a dummy node is 0, it does not affect the cumulative ASP of the real node.

### 6. TheoreticalsecurityanalysisofPP–GSM

We consider an adversarial server that wants to know the host data or network topology as an attacker of PP–GSM. The security of PP–GSM against such attacks can be demonstrated using the simulation-based security [32] in the semi-honest setting.

Let \( \mathcal{A} \) be an adversary who wants to obtain information about the host data. For the proof, we generate a simulator \( \mathcal{S} \) against the adversary \( \mathcal{A} \) as follows.

**The Simulator.** When the user encrypts host data using the evaluation key, the simulator generates an encryption of 0s instead of the host data. Note that the simulator has access to the evaluation key, which is public and allows to encrypt any data.

Now, we define the following games for PP–GSM framework \( \pi \) with environment \( \mathcal{E} \).
The game \( \text{REAL}_{(\pi, A, Z)} \): The PP–GSM framework \( \pi \) in the real world with environment \( Z \) and semi-malicious adversary \( A \).

The game \( \text{IDEAL}_{(\pi, S, Z)} \): It constructs the PP–GSM framework with the simulator \( S \). The difference from \( \text{REAL}_{(\pi, A, Z)} \) is that it encrypts \( 0 \)s instead of the host data.

We can prove the computational indistinguishability between real and ideal games as follows.

Claim. \( \text{REAL}_{(\pi, A, Z)} \) and \( \text{IDEAL}_{(\pi, S, Z)} \) are computationally indistinguishable.

Proof. The only difference between the two games is that the user encrypts the real input in \( \text{REAL}_{(\pi, A, Z)} \) while it encrypts \( 0 \) in the game \( \text{IDEAL}_{(\pi, S, Z)} \), if any. Regardless, the adversary cannot distinguish these two ciphertexts computationally because of the indistinguishability chosen plaintext attack (IND-CPA) security about HE [33]. In fact, both ciphertexts are computationally indistinguishable from the uniform random variable over the ciphertext space. □

According to the claim, we can conclude that a difference of advantage between these two games is negligible. However, network topology information can still be leaked because we do not encrypt the network topology. To prevent the leakage of network topology, we propose some heuristics called graph obfuscation in Section 5. We empirically analyze the effects of these heuristics in Section 7.

7. Experimental analysis

We evaluate the performance of PP–GSM in terms of the security assessment (Section 7.2) and the effectiveness of the graph obfuscation strategies (Section 7.3). We perform experiments with a 64-bit Ubuntu 18.04 LTS server that has an Intel Core i5-7600 CPU with four 3.80 GHz Xeon processors and 16 GB RAM for both the user and the CSP machines. We did not consider the transmission time between them. To reduce the bias of the evaluation results, we repeated the measurements for all our experiments until the standard error is less than 1% of the mean. As a scalable GSM, a two-layered Hierarchical Attack Representation Model (HARM) is implemented using an Attack Graph (AG) in the upper layer and Attack Trees (AT) in the lower layer (see [2,34] for full details). However, other GSMs can be used depending on the user requirements (e.g., to utilize features that can only be provided using other GSMs). We use various network structures in our experiment, as shown in Table 2, describing the number of nodes, number of edges, density, and the average node degree of the graph. In each network instance, all host data such as OS and installed applications are synthetically generated for the experiments, but real data would be used in real scenarios as in [35]. The networks used in our experiments are described as follows (each network’s graph is labeled as \( G_{\text{type}}^{\text{size}} \), where \( \text{type} \) includes random (R), Barabasi–Albert (BA), Watts–Strogatz (WS), and real-world autonomous system (AS)).

1. Random graph (\( G_{\text{R}}^{103} \)) was used for comparisons and has connections between nodes assigned randomly.

2. Barabasi–Albert network (\( G_{\text{BA}}^{60} \)) is used to represent a type of scale-free network generated using a preferential attachment mechanism, which corresponds to a power-law degree distribution that is observed in many real networks such as the Internet.

3. Watts–Strogatz network (\( G_{\text{WS}}^{60} \)) is used to represent a random graph with small-world properties, such as clustering and short average path length.

4. Autonomous system (\( G_{\text{AS}}^{103} \)) is used to represent a real-world autonomous system network data representing a collection of routers whose prefixes and routing policies are under the common administrative control.

### Table 2

Properties of the graphs used in experiments.

<table>
<thead>
<tr>
<th># Nodes</th>
<th># Edges</th>
<th>Density</th>
<th>Average degree</th>
<th>Average shortest path length</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_{\text{R}}^{103} )</td>
<td>60</td>
<td>92</td>
<td>0.052</td>
<td>3.07</td>
</tr>
<tr>
<td>( G_{\text{BA}}^{60} )</td>
<td>60</td>
<td>116</td>
<td>0.065</td>
<td>3.87</td>
</tr>
<tr>
<td>( G_{\text{WS}}^{60} )</td>
<td>60</td>
<td>60</td>
<td>0.034</td>
<td>2.00</td>
</tr>
<tr>
<td>( G_{\text{AS}}^{103} )</td>
<td>103</td>
<td>243</td>
<td>0.046</td>
<td>4.72</td>
</tr>
</tbody>
</table>

### Table 3

Time taken (in seconds) to execute basic operations using four HE libraries (denoted as mean (s.d.)).

<table>
<thead>
<tr>
<th>HE libraries</th>
<th>Encryption</th>
<th>Addition</th>
<th>Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFHE</td>
<td>0.001 (0.00)</td>
<td>0.77 (0.01)</td>
<td>11.46 (0.04)</td>
</tr>
<tr>
<td>HElib-BGV</td>
<td>0.20 (0.02)</td>
<td>0.0004 (0.00)</td>
<td>0.15 (0.00)</td>
</tr>
<tr>
<td>SEAL-CKKS</td>
<td>0.10 (0.01)</td>
<td>0.0008 (0.00)</td>
<td>0.01 (0.00)</td>
</tr>
<tr>
<td>SEAL-BFV</td>
<td>0.11 (0.00)</td>
<td>0.0013 (0.00)</td>
<td>0.35 (0.00)</td>
</tr>
</tbody>
</table>

7.1. HE libraries evaluation for PP–GSM

To determine the optimal HE library used for implementing PP–GSM, we first evaluate the performance of four different HE libraries that are widely used: TFHE [12], HElib-BGV [22], SEAL-CKKS [11], and SEAL-BFV [23,24]. For each HE library, we measure the execution times of basic operations such as encryption, addition, and multiplication. Table 3 shows the execution times of the four HE libraries. Since TFHE supports bit operations only, we additionally implemented 32-bit integer addition and multiplication operations using bit-level operations for evaluation.

Among HE libraries tested, SEAL-CKKS outperformed the other HE libraries for multiplications. It is also the only HE library that supports arithmetic operations for floating-point numbers. Therefore, we chose SEAL-CKKS for PP–GSM.

7.2. PP–GSM framework performance evaluation

The times for the key processing components of the PP–GSM framework except the host data encryption step are measured: setting an environment for SEAL (both user and server), generating the GSM (we used HARM [34] (server only), and computing security metrics (server only).

In our implementation, SEAL was installed on both the user and the cloud server. The setup time (step 1) took around 50.3 s for SEAL. Then, the user obfuscates the network topology, encrypts the host data, and sends them to the cloud server (step 2). For each host, an uploaded data consists of four elements: (i) encrypted OS list (vector type, size: 4.7 MB), (ii) encrypted application list (vector type, size: 4.7 MB), (iii) encrypted dummy node information (vector type, size: 4.7 MB), and (iv) its adjacent host information (vector type, size: \( m \) bits where \( m \) is the number of adjacent nodes). Our PP–GSM implementation contains 8192 OS versions and 8192 applications, respectively. Cloud generates the HARM with the user’s data (step 3).

Table 4 shows the mean processing time required for the host data encryption and HARM generation. In principle, the host data encryption time linearly increases as the number of nodes increases. Therefore, \( G_{\text{R}}^{103} \), \( G_{\text{BA}}^{60} \), and \( G_{\text{WS}}^{60} \) took similar times even with the differences in the number of edges. For the HARM generation, the time linearly increases as the number of nodes in a graph increases as described in [2].

Table 5 shows the standard deviations of \( P_{\text{ind}} \), attack cost/risk, and ROSI, which are quite large compared with their mean values, indicating that the execution times to compute those metrics are heavily dependent on the selected nodes in each graph. Notably, in \( G_{\text{AS}}^{103} \), PP–GSM takes less time than \( G_{\text{R}}^{103} \) or \( G_{\text{WS}}^{60} \) to calculate...
ranges in those graphs due to different execution time ranges. In contrast, in a more significant number of paths between nodes exist on those graphs. Therefore, we can deduce that the graph larger than the numbers of nodes in ROSI, the time for calculating cumulative ASP changes with the number of nodes in the GSM instead of the number of paths. Unlike independent ASP, attack cost, risk, and computing security metrics because there are only a few paths between nodes. Therefore, we can deduce that the graph would take a longer execution time to calculate the cumulative ASP than other graph topologies.

To demonstrate how the performance of PP–GSM changes with the attack path length and the number of attack paths, we conduct a sensitivity analysis using an artificially generated graph with 30 nodes. Fig. 4 shows the execution times taken to compute security metrics with respect to the attack path lengths and the number of attack paths, respectively.

When varying the length of attack paths (Fig. 4(a)), the execution time for $P_{\text{ind}}$ is relatively constant. This is due to the fast computation of additions, where all the calculations have been converted to logs. Similarly, $P_{\text{cum}}$ shows relatively constant trends regardless of the attack path lengths. On the other hand, the calculation times of ROSI and attack cost/risk increase linearly. This trend was expected as the number of required calculations increases linearly with respect to the attack path’s length. Since ROSI needs operations for calculating risk, the trend is similar to that of attack cost/risk.

For the effects of the number of attack paths (Fig. 4(b)), we observe a similar trend (near-linear) except for $P_{\text{ind}}$. We can see that the execution time for $P_{\text{ind}}$ also increases linearly with the number of attack paths. Moreover, calculating $P_{\text{ind}}$ requires a significant number of multiplications for several Taylor expansions. Therefore, it takes much longer to compute than other metrics when the number of paths increases.

As discussed in Section 4.1, we approximate $P_{\text{ind}}$ using Taylor expansion. To evaluate the accuracy of the approximation, we performed Spearman’s rank correlation [36], which measures the ranking relationship between the ground-truth independent ASP and the independent ASP computed by the proposed method. For both $G_{103}$ and $G_{60}$, the computed correlation coefficient was 0.99 ($p$-value $< 0.000001$), indicating that the proposed approximation method is highly accurate. For the other remaining metrics, PP–GSM provides the results of the computations, which are the same as the plaintext graph model (i.e., no approximations required).

### 7.3. Effectiveness evaluation of graph obfuscation strategies

We use the number of searched nodes to evaluate the effectiveness of the proposed graph obfuscation strategies. This metric directly relates to the attack efforts/costs, as attacking more nodes results in higher cost and time. For simplification, an attacker’s cost can be formulated with the number of searched nodes by assuming that a node is compromised when explored. Given a graph $G$, a start node $s$, and a target node $e$, we count the number of visited nodes on $G$ until the attacker finds the target node $e$ from the start node $s$. Without loss of generality, we assume that the same cost is required to compromise a host because all hosts’ attack costs are unknown to attackers due to the encryption of the host’s data, including its attack cost.

The effectiveness evaluation of our proposed graph obfuscation strategies against the attackers (A1) and (A2) are shown in Figs. 5 and 6, respectively. In this experiment, for the network instances in Table 2, we first added dummy elements to each network instance, respectively, using the four graph obfuscation strategies described in Section 5.2, and then applied the attack scenarios A1 and A2 with randomly chosen start and target nodes $s$ and $e$ in sequence.

**Attack scenario A1:** Firstly, adding more dummy nodes increased the percentage (%) of searched nodes in all the network instances linearly; indicating that a sufficient number of dummy nodes are needed to increase the attacker’s cost regardless of the graph obfuscation methods. Secondly, the best graph obfuscation method depends on the graph types. For $G_{103}$ and $G_{60}$, the hybrid addition strategy performed better than other strategies, while for $G_{60}$ and $G_{103}$, the high-centrality addition strategy performed the best. However, the high-centrality addition strategy produced the worst results for $G_{103}$ and $G_{60}$ while the hybrid addition strategy overall produced the best or the second-best results for all graph topologies. Therefore, knowing the underlying network topology, we can provide a suitable recommendation strategy based on our results. However, if we do not have sufficient prior information about the underlying network’s topological characteristics, the hybrid addition strategy is recommended.
Table 5. Time taken (in seconds) to calculate independent ASP ($P_{ind}$), Attack cost/Risk, ROSI, and cumulative ASP ($P_{cum}$).

<table>
<thead>
<tr>
<th>Operation</th>
<th>$P_{ind}$</th>
<th>Attack cost/Risk</th>
<th>ROSI</th>
<th>$P_{cum}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G^{60}_R$</td>
<td>124159.44 (55744.12)</td>
<td>38772.13 (16184.96)</td>
<td>28398.46 (13505.27)</td>
<td>30.34 (0.88)</td>
</tr>
<tr>
<td>$G^{60}_{BA}$</td>
<td>48350.21 (18785.09)</td>
<td>16182.59 (6123.31)</td>
<td>12289.77 (5661.43)</td>
<td>29.48 (1.02)</td>
</tr>
<tr>
<td>$G^{60}_{WS}$</td>
<td>6.56 (1.43)</td>
<td>2.83 (1.06)</td>
<td>2.18 (0.97)</td>
<td>30.11 (1.26)</td>
</tr>
<tr>
<td>$G^{103}_{AS}$</td>
<td>30187.55 (12169.32)</td>
<td>10058.34 (4349.53)</td>
<td>8435.39 (3419.02)</td>
<td>54.49 (1.16)</td>
</tr>
</tbody>
</table>

Fig. 5. Comparison of graph obfuscation strategies against A1.

**Attack scenario A2:** The attack scenario A2 deals with a more knowledgeable attacker, and as expected, the obfuscation strategies are not as effective compared with the attack scenario A1 except for $G^{60}_{WS}$. Interestingly, the obfuscation strategies are specifically effective in $G^{60}_{WS}$ for the attack scenario A2 because its attack cost would increase with the number of shortest paths. In the Watts–Strogatz model, new shortest paths would be easily created with dummy nodes.

A better performing obfuscation strategy is the hybrid addition strategy in almost all cases, as summarized in Section 5.2. Since the attacker knows exact locations, the attacker is unlikely to explore high importance nodes, and consequently targeting only high centrality nodes became less effective. Therefore, the hybrid addition strategy is better suited for the attack scenario A2. We surmise that the likelihood of including the dummy nodes in the attack path may increase as the network size increases because the length of shortest paths between nodes also increases.

Based on the above observations, our recommendation is to use the hybrid addition strategy in both attack scenarios (A1 and A2). The high-centrality addition strategy can be applied when the attack scenario A1 is the primary threat model and the underlying network is an AS network topology.

**8. Discussions and future work**

**Extending Graph Attributes:** In this work, we considered node attributes to show the feasibility of our approach, which can be used to compute various security metrics that are widely used, such as independent ASP, risk, ROSI, attack cost, and cumulative ASP [37]. In practice, however, edge attributes can also be important for analyzing the security of a network infrastructure. As a part of further work, we plan to extend our graph model into more generalized ones having both node and edge attributes.

**Use of PP–GSM for Large-sized Networks:** Our experimental results (see Section 7) have shown that PP–GSM could effectively analyze small-sized networks. However, because the computational overheads of PP–GSM are heavy due to the inherent computational limitations of HE, it may not be practical when large-sized or highly dense networks are used. On the other hand, we expect that PP–GSM can efficiently handle the network size in hundreds of nodes with a larger pool of computing resources in the cloud as supported by our experimental results. Moreover, partial networks (e.g., subnets) can be selectively analyzed (i.e., important subnets with highly valuable assets),

---

D. Lee, Y. Oh, J.B. Hong et al. Future Generation Computer Systems 142 (2023) 351–363
leading to the significant reduction of the computational overheads (e.g., divide and conquer approach). The security assessment results of subnets can hierarchically be incorporated with the remaining network components’ results again [34]. Moreover, we believe that such a performance issue would improve in the future with technical advances in underlying HE algorithms and hardware because our framework is flexibly designed and can be implemented with any other HE libraries.

Malicious CSP and the Cloud Server: We assumed that the cloud server is “honest-but-curious”, which correctly performs security assessment operations but wants to reveal hosts’ private data. However, this assumption may not hold in the real world (i.e., the cloud server can provide fake or incorrect security assessment results to users). To address this issue, users can outsource security assessment operations to multiple cloud services and compare their results to verify the correctness of those results. Another approach is to implement a verification method such as in [38], where checkpoints are implemented for the user to check. However, this was possible as the user also knew the computation being carried out (they outsourced linear programming), but in our case, the user may not know how the metric could be computed. Therefore, the implementation of process validation is required to ensure other attack models that are more invasive, and the defense against them will be explored in our future work.

Threats to the Validity: The threats to the validity of this work mainly come from two aspects. A possible threat is whether we used representative network models for evaluation. Our experiments were conducted on only four different network models. We have excluded large-sized network models from our evaluation because we do not have sufficient computational resources to process such large-sized network models. Another threat to validity is the generalizability of PP-GSM performance analysis. Because the current implementation of PP-GSM relies on SEAL’s implementation, our evaluation may not be applicable to different HE implementations. Analyzing the effects of HE implementations is part of our future work.

9. Related work

Security Assessment as a Service: There are a few works on outsourcing security assessment as a service implemented on the cloud. Kaliski et al. [39] and Hossain et al. [40] proposed a risk assessment in the cloud for the cloud and Supervisory Control and Data Acquisition (SCADA) system, discussing the importance and challenges faced to implement such systems. However, no testbed implementation and evaluation were made to evaluate their claims. Landoll et al. [41] also introduced a data-gathering method for security assessment or discovering vulnerabilities in existing security analysis. Theocharidou et al. [42] surveyed the issues and implementations of security as a service in the cloud, highlighting the lack of real implementations to validate previous work’s theories. Further, GSMs have not been considered in the framework, limiting the capabilities of privacy-preserving security assessment. Rak et al. [43] discussed and evaluated the ability to assess the cloud service provider’s security that satisfies the security requirements of enterprises storing data remotely. In a sense, this partially fulfills the security assessment of the user systems, but with only from the data point of view. Varadharajan and Tupakula [17] proposed security as a service model for the cloud environment, which also provides security features to the users, but still within the scope of the cloud. Similarly, Sun et al. [18] proposed security as a

![Fig. 6. Comparison of graph obfuscation strategies against A2.](image)
service for micro-services-based cloud applications. These works and others have focused on providing security as a service in the past, but their boundaries are usually limited to the scope of the cloud, and their implementations are limited as no experimental solutions are provided in general. Our proposed solution extends the capabilities to assess not only the cloud-based systems but also assess local systems and networks, with experimental analysis and implementations that can be utilized by others (further discussed in Section 3).

**Private Graph Analysis:** Metrics such as shortest distance has been proposed in [44]. Similarly, search queries have been one of the major focuses in the area of private graph analysis [45] with improved performance [46–48]. Unfortunately, these analysis methods focused more on data retrievals and searching, which are not sufficient functions when computing security metrics, which require more complex and dynamic calculations as shown in Section 3.4. As a result, it is not possible to transform existing private graph analysis methods to security metric calculations. Moreover, some security metrics are dynamic as shown in [37], but to the best of our knowledge, this has not been integrated into the private graph analysis. On the other hand, our proposed PP–GSM framework provides a platform where various security metrics can be computed while maintaining the privacy of the user. Our approach can be extended further to provide methods to compute other metrics (e.g., performance, reliability, etc.), providing functionalities towards outsourcing privacy-preserving cloud services, including security assessment as a service, for users.

10. Conclusion

Cloud platforms have advanced to provide various services, including SaaS, but users might hesitate to send their information to the cloud due to privacy concerns, which may be sensitive. To address this issue, we have proposed PP–GSM, a privacy-preserving security assessment framework using HE to carry out a security assessment of networks without revealing users’ sensitive data. Our experimental results have shown that PP–GSM produces highly accurate security assessment results. Moreover, our proposed graph obfuscation techniques (e.g., adding dummy elements to nodes having high betweenness centrality) can hide the network topology without affecting the security assessment results.

**CRediT authorship contribution statement**

**Dongwon Lee:** Methodology, Implementation, Evaluation, Writing – original draft. **Yongwoo Oh:** Visualization, Investigation. **Jin B. Hong:** Methodology, Writing – original draft, Writing – review & editing. **Hyoungshick Kim:** Supervision, Methodology, Writing – original draft, Writing – review & editing. **Dan Dongseong Kim:** Methodology, Writing – review & editing.

11. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**Data availability**

Data will be made available on request.

**Acknowledgments**

This work was supported by the Institute for Information & communications Technology Promotion (IITP) grants (No. 2018-0-00532 and No. 2022-0-01199) funded by the Korean government.

**References**


Dongwon Lee is an undergraduate student in the Department of Computer Engineering at Sungkyunkwan University. He was an intern at CSIRO’s Data61 from September 2019 to November. He is studying computer security supervised under Hyongshick Kim. His research interest is focused on cryptography.

Yongwoo Oh is an undergraduate student in the Department of Software, Sungkyunkwan University, South Korea. He is studying computer security supervised under Hyongshick Kim. His research interests are software security and usable security.

Jin B. Hong is a Lecturer in the Department of Computer Science and Software Engineering at the University of Western Australia, Australia. He received his Ph.D. degree in Computer Science from the University of Canterbury, New Zealand. His research interests are security modeling and analysis of computers and networks including cloud computing, SDN and IoT, and Moving Target Defense.

Hyongshick Kim is an associate professor in the Department of Software, Sungkyunkwan University. He received a BS degree from the Department of Information Engineering at Sungkyunkwan University, an MS degree from the Department of Computer Science at KAIST and a Ph.D. from the Computer Laboratory at the University of Cambridge in 1999, 2001 and 2012, respectively. After completing his Ph.D., he worked as a post-doctoral fellow in the Department of Electrical and Computer Engineering at the University of British Columbia. He was a distinguishing visiting scientist with CSIRO’s Data61 from 2019 to 2020. He previously worked for Samsung Electronics as a senior engineer from 2004 to 2008. His current research interest is focused on usable security and software security.

Dongseong Kim is an associate professor in the School of ITIE at the University of Queensland, Australia since Jan 2019. Prior to UQ, he led the Cybersecurity Lab at the University of Canterbury (UC), Christchurch, New Zealand from August 2011 to December 2018. From June 2008 to July 2011, Dr Kim was a postdoc at Duke University, North Carolina in the US. He worked on Metrics, Models, and Analysis of Network Security funded by the US NSF and Perforability Analysis of Cloud Computing funded by the IBM T.J. Watson and NEC Japan, respectively. He was a visiting scholar at the University of Maryland, College Park, Maryland in the US in 2007. Dr Kim’s research was supported by the NATO SPS programme, the New Zealand’s MBIE, the Qatar National Research Fund, the US Army Research Lab, and the Agency for Defense Development in South Korea. His research interests are in cybersecurity and privacy.