Privacy-Preserving Network Provenance

Yuankai Zhang
Georgetown University

Adam O'Neill
Georgetown University

Micah Sherr
Georgetown University

Wenchao Zhou
Georgetown University

{yuankai, adam, msherr, wzhou}@cs.georgetown.edu

ABSTRACT

Network accountability, forensic analysis, and failure diagnosis are becoming increasingly important for network management and security. *Network provenance* significantly aids network administrators in these tasks by explaining system behavior and revealing the dependencies between system states. Although resourceful, network provenance can sometimes be too rich, revealing potentially sensitive information that was involved in system execution. In this paper, we propose a cryptographic approach to preserve the confidentiality of provenance (sub)graphs while allowing users to query and access the parts of the graph for which they are authorized. Our proposed solution is a novel application of searchable symmetric encryption (SSE) and more generally structured encryption (SE). Our SE-enabled provenance system allows a node to enforce access control policies over its provenance data even after the data has been shipped to remote nodes (*e.g.*, for optimization purposes). We present a prototype of our design and demonstrate its practicality, scalability, and efficiency for both provenance maintenance and querying.

1. INTRODUCTION

Network provenance [56] allows network administrators to determine the precise causes and effects of network state, significantly easing otherwise burdensome tasks such as failure diagnostics, forensic analysis, network debugging, and network accounting. Efficient methods of storing and querying provenance [57] have enabled practical network provenance systems, spurring provenance products that have been deployed in enterprise networks [45].

To date, deployments of network provenance systems have been restricted to single administrative domains due in part to the security and privacy issues that arise when data are shared across administrative boundaries. However, since the Internet is governed by distributed protocols (BGP and DNS are notable examples), secure provenance schemes that operate across networks could provide enormous benefit. For example, a provenance-enabled Internet control-plane enables network administrators to reason about its own routing rules and better diagnose cross-domain routing faults such as bad gadgets [58]; with provenance support, participants of distributed hash tables can better understand sudden workload shifts and performance glitches.

Existing work investigates *secure* provenance systems in distributed environments that are robust against malicious or compromised endpoints or network elements [40, 58]. There, the focus has been on detecting false provenance information. This paper argues that authenticity is not by itself sufficient for practical cross-domain provenance systems. Provenance data collected on individual devices contains rich information that reveals details of system executions and decision making processes, some of which might be considered sensitive or confidential and should not be visible by unauthorized parties in the distributed system. When disseminating provenance information, we need to take into consideration organizations' privacy preferences. Prior position papers [6, 25] have previously identified such need, and suggested that partial and differential provenance access should be granted in a privacy-preserving manner.

This paper introduces *privacy-preserving network provenance* (PPNP), a distributed provenance scheme that supports the richness of network provenance while providing strong privacy guarantees over confidential data. We propose a cryptographic approach to preserve the confidentiality of provenance (sub)graphs while allowing authorized nodes to query and access the parts that are visible to them. In PPNP, the access control of the vertices and edges of a provenance graph is determined by their "labels." Provenance data owners can freely assign the labels (for example, based on the types of the corresponding system state/events) and their corresponding access control policies.

Our proposed PPNP scheme leverages previous work on a cryptographic primitive called Searchable Symmetric Encryption (SSE), introduced by Curtmola et al. [15]. Informally, SSE builds a *secure index* (also called a dictionary) that maps each searchable keyword to the corresponding data item(s), but is only searchable by keyword w if a user possesses a trapdoor for w. Even after the generated secure index is shipped to another entity, a node still controls the access of its data based on the roles of requesters. A richer primitive called Structured Encryption (SE) due to Chase and Kamara [10] considers arbitrary data structures such as graphs. Our proof-of-concept scheme of PPNP builds upon SSE and SE constructs by Cash et al. [9] and Chase and Kamara [10] and allows privacy-preserving retrieval of a labeled (sub)graphs of an encrypted graph. PPNP guarantees that the only portions of the provenance graph that are revealed are those for which the querier is authorized.

In particular, we adapt an SSE scheme of Cash et al. [9] for use in our main construction and optimize it to reduce bandwidth costs, making PPNP practical for large, distributed deployments. To demonstrate the practicality and scalability of PPNP, we con-

Proceedings of the VLDB Endowment, Vol. 10, No. 11

struct a proof-of-concept implementation which we call PrivProv and measure its performance and overhead for two diverse provenance tasks: inter-domain routing and Chord [51]. Our empirical results demonstrate that PrivProv achieves strong confidentiality guarantees with a negligible increase in latency and low bandwidth overhead.

2. BACKGROUND

Before describing our privacy-preserving network provenance scheme, we describe our system model (Section 2.1) and provide a brief background on network provenance (Section 2.2).

2.1 System Model

Our model of a distributed system is based on that described by Zhou et al. [57] in their exploration of efficient provenance techniques. We briefly review this model here.

We model a distributed system as a set $N = \{N_1, N_2, ..., N_n\}$ of nodes that communicate by sending messages over a network. The state of a node at a fixed point in time is expressed as a set of *tuples*. State can be affected either by *base tuples* that are inserted or deleted directly by the user, or by *derived tuples* that are derived from existing rules. Derived tuples may be communicated between nodes.

We use *Network Datalog* (NDlog) [34–37] to compactly describe the derivation rules and dependencies among tuples that can exist in the system. NDlog has been similarly applied to express rules in other application settings, including provenance systems [57], sensor and ad hoc networks [12, 33], and overlay networks [38, 49]. Although our choice of NDlog is motivated by its popularity in network provenance systems, our proposed PPNP scheme is agnostic to the particular provenance language; in particular, our model is compatible with legacy systems so long as they can express derivation rules and dependencies.

Example: Network Routing. We use the mincost protocol from ExSPAN [57] as a driving example. mincost computes the lowest-cost paths between each pair of nodes in a network. For this example, we assume an undirected topology. mincost is defined as the following NDlog *rules*:

```
mc1 cost(@S,D,C) :- link(@S,D,C).
mc2 cost(@S,D,C) :- link(@Z,S,C1),
                    mincost(@Z,D,C2), C=C1+C2.
mc3 mincost(@S,D,MIN<C>) :- cost(@S,D,C).
```

Each rule has the form "`ruleid p :- q1, q2, ..., qn.`" signifying that "`p` should be derived whenever `q1`, `q2`, ..., and `qn` all exist at the same time." Unlike Datalog, NDlog supports a *location specifier* in each predicate, written as `@X`, where `X` is the node on which the tuple is located. For example, `cost` tuples derived via rule `mc1` reside on the same node as the corresponding `link` tuples since both carry the same location specifier (`@S`).

In the above mincost example, a base tuple `link(@S,D,C)` exists if `S` has a direct link (an edge in the network topology) to `D`, at a cost of `C`. Rule `mc2` derives a `cost(@S,D,C)` tuple if there is a direct link from `S` to `Z` and a minimum cost route (defined by rule `mc3`) from `Z` to `D`. All paths with the same source and destination are aggregated in rule `mc3` to determine the minimal path cost.

Protocols run continuously in NDlog, and tuples are derived (or underived) in response to the insertion, deletion, or modification of base tuples. For example, `mincost` tuples may change if the cost of a link changes.

2.2 Network Provenance

The notion of provenance has been extensively explored and successfully applied to a variety of areas [7, 8, 13, 19, 22, 26, 27, 42, 43, 46, 47, 52, 53]. Of relevance here, *network provenance* [57–59] helps operators of distributed systems better understand and reason about the derivation history of network states in their dynamic distributed environments. Such capability has been shown to be widely applicable to network management tasks such as fault debugging, root cause analysis, and accountability.

For example, consider a scenario during the deployment of the mincost program, where a three-node network consists of nodes a, b, and c. The only link in the network is a link $b \rightarrow c$ with a cost 3. When a new link $b \rightarrow a$ with a cost of 1 is inserted into the network, a new path $c \rightarrow b \rightarrow a$ becomes available in response to the link insertion. The network provenance that explains the derivation of `mincost(c,a,4)` is as follows:

- when node `b` discovers the new link to node `a`, it is reflected as an insertion of base tuple **`link(@b,a,1)`** into `b`'s local `link` table;
- the insertion of the `link` tuple triggers rule `mc1`, resulting in **`cost(@b,a,1)`**;
- rule `mc3` is then used to derive **`mincost(@b,a,1)`**, since `cost(@b,a,1)` is the shortest path between `b` and `a`; and
- finally, combining the derived `mincost(@b,a,1)` tuple with the fact that there is a link with cost 3 between node `b` and c (represented as `link(@b,c,3)`), `b` concludes that there is a path with cost 4 between `a` and `c` (represented as `cost(@c,a,4)`). This information is also shipped to `c` to notify `c` about the new available route.

Network provenance is captured as a directed graph, where the vertices are the events (e.g., `link(@b,a,1)`) and rule execution instances (e.g., `mc1` and `mc2`), and the edges represent dependency relationships between the events and rule execution instances.

There have been multiple proposals and standards for provenance models; notable examples include Open Provenance Model (OPM) [41] and PROV [23]. While these provenance models vary in their detail (e.g., OPM introduces a comprehensive model that captures the causal relationships between three types of entities: artifacts, processes, and agents), they all share a common graph-based notation in which edges in the graph represent dependency or causal relationships among the vertices. This allows for an opportunity to develop a scheme that can be *generally* applied to a wide range of provenance models. The proposed privacy-preserving scheme employs structured encryption for general graph representation of provenance, and therefore is applicable to any graph-based provenance model. While in this paper, we focus on the network provenance model, the proposed private provenance scheme can be extended to support other graph-based provenance models by keeping additional annotations with the edges and vertices in the graph. (We elaborate how the proposed scheme can be adopted to the PROV model in Section 5.4.)

3. OVERVIEW AND GOALS

Although the use of provenance has been suggested for a variety of settings, we posit that the lack of confidentiality in existing provenance systems severely limits its practical use since data cannot be shared securely across administrative boundaries.

3.1 Case Studies: A Need for PPNP

Provenance information, by design, reveals information about derived and communicated state. In distributed environments with multiple data owners, such uninhibited sharing of information significantly hinders the deployment of provenance systems, particularly when data may be derived from sensitive sources (for example, in the case of inter-domain routing). To motivate the need

for confidentiality, we survey existing provenance applications and discuss how they could benefit from PPNP. For brevity, we discuss three types of provenance applications in this paper and refer interested readers to a technical report for a full survey [55].

In our survey, we consider a model in which the provenance graph is distributed among different *authority domains*. Each authority domain owns the data corresponding to its piece of the provenance graph. For example, in the case of provenance-enabled inter-domain routing, an autonomous system (AS) owns the portion of the provenance graph for a routing announcement that is based off of the AS' internal routing policies. To evaluate the benefits of PPNP for our surveyed use-cases, we consider whether—in an actual deployment—authority domains would be concerned about protecting the confidentiality of the data it shares with other authority domains.

Distributed system debugging [11, 54] requires strong confidentiality. An entity that wishes to debug its own system may depend on provenance data from external authorities. For example, an observed misbehavior at router A might be caused by configuration errors at router B which is several hops away and owned by a different authority. Even if different authorities are willing to cooperate for debugging purposes, they are unlikely to share plaintext provenance since it contains business secrets that are private and sensitive. Protections are needed to permit only authorized access to the sensitive data.

PPNP enables per-authority domain access controls, which makes it especially well-suited for such use-cases. At the same time, some derivations have distinguishable patterns in sizes, thus simply encrypting the derivations may still leak some information. We show later in Section 5 that PPNP provides strong confidentiality and strong derivation size hiding for these applications.

In a provenance-aware healthcare multi-agent system (HC-MAS) there are multiple authorities of different healthcare actors [31], which naturally makes the system decentralized and distributed. Such systems require very strong confidentiality due to the fact that healthcare records are highly private and sensitive. The Organ Transplant Management Application (OTMA) example described by Kifor et al. [31] demonstrates how a series of complex procedures are annotated with provenance, where each autonomous authority carries out a subset of processing steps. Crucially, it is critical to protect the confidentiality of internal decisions as these themselves may be based on highly sensitive information (for example, the results of HIV tests).

Internet-of-Things (IoT) devices produce enormous amounts of data, and provenance has been proposed as a mechanism to better understand and trace their data [50]. Since the data produced by IoT devices is very often sensitive, there is an obvious need to protect the confidentiality of this information, especially given the pervasive use of always-on IoT devices. Here, PPNP provides an important benefit by enabling operators to determine the causes and effects of IoT state in a manner that protects individuals' privacy.

Finally, in applications where provenance is not distributed and is held by a single authority [5, 17, 28, 32], PPNP is less useful since more straightforward encryption and role-based access control could be used to preserve privacy. In this paper, we focus on distributed provenance graphs that span administrative boundaries.

3.2 Security Goals

PPNP enables support for privacy-preserving provenance maintenance and querying. Our high-level goal is to enable operators of separately administered systems to share provenance data in a more controlled manner such that sensitive data are revealed only to authorized parties.

We conceptualize privacy-preserving network provenance by considering a coloring of the provenance graph. Let C be a set of *colors*, which can be roughly viewed as credentials. In our privacy-preserving provenance model, each data owner has one or more colors, and assigns exactly one of its colors to each of its tuples. Conceptually, the coloring of nodes defines the access control policy for the provenance graph.

Dismissed: simple encryption of tuples. At first blush, it may seem sufficient to simply encrypt the values of tuples stored in the provenance graph to provide privacy protections. In our initial exploration of privacy-preserving provenance, we considered using attribute-based encryption (ABE) [48] where colors represented attributes and tuples were encrypted to enforce access controls.

However, encrypting only the vertices in the provenance graph is insufficient to provide meaningful security since the structure of a provenance graph can also be revealing. For example, in the case of inter-domain routing, the structure of a provenance subgraph belonging to an autonomous system (AS) may reveal the size of that AS, its connections, and its business relationships (i.e., peers). More generally, merely obfuscating the "labels" in a graph has been dismissed by the security community as a poor method of information hiding [44].

We require a stronger security model in which any party, honest or malicious, learns only the portions of a given provenance graph for which it possesses the corresponding colors. We express our desired security properties more concretely next, and present and analyze a scheme based on Searchable Symmetric Encryption [15] (SSE) and more generally Structured Encryption (SE) [10] in Section 4. We view part of our contribution as identifying a novel application of SSE and SE to network provenance, which is also much lighter-weight compared to ABE.[1]

Security goals and threat model. Let $G = (V, E)$ be a simple directed graph[2] and $L : V \rightarrow 2^C$ be a labeling function of vertices (tuples) to colors. In a privacy-preserving network provenance system, in response to a query, a (honest or malicious) querier P with colors $C_P \subseteq C$ learns only a subgraph $G' \subseteq G$ such that $G' = (V', E')$ where $V' = \{v \in V | L(v) \in C_P\}$ and $E' = \{(v_i, v_j) | v_i, v_j \in V \text{ and } \{v_i, v_j\} \cap V' \neq \emptyset\}$.

We note that the subgraph may also contain *half-edges* in which only one of its adjacent vertices is in the subgraph; for the "invisible" node in half-edges, P learns only that there is a corresponding node in the full provenance graph.

We will use standard cryptographic primitives such as blockciphers and hash functions, and make standard assumptions about their security. In Section 4, we formally define and analyze security properties of PPNP.

Additional goals. A privacy-preserving network provenance scheme should provide the following properties:

- **Completeness:** Upon receiving a query, PPNP should return all vertices and edges that the requester is allowed to view.
- **Portability:** Provenance queries can be answered efficiently, and the results should remain secure and complete even when the provenance information is communicated to a remote third-party node (e.g., for caching purposes or to delegate

[1] One important difference between SE and ABE is that SE does not provide "collusion-resistance," meaning different users might be able to combine their keys to decrypt data neither could decrypt on their own. However, for us credentials will be atomic, i.e., not Boolean formulae on attributes, so this does not cause an issue.

[2] A simple directed graph is a directed graph with no self-loops or multiple edges.

storage to a cloud-based infrastructure) that may not have full access to plaintext provenance.
- **Performance:** The protocol should have reasonably low overhead, especially during provenance maintenance time. This is particularly important for long running distributed systems that have complex logic.

Importantly, PPNP does not identify misbehaving nodes. While there do exist techniques for discovering nodes in provenance systems that misbehave or equivocate when answering provenance queries [58], such secure network provenance systems depend on having unrestricted access to provenance information. We suspect—although we do not investigate it here—that secure provenance systems can be straightforwardly deployed alongside PPNP within administrative domains. We leave the study of the integration of secure network provenance with privacy-preserving network provenance as an interesting future research direction.

4. STRUCTURED ENCRYPTION FOR COLORED SUBGRAPHS

PPNP PPNP relies on a new structured encryption scheme [10] for "colored subgraph queries." Intuitively, such a scheme allows a user to encrypt a colored graph such that the encryptor can issue a per-color token that allows decrypting only the subgraph whose vertices have that color. The scheme leverages searchable symmetric encryption (SSE) [15] which allows data to still be efficiently searchable by a third-party even when it remains encrypted (Section 4.3).

Briefly, PPNP operates by constructing a dictionary in which the colors are the keys and each corresponding value contains the vertices with that color and the edges between them. It also contains a secret share of the corresponding cross-color edges, where the other secret share is in the value for the other color (Section 4.4). From the retrieved query result, an unauthorized user might be able to infer an approximation of graph size, but she will not know the exact size (due to padding) or details of individual edges as they are protected by encryption and/or secret sharing. We further describe in Section 4.5 fragmentation, a performance optimization, and present the complete leakage profile.

4.1 Notation and Conventions

All algorithms, including adversaries, are required to be efficient and may be randomized unless otherwise specified. If A is a randomized algorithm, then $x \leftarrow\!\!\$\ A(\ldots)$ denotes running A on the elided inputs and fresh random coins, and assigning the output to x. If A is deterministic, we drop the dollar-sign next to the arrow. We denote by $\Pr[\,A(\ldots) \Rightarrow x\,]$ the probability that A outputs x when run on the elided inputs and fresh random coins.

If S is a finite set, then $s \leftarrow\!\!\$\ S$ denotes sampling a uniformly random element from S and assigning it to s. We abbreviate $s_1 \leftarrow\!\!\$\ S\ ;\ s_2 \leftarrow\!\!\$\ S$ by $s_1, s_2 \leftarrow\!\!\$\ S$. Strings are binary. If s is a string, then s_i denotes its i-th bit. If s_1, s_2 are strings then $s_1 \| s_2$ denotes an encoding of s_1, s_2 from which they are uniquely recoverable. We often use other objects such as graphs as arguments to algorithms or with string notation, with the understanding that they are first implicitly encoded as strings in some canonical way.

4.2 Formal Model and Security

Formal Model. Let $G = (V, E)$ be a simple directed provenance graph, C be a set of colors, and $L : V \to 2^C$ be a labeling function. A *structured encryption scheme for privacy preserving network provenance* is a tuple of four algorithms:

$$\text{GE} = (\textsc{KeyGen}_G, \textsc{Enc}_G, \textsc{TokenGen}_G, \textsc{Search}_G)$$

The key generation algorithm $\textsc{KeyGen}_G$ outputs a key K. The encryption algorithm $\textsc{Enc}_G$ takes as inputs a key K, a provenance graph G, and a labeling function L, and outputs an encrypted graph γ. The deterministic token generation algorithm $\textsc{TokenGen}_G$ takes as inputs a key K, and a color c, and outputs a token τ_c. The deterministic query-answering algorithm $\textsc{Search}_G$ takes as input a list of tokens $(\tau_{c_1}, \ldots, \tau_{c_q})$ and an encrypted graph γ, and outputs a graph G'.

Correctness. We say that GE is **correct** if for all G, C, L as above, all subsets $C' \subseteq C$, all K output by $\textsc{KeyGen}_G$, all γ output by $\textsc{Enc}_G(K, G, L)$, all $c' \in C'$ and all $\tau_{c'}$ output by $\textsc{TokenGen}_G(K, c')$, it holds that $\textsc{Search}_G((\tau_{c'})_{c' \in C'}, \gamma)$ outputs the subgraph $G' = (V', E')$ of G such that $V' = \{v \in V | L(v) \in C'\}$ and $E' = \{(v_1, v_2) \in E | v_1, v_2 \in V'\}$.

Remark 4.1 *The structured encryption schemes we define are "response-revealing" in the terminology of [10]. Roughly, this means that there is not a separate decryption algorithm, and the search algorithm reveals the result of the query in the clear. This is because in our application the querier and the entity executing the* $\textsc{Search}_G$ *are actually the same. This will also be the case for the searchable symmetric encryption schemes we define later.*

Security. We consider a non-adaptive, simulation-based notion. Here "non-adaptive" refers to the fact that we only consider an adversary that chooses its queries before seeing the encrypted graph, and "simulation-based" captures the intuition that the adversary only learns some associated leakage of the graph. Namely, given a leakage function $\textsc{Lk}$, consider the following experiments with an adversary A against GE and simulator S. (Here and in similar experiments below, A and S maintain state that we omit for simplicity.) In the *real* experiment $\textsc{Real}_{\text{GE}}(A)$:

- The adversary A first outputs a graph G, set of colors C, labeling function L, and colors $c_1, \ldots, c_q \in C$.
- The experiment next generates $K \leftarrow\!\!\$\ \textsc{KeyGen}_G$, $\gamma \leftarrow\!\!\$\ \textsc{Enc}_G(K, G, L)$, and $\tau_{c_i} \leftarrow \textsc{TokenGen}(K, c_i)$ for all $i = 1$ to q. It then passes $(\gamma, \tau_{c_1}, \ldots, \tau_{c_q})$ to A.
- Finally, the adversary A outputs a bit b, which is output by the experiment.

In the *ideal* experiment $\textsc{Ideal}_{\textsc{Lk},\text{GE}}(A, S)$:

- The adversary A first outputs a graph G, set of colors C, labeling function L, and colors $c_1, \ldots, c_q \in C$.
- The experiment then generates $\ell \leftarrow \textsc{Lk}(G, C, L, c_1, \ldots, c_q)$ and passes ℓ to the simulator S.
- Finally, the simulator S outputs a bit b, which is output by the experiment.

We say that GE is **$\textsc{Lk}$-Secure** if for every adversary A there is a simulator S such that

$$\Pr[\textsc{Real}_{\text{GE}}(A) \Rightarrow 1] - \Pr[\textsc{Ideal}_{\textsc{Lk},\text{GE}}(A, S) \Rightarrow 1]$$

is sufficiently small (say 2^{-128}).

4.3 Searchable Symmetric Encryption

We provide some background on searchable symmetric encryption (SSE), first introduced by Curtmola *et al.* [15], which we will use in our main construction below.

Let W be a set of keywords (also called labels). A *dictionary* is a list of keyword-data pairs $\mathcal{I} = ((w_1, d_1), \ldots, (w_n, d_n))$, where $w_1, \ldots, w_n \in W$ are unique. We write $w \in \mathcal{I}$ to mean that $w = w_i$ for some $1 \le i \le n$. For $w \in \mathcal{I}$ we denote by $\mathcal{I}(w)$ the corresponding data d_i.

Syntax. An SSE scheme is a tuple of four algorithms

$$\text{SSE} = (\text{KeyGen}_S, \text{Enc}_S, \text{TrapGen}_S, \text{Search}_S)$$

defined as follows. The key-generation algorithm KeyGen_S outputs a key K. The encryption algorithm Enc_S takes as inputs a key K and a dictionary $\mathcal{I}$ and outputs an encrypted dictionary $\widetilde{\mathcal{I}}$. The deterministic trapdoor generation algorithm TrapGen_S takes as inputs a key K and a keyword w, and outputs a trapdoor T_w. The deterministic searching algorithm Search_S takes as inputs a trapdoor T_w and an encrypted dictionary $\widetilde{\mathcal{I}}$, and outputs a string d.

Correctness. We say that SSE is **correct** if for all $n \in \mathbb{N}$, all dictionaries $\mathcal{I} = ((w_1, d_1), \ldots, (w_n, d_n))$, all keywords $w \in W$, all K output by KeyGen_S, all $\widetilde{\mathcal{I}}$ output by $\text{Enc}_S(K, \mathcal{I})$, and all T_w output by $\text{TrapGen}_S(K, w)$, it holds that $\text{Search}_S(T_w, \widetilde{\mathcal{I}})$ outputs $\mathcal{I}(w)$ if $w \in \mathcal{I}$ and $\perp$ otherwise.

Security. As above, we consider a non-adaptive, simulation-based notion. Given a leakage function Lk, consider the following experiments with an adversary A against SSE and simulator S. In the *real* experiment $\text{Real}_{\text{SSE}}(A)$:

- The adversary A first outputs a dictionary $\mathcal{I}$ as above and keywords $w_1, \ldots, w_q \in W$.
- The experiment next generates $K \leftarrow\!\!\$\ \text{KeyGen}_S$, $\widetilde{\mathcal{I}} \leftarrow\!\!\$\ \text{Enc}_S(K, \mathcal{I})$, and $T_{w_i} \leftarrow \text{TrapGen}_S(K, w_i)$ for all $i = 1$ to q. It then passes $(\widetilde{\mathcal{I}}, T_{w_1}, \ldots, T_{w_q})$ to A.
- Finally, the adversary A outputs a bit b, which is output by the experiment.

In the *ideal* experiment $\text{Ideal}_{\text{Lk,SSE}}(A, S)$:

- The adversary A first outputs a database $\mathcal{I}$ as above and keywords $w_1, \ldots, w_q \in W$.
- The experiment then generates $\ell \leftarrow \text{Lk}(\mathcal{I}, w_1, \ldots, w_q)$ and passes ℓ to the simulator S.
- Finally, the simulator S outputs a bit b, which is output by the experiment.

We say that SSE is **Lk-Secure** if for every adversary A there is a simulator S such that

$$\Pr[\text{Real}_{\text{SSE}}(A) \Rightarrow 1] - \Pr[\text{Ideal}_{\text{Lk,SSE}}(A, S) \Rightarrow 1]$$

is sufficiently small.

4.4 Main Construction

We give a construction of structured encryption scheme for colored subgraphs from an SSE scheme. The techniques are inspired by the structured encryption scheme for labeled graphs by Chase and Kamara [10], but the details of our construction differ.

Scheme. Let $\text{SSE} = (\text{KeyGen}_S, \text{Enc}_S, \text{TrapGen}_S, \text{Search}_S)$ be an SSE scheme. We assume graphs have vertex labels of some fixed length. Our structured encryption scheme for colored subgraphs with nonce-length $k \in \mathbb{N}$ is defined as

$$\text{GE}_1 = (\text{KeyGen}_1, \text{Enc}_1, \text{TokenGen}_1, \text{Answer}_1)$$

in Figure 1. Correctness follows from taking the nonce-length to be sufficiently large to avoid collisions.

Intuitively, the scheme simply uses the underlying SSE scheme to store the data for each color. The data for each color contains the vertices with that color and the edges between them. It also contains a *secret share* of the corresponding cross-color edges, where the other secret share is in the data for the other color. We explain our algorithm in more details in Section 5, with a driving example that contains a concrete provenance graph.

Remark 4.2 *In the implementation described in Section 6, we do not store all the graph data in the dictionary as defined in the above scheme, but rather the data stored in the dictionary serves as pointers to encrypted vertices or edges, which can then be decrypted by the entity processing the query. This is done only for performance and does not affect security.*

Security Analysis. For the security analysis, we define leakage profile Lk_1 in terms of a given leakage profile Lk for the underlying SSE scheme, plus some additional leakage on cross-color edges. Namely, we define $\text{Lk}_1(G, C, L, c_1, \ldots, c_q)$ as $\text{Lk}(\mathcal{I}, c_1, \ldots, c_q)$, for $\mathcal{I}$ defined as in Figure 1, plus $\mathcal{H}$ and N_H, where

$$\mathcal{H} = \{(u, v) \mid L(u) = c_i \neq c_j = L(v), c_i, c_j \in (c_1, \ldots, c_q)\}$$

and

$$N_H = \{|(x, y)| \mid |(L(x), L(y)) \cap (c_1, \ldots, c_q)| = 1\} .$$

Note that N_H is the number of cross-color edges from(to) vertices within $(c_1, \ldots, c_q)$ to(from) vertices outside $(c_1, \ldots, c_1)$; the actual edges remain secret.

For the SSE schemes we consider, the data corresponding to the keywords for which trapdoors are requested is output by the leakage profile (as well as some additional leakage discussed later). This means GE_1 leaks not only the subgraph corresponding to the colors for which tokens are requested, but also those cross-color edges whose two colors are in $(c_1, \ldots, c_q)$, and the total number of cross-color edges to or from this subgraph to vertices with other colors that are not in $(c_1, \ldots, c_q)$. Leaking the total number of cross-color edges could be eliminated by using padding and a message authentication code, but we omit this from our main construction for bandwidth efficiency.

Formally, we prove the following theorem:

Theorem 4.3 *If* SSE *is* Lk*-secure, then* GE_1 *is* Lk_1*-secure.*

Proof Sketch. Given an adversary A_1 against GE_1, there is a corresponding adversary A against SSE defined in the natural way. Hence, by Lk-security of SSE, there is a simulator S such that

$$\Pr[\text{Real}_{\text{SSE}}(A) \Rightarrow 1] - \Pr[\text{Ideal}_{\text{Lk,SSE}}(A, S) \Rightarrow 1]$$

is sufficiently small. We use S to construct a simulator S_1 for A_1 as follows. Given $\text{Lk}_1(G, C, L, (c_1, \ldots, c_q)) = (\text{Lk}(\mathcal{I}, (c_1, \ldots, c_q)), \mathcal{H}, N_H)$ as defined above, S_1 runs S on input $\text{Lk}(\mathcal{I}, (c_1, \ldots, c_q))$. Let ℓ be the output of S. Then S_1 runs A_1 on ℓ and the rest of its input prepared appropriately using $\mathcal{H}$ and N_H, and returns its output. By the construction of A it follows that

$$\Pr[\text{Real}_{\text{GE}_1}(A_1) \Rightarrow 1] - \Pr[\text{Ideal}_{\text{Lk, GE}_1}(A_1, S_1) \Rightarrow 1]$$

is also sufficiently small. Hence GE_1 is Lk_1-secure.

4.5 SSE with Fragmentation

Finally, we describe a dictionary-based SSE scheme with data fragmentation based on the π_{pack} scheme from Cash et al. [9], which we optimize for bandwidth efficiency. We use this SSE scheme in our structured encryption scheme for colored subgraph queries above. Intuitively, the SSE scheme "chops up" the data for each keyword into equal-sized fragments of some length parameter, after padding to a multiple of the fragment length. It is described as "pack"ing multiple items into one fixed length chunk in the original π_{pack} scheme.

More formally, to define the SSE scheme, let $\text{PRF} : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^*$ be a function family. We also use a dictionary data structure, which can be formalized in standard ways. The

function KEYGEN_1
  return $K \leftarrow\!\!\$\ \text{KEYGEN}_S$
function $\text{ENC}_1(K, G, L)$
  Construct Dictionary:
    For each $(u, v) \in E$ do: $p_{uv}, q_{uv} \leftarrow\!\!\$\ \{0,1\}^{|u|}$, $n_{uv} \leftarrow\!\!\$\ \{0,1\}^k$
    For each $c \in C$ do:
      $l_{cV} \leftarrow \{v \mid L(v) = c, v \in V\}$
      $l_{cE} \leftarrow \{(u, v) \mid L(u) = L(v) = c, (u, v) \in E\}$
      $l_{cH_1} \leftarrow \{(u \oplus p_{uv}, q_{uv}, n_{uv}) \mid L(v) \neq L(u) = c, (u, v) \in E\}$,
      $l_{cH_2} \leftarrow \{(p_{xy}, y \oplus q_{xy}, n_{xy}) \mid L(x) \neq L(y) = c, (x, y) \in E\}$,
      $l_c \leftarrow (l_{cV} \| l_{cE}, \| l_{cH_1} \| l_{cH_2})$
    $\mathcal{I} \leftarrow (c, l_c)_{c \in C}$
  Encrypt Dictionary:
    return $\gamma \leftarrow \text{ENC}_S(K, \mathcal{I})$
function $\text{TOKENGEN}_1(K, c)$
  return $\tau \leftarrow \text{TRAPGEN}_S(K, c)$
function $\text{SEARCH}_1((\tau_{c_1}, \ldots, \tau_{c_q}), \gamma)$
  Search Encrypted Dictionary:
    For each c_i, $1 \leq i \leq q$, do: $l_{c_i} \leftarrow \text{SEARCH}_S(\gamma, \tau_{c_i})$
  Reconstruct Graph:
    For each c_i, $1 \leq i \leq q$:
      Parse l_{c_i} as $(l_{c_iV} \| l_{c_iE} \| l_{c_iH_1} \| l_{c_iH_2})$
      For each $v \in l_{c_iV}$, $V' \leftarrow V' \cup v$
      For each $(u, v) \in l_{c_iE}$, $E' \leftarrow E' \cup (u, v)$
    For each n in $(u' \oplus p, q, n)$ and $(p, v' \oplus q, n)$:
      Compute (u', v') and $E' \leftarrow E' \cup (u', v')$
    return $G' = (V', E')$

Figure 1: Structured Encryption Scheme for Colored Subgraphs GE_1

scheme SSE_F with fragmentation-length $f \in \mathbb{N}$ and label-length $k \in N$ is defined as

$$\text{SSE}_F = (\text{KEYGEN}_F, \text{ENC}_F, \text{TRAPGEN}_F, \text{SEARCH}_F)$$

in Figure 2. Correctness follows from taking the label-length to be sufficiently large to avoid collisions.

Security Analysis. For the security analysis, we adopt the *query pattern leakage* as defined by Chase and Kamara [10]. Namely, define $\mathsf{QP}(w_1, \ldots, w_q)$ to be the $q \times q$ binary matrix with a 1 at position i, j if $w_i = w_j$ and 0 otherwise. Then we define the leakage function

$$\text{LK}_F(\mathcal{I}, w_1, \ldots, w_q) = (\{\mathcal{I}(w_i)\}_{i \in [q]}, \sum_{i=1}^{n} \lceil |\mathcal{I}(w_i)|/f \rceil, \mathsf{QP}(w_1, \ldots, w_q)) .$$

The corresponding security theorem follows from the analysis of the π_{pack} scheme in Cash et al. [9].

Intuitively, it hides the length of each data item (while avoiding padding each data item to the maximum length) for which the adversary does not have the token for the corresponding keyword. Interestingly, the fact that these lengths are hidden does not seem to depend on f. However, as f increases, the adversary gets a better upper-bound on the number of keywords, since $\sum_{i=1}^{n} \lceil |\mathcal{I}(w_i)|/f \rceil$ is closer to n. In terms of bandwidth efficiency, as f decreases, more pseudorandom labels need to be stored in the dictionary.

Hence, there is a tradeoff: bandwidth depends on how close the length of each data item is to f, due to padding. If f is large and these lengths are not close to f, then poor bandwidth efficiency is achieved. The optimal choice of f for bandwidth efficiency is therefore data-dependent.

Let $\{c_1, \ldots, c_n\}$ be the color set, k be the length of pseudorandom labels and $\mathcal{I} = ((c_1, d_1), \ldots, (c_n, d_n))$ be the constructed dictionary. The optimal fragmentation size F can be derived by

$$\min_F \sum_{i=1}^{n} \left(\left\lceil \frac{|d_i|}{F} \right\rceil \times (F + k) \right) .$$

5. PPNP DESIGN

In this section, we introduce PPNP, a novel privacy-preserving network provenance system. PPNP uses the structured encryption scheme presented in Section 4 as a cryptographic primitive to enforce access control policies.

The access control policies are specified as the colors of the vertices and edges in the provenance graph, as determined by the provenance data owners. For simplicity, we assume that the color of an edge is determined by the colors of its two end vertices. (The proposed technique can be extended to support the case where edges are labeled in isolation with vertex colors.)

Figure 4 shows an overview of PPNP's design through a simplified example. Here, a node Q intends to query a provenance entry from node X, who already has a cached encrypted provenance graph from node Y, where the complete provenance graph is as is depicted in Figure 3.

- Upon receiving a provenance query for V_1 from Q, X sends its own encrypted provenance graph to Q, along with the cached encrypted provenance graph from Y.
- Having received both X's and Y's encrypted provenance graph, Q sends out *token requests* to X and Y, asking for tokens to access their respective authorized colors.
- X and Y send back to Q the tokens—that is, the corresponding keys for the colors that Q are authorized to access: green from X, and blue and green from Y.
- Once Q receives the tokens, it can then extract a sub-provenance graph for V_1 that Q is allowed to view.

To illustrate differential access control in PPNP, we show the view from another node P which intends to query the same graph as Q in Figure 5. P is authorized to access red and green from X, and blue from Y, while Q is authorized to access green from X, and blue and green from Y.

We also show components, i.e. nodes and (half)edges, of Q's view in Figure 6. Q decrypts ciphertext-blocks from X and Y separately. From X, Q successfully decrypts two vertices (V2 and V5), one full edge ((5,2)), and three halfedges ((2,1), (4,2), (3,2)). Similarly, from Y, Q gets two vertices (V3 and V6), and three halfedges ((3,2), (6,3), (6,3)). For halfedges, We use "cut shapes" to denote matching halfedges visually. The actual matching is done by the nonce associated with each halfedge. Note that for full edge, it is not "cut". Also, in order to recover the full edge, a node must possess both halfedges. Thus Q would match the two wavy halfedges to recover edge (6,3), and two sawtooth halfedges to recover edge (3,2). Halfedges (2,1) and (4,2) is not matched, and Q can only infer that these two halfedges have one green vertices.

5.1 Answering Provenance Queries

Upon receiving a provenance query, a node first prepares its provenance graph as a dictionary, and then replies with the encrypted graph as obtained using the ENC_1 function.

Assume at node X, the provenance graph is denoted by $G = (E, V)$, where V is set of vertices and E is set of edges in G.

function $\text{ENC}_F(K, \mathcal{I})$
  Fragment and Encrypt Index:
    Parse $\mathcal{I}$ as $((w_1, d_1), \ldots, (w_n, d_n))$
    Initialize dictionary D
    For $1 \leq i \leq n$ do:
      $K_1 \| K_2 \leftarrow \mathsf{PRF}_K(w, 1^{2k})$
      Pad d_i to length $\lceil \frac{|d_i|}{f} \rceil$
      Parse d_i as $d_{i,1} \| \ldots \| d_{i,\ell_i}$ where each $|d_{i,j}| = f$
      $ctr \leftarrow 1$
      For each $1 \leq j \leq \ell_i$ do:
        Add $(\mathsf{PRF}_{K_1}(ctr, 1^k), \mathsf{PRF}_{K_2}(ctr, 1^f) \oplus d_{i,j})$ into D
        $ctr \leftarrow ctr + 1$
    return D

function KEYGEN_F
  return $K \leftarrow \mathcal{K}$
function $\text{TRAPGEN}_F(K, w)$
  return $\mathsf{PRF}_K(w, 1^{2k})$
function $\text{SEARCH}_F(K_1 \| K_2, D)$
  For $ctr = 1$ until Retrieve returns $\perp$
    Retrieve $\mathsf{PRF}_{K_1}(ctr, 1^k)$ from D to get data c
    $d_{i,ctr} \leftarrow c \oplus \mathsf{PRF}_{K_2}(ctr, 1^f)$
    $ctr \leftarrow ctr + 1$
  Remove padding from $d_{i,ctr-1}$
  return $d_{i,1} \| \ldots \| d_{i,ctr-1}$

Figure 2: SSE scheme with fragmentation SSE_F

Table 1: Provenance dictionaries, before fragmentation

term(c)	vertices(l_{cV})	edges(l_{cE})	half-edges(l_{cH})
@X:			
red	V_1, V_4	$\varnothing$	$(p_{21}, V_1 \oplus q_{21}, n_{21}), (V_4 \oplus p_{42}, q_{42}, n_{42})$
green	V_2, V_5	V_5V_2	$(V_2 \oplus p_{21}, q_{21}, n_{21}), (p_{42}, V_2 \oplus q_{42}, n_{42}), (p_{32}, V_2 \oplus q_{32}, n_{32})$
@Y:			
blue	V_3	$\varnothing$	$(p_{63}, V_3 \oplus q_{63}, n_{63}), (V_3 \oplus p_{32}, q_{32}, n_{32})$
green	V_6	$\varnothing$	$(V_6 \oplus p_{63}, q_{63}, n_{63})$

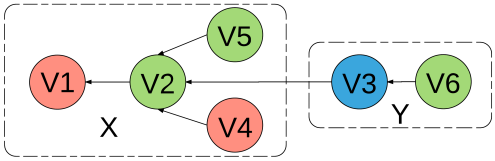


Figure 3: An example provenance graph.

Additionally, for each $v \in V$, we associate a color c with v. To construct the data structure for SSE, we create the dictionary as follows: (For simplicity, and in order to achieve a solution using only light-weight symmetric-key cryptographic primitives, we impose a constraint that a vertex is associated with exactly one color. Extending our scheme to allow vertices with multiple colors, or more generally Boolean formulae on the colors, is an interesting direction of future work.)

The ENC_1 function (shown in Figure 1) produces an encrypted dictionary. It begins by constructing a list l_{cV} that contains all vertices that are labeled with the color c, for each color c in the color set C. It then creates a a list l_{cE} that contains all edges whose ends are labeled as c. Then, a list l_{cH} is constructed such that all *half-edges* of c are covered, where the half-edges of a color c are defined as the edges $e = (u, v)$ that have either u or v labeled as c, but not both. Finally, ENC_1 concatenates the lists l_{cV}, l_{cE}, and l_{cH} into a single list, l_c. The function constructs l_c for each $c \in C$.

For a half-edge $e = (u, v)$ where u is labeled as c_u and v is labeled as c_v ($c_u \neq c_v$), we randomly generate p, q, and n such that $|u| = |v| = |p| = |q| = |n|$. In l_{c_uH}, we include a tuple $(u \oplus p, q, n)$ as an element in l_{c_uH}, and its corresponding tuple $(p, v \oplus q, n)$ as an element in l_{c_vH}.

After building the dictionary, each (c, l_c) pair is encrypted by a secret key K_c^X, where c is the color and X is the node where G is located. ENC_1 returns $\text{ENC}_S(K, \mathcal{I})$, the encrypted dictionary.

As an example, Table 1 provides the constructed dictionary for the provenance graph depicted in Figure 3. As one can see from the Table, the values of different keys in the dictionary may have different lengths. Unless otherwise masked (see below), this may leak

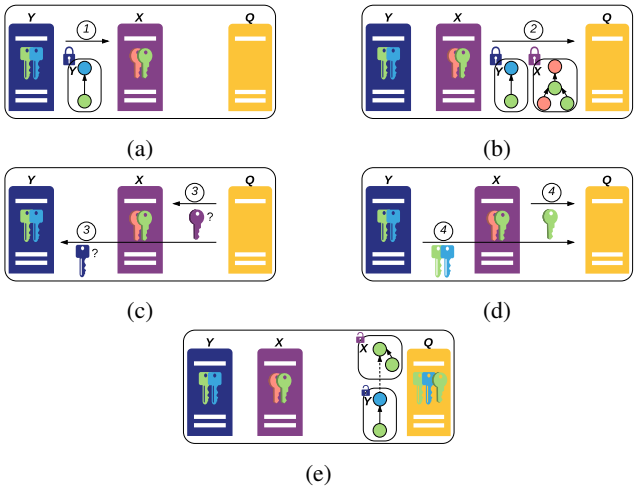
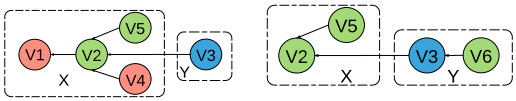


Figure 4: Answering Provenance Query. (a) At earlier time, X caches Y's encrypted provenance graph. (b) Upon receiving query from Q, X sends Q both its own encrypted and cached provenance graphs. (c) Q asks X and Y for authorized tokens. (d) X and Y send back tokens for colors that Q are authorized to access. (e) Q decrypts provenance (sub)graphs that it is allowed to view. Note that the dashed edge was recovered by cross-node handling mechanism.

additional information about the underlying provenance graph. For instance, when the vertices are labeled in an unbalanced fashion—e.g., a majority of vertices share one same label and a small minority of vertices share another label—an adversary would be able to distinguish the two labels from the lengths of document lists.

A typical approach to eliminating this information leakage is to perform padding to the dictionary such that all keys have same-lengthed values. However, such an approach introduces significant storage and communication overhead. As described in Section 4.5, we instead adopt an SSE scheme that incorporates data fragmentation to mitigate the information leakage without incurring these large costs. Table 2 shows the final fragmented dictionary, with fragment size $f = 5 \times |c|$ and $|pad| = |c|$.



P's view Q's view

Figure 5: Views from different parties of provenance graph in Figure 3.

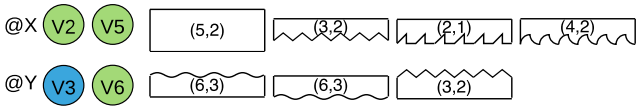


Figure 6: Components of Q's view in Figure 5.

5.2 Handling Cross-node Communications

Consider an edge $e = (u, v)$ that crosses two physical nodes: u is located on node Y and has label c_u; v is located on node X and has label c_v. It should be noted that a querier who has access to vertex v should not automatically be granted access to any information of u (i.e., the ID and label of u). Unless explicitly granted the access to u, the querier should only be aware that v is connected to some *unknown* vertex located on node Y.

Since each node encrypts its own provenance graph by its own secret key, it is non-trival for two nodes to share information of such a cross-node edge without compromising the aforementioned property. To achieve this, we resort to an approach similar to how we treated half-edges in Section 5.1.

When Y prepares its provenance graph, it randomly generates p, q and n such that $|u| = |v| = |p| = |q| = |n|$. Similar to half-edges, it includes tuple $(u \oplus p, q, n)$ in $l_{c_u H}$, and adds an auxiliary element, (p, q, n), to the final inverted-index. Note that the auxiliary elements of an inverted-index are not encrypted.

When X receives the encrypted inverted-index which includes the auxiliary element) from Y, it parses p, q, n and adds tuple $(p, v \oplus q, n)$ in $l_{c_v H}$ of its inverted-index. Table 1 provides an example of the generated inverted-indices for Figure 3, where cross-node communications are involved.

5.3 Reconstruct Provenance Graph

After the query node receives the final answer to its query and has been granted all tokens, it starts solving the provenance graph jigsaw puzzle by assembling the separated pieces together.

For each puzzle piece, it decrypts the dictionary with its corresponding tokens row-by-row. In more detail, from document list l_{cV} and l_{cE} for label c, the vertices and edges with label c can be recovered, as detailed in the SEARCH$_1$ function listed in Figure 1. For half edges in l_{cH}, a hash map is maintained, where n in tuple (p, q, n) is the search key. The purpose of this hash map is to match every half edge with its other half using the cuts of the jigsaw.

Recall that each half edge $e = (u, v)$ with $c_u \neq c_v$ appears twice in the dictionary: once in the document list for c_u in the form of $(u \oplus p, q, n)$, and once for c_v in the form of $(p, v \oplus q, n)$. With the two tuples matched by n, the original edge (u, v) can be recovered by $(u \oplus p, q) \oplus (p, v \oplus q)$.

The same matching method applies to cross-node edges as well. Each cross-node edge $e = (u, v)$ with $u \in Y$ and $v \in X$ also appears twice: once in the dictionary reported by Y in the form of $(u \oplus p, q, n)$, and once in the dictionary reported by X in the form of $(p, v \oplus q, n)$. From here, the provenance graph can be recovered.

5.4 Application of PPNP

While we focus on the application of PPNP to network provenance; it is not difficult to apply PPNP to other graph-based provenance models.

Table 2: Fragmentation of the inverted-indices (dictionaries) from Table 1. F is a PRF. Colors are abbreviated as r, g, b. To improve readability, parentheses and commas have been omitted.

term(c)	document-list(l_c)				
@X:					
$F(K_r^X, 1)$	V_1	V_4	p_{21}	$V_1 \oplus q_{21}$	n_{21}
$F(K_r^X, 2)$	$V_4 \oplus p_{42}$	q_{42}	n_{42}	pad	pad
$F(K_g^X, 1)$	V_2	V_5	V_5	V_2	$V_2 \oplus p_{21}$
$F(K_g^X, 2)$	q_{21}	n_{21}	p_{42}	$V_2 \oplus q_{42}$	n_{42}
$F(K_g^X, 3)$	p_{32}	$V_2 \oplus q_{32}$	n_{32}	pad	pad
@Y:					
$F(K_b^Y, 1)$	V_3	p_{63}	$V_3 \oplus q_{63}$	n_{63}	$V_3 \oplus p_{32}$
$F(K_b^Y, 2)$	q_{32}	n_{32}	pad	pad	pad
$F(K_g^Y, 1)$	V_6	$V_6 \oplus p_{63}$	q_{63}	n_{63}	pad

Applying PPNP to PROV. For concreteness, we briefly describe the application of PPNP to the PROV provenance model. PROV adopts a graph-based provenance model where a vertex in the graph can be one of three types: (a) *entities*, such as physical or digital items; (b) *activities*, such as actions or processes that derives entities; or (c) *agents*, such as persons, organizations or software that are partly responsible for activities. Edges in the graph identify direct *relationships* between the vertices. For example, an edge between an entity and an activity can describe that the entity was generated by the activity. The exact relationship that an edge identifies is encoded as an annotation of the edge (e.g., a "wasGeneratedBy" annotation for the previous example).

Assume that a set of provenance stores collectively maintain a distributed PROV provenance graph, and that the PROV provenance graph has been "colored" based on an imposed access control policy by the owners. The PROV provenance graph can be constructed as a colored graph with annotations on the edges and vertices. The SSE-based structured encryption scheme presented in Section 4 would still be applicable with a minor extension to further incorporate the annotations: we additionally include annotations to their corresponding vertices, edges and half-edges in the constructed dictionary. The retrieved partial view of the provenance will then naturally contain the annotations of the vertices and edges that the querier is permitted to access.

Customizing PPNP for deployment. We also note that there are several tunable knobs that can be customized based on the privacy requirement of individual applications.

For distributed system debugging [11, 54] and IoT tracking [50] that require strong confidentiality, it is necessary to apply GE$_1$ from Figure 1 with the fragmentation-aware SSE$_F$ scheme in Figure 2. For confidentiality, any querier who does not hold the authorized token for a color c would not be able to access the vertices and edges with that color, even if it holds the ciphertext the for entire graph. With fragmentation, the querier cannot deduce from the ciphertext sensitive information such as the names and the number of colors/lables, or the provenance graph size (due to padding).

In provenance-aware healthcare multi-agent systems (HC-MAS) [31], one can simply apply GE$_1$ with any underlying SSE scheme, as it is unnecessary to apply fragmentation for hiding the names and the number of colors/labels in the provenance graph. In this case, the colors correspond to medical procedures which are publicly available information.

6. EVALUATION

We constructed PrivProv in RAPIDNET, a declarative NDlog engine built upon the ns-3 network simulator. In order to compare PrivProv with previous network provenance systems (e.g., ExS-

Table 3: Experiment configurations.

Configuration	Dictionary	Fragment	Encrypt	Token
ExSPAN	No	No	No	No
ExSPAN+Dictionary	Yes	No	No	No
PrivProv w/o Token	Yes	Yes	Yes	No
PrivProv w/ Token	Yes	Yes	Yes	Yes

PAN [57]), we obtained the source code of the ExSPAN system from Zhou et al. [57] and modified it to support PrivProv's privacy-preserving features. Specifically, we added functionality for constructing inverted-indices from provenance graphs, fragmenting and encrypting the inverted-indices, decrypting ciphertexts, and reconstructing provenance graphs from inverted-indices. In addition, we modified the RapidNet compiler to include labels for provenance graphs, and implemented user-defined functions in RapidNet to interact with SSE. Cryptographic functions used in our PrivProv implementation were imported from the Crypto++ library [14].

By default, ExSPAN uses "string polynomials" to send digests of provenance graphs. For example, the provenance graph in Figure 3 would be transmitted as $V_1(V_2(V_4 + V_5 + V_3(V_5)))$. To ensure a fair comparison to PrivProv, we modified user-defined functions in ExSPAN to include "full" provenance graphs in which a full list of vertices and edges are transmitted to represent a graph.

The source code of our prototype implementation, published under an open-source license, is available for download [1].

6.1 Evaluation Settings

To understand the relative overheads of building the dictionary, encryption, and performing a token query in PrivProv, we evaluated the following four configurations:

- **ExSPAN** serves as our baseline, which returns results of provenance queries in plaintext. It does not construct or communicate the secure dictionaries used for supporting SSE.
- **ExSPAN+Dictionary** returns provenance queries in the format of a dictionary that maps colors to their corresponding vertices and edges in the provenance graph. For an edge with different colored ends, the edge will be included in the entries of both colors.
- **PrivProv w/o token** additionally applies cryptography to construct the privacy preserving query result.
- **PrivProv w/ token** further includes the token query that retrieves, from the original owner, the token required for reconstructing the provenance graph.

Table 3 presents a summary of the four configurations.

Applications. We evaluated PrivProv with two representative applications: mincost (introduced in Section 2) and the Chord [51] distributed hash table (DHT) which provides the capability of inserting and retrieving key value pairs in a distributed fashion. For Chord DHT, we focused on the provenance of the lookup process that, upon a user's request, identifies the node responsible for hosting a provided key. This process can be succinctly implemented in NDlog using the following rules [34]:

```
l1 lookupResults(@R,K,S,SI,E):- lookup(@NI,K,R,E),
       node(@NI,N), bestSucc(@NI,S,SI), K in (N,S].
l2 bestLookupDist(@NI,K,R,E,a_MIN<D>):- node(@NI,N),
       lookup(@NI,K,R,E), finger(@NI,I,B,BI),
       D := K-B-1, B in (N,K).
l3 forwardLookup(@NI,a_MIN<BI>,K,R,E):- node(@NI,N),
       bestLookupDist(@NI,K,R,E,D),
       finger(@NI,I,B,BI), D == K-B-1, B in (N,K).
l4 lookup(@BI,K,R,E):- forwardLookup(@NI,BI,K,R,E),
       f_typeOf(BI) != null.
```

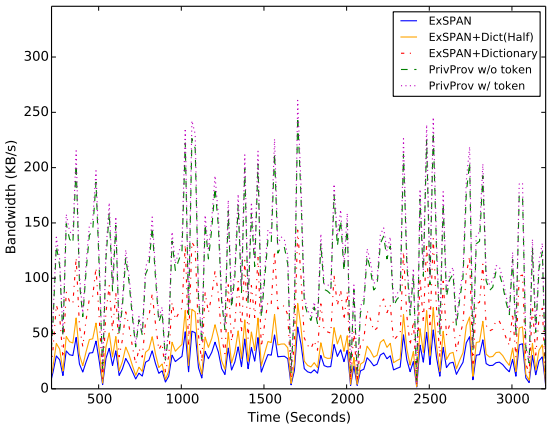


Figure 7: Bandwidth utilization of provenance queries.

Experimental setup. We performed the experiments on a Dell OptiPlex 7040 desktop running Ubuntu 14.04 LTS kernel version 3.19.0 with an Intel Core i7-6700 processor, 32GB 2133MHz DDR4 RAM, and a M.2 256GB SATA SSD.

Our experiments used ns-3's simulation functionality, which allows evaluations of PrivProv in a sizable network topology. Network topologies in all experiments are generated by the GT-ITM topology generator [24] using Transit-Stub type networks. On average, each transit node is connected to eight stub domains; each stub domain contains on average three nodes, while each transit domain contains four nodes. Thus, on average, each transit domain contains 50 nodes. We increase the number of transit domains to vary the size of network topologies. The links between transits are configured with 1Gbps of bandwidth and 20ms latency, while links within a transit have 100Mbps bandwidth and 50ms latency.

Unless explicitly specified, our experiments ran on a 100-node topology generated with the settings described above.

Workflow. After applications reach a fixed point—that is, all pairwise shortest paths are computed (for mincost), and the Chord ring and finger tables are stabilized (for Chord). 1000 provenance queries are issued against randomly chosen min-cost paths (for mincost) and lookup results (for Chord).

PrivProv supports a general graph coloring scheme. In this paper, we adopted a simple scheme in which provenance vertices are colored based on the IP address of the vertices' host machine. This corresponds to a straightforward role-based access policy. For simplicity, we use the SHA-1 digest of IP addresses as vertex colors.

We set the fragment size to 300, except for the evaluation of fragmentation optimization, which uses a varying fragment size.

6.2 Evaluation Results

PrivProv does not introduce any modification to provenance maintenance. (We confirm this below.) We thus focus on evaluating the performance of provenance querying.

6.2.1 Communication Overhead

Figure 7 presents the aggregate bandwidth utilization over time during the processing of 1000 random queries.

We observed that PrivProv incurs roughly a fourfold communication overhead compared with ExSPAN, where 50% of the overhead comes from the construction of the inverted-index. This is expected since almost every edge is a cross-color edge (recall that vertices are colored based on the hosting machine), each edge would

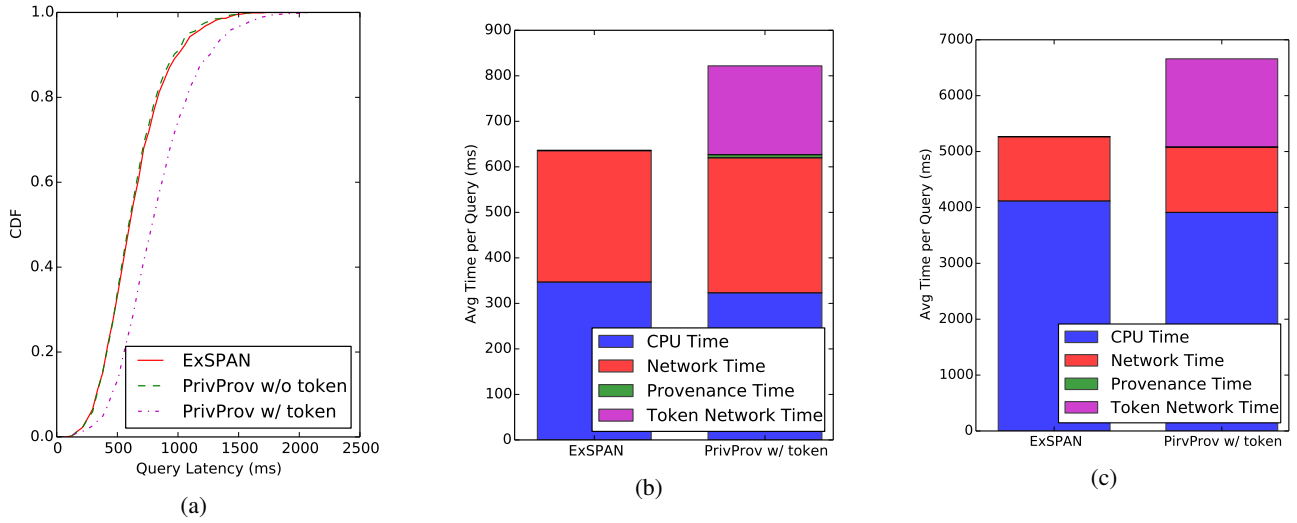


Figure 8: The cumulative distribution of provenance query latencies (figure *a*), a breakdown of provenance query latencies (figure *b*), and the query latency composition of 100 node topology running 1000 uniformly random queries in Chord (figure *c*).

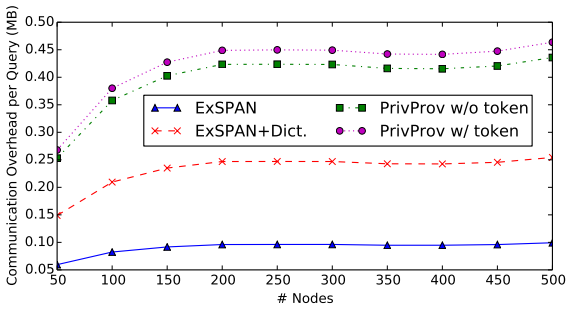


Figure 9: Communication overhead with varying network sizes.

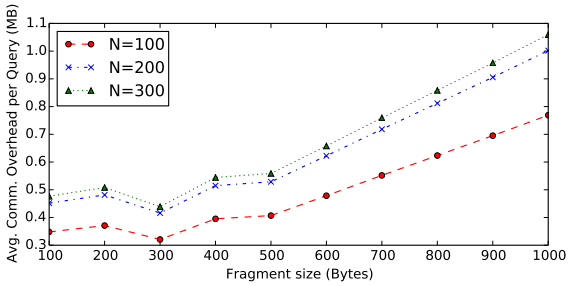


Figure 10: Communication overhead with varying fragmentation sizes.

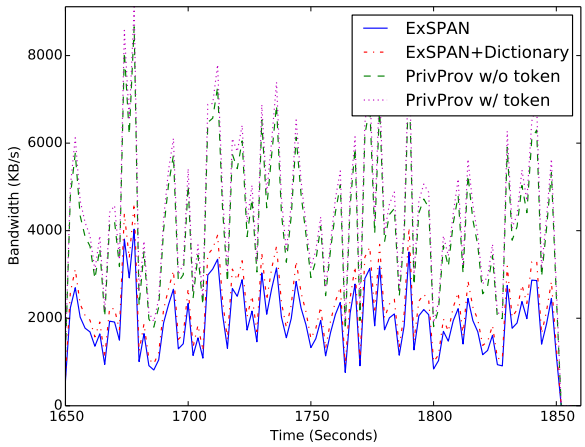


Figure 11: Bandwidth utilization of 100 node topology running 1000 uniformly random queries in Chord.

appear exactly twice in the inverted-index. For applications that require less confidentiality of cross-color edges—for example, if it is acceptable to reveal cross-color edges even when a user has tokens for only one end of the edges—the communication overhead would be significantly reduced, as shown by ExSPAN+Dict(Half) in the figure, where a cross-color edge is revealed under edge-tail's token. The average bandwidth for ExSPAN and ExSPAN+Dict(Half) are 28.91KB/s and 37.85KB/s respectively, compared to 65.27KB/s for ExSPAN+Dict. Other factors that contribute to the overhead include fragmentation, padding that ensures the inverted list of all colors have the same length, and auxiliary information for handling cross-machine edges. The overhead of token queries is negligible.

6.2.2 *Query Latency*

Figure 8 (a) presents the Cumulative Distribution Function (CDF) of the query latencies of 1000 random provenance queries. The figure shows that PrivProv w/o token does not introduce much latency. In fact, it almost overlaps with ExSPAN. Token queries, however, require an additional round trip time, which slows the provenance query by 20%. We remark that tokens can be cached and hence the token query overhead can be amortized over multiple provenance queries.

Figure 8 (b) presents the breakdown of the query latencies. "CPU time" counts the time spent in retrieving plaintext provenance from maintained provenance tables; "provenance time" counts the time spent in constructing and assembling query results based on the retrieved plaintext provenance; "network time" counts simulation time spent in transmitting provenance query results; and "token network time" counts simulation time spent in network delay and transmission delay for token queries.

Since the experiments were run in simulation mode, we calculated query latencies in two parts: we tracked CPU time and provenance time by recording the actual wall clock time during the execution; for network time, we considered ns-3's simulated network transmission time.

We observed that PrivProv does not incur much computation overhead (as indicated by the low provenance time) for preparing and parsing the secure dictionary of provenance graphs. PrivProv

introduces a small increase in network time due to increased size of messages. The significant delay comes from processing the token queries, which introduces one additional round trip time. We note that these findings align with the latency results of Figure 8(a).

One counterintuitive point is that CPU time of PrivProv is slightly lower than that of ExSPAN. Upon further investigation, we found that the additional CPU time in ExSPAN is caused by the implementation of RAPIDNET where objects are deep-copied. In contrast, PrivProv uses a flat encoding for the encrypted secure dictionary and thus has a less "structured" encoding of provenance objects as compared to ExSPAN.

6.2.3 Scalability

Figure 9 presents the communication costs for 1000 random provenance queries for various sized network topologies. We make two observations: (i) The communication overhead of provenance queries increases sublinearly with network size. This is expected since for mincost the provenance complexity is directly related to the network diameter, the latter of which also increases sublinearly with network size. (ii) Second, the relative overhead of constructing the dictionary (ExSPAN+Dictionary), creating the secure dictionary (PrivProv w/o token), and processing token queries (PrivProv w/ token) remains unchanged, which reconfirms our observation from Section 6.2.1.

Figure 9 shows that the overhead plateaus after 200 nodes. This is due to the fact that the network diameter remains the same. In our experiment setting, we increased network size by adding more transit domains in the network, which did not increase network diameter after four transit domains. Thus, the depth of the resulted provenance graphs did not grow further. The scalability relies on the *underlying* protocol. PrivProv should not affect the scalability of provenance maintenance and querying since it adds a constant factor of overhead, in both computation and communication.

6.2.4 The Effect of Fragmentation

Figure 10 presents the effect of various fragmentation size on communication overhead. We performed the experiments on three network topologies with 100, 200, and 300 nodes respectively. As we can observe clearly from the figure that the "V" shaped lines indicate an optimal fragmentation size which minimizes network communication overhead. The intuition is that with small fragmentation size, the overhead of maintaining meta-data such as random labels is significant, analogous to sending small payload TCP segments, while with large fragmentation size, padding becomes a significant contributing factor to the communication overhead. As discussed in Section 4, the optimal fragmentation size is determined by the length distribution of d_i. The evaluation assumes a coloring scheme where provenance vertices are colored based on IP address of the vertices' host machine (Section 6.1). As a result, each color contains provenance vertices and edges related to `cost` and `mincost` tuples on one physical node. Although the network size increases, the distribution of d_i remains roughly unchanged, and so is the optimal fragmentation size.

6.2.5 Evaluation of Chord DHT

Figure 11 presents bandwidth consumption over time for processing 1000 random provenance queries when the underlying protocol is the Chord DHT. We observed that PrivProv incurs 2-3x overhead in bandwidth consumption. For Chord, the majority of the edges (around 80%) in a provenance graph have the same color for both ends; the relative cost for constructing the dictionary is thus significantly lower than that of mincost. The relative cost for constructing the secure dictionary, on the other hand, remains similar to mincost, as expected.

Figure 8 (c) presents the breakdown of query latency for Chord. The figure shows that CPU time dominates query latency, due to the increase in program complexity of Chord, compared with mincost. Again, the increases in provenance time and network time are nearly negligible.

Token network time delay adds one more round trip time. The increase in token network time delay compared with mincost is due to the increased number of physical nodes and hops involved in one provenance query. For mincost, the number of physical nodes and hops involved in one query is capped by the network diameter. For Chord, however, since it is an overlay application, each hop corresponds to multiple physical hops in the underlying network.

7. RELATED WORK

Provenance. Provenance is a concept initiated from the database community [7], but it has recently been applied in several other areas, including distributed systems [54, 57, 58], storage systems [42], operating systems [20], and mobile platforms [18]. Our work is mainly related to projects that use network provenance for diagnostics. In this area, ExSPAN [57] was the first system to maintain network provenance at scale; SNP [58] added integrity guarantees in adversarial settings, DTaP [59] a temporal dimension, and Y! [54] support for negative events. Bao et al. developed efficient schemes for labeling workflow provenance [2, 3]. Glavic et al. presented query rewriting techniques for processing provenance in [21].

Position papers, such as those by Hasan et al. [25] and Bertino et al. [6], highlight the need and challenges for privacy in provenance systems. Privacy concerns of scientific workflow provenance are raised by Davidson et al. [17]. PPNP presents a solution to these challenges by performing queries on encrypted provenance graphs in a privacy-preserving manner.

The Secure-View problem, introduced by Davidson et al. [16], aims to hide module information without losing too much utility of provenance. Maruseac et al. [39] propose an obfuscation scheme to provide differential privacy of provenance paths; the obfuscation scheme allows for statistically similar results (with a bounded error), unlike the exact provenance query results provided by PPNP. Bates et al. [4] present a complete architecture for provenance management in the cloud. The work is complementary to PPNP, where it can be used as a basis for setting up attribute-based (or role-based) access control. Finally, Khan et al. [30] propose secure Location Provenance which enforces the security guarantees (including privacy) by employing a collection of cryptographic constructs. The proposed solution focuses on Location Provenance which assumes a *chain-based* provenance model, whereas PPNP is applicable to the more general graph-based provenance model.

Searchable Symmetric and Structured Encryption. We do not attempt to provide a comprehensive overview of work on searchable symmetric encryption or more generally structured encryption, and instead refer the reader to an excellent talk of Kamara [29] for an overview of these primitives and other approaches for searching on encrypted data.

8. CONCLUSION

This paper presents PPNP, a novel distributed privacy-preserving network provenance scheme that supports the richness of network provenance while providing strong privacy guarantees over confidential data. We formally prove that our proposed cryptographic-based PPNP scheme is secure, and show how PPNP can be applied to existing provenance systems that require heterogeneous privacy

preferences. We develop a prototype implementation, PrivProv, and evaluate its performance on two distinctly different provenance applications. Our evaluation results demonstrate that PrivProv incurs negligible increase in latency and a reasonable bandwidth overhead, making it practical for large, distributed deployments.

Acknowledgments

We thank Mohammad Zaheri for several insightful conversations about this work, and the anonymous reviewers for their insightful comments. This paper is partially funded from CNS-1149832, CNS-1527401, CNS-1453392, CNS-1513734 and CNS-1650419. The findings and opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] PPNP Code Release. http://security.cs.georgetown.edu/~yuankai/ppnp-code.tar.gz.

[2] Z. Bao, S. B. Davidson, S. Khanna, and S. Roy. An optimal labeling scheme for workflow provenance using skeleton labels. In *Proceedings of SIGMOD*, 2010.

[3] Z. Bao, S. B. Davidson, and T. Milo. Labeling recursive workflow executions on-the-fly. In *Proceedings of SIGMOD*, 2011.

[4] A. Bates, B. Mood, M. Valafar, and K. Butler. Towards secure provenance-based access control in cloud environments. In *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013.

[5] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer. Trustworthy whole-system provenance for the linux kernel. In *USENIX Security*, 2015.

[6] E. Bertino, G. Ghinita, M. Kantarcioglu, D. Nguyen, J. Park, R. Sandhu, S. Sultana, B. Thuraisingham, and S. Xu. A roadmap for privacy-enhanced secure data provenance. *Journal of Intelligent Information Systems*, 2014.

[7] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *Proceedings of ICDT*, 2001.

[8] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. VisTrails: Visualization meets data management. In *Proceedings of SIGMOD*, 2006.

[9] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. *IACR Cryptology ePrint Archive*, 2014.

[10] M. Chase and S. Kamara. Structured Encryption and Controlled Disclosure. In *ASIACRYPT*, 2010.

[11] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo. The Good, the Bad, and the Differences: Better Network Diagnostics with Differential Provenance. In *Proceedings of SIGCOMM*, 2016.

[12] D. C. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica. The Design and Implementation of a Declarative Sensor Network System. In *Proceedings of SenSys*, 2007.

[13] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson. Addressing the provenance challenge using zoom. *Concurrency and Computation : Practice and Experience*, 2008.

[14] Crypto++. https://www.cryptopp.com.

[15] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of CCS*, 2006.

[16] S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *Proceedings of PODS*, 2011.

[17] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen. On provenance and privacy. In *Proceedings of ICDT*, 2011.

[18] M. Dietz, S. Shekhar, Y. Pisetsky, A. Shu, and D. S. Wallach. Quire: Lightweight provenance for smart phone operating systems. In *USENIX Security*, 2011.

[19] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of SSDBM*, 2002.

[20] A. Gehani and D. Tariq. Spade: support for provenance auditing in distributed environments. In *Proceedings of International Middleware Conference*, 2012.

[21] B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *Proceedings of ICDE*, 2009.

[22] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating collaborative data sharing. In *Proceedings of SIGMOD*, 2007.

[23] P. Groth and L. Moreau. Prov-overview. an overview of the prov family of documents. 2013.

[24] GT-ITM. http://www.cc.gatech.edu/projects/gtitm/.

[25] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*. ACM, 2007.

[26] R. Hasan, R. Sion, and M. Winslett. Preventing history forgery with secure provenance. *ACM Transactions on Storage (TOS)*, 2009.

[27] R. Ikeda, H. Park, and J. Widom. Provenance for generalized map and reduce workflows. In *Proceedings of CIDR*, 2011.

[28] M. Interlandi, K. Shah, S. D. Tetali, M. A. Gulzar, S. Yoo, M. Kim, T. Millstein, and T. Condie. Titian: Data provenance support in spark. *Proceedings of the VLDB Endowment*, 2015.

[29] S. Kamara. How to Search on Encrypted Data. https://cs.brown.edu/~seny/slides/encryptedsearch-full.pdf.

[30] R. Khan, S. Zawoad, M. M. Haque, and R. Hasan. Otit: Towards secure provenance modeling for location proofs. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014.

[31] T. Kifor, L. Z. Varga, J. Vazquez-Salceda, S. Alvarez, S. Willmott, S. Miles, and L. Moreau. Provenance in agent-mediated healthcare systems. *IEEE Intelligent Systems*, 2006.

[32] L. Kot. Tracking personal data use: Provenance and trust. In *CIDR*, 2015.

[33] C. Liu, R. Correa, X. Li, P. Basu, B. T. Loo, and Y. Mao. Declarative policy-based adaptive mobile ad hoc networking. *IEEE/ACM Transactions on Networking (TON)*, 2011.

[34] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing Declarative Overlays. In *Proceedings of SOSP*, 2005.

[35] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *Proceedings of SIGCOMMM*, 2005.

[36] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking: Language, Execution and Optimization. In *Proceedings of SIGMOD*, 2006.

[37] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative Networking. *Communication of ACM*, 2009.

[38] Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith. MOSAIC: Unified Platform for Dynamic Overlay Selection and Composition. In *Proceedings of CoNEXT*, 2008.

[39] M. Maruseac, G. Ghinita, and R. Rughinis. Privacy-preserving publication of provenance workflows. In *Proceedings of the 4th ACM conference on Data and application security and privacy*. ACM, 2014.

[40] P. McDaniel, K. Butler, S. McLaughlin, R. Sion, E. Zadok, and M. Winslett. Towards a Secure and Efficient System for End-to-End Provenance. In *2nd Workshop on the Theory and Practice of Provenance (TAPP)*, February 2010.

[41] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. The open provenance model. 2007.

[42] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proceedings of USENIX ATC*, 2006.

[43] K.-K. Muniswamy-Reddy, P. Macko, and M. Seltzer. Provenance for the cloud. In *Proceedings of FAST*, 2010.

[44] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE S&P*, 2008.

[45] Netsil. http://netsil.com/.

[46] T. Oinn, M. Addis, J. Ferris, D. Marvin, T. Carver, M. R. Pocock, and A. Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 2004.

[47] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of ICDE*, 2007.

[48] A. Sahai and B. Waters. Fuzzy Identity-based Encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005.

[49] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Proceedings of NDSS*, 2010.

[50] J. A. Stankovic. Research directions for the internet of things. *IEEE Internet of Things Journal*, 2014.

[51] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMMM*, 2001.

[52] W. System, I. Altintas, O. Barney, and E. Jaeger-frank. Provenance collection support in the kepler scientific workflow system. In *Proceedings of IPAW*, 2006.

[53] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of CIDR*, 2005.

[54] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo. Diagnosing missing events in distributed systems with negative provenance. In *ACM SIGCOMM Computer Communication Review*, 2014.

[55] Y. Zhang, A. O'Neill, M. Sherr, and W. Zhou. Privacy-preserving network provenance. Technical report. Available at https://security.cs.georgetown.edu/~yuankai/ppnp-tr.pdf.

[56] W. Zhou, E. Cronin, and B. T. Loo. Provenance-Aware Secure Networks. In *International Conference on Data Engineering Workshop (ICDEW)*, 2008.

[57] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao. Efficient querying and maintenance of network provenance at Internet-scale. In *Proceedings of SIGMOD*, 2010.

[58] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. Secure network provenance. In *Proceedings of SOSP*, 2011.

[59] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr. Distributed time-aware provenance. *PVLDB*, 2013.