A Framework for Utility-Driven Network Trace Anonymization

Abhinav Parate, Gerome Miklau
University of Massachusetts, Amherst
Department of Computer Science
140 Governors Drive, Amherst, MA
aparate@cs.umass.edu, miklau@cs.umass.edu

Technical Report No. 2008-038

Abstract

The publication of network traces is critical for network research but their release is highly constrained by privacy and security concerns. The importance of a framework for anonymizing traces to provide different levels of security and utility to promote trace publication has been identified in the literature. However, the current state-of-art anonymization techniques have failed to provide the guarantees on privacy and security. In this paper, we propose a framework in which a trace owner can match an anonymizing transformation with the requirements of analysts. The trace owner can release multiple transformed traces, each customized to an analysts needs, or a single transformation satisfying all requirements. The framework enables formal reasoning about anonymization policies, for example to verify that a given trace has utility for the analyst, or to obtain the most secure anonymization for the desired level of utility. We validate our techniques by applying them to a real enterprise network trace and measuring the success of attacks by an informed adversary. The proposed framework is extensible and it allows for the addition of anonymization techniques as they evolve.

1 Introduction

Sharing network traces across institutions is critical for advancing network research and protecting against cyber-infrastructure attacks. But the public release of network traces remains highly constrained by privacy and security concerns[5, 19]. Traces contain highly sensitive information about users in the network, proprietary applications running in enterprises, as well as network topology and other network-sensitive information that could be used to aid a cyber-attack.

The most common approach to enabling secure trace analysis is through *anonymizing transformations*. The original trace is transformed by removing sensitive content and obscuring sensitive fields and the result is released to the public. The appeal of trace anonymization is that the trace owner can generate a single anonymized trace, publish it, and analysts can perform computations on the trace independently of the trace owner. A number of anonymizing transformations have been proposed for network traces, with IP packet traces receiving the most attention. Proposed anonymization techniques include *tcpurify* [3], the *tcpdpriv* [2] and Crypto-PAn [8] tools (which can preserve prefix relationships among anonymized IP addresses), as well as frameworks for defining transformations, such as *tcpmkpub* [15].

Unfortunately, reliable standards for the privacy and utility of these transformations has been elusive. A number of recent attacks on trace transformation techniques have been demonstrated by researchers [18, 7, 12, 4, 6] and the research community is still actively pursuing metrics for trace privacy that can guide trace owners [18, 7]. In addition, the utility of anonymized traces to analysts has received less attention than privacy, and the benefits of improved anonymizing transformations are sometimes at the cost of usefulness to analysts of the published trace.

In this paper we address the problem faced by a *trace owner* who wishes to allow a group of independent *analysts* to study an IP-level network trace. Our trace publication framework allows the trace owner to anonymize a trace for the needs of a particular analysis, or a related set of analyses. The published traces can be more secure because they provide only the needed information, omitting everything else. In addition, we provide procedures for formally verifying that a published trace meets utility and privacy objectives.

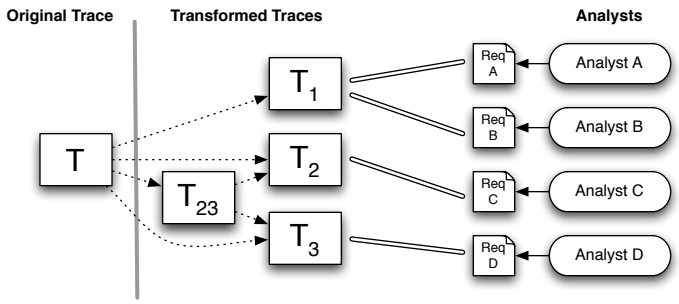


Figure 1: The proposed trace protection framework: the original trace T may be transformed in multiple ways (T_1, T_2, T_3, T_{23}) to support the requirements of different analysts.

Our publication framework is illustrated informally in Figure 1. The figure shows an original trace T transformed in four different ways, for use by different analysts. Trace T_1 contains sufficient information for both analysts A and B. Trace T_2 is devised for use exclusively by the analyst C, and trace T_3 is customized for the needs of analyst D. An alternative to publishing both trace T_2 and T_3 is to derive the single trace T_{23} which can support analysts C and D simultaneously.

The goal of this paper is *not* to propose a novel anonymizing transformation for network traces. Instead, our goal is to create a framework in which basic trace transformation operations can be applied with a precise, formal understanding of their impact on trace utility and privacy. Our framework consists of the following:

- **Formal transformations** A set of simple, formally-defined *transformation operators* that are applied to the trace to remove or obscure information. These include encryption, field removal, domain translation, etc. Transformation operators can be combined to form composite transformations. The output of the chosen composite transformation is published (along with the description of the transformation).
- **Input from the analyst** We assume the requesting trace analyst provides a description of the information needed for analysis. We propose a simple language for *utility constraints* which express the need for certain relationships to hold between the original trace and the published trace. The constraints can require, for example, that certain fields are present and unmodified, or that other values can be obscured as long as they preserve the ordering relationship present in the original trace. It is usually straightforward to determine the constraints that are needed to support a particular analysis; we provide examples of recent network studies of IP traces and their utility requirements in Section 2 and Appendix A.1.
- **Formally evaluating privacy and utility** Because both the transformations and the utility requirements of analysts are specified formally, it is possible for the trace owner to:
  - decide whether a composite trace transformation satisfies an analyst's requirements, implying that the desired analysis can be carried out on the transformed trace with perfect utility.
  - compute the most secure transform satisfying a given set of analyst requirements.
  - compare the security of transforms or analyze the impact of a collusion attack that might allow published traces to be combined.

Publishing multiple traces Because our trace transformations are often customized to the needs of individual analyses, it is worth comparing our framework with an alternative to trace anonymization recently proposed by Mogul et al [14]. In that work, the authors propose an execution framework in which analysts submit code to the trace owner. The trace owner executes the code locally on the trace and publishes only the program output to the trace analyst. The principle challenge is verifying whether the program output can be safely released without violating privacy.

In our framework, the analyst submits a set of utility constraints – not a general-purpose program. Therefore, the trace owner does not bear the significant burden of evaluating the safety of a program or the danger of covert channels. In addition, we still publish a transformed trace which both relieves the trace owner of the need to provide computational resources, and allows the analyst to refine their computations on the trace. We believe that our utility

constraints are therefore the right methodology for supporting a trace publisher in customizing trace transformation to the needs of analysts.

The advantages of our framework do entail some challenges for the trace owner. Compared with conventional trace anonymization, the trace owner in our framework must make more fine-grained choices about which transformed traces to publish to which users, and must compute and publish multiple anonymized traces instead of just one. We believe that the benefits to trace security warrant this effort. Our transformation operators are efficient to apply, and we provide a number of tools to help the trace owner make publication decisions. In addition, the trace owner can choose to publish a single trace supporting multiple analysts. For example, in Figure 1, publishing trace T_{23} may be easier than publishing both T_2 and T_3, but may require a sacrifice in privacy as, intuitively, Analyst C will receive some additional information used in the trace analysis D.

An additional concern with publishing multiple traces are attacks in which a single party poses as two or more analysts, or two or more analysts collude. In this case different published views of the trace could be combined to reveal more information than intended. In the absence of a trustworthy authority validating the identities, it is a challenge to counter such attacks. We can however analyze the risk of collusion formally: in Section 4 we show that it is possible to bound how much a group of colluding parties can learn, distinguishing between cases where collusion is a serious risk and cases where little can be gained from collusion.

The remainder of the paper is organized as follows. In Section 2, we describe a case study in which we apply the main components of our framework to a trace analysis of TCP connection characteristics. Section 3 describes our trace transformation operators and our language for specifying analyst requirements. Section 4 details the formal steps in utility analysis and the computation of most secure transformation for an analysis. Section 5 measures the privacy of sample transformations quantitatively through experiments on a real network trace. We discuss related work in Section 6 and conclude in Section 7.

2 Example: Inferring TCP Connection Characteristics

In this section, we provide an overview of our framework by describing its use in enabling a real study of TCP connection characteristics carried out by Jaiswal et al. [10]. First, we explain the analysis reseachers wish to perform and derive the basic requirements that must be satisfied by any usable trace. Then we describe an anonymizing transformation, and finally, we verify the transformation for the requirements satisfaction and give a brief statement on privacy of the network trace.

Analysis Description

A TCP connection is identified by two IP addresses $(ip1, ip2)$ and two ports $(pt1, pt2)$, corresponding to the sender and receiver. Jaiswal et al. study the characteristics of TCP connections through passive monitoring [10]. Their study focuses on measuring the sender's congestion window (*cwnd*) and the connection round-trip time (*RTT*). In this analysis, the congestion window of a connection is estimated using a finite state machine (FSM). The transitions in this FSM are triggered by receiver-to-sender ACKs or by out-of-sequence packet retransmissions. The FSM processes the packets in connection in the order they were observed at observation point, O. The estimation of variable, RTT, is done indirectly by adding the trip time of packet from point O to the receiver and then, back to O with the trip time between point O, the sender and O again. The full details of this estimation can be seen in [10].

Utility Requirements

Based on this description, we present the requirements or the sufficient conditions that must be satisfied by any transformed trace supporting the analysis described.

1. **R1** The trace must include the $type$ of the packet (SYN or ACK).
2. **R2** The trace must allow the analyst to identify if any given two records belong to the same connection or not.
3. **R3** The trace must allow the analyst to order packets in the same connection by sequence numbers(seq_no) or timestamps(ts).
4. **R4** The trace should preserve the difference between ts values for packets in same connection.
5. **R5** The trace should preserve the difference between *seq_no* for packets in same connection.
6. **R6** The sequence numbers of the sender's packets and the acknowledgement numbers(*ack_no*) of receiver's packets of same connection should be comparable for equality in the trace.

Table 1: *A formal description of utility requirements sufficient to support the example analysis of TCP connection properties.*

Formal Utility Requirements
$Any(t) \Rightarrow t.syn = \phi(t).syn$
$Any(t) \Rightarrow t.ack = \phi(t).ack$
$Any(t1, t2) \Rightarrow$ ((t1.ip1== t2.ip1) && (t1.ip2 ==t2.ip2) && (t1.pt1== t2.pt1)&& (t1.pt2 ==t2.pt2)) = ($\phi(t1)$.ip1== $\phi(t2)$.ip1 && $\phi(t1)$.ip2 ==$\phi(t2)$.ip2 && $\phi(t1)$.pt1== $\phi(t2)$.pt1 && $\phi(t1)$.pt2 ==$\phi(t2)$.pt2)
Same-Conn(t1,t2) $\Rightarrow (t1.seq\_no \leq t2.seq\_no) = (\phi(t1).seq\_no \leq \phi(t2).seq\_no)$
Same-Conn(t1,t2) $\Rightarrow (t1.ts \leq t2.ts) = (\phi(t1).ts \leq \phi(t2).ts)$
Same-Conn(t1,t2) $\Rightarrow (t1.seq\_no - t2.seq\_no) = (\phi(t1).seq\_no - \phi(t2).seq\_no)$
Same-Conn(t1,t2) $\Rightarrow (t1.ts - t2.ts) = (\phi(t1).ts - \phi(t2).ts)$
Opp-Pckts(t1,t2) $\Rightarrow (t1.seq\_no == t2.ack\_no) = (\phi(t1).seq\_no = \phi(t2).ack\_no)$
$Any(t) \Rightarrow t.window = \phi(t).window$
$Any(t) \Rightarrow t.dir = \phi(t).dir$
Qualifiers
$Any(t)$ { }
$Any(t1, t2)$ { }
Same-Conn(t1,t2)$\{(t1.ip1 == t2.ip1)$, $(t1.ip2 == t2.ip2)$, $(t1.pt1 == t2.pt1)$, $(t1.pt2 == t2.pt2)\}$
Opp-Pckts(t1,t2) $\{(t1.ip1 == t2.ip1), (t1.ip2 == t2.ip2), (t1.pt1 == t2.pt1), (t1.pt2 == t2.pt2), (t1.dir! = t2.dir)\}$

7. **R7** The actual value of TCP field, $window$, should be present in the transformed trace.

The above requirements are specified formally as a set of constraints given in Table 1. The formal requirements are constraints stating that certain relationships must hold between the original trace and the anonymized trace. The full description of our specification language is given in Section 3.2.

Trace Anonymization

Next we describe a simple transformation that provably satisfies the above utility requirements. For this transformation, we concatenate the fields $(ip1, ip2)$, encrypt the concatenated string to obtain anonymized $(ip1, ip2)$ fields. Similarly, we obtain anonymized $(pt1, pt2)$ by concatenating $(pt1, pt2, ip1, ip2)$ and encrypting it. This will map same $(pt1, pt2)$ values to different values if they are from different connections. The fields *seq_no, ack_no* are anonymized by linear translation such that minimum value in these fields becomes 0. For example, the values $(150, 165, 170)$ will be linearly translated to $(0, 15, 20)$. The field *ts* is anonymized similarly. We do not anonymize any other field. Finally, we remove any field which is not required in the analysis.

In Section 3.1, we provide a basic set of formal transformation operators. The anonymization scheme mentioned above can be expressed as a composite function of the formal transformations on individual fields. This composite transformation function ϕ is given by:

$$\phi = \Pi_X \circ E_{\{ip1,ip2\},\kappa_1} \circ E_{\{pt1,pt2\}(ip1,ip2),\kappa_2} \circ T_{\{ts\}(C)} \circ T_{\{seq\_no,ack\_no\}(C)} \circ I_{\{dir,window,syn,ack\}}$$

Here $C = \{ip1, ip2, pt1, pt2\}$, E is encryption operator, T is translation operator, Π is projection operator, I is identity operator, X is the set of required attributes and κ_1, κ_2 are keys for encryption function.

In Table 2, a simplified example of a network trace is given. The records in this table are then transformed using above transformation function ϕ, to obtain the anonymized view given in Table 3. The encrypted values have been replaced by variables for clarity.

Provable Utility

The utility analysis verifies that the anonymization scheme, defined by the composite transformation function, satisfies the constraints.

Informally, as we do not anonymize *syn* and ack bits in the trace, the type information of the packet is available, satisfying *R1*. The encryption of connection fields still supports grouping together of records in same connection (*R2*). The linear translation of *ts* and *seq_no* preserves the relative order (*R3*) and the relative differences in these fields (*R4*,*R5*). By using the same transformation for *seq_no* and *ack_no*, we make sure that *R6* is satisfied, allowing the equality tests on these fields. *R7* is satisfied as field,$window$ is not anonymized.

The formal verification process requires formal description of requirements and the anonymization scheme and it has been described in detail in section 4.1.

Privacy Analysis

Table 2: *Original Connection Tuples*

ts	ver	$ip1$	$ip2$	$pt1$	$pt2$	dir	seq_no	ack_no	$window$	syn	ack
30	4	172.31.1.34	172.31.2.212	22	22	$\rightarrow$	5000	7280	8760	0	1
30	4	172.31.1.34	172.31.2.89	80	9080	$\rightarrow$	1000	1280	17424	0	1
31	4	172.31.1.34	172.31.2.212	80	9080	$\rightarrow$	4780	8214	6432	0	1
31	4	172.31.1.34	172.31.2.212	22	22	$\rightarrow$	5012	7280	8760	0	1
31	4	172.31.1.34	172.31.2.89	80	9080	$\rightarrow$	1012	1280	17424	0	1
32	4	172.31.1.34	172.31.2.212	22	22	$\leftarrow$	7280	5024	65110	0	1
32	4	172.31.1.34	172.31.2.212	22	22	$\rightarrow$	5024	7280	8760	0	1
32	4	172.31.1.34	172.31.2.89	80	9080	$\rightarrow$	1024	1280	17424	0	1

Table 3: *Transformed Tuples*

ts	$ip1$ $ip2$	$pt1$ $pt2$	dir	seq_no	ack_no	$window$	syn	ack
0	c_1	p_1	$\rightarrow$	0	2280	8760	0	1
1	c_1	p_1	$\rightarrow$	12	2280	8760	0	1
2	c_1	p_1	$\leftarrow$	2280	24	65110	0	1
2	c_1	p_1	$\rightarrow$	24	2280	8760	0	1
0	c_1	p_1'	$\rightarrow$	0	3434	6432	0	1
0	c_2	p_2	$\rightarrow$	0	280	17424	0	1
1	c_2	p_2	$\rightarrow$	12	280	17424	0	1
2	c_2	p_2	$\rightarrow$	24	280	17424	0	1

Any anonymized view of the trace must be checked for the potential leakage of sensitive information about the users and the network involved. We have described a formal measure for evaluating the trace in section 5. The decision must be taken by the publisher if this measure indicates risk of information leakage.

Without the measure, we can still give some statements on privacy based on the transformations chosen. In this example, we can see that the ip addresses and ports have been encrypted together. As a result, the information about individual hosts is lost. The frequency analysis of individual attributes like IP addresses, which has been key to many de-anonymizing attacks, is not possible. Also, simple transformation like translation of timestamps can have significant impact on the privacy and security. Due to the translation, every connection in the anonymized trace starts at time 0 and hence, inter-arrival information of packets across connections is lost. These translations of *ts, seq_no,*and *ack_no* make a fingerprinting attack difficult for the adversary. The removal of unrequired fields from the trace prevents leakage of any undesired fingerprints. Yet, some of the connections and users may have unique feature, which must be identified using the measure in Section 5 and the decision must be taken by the publisher before releasing the view.

3 The Trace Transformation Framework

In this section we describe the two main objects of our framework: *operators*, used by the trace owner to define trace transformations, and *constraints*, used by analysts to express utility requirements.

3.1 Trace transformation operators

The following transformation operators are applied to a network trace in order to obscure, remove, or translate field values. Each transformation operator removes information from the trace, making it more difficult for an adversary to attack, but also less useful for analysts. The trace owner may combine individual operators to form composite transformations, balancing utility and security considerations. The output of a composite transformation will be released to the analyst.

Operator descriptions

We consider a network trace as a table consisting of *records* representing IP packets. Each record contains packet header *fields* and a timestamp field, but does not include the packet payload.

Projection The simplest operator is *projection*, which removes from the input trace one or more designated fields. The term projection is a reference to the relational algebra operator of the same name. Projection is denoted Π_X

where X is a set of fields *to be retained*; all other fields are eliminated in the output trace.

Encryption The encryption operator hides target fields by applying a symmetric encryption function to one or more fields. The encryption operator is denoted $E_{X(Y),\kappa}$ where X is set of target fields to be encrypted, Y is an optional set of grouping attributes for encryption, and κ is a secret encryption key.

The encryption operation is applied as follows. For each record in the trace, the values of attributes from set X are concatenated with the values of attributes from set Y to form a string. The string is appropriately padded and then encrypted under a symmetric encryption algorithm using κ as the key. The ciphertext output replaces the fields of X in the output trace; the values for attributes in Y are not affected.

The encryption key is never shared, so the output trace must be analyzed without access to the values in these fields. A different encryption key is used for each encryption operator applied, but the same encryption key is used for all values of the fields in X. Thus, common values in an encrypted field are revealed to the analyst. However, if two records agree upon values in X but differ in values in Y, then the encrypted values of X will be different for these records. As a result, the encryption of two records will be same only if they agree upon values for X as well as for Y.

Table 3 shows the result of applying encryption operators $E_{\{ip1,ip2\},\kappa}$ and $E_{\{pt1,pt2\}(ip1,ip2),\kappa}$ to Table 2. The encryption allows connections (identified by source and destination IP, port fields) to be differentiated. However, it is not possible see that two connections share the same destination port, for example. Further, because source and destination IP are used as input for encryption of ports, it is not possible to correlate ports across different connections.

Canonical Ordering The *canonical ordering* operator is used to replace fields whose actual values can be eliminated as long as they are replaced by synthetic values respecting the ordering of the original values. The ordering operator is denoted $O_{X(Y)}$ where X is the set of target fields to be replaced, and Y is an optional set of grouping fields. If the input set Y is empty, the data entries in fields of X are sorted and replaced by their order in the sorted list, beginning with zero. If the input set Y is not empty, then the ordering operation is done separately for each group of records that agree on values for the columns in Y.

Translation The translation operation is applied to numerical fields in the trace, shifting values through addition or subtraction of a given constant. The operator is denoted $T_{X(Y)}$ where X is a set of target columns that are translated by the operator. The operator can optionally have another set of columns Y called grouping columns, which are not affected by the operation.

If the input set Y is empty, all the data-entries in target columns in X are shifted by a parameter c. The shift is caused by subtracting a random parameter c from each entry in the columns. If the input set Y is not empty, then all the records in a trace are formed into groups such that the records in each group have same data-entries for columns in Y. For records in each group, the target columns X are shifted by a parameter c where the value of parameter is dependent on the group. The parameter value can be chosen randomly or by using a function that takes data-entry of Y for the group as input.

Scaling The scaling operation scales all the values in a given field by multiplying it with a constant multiplier. The scaling operator is denoted $S_{X,k}$ for a set of target fields X. The scaling operator acts scales all the values in fields in X by a factor of k.

It is sometimes convenient to consider the identity transformation, denoted I_X, which does not transform field X, including it in the output without modification.

Composite Transformations

The operators above can be combined to form composite transformations for a network trace. We assume in the sequel that composite transformations ϕ are represented in the following normal form:

$$\phi = \Pi_X \circ \ \phi^1_{X_1} \circ \phi^2_{X_2} \circ ... \circ \phi^n_{X_n} \tag{1}$$

where $\phi^i_{X_i}$ refers to $(i+1)^{th}$ operator in ϕ which acts on attribute set X_i and for all i, $\phi^i_{X_i} \in \{E, T, O, S, I\}$. We denote the set of all such transformations Φ. The last operation applied to the trace is the projection Π_X. Since X is

the set of attributes retained by Π, with all others removed, any other operators on fields not in X are irrelevant and can be disregarded. Thus, without loss of generality we assume $\forall i, X_i \subseteq X$. Further we restrict our attention to composite operations in which each field in the trace is affected only by one operation: $\forall i, j, X_i \cap X_j = \{\}$. In the paper, we will assume Π_X to be present even if not mentioned in ϕ. For example, $E_{X_1} \circ T_{X_2}$ and $\Pi_{X_1 \cup X_2} \circ E_{X_1} \circ T_{X_2}$ will mean the same.

Other operators Our framework can easily accommodate other transformation operators. We have found that this simple set of operators can be used to generate safe transformations supporting a wide range of network analyses performed in the research literature (in addition to the example in Section 2, we consider other examples in Appendix A.1). In many cases, adding additional transformation operators to our framework requires only minor extensions to the algorithms described in Section 4. For example, prefix-preserving encryption of IP addresses could be added as a special transformation operator. We do not consider it here, since it has been thoroughly discussed in the literature [8, 15], and we have focused on supporting the wide range of networking studies that do *not* require prefix preservation. However, it is worth noting that some potentially important operators (e.g. random perturbation of numeric fields, or generalization of field values) will lead to analysis results that are approximately correct, but not exact. In this initial investigation, we are concerned with supporting exact analyses. We leave as future work the consideration of such operators and the evaluation of approximately correct analysis results.

3.2 Specifying Utility Requirements

In our framework, the analyst seeking access to a network trace must specify their utility requirements formally. These requirements are expressed as a set of *constraints* asserting a given relationship between fields in the original trace and fields in the anonymized trace. The analyst is expected to specify constraints that are sufficient to allow the exact analysis to be carried out on the trace. Each constraint states which item of information must be preserved while anonymizing the trace. An item of information in a network trace can be either: (i) the value of some field in the trace, or (ii) the value of some arithmetic or boolean expression evaluated using the fields from network trace only. The syntax of notation for the constraint is as follows:

Definition 1 (Utility Constraint)**.** *A utility constraint is described by a rule of the following form:*

$$\langle \textit{qualifier} \rangle \Rightarrow (expr(orig) = expr(anon))$$

A complete grammar for utility constraints is given in Table 4. We use record.field and ϕ(record).field, which are terminal symbols in the grammar, to mean any valid field in the original network trace and the transformed trace, respectively.

The above constraint can be interpreted as "if there are one or more records in a trace that satisfy the qualifying condition given in $\langle$*qualifier*$\rangle$, then the value of expression $expr$ evaluated over these records must be equal to the value of same expression when evaluated over corresponding anonymized records". We can use this grammar to generate complex arithmetic expressions involving any fields in the trace.

A constraint rule is *unary* if its conditions refer to a single record, or *binary* if it refers to two records. For example, if an analyst wants to test two port numbers involved in a connection for equality, this can be expressed as the following unary constraint:

$$Any(t) \Rightarrow ((t.pt1 == t.pt2) = (\phi(t).pt1 == \phi(t).pt2))$$

The qualifier $Any(t)$ is true for any record in the the trace. The *constraint* says that if the two port numbers in a record have same value then the corresponding values in the anonymized record should also be the same. We can see that the information that needs to be preserved is the equality of port numbers. A transformation need not preserve the actual values of port numbers in order to satisfy this rule.

A *binary constraint* requires two records for evaluating its expression. For example, the analyst may want to verify that the acknowledgement number in one packet is equal to the sequence number of another packet moving in opposite direction in a TCP connection. This requirement can be expressed as following constraint:

$$Opp(t1, t2) \Rightarrow (t1.ack == t2.seq) = (\phi(t1).ack == \phi(t2).seq)$$

The information that needs to be preserved here is the equality of two fields across records. The actual values need not be preserved. The qualifier $Opp(t1, t2)$ is user-defined and it states the conditions that must be true for two

Table 4: *Constraints Specification Language*

constraint-spec:	qualifier ⇒ constraint
qualifier:	User-defined list of boolean conditions that must be true for qualifying record(s)
constraint:	expr<**orig**> = expr<**anon**>
expr<*T*>:	arith-expr<*T*> | bool-expr<*T*>
T:	**orig** | **anon**
arith-expr<*T*>:	arith-expr<*T*> arith-op arith-expr¡*T*>
arith-expr<*T*>:	(arith-expr<*T*>)
arith-expr<*T*>:	field<*T*>
arith-op:	+ | - | * | /
bool-expr<*T*>:	bool-expr<*T*> cond-op bool-expr<*T*>
bool-expr<*T*>:	(bool-expr<*T*>)
bool-expr<*T*>:	field<*T*> bool-op field<*T*>
cond-op:	′&&′ | ′||′
bool-op:	< | > | ≤|≥| == |! =
field<**orig**>:	record.field
field<**anon**>:	ϕ(record).field

Table 5: *Lookup table for unary constraint used for verifying utility of transformation*

expression	**transformations**
t.a ≤ t.b t.a ≥ t.b t.a ≤ t.b t.a ≥ t.b	$O_{X(Y)}, T_{X(Y)},$ $I_X, S_{X,k}$ $E^o_{\{a\}(Y),\kappa} \circ E^o_{\{b\}(Y),\kappa}$ $a \notin Y, b \notin Y, \{a,b\} \subseteq X$
$t.a == t.b$ $t.a! = t.b$	$O_{X(Y)}, T_{X(Y)}, S_{X,k}$ $E_{\{a\}(Y),\kappa} \circ E_{\{b\}(Y),\kappa}, I_X$ $a \notin Y, b \notin Y, \{a,b\} \subseteq X,$
$t.a - t.b$	$T_{X(Y)}, I_X$ $a \notin Y, b \notin Y, \{a,b\} \subseteq X$
$t.a + t.b$	I_X $\{a,b\} \subseteq X$
$t.a \times t.b$	I_Z $\{a,b\} \subseteq Z$
$t.a/t.b$	$S_{X,k}, I_X$ $\{a,b\} \subseteq X$
t.a	$I_X, a \in X$

records to belong to packets moving in opposite directions in a connection. In this case, the list of conditions is $\{(t1.ip1 == t2.ip1), (t1.ip2 == t2.ip2), (t1.pt1 == t2.pt1), (t1.pt2 == t2.pt2), (t1.dir! = t2.dir)\}$.

We believe trace analysts will be able to use these constraint rules to accurately describe the properties of a trace required for accurate analysis. In most cases it is not difficult to consider a trace analysis and derive the fields whose values must be unchanged, or the relationships between values that must be maintained. (See Appendix A.1 for examples.) We note that it is in the interest of the analyst to choose a set of constraint rules which specific to the desired analysis task. Obviously, if all fields of all records in the trace are required to be unmodified, then the only satisfying trace will be the original, unanonymized trace. Our framework does not impose any explicit controls on the utility requirements submitted by analysts, except that the trace owner is likely to reject requests for constraint requirements that are too general.

4 Formal analysis of trace transformations

An important feature of our framework is that it enables the trace owner to reason formally about the relationship between utility requirements and anonymizing transformations. In this section we show how the trace owner can determine conclusively that a published trace will satisfy the utility requirements expressed by analysts. In addition, we show how the trace owner can derive the *most secure transformation* satisfying a desired set of utility constraints. Lastly, we show how the trace owner can compare alternative publication strategies and analyze the potential impact of collusion amongst analysts who have received traces.

We refer to the formal reasoning about trace transformations and utility constraints as *static* analysis because these relationships between transformations hold for all possible input traces. Other aspects of trace privacy cannot be assessed statically; in Section 5 we measure the privacy of real traces under sample transformations.

4.1 Verifying the utility of a transformation

We now show that it is possible to test efficiently whether a given transformation will always satisfy the utility requirements expressed by a set of constraints.

Definition 2 (Utility constraint satisfaction)**.** *Given a set of constraints C and a transformation ϕ, we say ϕ satisfies C (denoted $\phi \models C$) if, for any input trace, the output of the transformation satisfies each constraint in C.*

Checking utility constraint satisfaction is performed independently for each constraint rule in C by matching the conditions specified in a constraint to the operators which impact the named fields. Recall that the general form for unary constraints is $\langle$*qualifier*$\rangle \Rightarrow (expr(t) = expr(\phi(t)))$ where $expr$ can either be conjunctive normal form of one or more comparisons, or an arithmetic expression. Since the unary constraint has only one record, each comparison

expression must involve two attributes from the network trace. For each comparison or arithmetic expression in $expr$, we look for the corresponding entry in Table 5 which lists expressions and compatible transformation operators. If the composite transform function ϕ has a matching transformations in Table 5, then we proceed to the next comparison or sub-expression. Otherwise we conclude that ϕ does not satisfy the constraint. If ϕ has a matching transformation for each of the sub-expressions, the constraint is said to be satisfied by the transformation.

The procedure for verifying binary constraints is similar, with some minor modifications. We describe the verification process in Appendix A.2.

4.2 A partial order for transformation operators

Since each transformation operator removes information from the trace, some composite transformations can be compared with one another in terms of the amount of information they preserve. We show here that there is a natural partial order on transformations.

Definition 3 (Inverse Set of a transformed Trace)**.** *Let $\mathcal{N}$ be a network trace transformed using transformation ϕ to get transformed trace $\phi(\mathcal{N})$. Then, the inverse set for trace $\phi(\mathcal{N})$ is given by all possible network traces that have the same algebraic properties as in $\phi(\mathcal{N})$ and hence, can give $\phi(\mathcal{N})$ as output when transformed using ϕ. We use notation $\phi^{-1}[\mathcal{N}]$ to represent this set.*

Example

Let us consider a transformed trace $\phi(\mathcal{N})$ obtained by applying $\phi = \Pi_{\{A\}} \circ O_{\{A\}}$ on $\mathcal{N}$. In our example, we consider $\phi(\mathcal{N})$ to be {1,2,3}. We can see that if we apply ϕ on network traces {1,5,100} and {4120, 5230, 6788}, the result will be same as $\phi(\mathcal{N})$. Infact there are $\binom{N}{3}$ such traces that have same algebraic properties as retained by ϕ in $\phi(\mathcal{N})$ where N is size of domain of attribute A. Thus, the inverse set $\phi^{-1}[\mathcal{N}]$ consists of $\binom{N}{3}$ traces.

Definition 4 (Equivalence of Traces under transformation ϕ)**.** *Two traces $\mathcal{N}_1$ and $\mathcal{N}_2$ are equivalent under transformation ϕ iff $\phi^{-1}[\mathcal{N}_1] = \phi^{-1}[\mathcal{N}_2]$. We denote this relation as $\mathcal{N}_1 \sim_\phi \mathcal{N}_2$.*

It can be seen that the relation $\sim_\phi$ is transitive. Hence, we can divide the entire domain of network traces into equivalence classes where all the network traces in an equivalence class are equivalent under ϕ.

Lemma 1 (Equivalence Class)**.** *For any network trace $\mathcal{N}$ and transformation ϕ, the equivalence class containing $\mathcal{N}$ (denoted by $e_\phi(\mathcal{N})$) is same as the inverse set $\phi^{-1}[\mathcal{N}]$.*

The proof for this lemma is given in Appendix A.3

Definition 5 (Equivalence of Transformations)**.** *Two transformations ϕ_1 and ϕ_2 are equivalent if the relations $\sim_{\phi_1}$ and $\sim_{\phi_2}$ divide the domain of network traces into same equivalence classes.*

The implication of this definition is that for any trace $\mathcal{N}$, the inverse sets $\phi_1^{-1}[\mathcal{N}] = \phi_2^{-1}[\mathcal{N}]$. In other words, the information retained under two transformations is exactly the same.

Definition 6 (Strictness Relation)**.** *Given two composite transformations ϕ_1 and ϕ_2, we say that ϕ_1 is stricter than ϕ_2 if -*

$$\forall \textit{ Network Trace } \mathcal{N},\ e_{\phi_2}(\mathcal{N}) \subseteq e_{\phi_1}(\mathcal{N})$$

This definition implies that if the transformation ϕ_1 is stricter than ϕ_2, then -

$$\forall \text{ Network Trace } \mathcal{N},\ \phi_2^{-1}[\mathcal{N}] \subseteq \phi_1^{-1}[\mathcal{N}]$$

In other words, ϕ_1 contains less information about the original trace and hence, the size of inverse set is bigger than that obtained using ϕ_2.

Using the definition of strictness, the most strict transformation is $\Pi_\emptyset$, which removes all attributes of the trace. The least strict transformation is $I_{\bar{X}}$, which simply applies the identity transformation to all attributes, returning the original trace without modification. All other transformations fall between these two in terms of strictness. For example, we have $E_{X(Y)} \prec O_{X(Y)}$ because encryption removes the ordering information from the data-entries. Also, $E_{X(Y')} \prec E_{X(Y)}$ if $Y \subseteq Y'$ as $E_{X(Y')}$ removes the equality information of X-entries from records which have the

same entries for Y but differ in some attribute in $(Y' - Y)$. More strictness relations for basic operators are given in Lemma 4 in Appendix A.3.

Recall that Φ denotes the set of all composite transformations. Then the following theorems show that the strictness relation has a number of convenient properties.

Theorem 1. $(\Phi, \preceq)$ *is a partially ordered set.*

Theorem 2. $(\Phi, \preceq)$ *is a join-semilattice i.e. for any two transformations ϕ_1 and ϕ_2, there is another transformation in Φ, denoted $\phi_1 \vee \phi_2$, which is the least upper bound of ϕ_1 and ϕ_2.*

The proofs of these results are included in Appendix A.4. Theorem 2 can be easily extended to conclude that any set of transforms has a unique least upper bound and this fact has a number of important consequences for the trace publisher:

- First, given a set of constraints C it is important for the trace publisher to compute the most secure transformation satisfying C. Theorem 2 shows that such a transformation always exists.
- Next, imagine that the trace publisher has derived three transforms ϕ_1, ϕ_2, ϕ_3 specific to three analyst requests. The publisher may wish to consider publishing a single trace that can satisfy all three requests simultaneously. The least upper bound of these three transformations, denoted $lub(\phi_1, \phi_2, \phi_3)$ is the transformation the publisher must consider.
- Similarly, if the publisher has already released the traces derived from ϕ_1, ϕ_2, ϕ_3 and fears that the analysts may collude, then the least upper bound transformation $lub(\phi_1, \phi_2, \phi_3)$ is a conservative bound on the amount of information the colluding parties could recover by working together.

Input : Set of Constraints C
Output: Composite Transform

Let S = $\{\}$ be empty set of attributes;
Let map = $\{\}$ Constraint-set to Transformation Map;
foreach *constraint c in* C **do**
    foreach *attribute a present in c* **do**
        $S = S \cup \{a\}$;
    end
    ρ=Most Secure Operator from look-up table that satisfies c;
    PUT(*map*,$\{c\}$,ρ);
end
while $\exists$ *dependent sets* C_1, C_2 *in* map **do**
    ρ_1 = GET(map,C_1);
    ρ_2 = GET(map,C_2);
    ρ = LEAST-UPPER-BOUND(ρ_1, ρ_2);
    REMOVE(map, C_1);
    REMOVE(map, C_2);
    PUT($map, C_1 \cup C_2, \rho$);
end
ϕ = $\prod_S$;
foreach *set* C *in* map **do**
    ρ= GET(map,C);
    $\phi = \phi \circ \rho$;
end
return(ϕ)

Algorithm 1: Most Secure Transform

4.3 Computing maximally secure transformations

The strictness relation can be used as the basis of an algorithm for finding the most secure transformation satisfying a set of utility requirements.

Definition 7 (Most Secure Transformation). *Given a set of constraints C, the most secure transformation is the minimum element in $\Phi[C]$, denoted* $\min(\Phi[C])$.

We denote by $\Phi[C]$ the set of transformations satisfying the constraints of C. Algorithm 1 computes the most secure transform given a set of constraints. The algorithm uses a map data-structure which keeps the mapping of a set of constraints to its most secure transform. It starts by forming $|C|$ different constraint sets with each set having exactly one constraint. Using look-up table for constraints, the strictest operator is obtained for each constraint and the entry is made in the map (Lines 3-8).

As a next step, two constraint sets (C_1, C_2) are chosen such that there exist an attribute which is referred by atleast one constraint in each set. The composite transforms for C_1 and C_2 can operate differently on this common attribute. Thus, the least upper bound of these transforms is computed to get the most secure transform having properties of both the transforms (Lines 11-13). The steps for obtaining lub can be seen in proof of Theorem 2 (given in Appendix A.4). The constraint sets C_1 and C_2 are now merged to obtain a single set and is put into the map along with lub. The previous entries for the two sets are removed from the map. (Lines 14-16).

The above steps are repeated until no dependent constraint sets are left. Now, all the transforms in the map transform disjoint set of attributes and do not conflict. As a final step, the composition of all these transforms is done. The resulting composition operator along with the required projection operator is returned as the most secure transform(Lines 18-23).

4.4 Evaluating collusion risk

Customizing published traces to the needs of analysts means that any given analyst will have the information they need, but not more than they need. However, if a trace owner publishes a set of traces, we must consider the implications of one party acquiring these traces and attempting to combine them.

The ability to correlate the information present in two (or more) transformed traces depends greatly on the particular transformations. As a straightforward defense against the risks of collusion, the trace owner can always consider the least upper bound, *lub*, of the published transformations. The *lub* provides a conservative bound on the amount of information released since each of the published traces could be computed from it. Therefore the trace owner can evaluate the overall privacy of publishing the *lub* transformation; if it is acceptable, then the risk of collusion can be ignored.

In many practical cases the *lub* is overly conservative, as it is not possible to merge the traces accurately. For example, if the two transformations $E_{\{ip1,ip2,pt1,pt2\},\kappa_1} \circ I_{win}$ and $E_{\{ip1,ip2\},\kappa_2} \circ I_{TTL}$ are released, it is difficult to correlate them. One trace includes a window field for each encrypted connection, while the other includes a TTL field for each pair of hosts. Because distinct port pairs are included in the encryption in the first trace, but removed from the second, it would be very difficult to relate packets across the two traces. In general, the relationship between the information content present in two transformations σ_1, σ_2 and the information present in the $lub(\sigma_1, \sigma_2)$ depends on (i) the transformation operators applied to fields common to σ_1 and σ_2, and (ii) the degree to which these fields uniquely identify packets in the trace. Static analysis tools can be used to evaluate (i), however (ii) may depend on the actual input trace and requires quantitative evaluation, similar to that described next.

5 Quantifying trace privacy

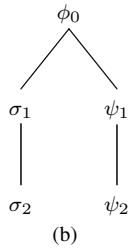
The techniques in the previous section allow the trace owner to compare the information content of some transformations, and to find the most secure transformation that satisfies a given set of utility requirements. This provides a *relative* evaluation of trace privacy, but it does not allow the trace owner decide whether a specific transformation meets an acceptable privacy standard.

In this section we measure quantitatively the privacy of sample trace transformations by applying the transformations to a real enterprise IP trace, simulating attacks by an informed adversary, and measuring the risk of disclosure. In addition to providing a reliable evaluation of the absolute security of a transformation, the quantitative analysis also allows us to compare the security of transformations that are not comparable (recall that these cases occur because the relation $\preceq$ in Section 4 is only a partial order). Also, we can quantify the improved security that results from publishing two traces customized for separate analyses, as compared with publishing a single trace that can support both.

Figure 2: *(a) Five example transformations used in the quantitative evaluation of host anonymity. (b) Tree representing strictness relationships between the example transformations (i.e. [child] $\preceq$ [parent] in each case).*

$\sigma_1 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1\},\kappa_2} \circ E_{\{pt2\},\kappa_2} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{win\}}$
$\sigma_2 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1,pt2\},\kappa_2} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{win\}}$
$\psi_1 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1\},\kappa_2} \circ E_{\{pt2\},\kappa_2} \circ O_{\{ts\}(C)} \circ O_{\{seq,ack\}(C)} \circ O_{\{ipid\}(C)}$
$\psi_2 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ E_{\{pt1,pt2\},\kappa_2} \circ O_{\{ts\}(C)} \circ O_{\{seq,ack\}(C)} \circ O_{\{ipid\}(C)}$
$\phi_0 = E_{\{ip1\},\kappa} \circ E_{\{ip2\},\kappa} \circ T_{\{ts\}(C)} \circ T_{\{seq,ack\}(C)} \circ I_{\{pt1,pt2,win,TTL\}} \circ O_{\{ipid\}(C)}$

(a)



(b)

Experimental setup We use a single IP packet trace collected at a gateway router of a public university. The trace consists of 1.83 million packets and has 41930 hosts, both external and internal. The trace was stored as a relational table in the open-source PostgreSQL database system running on a Dell workstation with Intel Core2 Duo 2.39 GHz processor and 2GB RAM. Each transformation was applied to the trace using the database system.

Attack model We focus on one of the most common threats considered in trace publication – the re-identification of anonymized host addresses. We assume the adversary has access (through some external means) to information about traffic properties of the hosts participating in the collected trace. We call these host fingerprints. The adversary attacks the published trace by matching fingerprints of these hosts to the available attributes of hosts in the published, transformed trace. The result of the attack is a set of unanonymized candidate hosts that could feasibly correspond to a target anonymized host.

Adversary knowledge We consider a powerful adversary who is able to fingerprint hosts using the collection of host attributes described in Figure 4. The $port1$ in Figure 4 refers to the ports on which the fingerprinted host receives packets, whereas $port2$ and *ip-address2* corresponds to hosts communicating with the fingerprinted host. The rest of the fields have their usual meanings.

We do not require an exact match of fingerprints and trace attributes. Instead, the adversary applies a similarity metric to the host pairs, and any un-anonymized host having similarity to the anonymized host, $\mathcal{A}$, above certain threshold is added to the candidate set of $\mathcal{A}$. A higher $threshold$ value reflects the high confidence of adversary about the accuracy of his fingerprints. In order to simulate a strong adversary, we compute the fingerprints available to adversary from the original trace and choose a high threshold value of 0.98. If the fingerprints being matched are sets of values X and Y, then the similarity is given by $sim(X,Y) = 1 - \frac{(|X \cup Y| - |X \cap Y|)}{|X \cup Y|}$. The similarity metric for continuous numeric-value fingerprints is given by $sim(x,y) = 1 - \frac{|x-y|}{\max(x,y)}$ Finally, we use the Pearson correlation coefficient to compare fingerprints which are chains of values (e.g. the chain of *seq_no* for a connection). We average the similarity of all the available fingerprints to compute overall similarity of the host.

Privacy Measure We measure privacy by computing, for various k, the value of $\mathcal{N}(k)$: the number of anonymized hosts in the trace that have a candidate set of size less than or equal to k. For example out of the total number of hosts in the trace, $\mathcal{N}(1)$ indicates the number of addresses the adversary is able to uniquely de-anonymize. Clearly, a lower value of $\mathcal{N}(k)$ indicates a more privacy-preserving trace.

Transformations We evaluate the anonymity of the five transformations shown in Figure 2(a), which were motivated by some of the sample analyses considered earlier in the paper. The first two transformations σ_1, σ_2 support the example analysis given in Section 2. Using strictness relations in basic operators (Appendix **??**), we can see that $\sigma_2 \preceq \sigma_1$. In transform σ_1, ports are encrypted separately allowing an adversary to use external information on the entropy of the ports, a host sends traffic on. This information is unavailable in transformation σ_2 because ports are encrypted together.

Figure 3: *Results giving number of hosts, $\mathcal{N}(k)$, whose candidate set size was $\leq k$ for $k = 1, 5, 10$*

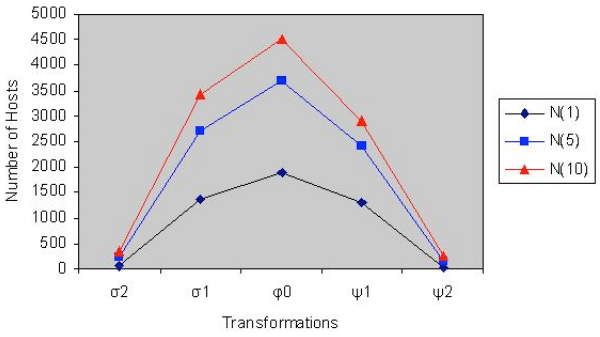


Figure 4: *Fingerprints Table*

field	finger-prints
port1(pt1)	Well-known ports, Entropy
port2(pt2)	Entropy
ip-address2(ip2)	Entropy
TTL	Maxmimum value, Minimum value
timestamps(ts)	Duration of each connection
seq number	Signature chains
ack number	-
window(win)	-
IP Identifier(ipid)	-

The transformations ψ_1 and ψ_2 allow the analyst to count the number of hosts which are involved in connections with a high rate of out-of-sequence packets. The identification of out-of-sequence packets requires only the order-information of sequence numbers, timestamps, acknowledgement numbers and IP identifiers in a connection. As these fields have been transformed using the O operator, none of the fingerprints are available for fingerprinting. It is easy to observe that $\psi_2 \preceq \psi_1$.

The transformation ϕ_0 has been chosen as a base case, in which all the fingerprints listed in Figure 4 are available to the adversary. Also, ϕ_0 is a relaxation of σ_1 and ψ_1. As such, it is capable of supporting both the above-mentioned analyses. Figure 2(b) illustrates the strictness relationships between the five transformations studied here.

Results We have summarized our results in Figure 3. The privacy measure $\mathcal{N}(1)$ gives the number of uniquely identified host and is greatest for ϕ_0, as expected. All the fingerprints are available in ϕ_0 leading to identification of 1904 of 41930 hosts. Also, the lower value of $\mathcal{N}(1)$ for σ_2 when compared with σ_1 validates the relation $\sigma_2 \prec \sigma_1$. It can be seen that the results are valid for similar relations among other operators.

In addition, the privacy measure allows us to compare transforms σ_2 and ψ_2 which are incomparable statically. The lowest $\mathcal{N}(1)$ value for ψ_2 indicates that it is the safest transform among the set of transforms considered. We can also study the significance of fields to the adversary using these results. For example, the significant difference in $\mathcal{N}(1)$ value for ψ_1 and ψ_2 indicates that the entropy of ports is highly informative for the adversary.

Finally, we can see that releasing two secure views σ_2 and ψ_2 for two different analysts results in disclosure of only 66 and 25 hosts respectively and is much less than 1904 hosts disclosed by a general view ϕ_0. This illustrates the significant gain in anonymity that results from publishing two traces customized for individual analyses, rather than publishing a single trace to satisfy both.

6 Related Work

Slagell et al [19] recognized the need for a framework to support the trade-off between the security and utility of the trace and provide multiple levels of anonymization. They stressed the need of metric to compare the utility of two traces based on fields available in them. In [13], the authors have proposed a measure for evaluating utility of network trace for intrusion detection. The proposed measure is specific to intrusion detection and cannot be applied to other analysis.

As described in the introduction, a wide range of anonymizing transformations have been considered for network traces. In [2], Xu et al proposed a cryptography based prefix-preserving anonymization scheme for the ip-addresses. This scheme provides one-to-one consistent anonymization for ip-addresses and is not applicable for the transformation of other fields. In [16], Pang et al have proposed a framework with some of the possible transformation for the different packet fields. However, its main focus is transforming the information contained in packet payload and not in the packet header fields. In [15], the various header fields have been studied in some detail and a framework to support anonymization of different fields is proposed. This framework allows for the anonymization policies to control the various fields. In theory, it can publish multiple views but it lacks the tools for analyzing utility of the different views.

In [14], Mogul et al propose a framework that requires an analyst to write the analysis program in the language supported by framework. This program is then reviewed by experts for any privacy or security issues. The approved program is executed by the trace owner and results are provided to the analyst. We compared our framework with this approach in Section 1.

The PREDICT [1] repository has been established to make network traces available for research. The network traces are distributed at various data hosting sites, and access to the traces is authorized only after the purpose and identity of researchers is reviewed and verified. To the best of our knowledge, the anonymization of traces is not customized to the needs analysts and multiple versions of traces are not published.

7 Conclusion

We have described a publication framework which allows a trace owner to customize published traces in order to minimize information disclosure while provably meeting the utility of analysts. Using our framework, the trace owner can verify a number of useful privacy and utility properties statically. Such properties hold for all possible traces, and can be verified efficiently. However, some aspects of trace privacy must be evaluated on each particular trace. We have implemented our techniques and quantified trace privacy under example transformations.

Our preliminary implementation suggests that our trace transformation operators can be applied efficiently. In future work we would like to implement a trace transformation infrastructure to efficiently support multiple transformations of a trace using indexing and parallel data access. We would also like to extend our framework to support additional transformation operators, and to pursue further analysis of collusion risks.

References

[1] Predict. *https://www.predict.org/*.

[2] Tcpdpriv. *http://ita.eee.lbl.gov/html/contrib/tcpdpriv.html*.

[3] Tcpurify. *http://irg.cs.ohiou.edu/ eblanton/tcpurify*.

[4] Anonymization of ip traffic monitoring data - attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *Proceedings of the Workshop on Privacy Enhancing Technologies*, pages 179–196, May 2005.

[5] K. Claffy. Ten things lawyers should know about internet research. *http://www.caida.org/publications/papers/2008/lawyers_top_ten/*.

[6] S. Coull, C. Wright, F. Monrose, M. Collins, and M. K. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *Proceedings of the* 14^{th} *Annual Network and Distributed System Security Symposium*, pages 35–47, February 2007.

[7] S. E. Coull, C. V. Wright, A. D. Keromytis, F. Monrose, and M. K. Reiter. Taming the devil: Techniques for evaluating anonymized network data. *Proceedings of the 15th Network and Distributed System Security Symposium*, February 2008.

[8] J. Fan, J. Xu, M. Ammar, and S. Moon. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. *Computer Networks*, 46(2):263–272, October 2004.

[9] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. In *Proceedings of IEEE INFOCOM*, pages 1199–1209, April 2003.

[10] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring tcp connection characteristics through passive measurements. In *Proceedings of INFOCOMM*, 2004.

[11] H. Jiang and C. Dovrolis. Source-level ip packet bursts: causes and effects. *ACM Internet Measurements Conference (IMC)*, 2003.

[12] D. Koukis, S. Antonatos, and K. Anagnostakis. On the privacy risks of publishing anonymized ip network traces. In *Proceedings of Communications and Multimedia Security*, pages 22–32, Oct 2006.

[13] K. Lakkaraju and A. Slagell. Evaluating the utility of anonymized network traces for intrusion detection. In *4th Annual SECURECOMM Conference*, September 2008.

[14] J. C. Mogul and M. Arlitt. Sc2d: an alternative to trace anonymization. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 323–328, New York, NY, USA, 2006.

[15] R. Pang, M. Allman, V. Paxon, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, January 2006.

Table 6: *Utility Requirements for Different Real Network Analyses along with the supporting transformation*

Formal Utility Requirements	Transform
Study of TCP connection interarrivals	$\phi_1 = I_{\{syn,ack\}} \circ E_{\{ip1,ip2,pt1,pt2\}} \circ T_{\{ts\}}$
$Any(t) \Rightarrow t.syn = \phi(t).syn$	
$Any(t) \Rightarrow t.ack = \phi(t).ack$	
$Any(t1, t2) \Rightarrow$ ((t1.ip1== t2.ip1) && (t1.ip2 ==t2.ip2) && (t1.pt1== t2.pt1)&& (t1.pt2 ==t2.pt2)) = ($\phi(t1)$.ip1== $\phi(t2)$.ip1 && $\phi(t1)$.ip2 ==$\phi(t2)$.ip2 && $\phi(t1)$.pt1== $\phi(t2)$.pt1 && $\phi(t1)$.pt2 ==$\phi(t2)$.pt2)	
Any(t1,t2) $\Rightarrow (t1.ts - t2.ts) = (\phi(t1).ts - \phi(t2).ts)$	
Study of Packet-Bursts: (addition to above requirements)	$\phi_2 = I_{\{syn,ack\}} \circ T_{\{ts\}} \circ E_{\{ip1,ip2,pt1,pt2\}} \circ T_{\{seq\_no,ack\_no\}(C)}$
Same-Conn(t1,t2) $\Rightarrow (t1.seq\_no \leq t2.seq\_no) = (\phi(t1).seq\_no \leq \phi(t2).seq\_no)$	
Same-Conn(t1,t2) $\Rightarrow (t1.seq\_no - t2.seq\_no) = (\phi(t1).seq\_no - \phi(t2).seq\_no)$	
Opp-Pckts(t1,t2) $\Rightarrow (t1.seq\_no == t2.ack\_no) = (\phi(t1).seq\_no = \phi(t2).ack\_no)$	
Classification of Out-Of-Sequence Packets	$\phi_3 = \phi_2 \circ I_{\{window,dir\}} \circ E_{ipid(C)}$
Same-Conn(t1,t2) $\Rightarrow (t1.ipid == t2.ipid) = (\phi(t1).ipid = \phi(t2).ipid)$	
Rest of the requirements for this case are exactly same as given in Section 2	

[16] R. Pang and V. Paxson. A high-level programming environment for packet trace anonymization and transformation. In *Proceedings of ACM SIGCOMM '03*, pages 339–351, New York, NY, USA, 2003.

[17] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.

[18] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley. Analyzing privacy in enterprise packet trace anonymization. In *Proceedings of the* 15^{th} *Network and Distributed Systems Security Symposium*, 2008.

[19] A. Slagell and W. Yurcik. Sharing computer network logs for security and privacy: A motivation for new methodologies of anonymization. *IEEE/CREATENET SecureComm*, pages 80–89, 2005.

APPENDIX

A Formal Analysis of Trace Transformations

A.1 Example of Real World Analyses

We provide some examples of the real world analyses along with their description and the utility constraints. It can be seen that the utility constraints are easily derivable from the description of the analysis with some backhround knowledge about the networks.

In [17], TCP connection interarrivals have been studied to determine if it can be modeled correctly with Poisson modeling. This study requires the starting time(can be relative) for each connection which can be obtained from SYN/FIN packets.

In [11], the causes and effects of packet bursts from individual flows have been studied. The packet burst is identified by several packets sent back-to-back in a flow. Their impact on aggregate traffic is studied by observing the scaling behavior of the traffic in trace over range of timescales.

In [9], the classification of out-of-sequence packets is done. This has similar requirements as given in Table 1. The additional requirement imposed is the availability of IP Identifier with its equality information preserved across records in connection.

The utility constraints and the composite transformation satisfying them is given in Table 6.

A.2 Verification of Binary Constraints

Given transformation ϕ and constraint set C, checking the utility constraint satisfaction requires checking satisfiability of each constraint (unary or binary) in C. Here we present the key concepts in understanding the verification of binary constraints.

Qualifying Conditions

The qualifying conditions in a constraint are the set of conditions given in *qualifier* that must hold true for the records qualified for constraint satisfaction. These conditions provide the comparisons of the fields in these records. The example of qualifier and its set of conditions is given in Table 1.

Table 7: *Lookup table for binary constraint required for verifying utility of a transformation*

qualifier-set	expression	transformations
Q	t1.a $\leq$ t2.a t1.a $\geq$ t2.a t1.a $\leq$ t2.a t1.a $\geq$ t2.a	$O_{X(Y)}, T_{X(Y)}$, $I_X, S_{X,k}$ $E^o_{\{a\}(Y),\kappa}$ $a \in X, Q_Y \subseteq Q$
	$t1.a - t2.a$	$T_{X(Y)}, I_X$ $a \in X, Q_Y \subseteq Q$
	$t1.a + t2.a$	I_X $a \in X$
	$t1.a \times t2.a$	I_X $a \in X$
	$t1.a/t2.a$	$S_{X,k}, I_X$ $a \in X$
	t1.a	$I_X, a \in X$
Q	t1.a $\leq$ t2.b t1.a $\geq$ t2.b t1.a $\leq$ t2.b t1.a $\geq$ t2.b	$O_{X(Y)}, T_{X(Y)}$, $I_X, S_{X,k}$ $E^o_{\{a\}(Y),\kappa} \circ E^o_{\{b\}(Y),\kappa}$ $\{a,b\} \subseteq X\ Q_Y \subseteq Q$
	$t1.a - t2.b$	$T_{X(Y)}, I_X$ $\{a,b\} \subseteq X, Q_Y \subseteq Q$
	$t1.a + t2.b$	I_X $\{a,b\} \subseteq X, Q_Y \subseteq Q$
	$t1.a \times t2.b$	I_Z $\{a,b\} \subseteq Z$
	$t1.a/t2.b$	$S_{X,k}, I_X$ $\{a,b\} \subseteq X$
Q	t1.a ! = t2.b t1.a == t2.b	$O_{X(Y)}, T_{X(Y)}$, $I_X, S_{X,k}$ $E^o_{\{a\}(Y),\kappa} \circ E^o_{\{b\}(Y),\kappa}$ $\{a,b\} \subseteq X, Q_Y \subseteq Q$

Definition 8. *Given two sets of conditions Q_1 and Q_2 and $Q_1 \neq Q_2$, we say that Q_1 is relaxed as compared to Q_2 if any records which satisfy conditions in Q_2, also satisfies the conditions in Q_1. Q_1 is relaxed as compared to Q_2 only if $Q_1 \subseteq Q_2$.*

Grouping Conditions

The transformation operators like $T_{X(Y)}, O_{X(Y)}$ or $E_{X(Y),\kappa}$ divides the trace into group of records where each group agrees upon its values for attributes in set Y. The transformed attributes X can be compared across records only if these two records belong to the same group. Thus, set $Y = \{y_1, ..., y_n\}$ imposes a *grouping condition* and is given by-

$$Q_Y = \{(t1.y_1 == t2.y_1), (t1.y_2 == t2.y_2), ..., (t1.y_n == t2.y_n)\}$$

Any transformation with *grouping condition* Q_Y can satisfy constraint with qualifying condition Q only if the records selected by Q are subset of records grouped by Q_Y. This holds true if $Q_Y \subseteq Q$.

Verification Process

The general form for the binary constraint is-

$$qualifier(t1, t2) \Rightarrow (expr(t1, t2) = expr(\phi(t1), \phi(t2))$$

Let us assume that the binary constraint has Q as the set of qualifying conditions and ϕ is the transformation being checked. As with the unary constraints, we divide the expression $expr$ into atomic sub-expressions where each sub-expression can be either comparison or an arithmetic expression. We look for satisfiability of each sub-expression.

For any sub-expression except of type $(t1.a == t2.a)$, we look for the entry for the subexpression in Table 7 which lists the expressions with compatible transformation operators and the requisites. If the transformation ϕ has a matching transformation in Table 7 and it satisfies the requisites mentioned, the sub-expression is satisfied and we look for the next subexpression.

For subexpression of type $(t1.a == t2.a)$, we find out the set X of attributes such that $\forall x \in X$, $(t1.x == t2.x)$ is present in $expr(t1, t2)$. These sub-expressions are satisfied if following holds true-

- For any encryption function of type $E_{Y,\kappa}$ or $E_{Y(Z),\kappa}$ in ϕ, $(Y - X == \{\})$ must be true. In addition, either $(Z \subseteq X)$ or $Q_Z \subseteq Q$ must hold.
- For any transformation of type $T_{Y(Z)}$ or $O_{Y(Z)}$ such that $(Y \cap X! = \{\})$ then either $Z \subseteq X$ or $Q_Z \subseteq Q$.

If ϕ satisfies all the sub-expressions in constraint, the constraint is satisfied.

A.3 Strictness Relations and its properties

In this section, we describe the properties of strictness relations. We begin with the proof of Lemma 1.

Lemma (Equivalence Class)**.** *For any network trace $\mathcal{N}$ and transformation ϕ, the equivalence class containing $\mathcal{N}$ (denoted by $e_\phi(\mathcal{N})$) is same as the inverse set $\phi^{-1}[\mathcal{N}]$.*

Proof. Let us consider any trace $\mathcal{N}'$ such that $\mathcal{N} \in e_\phi(\mathcal{N}')$ i.e. $\mathcal{N}' \sim_\phi \mathcal{N}'$. Hence, using the definition of equivalence (refer definition 4), we have $\phi^{-1}[\mathcal{N}] = \phi^{-1}[\mathcal{N}']$. Since, $\mathcal{N} \in \phi^{-1}[\mathcal{N}]$, $\mathcal{N}' \in \phi^{-1}[\mathcal{N}']$ and $\phi^{-1}[\mathcal{N}] = \phi^{-1}[\mathcal{N}']$, we get $\mathcal{N}' \in \phi^{-1}[\mathcal{N}]$. Thus, any trace $\mathcal{N}' \sim_\phi \mathcal{N}$ is present in the inverse set $\phi^{-1}[\mathcal{N}]$.

Now, we show that any two traces $\mathcal{N}_1$ and $\mathcal{N}_2$ in $\phi^{-1}[\mathcal{N}]$ belong to the equivalence class of $\mathcal{N}$. We see that both the $\mathcal{N}_1, \mathcal{N}_2$ have all the algebraic properties retained by ϕ in $\phi(\mathcal{N})$. Since ϕ retains only these properties, the traces $\phi(\mathcal{N}_1)$ and $\phi(\mathcal{N}_2)$ will have the same properties as $\phi(\mathcal{N})$. As a result, the inverse set of $\phi(\mathcal{N}_1)$ and $\phi(\mathcal{N}_2)$ will be the same as the inverse set of $\phi(\mathcal{N})$. Hence, proved.

□

Definition 9 (Independent Transformations)**.** *Two transformations are independent of each other if they transform and project disjoint set of attributes.*

Lemma 2. *If $\phi_1 \prec \phi_2$ and ϕ is a transformation independent of ϕ_1 and ϕ_2, then $\phi_1 \circ \phi \preceq \phi_2 \circ \phi$.*

Proof. It is given that ϕ_1 and ϕ are independent of each other. Thus, if X and Y are the attribute sets transformed and projected by ϕ_1 and ϕ respectively, then $X \cap Y = null$.

If these two transforms are used to transform a network trace $\mathcal{N}$, then $\phi_1(\mathcal{N})$ does not retain any information about attribute set Y. As a result, $\phi_1^{-1}[\mathcal{N}]$ contains traces with all possibilities for Y from its domain. However, $\phi^{-1}[\mathcal{N}]$ consists of traces with fewer possibilities for Y but it has all the possibilities for X from its domain. But when we apply $\phi_1 \circ \phi$, it retains information about X as well as Y and it can be seen that -

1. $(\phi_1 \circ \phi)^{-1}[\mathcal{N}] \subset \phi_1^{-1}[\mathcal{N}]$
2. $(\phi_1 \circ \phi)^{-1}[\mathcal{N}] \subset \phi^{-1}[\mathcal{N}]$
3. $(\phi_1 \circ \phi)^{-1}[\mathcal{N}] = \phi_1^{-1}[\mathcal{N}] \cap \phi^{-1}[\mathcal{N}]$

Similar statements can be made about $\phi_2 \circ \phi$ because ϕ_2 and ϕ are independent as well. Thus, we have -

1. $(\phi_1 \circ \phi)^{-1}[\mathcal{N}] = \phi_1^{-1}[\mathcal{N}] \cap \phi^{-1}[\mathcal{N}]$
2. $(\phi_2 \circ \phi)^{-1}[\mathcal{N}] = \phi_2^{-1}[\mathcal{N}] \cap \phi^{-1}[\mathcal{N}]$
3. $\phi_2^{-1}[\mathcal{N}] \subset \phi_1^{-1}[\mathcal{N}]$ $(\because \phi_1 \prec \phi_2)$

From the set theory, we know that if $A \subset B$ then $A \cap U \subseteq B \cap U$. Using this, we get

$$(\phi_2 \circ \phi)^{-1}[\mathcal{N}] \subseteq (\phi_1 \circ \phi)^{-1}[\mathcal{N}]$$

This is true for any network trace $\mathcal{N}$. Hence, we have shown that $\phi_1 \circ \phi \preceq \phi_2 \circ \phi$. □

Corollary 1 (Commutative Property)**.** *If ϕ_1 and ϕ_2 are independent of each other, then $\phi_1 \circ \phi_2 = \phi_2 \circ \phi_1$.*

Corollary 2 (Transitive Relation)**.** *The strictness relation is transitive i.e. if $\phi_1 \prec \phi_2$ and $\phi_2 \prec \phi_3$, then $\phi_1 \prec \phi_3$.*

Corollary 3 (Strictness Chain)**.** *If there exists a chain of strictness relations from ϕ_2 to ϕ_1 such that $\phi_1 \prec \rho_1 \prec ... \prec \rho_n \prec \phi_2$ then $\phi_1 \prec \phi_2$*

The proofs for these corollaries are straightforward using strictness definition (refer 6). The Corollary 3 allows us to compute the strictness relations from the known strictness relations.

Example

Assuming that we have already computed the following strictness relations using the definition of strictness - $E_{X_1} \circ E_{X_2} \succ E_{X_1 \cup X_2}$ and $T_Y \succ O_Y$ for any X_1, X_2 and Y. Let us compare $E_{X_1} \circ E_{X_2} \circ T_Y$ and $E_{X_1 \cup X_2} \circ O_Y$.

$$\begin{aligned} E_{X_1} \circ E_{X_2} \circ T_Y &\succ E_{X_1 \cup X_2} \circ T_Y \\ &\succ E_{X_1 \cup X_2} \circ O_Y \end{aligned}$$

Thus, there exists a chain where each step in chain has been obtained using the precomputed relations, Lemma 2 and the commutative property given in Corollary 1. Finally, Corollary 3 implies that $E_{X_1 \cup X_2} \circ O_Y \prec E_{X_1} \circ E_{X_2} \circ T_Y$.

Now, we give a theorem which allows us to get the minimal set of strictness relations that must be pre-computed in order to compare any two transformations.

Lemma 3. *If $\phi_1, \phi_2, ..., \phi_n$ are composite transforms independent to each other and each $\phi_i \prec \phi$, then*

$$\phi_1 \circ \phi_2 \circ ... \circ \phi_n \preceq \phi$$

Proof. We provide the informal proof to this lemma as follows- As each ϕ_i is stricter than ϕ, the information contained in each ϕ_i is upper-bounded by the information in ϕ. Since ϕ_is are independent, each has information about distinct set of attributes. As a result, the overall information contained in ϕ_is can be at most equal to the information in ϕ. Thus, the composition of ϕ_is is stricter or equal to ϕ. □

Theorem 3 (Basis Set of Strictness Relation)**.** *If $\mathcal{S}$ is the set of basic operators, then we can derive a relation between any two comparable composite transforms (in normal form) using the basis set of strictness relations which is given by the relations for following pairs-*

1. $\{\alpha_{X(Y)}, \alpha_{X(Y')}\}$ *where* $Y \subset Y'$ $\forall \alpha \in \mathcal{S}$.
2. $\{\alpha_{X(Y)}, \alpha_{X'(Y)}\}$ *where* $X \subset X'$ $\forall \alpha \in \mathcal{S}$.
3. $\{\alpha_{X(Y)}, \alpha_{X_1(Y)} \circ \alpha_{X_2(Y)}\}$ *where* $X = X_1 \cup X_2$ $\forall \alpha \in \mathcal{S}$.
4. $\{\alpha_{X(Y)}, \beta_{X(Y)}\}$ $\forall \alpha, \beta \in \mathcal{S}$.

Proof. We need to prove that we can find a strictness chain for any two comparable composite transformations ϕ_1 and ϕ_2. Let $\phi_1 \prec \phi_2$. Let X_1, X_2 be the set of attributes transformed and projected by ϕ_1 and ϕ_2 respectively.

First, we show that $X_1 \subseteq X_2$. It is because if X_1 had some attribute which is not in X_2, then the transformation ϕ_1 will have atleast some information about this attribute and can be used to reduce the possible values of this domain. This is not possible in ϕ_2. Hence, for any network trace $\mathcal{N}, \phi_2^{-1}[\mathcal{N}] \not\subset \phi_1^{-1}[\mathcal{N}]$. This is contradiction as $\phi_1 \prec \phi_2$. Hence, $X_1 \subseteq X_2$.

Case I: $\phi_1 = \alpha_{X_1(Y_1)}, \phi_2 = \alpha_{X_2(Y_2)}$ where $X_1 \subseteq X_2$

First, we observe that in order to have $\phi_1 \prec \phi_2$, we must have $Y_2 \subseteq Y_1$. It is because if Y_1 was subset of Y_2, then ϕ_1 will have records grouped by Y_1 which will be bigger than the group of records formed by Y_2. Thus, ϕ_1 will have more information retained. Hence, we cannot have $Y_1 \subseteq Y_2$. These two sets cannot be disjoint either as that will make operators incomparable. Thus, $Y_2 \subseteq Y_1$.

From the basis set, we can compare $\phi_2 = \alpha_{X_2(Y_2)}$ with $\phi_3 = \alpha_{X_2(Y_1)}$ using relation of type 1. This relation must imply that $\phi_3 \prec \phi_2$. ($\because \phi_1 \prec \phi_2$ for any X_1, Y_1, X_2, Y_2 where $X_1 \subseteq X_2$ and $Y_2 \subseteq Y_1$, then we must have $\phi_3 \prec \phi_2$). Now, we can compare ϕ_3 with ϕ_1 using relation of type 2 in basis set. This relation must imply that $\phi_1 \prec \phi_3$. Thus, we can have a chain of comparisons $\phi_1 \prec \phi_3 \prec \phi_2$ obtainable from basis set of relations.

Case II: $\phi_1 = \alpha_{X_1(Y_1)}, \phi_2 = \beta_{X_2(Y_2)}$ where $X_1 \subseteq X_2$

From the basis set, we can compare $\phi_1 = \alpha_{X_1(Y_1)}$ with $\phi_3 = \beta_{X_1(Y_1)}$ using relation of type 4. This relation must imply that $\phi_1 \prec \phi_3$ ($\because \phi_1 \prec \phi_2$ for any X_1, Y_1, X_2, Y_2 where $X_1 \subseteq X_2$, then we must have $\phi_1 \prec \phi_3$). Now, we look at $\phi_4 = \beta_{X_2(Y_1)}$. We can compare ϕ_4 to ϕ_3 using relation of type 2 in basis set. As $X_1 \subseteq X_2$, we can argue that $\phi_3 \prec \phi_4$. As in Case I, we can show that $Y_2 \subseteq Y_1$. Now, using relation of type 1, we can compare ϕ_4 to ϕ_2. Also, it must imply $\phi_4 \prec \phi_2$. Since $Y_2 \subseteq Y_1$ implies that ϕ_2 must have more information than ϕ_4. Thus, we can have chain of comparison from ϕ_2 to ϕ_1 using relations in basis set only.

Case III: $\phi_1 = \alpha_{X(Y)}, \phi_2 = \beta^1_{X_1(Y_1)} \circ \beta^2_{X_2(Y_2)} ... \circ \beta^k_{X_k(Y_k)}$ where $X \subseteq \bigcup_1^k X_i$

Let us consider $\phi_3 = \beta^1_{X'_1(Y)} \circ \beta^2_{X'_2(Y)} ... \circ \beta^k_{X'_k(Y)}$ where $X'_i = X_i \cap X \subseteq X$. Using case I, we can get the chain

of relations from ϕ_2 to ϕ_3 such that $\phi_3 \preceq \phi_2$. Using relations of type 4 in basis set, we can get a chain of relations from ϕ_3 to $\phi_4 = \alpha^1_{X'_1(Y)} \circ \alpha^2_{X'_2(Y)} ... \circ \alpha^k_{X'_k(Y)}$ such that $\phi_4 \prec \phi_3$. Finally, relations of type 3 in basis set allows us to get a chain of relations from ϕ_4 to $\phi_1 = \alpha_{\bigcup_1^k X'_i(Y)} = \alpha_{X(Y)}$. Thus, we have a chain of relation from ϕ_2 to ϕ_1 using relations in basis set only.

Case IV: $\phi_1 = \alpha^1_{X_1(Y_1)} \circ \alpha^2_{X_2(Y_2)} ... \circ \alpha^m_{X_m(Y_m)}, \phi_2 = \beta^1_{X'_1(Y'_1)} \circ \beta^2_{X'_2(Y'_2)} ... \circ \beta^n_{X'_n(Y'_n)}$ where $\bigcup_1^m X_i \subseteq \bigcup_1^n X'_i$
Since $\phi_1 \prec \phi_2$ and ϕ_1 and ϕ_2 are such that each attribute is transformed by exactly one operator, the information retained by each operator in ϕ_1 is unique and is less than the overall information in ϕ_1. Thus, we have each $\alpha^i_{X_i(Y_i)} \prec \phi_1$. Since, $\phi_1 \prec \phi_2$, we must have $\alpha^i_{X_i(Y_i)} \prec \phi_2$. Using the steps in Case III, we can prove that each $\alpha^i_{X_i(Y_i)} \prec \phi_2$ using relations from basis set. Now, using lemma 3, $\alpha^i_{X_i(Y_i)} \prec \phi_2$ for $i = 1$ to m implies that -

$$\alpha^1_{X_1(Y_1)} \circ \alpha^2_{X_2(Y_2)} ... \circ \alpha^m_{X_m(Y_m)} \prec \phi_2$$

The above expression implies $\phi_1 \prec \phi_2$. Since, Case IV represents the most general case, we have shown that we can compare any two comparable composite transforms ϕ_1 and ϕ_2 using only relations present in basis set. □

In the following lemma, we have listed the minimal set of strictness relations among basic operators in $\{E, O, T, S, I, \Pi\}$. These relations can be verified using the definition of strictness. Any pair of composite transformations can be compared using these relations.

Lemma 4. ***Strictness Relation among Basic Operators***

1. $\phi_{X(Z)} \prec \phi_{X(Y)}$ *if* $Y \subset Z$ *and* $\phi \in \{E, O, T, S\}$
2. $\phi_{X_1(Y)} \circ \phi_{X_2(Y)} \prec \phi_{X(Y)}$ *if* $X = X_1 \cup X_2$ *and* $\phi \in \{O, T, S\}$
3. $E_{X(Y)} \prec E_{X_1(Y)} \circ E_{X_2(Y)}$ *if* $X = X_1 \cup X_2$
4. $E_{X(Y)}$ *and* $E_{X'(Y)}$ *are incomparable if* $X \subset X'$
5. $\phi_{X(Y)} \prec \phi_{X'(Y)}$ *where* $X \subset X'$ *and* $\phi \in \{O, T, S, I\}$.
6. $\Pi_{\{\}} \prec E_{X(Y)} \prec O_{X(Y)}$
7. $O_{X(Y)} \prec S_X$
8. $O_{X(Y)} \prec T_{X(Y)}$
9. $\phi_{X(Y)} \prec I_X$ *if* $\phi \in \{E, O, T, S\}$
10. $\Pi_X \prec \Pi_{X'}$ *if* $X \subset X'$
11. $\phi_{X_1} \circ \phi_{X_2} = \phi_X$ *where* $\phi \in \{\Pi, I\}$

A.4 Proof of Theorem 2

The proof for Theorem 2 requires a small result which we have given here as a lemma. This states that there exists a least upper bound in set of composite transforms Φ for any two basic operators.

Lemma 5 (Most Secure Transform for Basic Operators)**.** *If* $\rho_{X_1(Y_1)}$ *and* $\psi_{X_2(Y_2)}$ *are two operators from set* $\{E, O, T, S, I\}$*, then the most secure transform is the least upper bound* $\rho_{X_1(Y_1)} \vee \psi_{X_2(Y_2)}$*, and it is given by* ϕ *-*

- $\phi = \rho_{X_1(Y_1)} \circ \psi_{X_2(Y_2)}$ *if* $(X_1 \cap X_2) = \{\}$
- $\phi = E_{X(Y)} \circ E_{X_1 - X(Y_1)} \circ E_{X_2 - X(Y_2)}$ *if* $\rho = \psi = E$
- $\phi = \rho_{X_1 \cup X_2(Y)}$ *if* $\rho \preceq \psi$*,*$\psi \neq E$ *and* $(X_1 \cap X_2)! = \{\}$
- $\phi = E_{X_1 - X(Y_1)} \circ \psi_{X_2(Y)}$ *if* $\rho = E, \psi \prec E$
- $\phi = I_{X_1 \cup X_2}$ *if* $\rho = S, \psi = T$ *and* $(X_1 \cap X_2)! = \{\}$

where $X = X_1 \cap X_2$ *and* $Y = Y_1 \cap Y_2$*.*

Theorem. $(\Phi, \preceq)$ *forms a join-semilattice i.e. any two elements in* Φ *have a least upper bound in* Φ*.*

$$\forall \phi_1, \phi_2 \in \Phi \quad \exists (\phi_1 \vee \phi_2) \in \Phi$$

Proof. We will use notation $\{\phi, \psi, \chi, \omega\}$ for composite transforms and $\{\sigma, \delta, \xi\}$ for basic operators. We will say that two transforms are *independent* if they transform disjoint set of attributes. We will say one transform is in *conflict* with another if there exists an attribute which is transformed by both.
Case I $\phi_1 = \sigma$ and $\phi_2 = \delta$
For any two basic operators, there exists a least upper bound(lub) and is given in Lemma 5.
Case II ϕ_1 and ϕ_2 are independent.
This is a trivial case. Since the transforms ϕ_1, ϕ_2 act on two disjoint attribute sets, the information in one is unrelated to another. Thus, the lub is given by composition $\phi_1 \circ \phi_2$.
Case III $\phi_1 = \psi \circ \sigma$ and $\phi_2 = \delta$, ψ and δ are independent but δ and σ are in conflict..
We show that there exists a lub for $\{(\psi \circ \sigma), \delta\}$ in $\Phi$and is given by $\omega = (\sigma \vee \delta) \circ \psi$.
From Case II, we know that ω is lub of ψ and $(\sigma \vee \delta)$. Since, $(\sigma \vee \delta)$ is lub of σ and δ. Hence, ω is lub of $\{\psi, \sigma, \delta\}$. Also, anything greater than ψ and σ is also greater than $\psi \vee \sigma$. Thus, $\omega \succeq (\psi \vee \sigma)$ and $\psi \vee \sigma = \psi \circ \sigma$ implies $\omega \succeq (\psi \circ \sigma)$. Using this with $\omega \succeq \delta$, we get $\omega \succeq lub((\psi \circ \sigma), \delta)$. Thus, ω is upper bound of ϕ_1 and ϕ_2.
Let us say χ be some upper bound of $\{(\psi \circ \sigma), \delta\}$. Hence, $\chi \succeq (\psi \circ \sigma) \Rightarrow (\chi \succeq \psi)\ and\ (\chi \succeq \sigma)$. Also, $\chi \succeq \delta$.
We conclude that $\chi \succeq \sigma$, $\chi \succeq \delta$ and $\chi \succeq \psi$. The first two conclusion implies that $\chi \succeq (\sigma \vee \delta)$. Since ψ and $\sigma \vee \delta$ transform disjoint sets, $\chi \succeq (\sigma \vee \delta)$ and $\chi \succeq \psi$ implies $\chi \succeq ((\sigma \vee \delta) \circ \psi)$. Hence, we have shown that any upper bound χ for $\{\phi_1, \phi_2\}$ is greater than ω which itself is an upper bound. Hence, ω is the lub.
Case IV $\phi_1 = \sigma_1 \circ \sigma_2 \circ ... \circ \sigma_n$ and $\phi_2 = \delta$. δ in conflict with more than one σ_i.
Let us begin with computing $\chi = \delta \vee \sigma_1$. From Lemma 5, we can observe that there can be atmost one basic operator ξ in χ which transforms attributes of δ only. Other basic operators in χ contains attributes of σ_1. Hence, there is atmost one operator in χ which can conflict with σ_2 as it can conflict with operator containing attributes of δ only. Using case III, there exists lub $\chi \vee \sigma_2$. We can continue to get $\omega = ((...(\delta \vee \sigma_1) \vee \sigma_2)...\sigma_n)$. Now, we prove that ω is the lub of $\{\delta, \sigma_1 \circ ... \circ \sigma_n\}$.Let there be some upper bound χ of $\{\delta, (\sigma_1 \circ ... \circ \sigma_n)\}$. Then, we have $\chi \succeq (\sigma_1 \circ ... \circ \sigma_n)$ which implies $\chi \succeq \sigma_1, \chi \succeq \sigma_2$ and so on. We have-

$$\begin{aligned} \chi &\succeq \delta \\ &\succeq \delta \vee \sigma_1 \ \ (\because \chi \succeq \sigma_1) \\ &\succeq (\delta \vee \sigma_1) \vee \sigma_2 \ \ (\because \chi \succeq \sigma_2) \\ &\succeq (...(\delta \vee \sigma_1) \vee \sigma_2 ... \vee \sigma_n) = \omega \end{aligned}$$

Thus, any upper bound χ is greater than ω. Also, ω can be shown to be upper bound. Hence, ω is the least upper bound.
Case V$\phi_1 = \sigma_1 \circ \sigma_2 \circ .... \circ \sigma_n$ and $\phi_2 = \delta_1 \circ \delta_2 \circ .... \circ \delta_m$.
Using case IV, we can compute $(\phi_1 \vee \delta_1)$. Similarly, we can compute $((\phi_1 \vee \delta_1) \vee \delta_2)$. We can continue to get $\omega = (..(\phi_1 \vee \delta_1) \vee \delta_2) ... \vee \delta_m)$. Using similar steps as in case IV, we can prove that any upper bound χ of $\{\phi_1, \phi_2\}$ is also an upper bound of ω. Also, we can show that ω itself is upper bound of ϕ_1 and ϕ_2. Thus, ω is the lub. Hence, we proved that there exists a least upper bound in Φ for ϕ_1 and ϕ_2 in Φ. In addition, the proof outlines the way it can be obtained. □