A Semantic Hierarchy for Erasure Policies*

Filippo Del Tedesco[1], Sebastian Hunt[2], and David Sands[1]

[1] Chalmers University of Technology, Sweden
[2] City University London

Abstract. We consider the problem of logical data erasure, contrasting with physical erasure in the same way that end-to-end information flow control contrasts with access control. We present a semantic hierarchy for erasure policies, using a possibilistic knowledge-based semantics to define policy satisfaction such that there is an intuitively clear upper bound on what information an erasure policy permits to be retained. Our hierarchy allows a rich class of erasure policies to be expressed, taking account of the power of the attacker, how much information may be retained, and under what conditions it may be retained. While our main aim is to specify erasure policies, the semantic framework allows quite general information-flow policies to be formulated for a variety of semantic notions of secrecy.

1 Introduction

Erasing data can be difficult for many reasons. As an example, recent research on SSD-drives has shown that the low-level routines for erasing data often inadvertently leave data behind [26]. This is due to the fact that information on an SSD (in contrast to a more conventional magnetic hard drive) gets copied to various parts of memory in order to even out wear. The naive firmware sanitization routines do not have access to the movement-history of data, and so leave potentially large amounts of data behind.

This paper is not focused on low-level erasure per se. The requirement that data is used but not retained is commonplace in many non hardware-specific scenarios. As an everyday example consider the credit card details provided by a user to a payment system. The expectation is that card details will be used to authorize payment, but will not be retained by the system once the transaction is complete.

An erasure policy describes such a limited use of a piece of data. But what does it mean for a system to correctly erase some piece of data? One natural approach taken here is to view erasure as an information-flow concept – following [7]. To erase something means that after the point of erasure there is no information flowing from the original data to observers of the system. This gives a natural generalization of the low-level concept of *physical erasure* to what one might call *logical erasure*. Logical erasure specifies that a system behaves *as if* it has physically erased some data *from the viewpoint of a particular observer*. The observer viewpoint is more than just a way to model erasure in a multi-level security context (as in [7]). To understand the importance of the attacker viewpoint, consider a system which receives some data subject to an erasure policy. The system then receives a random key from a one-time pad and XORs it with the secret. The key is then overwritten with a constant. Does such a system erase the

* For extended version with proofs visit: `tinyurl.com/4ysm5k9`

data? The answer, from an information-flow perspective, depends on what the observer (a.k.a. *the attacker*) can see/remember about the execution. An attacker who can see the exact final state of the system (including the encrypted data) and nothing more, cannot deduce anything about the subject data, and so we can conclude that it is erased for that attacker. But if the attacker could also observe the key that was provided, then the system is not erasing. Different situations may need to model different attacker powers.

In practice the concept of erasure is a subtle one in which many dimensions play a role. This is analogous to the various "dimensions" of declassification [25]. In this paper we develop a semantic model for erasure which can account for different *amounts* of erasure, covering the situation where some but not necessarily all information about the subject is removed, and different varieties of *conditional erasure*, which describes both what is erased, and under what conditions.

The contribution of this work is to identify (Section 2) and formalise (Section 4) a hierarchy of increasingly expressive erasure policies which captures various dimensions of erasure. To do this we build on a new possibilistic information-flow model (Section 3) which is parameterised by (i) the *subject* of the information flow policy (e.g. the data to be erased), (ii) the attacker's observational power. This is done taking into account the *facts* that an attacker might be interested to learn, and the *queries* which he can or will be able to answer about the subject.

2 Erasure case studies

We consider a series of examples of erasing systems which differ according to the way they answer the following questions:

1. *How much* of the erasure subject is erased?
2. *Under which conditions* is erasure performed?

In Section 4 we revisit these examples, showing how the erasure behaviour of each can be captured by a particular type of erasure policy.

2.1 Total erasure

Consider a ticket vending machine using credit cards as the payment method. A partial implementation, in simplified form, is shown in Listing 1.1. Line 1 inputs the card number; line 2 executes the payment transaction; line 3 writes the transaction time to a log for audit purposes; line 4 deletes the card number.

```
1 get(cc_number);
2 charge(ticket_cost,cc_number);
3 log(current_time());
4 cc_number=null;
```

Listing 1.1. Ticket vending machine, total and unconditional erasure

This is an example of an erasing program: once line 4 is executed, the card number has been erased from the system. This statement can be refined further with respect to our original questions: 1) the system is *totally* erasing (no information about the card number is retained) and 2) erasure occurs *unconditionally*, since control flow always reaches line 4.

2.2 Partial erasure

Consider a variant of the vending machine (Listing 1.2) which logs the last four digits of the card number of each transaction, enabling future confirmation of transactions in response to user queries. The difference to Listing 1.1 is in line 3, where additional

```
1 get(cc_number);
2 charge(ticket_cost,cc_number);
3 log(current_time(),last4(cc_number));
4 cc_number=null;
```

Listing 1.2. Ticket vending machine, partial and unconditional erasure

data is written to the log. With this change, line 4 no longer results in total erasure since, even after cc_number is overwritten, the last four digits of the card number are retained in the log.

2.3 Low dependent erasure

Consider a further elaboration of the vending machine example (Listing 1.3) which allows the user to *choose* whether the last four digits are retained. In line 3 the program acquires the user choice, then it either proceeds as Listing 1.2 or as Listing 1.1, according to the choice. Now the question about how much information is erased has two different answers, depending on the second user input. Since this dependency is not related to the erasure subject itself, we call this *low dependent* erasure.

```
get(cc_number);                                          1
charge(ticket_cost,cc_number);                           2
get(choice);                                             3
if choice="Allow"                                        4
 then log(current_time(),last4(cc_number));              5
 else log(current_time());                               6
cc_number=null;                                          7
```

Listing 1.3. Ticket vending machine, low dependent erasure

2.4 High dependent erasure

Suppose there is a brand of credit cards, StealthCard, which only allows terminals enforcing a strict confidentiality policy to be connected to their network. This requires a further refinement of the program (Listing 1.4), since StealthCard users are not permitted a choice for the logging option. At line 3 the credit card number is inspected and, if it is a StealthCard, the system proceeds like 1.1.

```
get(cc_number);                                          1
charge(ticket_cost,cc_number);                           2
if (cc_number is in StealthCard)                         3
  then log(current_time());                              4
  else get(choice);                                      5
    if choice="Allow"                                    6
      then log(current_time(),                           7
               last4(cc_number));                        8
      else log(current_time())                           9
cc_number=null;                                          10
```

Listing 1.4. Ticket vending machine, high dependent erasure

Compared to the previous case, this example has an additional layer of dependency, since the amount of data to be erased is itself dependent on the erasure subject. We refer to this as *high dependent* erasure.

3 An abstract model of information flow

We formalise erasure policies as a particular class of information flow policies. In this section we define the basic building blocks for describing such policies. We consider trace-based (possibilistic) models of system behaviour and we interpret information flow policies over these models. We make the standard conservative assumption that the attacker has perfect knowledge of the system model.

3.1 Trace models

The behavioural "atom" in our framework is the *event* (in our examples this will typically be an input ($?v$) or output ($!v$) but internal computation steps can be modelled in the same way). Traces, ranged over by s, t, s_1, t_1, etc, are finite or countably infinite sequences of events. We write $t.e$ for the trace t extended with event e and we write $s.t$ for the concatenation of traces s and t. In what follows we assume given some set T of traces.

A *system* is considered to be a set $S \subseteq T$ (the assumption is that S is the set of maximal traces of the system being modeled). Certain parts of system behaviour will be identified as the *subject* of our policies and we define these parts by a function $\Phi : T \to D$, for some set D (typically, Φ will be a projection on traces). For a confidentiality property the subject might represent the secret that we are trying to protect (an input or a behaviour of a classified agent). For erasure the subject will be the input which is to be erased.

Given a system S, we denote by $\Phi(S)$ the subset of D relevant for S:

$$\Phi(S) = \{\Phi(t) | t \in S\}$$

We call this the *subject domain* of S. Let $\mathrm{Sys}(V)$ be the set of all systems with subject domain V. Our flow policies will be specific to systems with a given subject domain.

3.2 Equivalence relations and partitions

The essential component of a flow policy is a *visibility* policy which specifies how much an attacker should be allowed to learn about the subject of a system by observing its behaviour. Following a standard approach in the information flow literature – see, for example [18, 24] – we use equivalence relations for this purpose. A flow policy for systems in $\mathrm{Sys}(V)$ is $R \in \mathrm{ER}(V)$, where $\mathrm{ER}(V)$ denotes the set of all equivalence relations on V. The intention is that attackers should not be able to distinguish between subjects which are equivalent according to R. An example is the "have the same last four digits" relation, specifying that the most an attacker should be allowed to learn is the last four digits of the credit card number (put another way, all cards with the same last four digits should look the same to the attacker).

In what follows we make extensive use of two key, well known facts about equivalence relations:

- The set of equivalence relations on V, ordered by inclusion of their defining sets of pairs, forms a complete lattice, with the identity relation (which we denote Id_V) as the bottom element, and the total relation (which we denote All_V) as the top.
- The set of equivalence relations on V is in one-one correspondence with the set of *partitions* of V, where each of the disjoint subsets making up a partition is an equivalence class of the corresponding equivalence relation. We write $\mathrm{PT}(V)$ for the set of all partitions of V. Given $P \in \mathrm{PT}(V)$, we write $\mathcal{E}(P)$ for the corresponding equivalence relation: $v_1 \ \mathcal{E}(P) \ v_2$ iff $\exists X \in P. v_1, v_2 \in X$. In the other direction, given $R \in \mathrm{ER}(V)$ and $v \in V$ we write $[v]_R$ for the R-equivalence class of v: $[v]_R = \{v' \in V | v' \ R \ v\}$. We write $[R]$ for the partition corresponding to R: $[R] = \{[v]_R | v \in V\}$.

In the current context, the significance of $R_1 \subseteq R_2$ is that R_1 is more discriminating - ie has smaller equivalence classes - than R_2. Hence, as visibility policies, R_1 is more permissive than R_2. The lattice operation of interest on $\mathrm{ER}(V)$ is *meet*, which is given by set intersection. Given a family of equivalence relations $\{R_i\}_{i \in I}$, we write their meet as $\bigwedge_{i \in I} R_i$ (the least permissive equivalence relation which is nonetheless more permissive than each R_i).

The order relation on partitions corresponding to subset inclusion on equivalence relations will be written $\preceq_{\mathrm{ER}}$, thus $[R_1] \preceq_{\mathrm{ER}} [R_2]$ iff $R_1 \subseteq R_2$. We overload the above notation for meets of partitions in this isomorphic lattice: $\bigwedge_{i \in I} P_i = [\bigwedge_{i \in I} \mathcal{E}(P_i)]$.

3.3 Attacker models and K-spaces

As discussed in the introduction, whether or not a system satisfies a policy will depend on what is observable to the attacker. We specify an *attacker model* as an equivalence relation on traces, $A \in \mathrm{ER}(T)$. Note that this is a passive notion of attacker - attackers can observe but not interact with the system.

To compare what the attacker actually learns about the subject with what the visibility policy permits, we define, for each attacker observation $O \in [A]$, the corresponding *knowledge set* $\mathrm{K}_S(O) \subseteq V$, which is the set of possible subject values which the attacker can deduce from making a given observation[3]: $\mathrm{K}_S(O) = \{\Phi(t) | t \in O \cap S\}$.

The *K-space of A for S*, denoted $\mathcal{K}_S(A)$, is the collection of all the attacker's possible (ie non-empty) knowledge sets when observing S:

$$\mathcal{K}_S(A) = \{\mathrm{K}_S(O) | O \in [A], O \cap S \neq \emptyset\}$$

Lemma 1. *Let $S \in Sys(V)$ and $A \in ER(V)$. Then the K-space of A for S* covers V*, by which we mean that every member of $\mathcal{K}_S(A)$ is non-empty and $\bigcup \mathcal{K}_S(A) = V$.*

From now on, for a given V, we use the term K-space to mean any collection of sets which covers V.

In the special case that a system's behaviour is a *function* of the subject, each K-space will actually define an equivalence relation on V:

Proposition 1. *Say that $S \in Sys(V)$ is* functional *just when, for all $t, t' \in S$, $t \neq t' \Rightarrow \Phi(t) \neq \Phi(t')$. In this case, for all $A \in ER(T)$, $\mathcal{K}_S(A)$ partitions V.*

When S is functional, the K-space $\mathcal{K}_S(A)$, being a partition, can be interpreted as the equivalence relation $\mathcal{E}(\mathcal{K}_S(A))$. So, in the functional case there is a straightforward way to compare a visibility policy with an attacker's K-space: we say that the policy R is satisfied just when R is more discriminating than this induced equivalence relation. Formally, when S is functional, *S satisfies R for attacker A*, written $S \vdash_A R$, just when $R \subseteq \mathcal{E}(\mathcal{K}_S(A))$ or, equivalently:

$$S \vdash_A R \text{ iff } [R] \preceq_{\mathrm{ER}} \mathcal{K}_S(A) \tag{1}$$

We now consider how to extend this definition to the general case, in which a system has other inputs apart from the policy subject.

[3] A more reasonable but less conventional terminology would be to call this an *uncertainty set*.

3.4 Comparing K-Spaces: facts and queries

In general, a system's behaviour may depend on events which are neither part of the policy subject nor visible to the attacker. In this case, the attacker's knowledge of the subject need not be deterministic, resulting in a K-space which is not a partition. This raises the question: when is one K-space "more discriminating" than another?

Here we motivate a variety of orderings by considering some basic modes in which an attacker can use observations to make deductions about the subject of a system:

Facts A fact F is just a set of values. A given knowledge set X *confirms fact* F just when $X \subseteq F$. Dually, X *has uncertainty* F when $F \subseteq X$. For example a fact of interest (to an attacker) might be the set of "Platinum" card numbers. In this case an observation might confirm to the attacker that a card is a Platinum card by also revealing exactly which platinum card it is. For a given K-space K we then say that
- K *can confirm* F if there exists some $X \in K$ such that X confirms F.
- K *can have uncertainty* F if there exists some $X \in K$ such that X has uncertainty F.

Queries A query Q is also just a set of values. We say that a given knowledge set X *answers query* Q just when either $X \subseteq Q$ or $X \subseteq V \setminus Q$. For a given K-space K we then say that
- K *will answer* Q if for all $X \in K$, X answers Q, and
- K *can answer* Q if there exists some $X \in K$ such that X answers Q.

In a possibilistic setting it is natural to focus on those "secrets" which it is *impossible* for a given system to reveal, where revealing a secret could mean either confirming a fact or answering a query. Two of the four K-space properties defined above have an immediate significance for this notion of secrecy:

- Say that S *keeps fact* F *secret from attacker* A iff there are no runs of S for which A's observation confirms F, ie iff: $\neg(\mathcal{K}_S(A)$ can confirm $F)$.
- Say that S *keeps query* Q *secret from attacker* A iff there are no runs of S for which A's observation answers Q, ie iff: $\neg(\mathcal{K}_S(A)$ can answer $Q)$.

The possibilistic secrecy significance of "has uncertainty" and "will answer" is not so clear. However, as we will show, we are able to define flow policies and a parameterized notion of policy satisfaction which behaves well with respect to all four properties.

Using the ability of a K-space to confirm facts and answer queries, we can order systems in different ways, where a "smaller" K-space (ie one lower down in the ordering) allows the attacker to make more deductions (and so the system may be regarded as less secure). Define the following orderings between K-spaces:

Upper: $K_1 \preceq_{\mathrm{U}} K_2$ iff $\forall F.K_2$ can confirm $F \Rightarrow K_1$ can confirm F. Note that $K_1 \preceq_{\mathrm{U}} K_2$ iff K_2 keeps more facts secret than K_1.
Lower: $K_1 \preceq_{\mathrm{L}} K_2$ iff $\forall F.K_1$ can have uncertainty $F \Rightarrow K_2$ can have uncertainty F.
Convex (Egli-Milner): $K_1 \preceq_{\mathrm{EM}} K_2$ iff $K_1 \preceq_{\mathrm{U}} K_2 \wedge K_1 \preceq_{\mathrm{L}} K_2$.
Can-Answer: $K_1 \preceq_{\mathrm{CA}} K_2$ iff $\forall Q.K_2$ can answer $Q \Rightarrow K_1$ can answer Q. Note that $K_1 \preceq_{\mathrm{CA}} K_2$ iff K_2 keeps more queries secret than K_1.
Will-Answer: $K_1 \preceq_{\mathrm{WA}} K_2$ iff $\forall Q.K_2$ will answer $Q \Rightarrow K_1$ will answer Q.

It is straightforward to verify that these orders are reflexive and transitive, but not anti-symmetric. The choice of names for the upper and lower orders is due to their correspondence with the powerdomain orderings [22]:

Proposition 2. $K_1 \preceq_U K_2$ *iff* $\forall X_2 \in K_2.\exists X_1 \in K_1.X_1 \subseteq X_2$
$K_1 \preceq_L K_2$ *iff* $\forall X_1 \in K_1.\exists X_2 \in K_2.X_1 \subseteq X_2$

We can compare the K-space orders 1) unconditionally, 2) as in the case of policy satisfaction, when we are comparing a partition with a K-space, and, 3) when the K-spaces are both partitions, yielding the following results:

Proposition 3. *1.* $\preceq_{EM} \subsetneq \preceq_L \subsetneq \preceq_{WA}$ *and* $\preceq_{EM} \subsetneq \preceq_U \subsetneq \preceq_{CA}$.
2. *Additionally, when* P *is a partition:* $P \preceq_{CA} K \Rightarrow P \preceq_{WA} K$ *(the reverse implication does not hold in general).*
3. $\preceq_{ER}$, $\preceq_{EM}$, $\preceq_L$, *and* $\preceq_{WA}$ *all coincide on partitions. Furthermore, when* P_1 *and* P_2 *are partitions:* $P_1 \preceq_{ER} P_2 \Rightarrow P_1 \preceq_U P_2 \Rightarrow P_1 \preceq_{CA} P_2$ *(the reverse implications do not hold in general).*

These orderings give us a variety of ways to extend the definition of policy satisfaction from functional systems (Equation 1) to the general case. The choice will depend on the type of security condition (eg protection of facts versus protection of queries) which we wish to impose.

4 The policy hierarchy

We specify a three-level hierarchy of erasure policy types. All three types of policy use a structured collection of equivalence relations on the subject domain to define what information should be erased. A key design principle is that, whenever a policy permits part of the erasure subject to be retained, this should be *explicit*, by which we mean that it should be captured by the conjunction of the component equivalence relations.

For each type of policy, we define a satisfaction relation, parameterized by a choice of K-space ordering $o \in \{U, L, EM, CA, WA\}$.

Assume a fixed policy subject function $\Phi : T \to D$. Given a subset $V \subseteq D$, let $T_V = \{t \in T | \Phi(t) \in V\}$. Note that if S belongs to $\mathrm{Sys}(V)$ then $S \subseteq T_V$.

Type 0 policies

Type 0 policies allow us to specify *unconditional* erasure, corresponding to the two examples shown in Section 2 in Listings 1.1 and 1.2.

A Type 0 erasure policy is just a visibility policy. We write Type-$0(V)$ for the set of all Type 0 policies for systems in $\mathrm{Sys}(V)$ (thus Type-$0(V) = \mathrm{ER}(V)$). The definition of satisfaction for a given attacker model A and system S uses a K-space ordering (specified by parameter o) to generalise the satisfaction relation of Equation 1 to arbitrary (ie not-necessarily functional) systems:

$$S \vdash^o_A R \text{ iff } [R] \preceq_{\mathrm{o}} \mathcal{K}_S(A)$$

For functional systems note that, by Proposition 3, choosing o to be any one of EM, L or WA yields a notion of satisfaction equivalent to Equation 1, while U and CA yield strictly weaker notions.

Example. Consider the example in Listing 1.2. The subject domain is CC, the set of all credit card numbers, and (since the erasure subject is the initial input) the subject function is the first projection on traces. The policy we have in mind for this system is that it should erase all but the last four digits of the credit card number. We extend this example so that it uses a method call `erased()` to generate an explicit output event η (signalling that erasure should have taken place) followed by a dump of the program memory (thus revealing all retained information to a sufficiently strong attacker).

```
1 get(cc_number);
2 charge(ticket_cost,cc_number);
3 log(current_time(),last4(cc_number));
4 cc_number=null;
5 erased();
6 dump();
```

Listing 1.5. Ticket vending machine, partial and unconditional erasure: extended

If we restrict attention to systems (such as this one) where each run starts by inputting a credit card number and eventually outputs the erasure signal exactly once, we can assume a universe of traces T such that all $t \in T$ have the form $t = ?cc.s.\eta.s'$, where s, s' are sequences not including η. Let S be the trace model for the above system. The required visibility policy is the equivalence relation L4 $\in$ ER(CC) which equates any two credit card numbers sharing the same last four digits. An appropriate attacker model is the attacker who sees nothing before the erasure event and everything afterwards. Call this the *simple erasure attacker*, denoted AS:

$$\text{AS} = \{(t_1, t_2) \in T \times T | \exists s_1, s_2, s_3.\ t_1 = s_1.\eta.s_3\ \wedge\ t_2 = s_2.\eta.s_3\}$$

Informally, it should be clear that, for each run of the system, AS will learn the last four digits of the credit card which was input, together with some other log data (the transaction time) which is independent of the card number. Thus the knowledge set on a run, for example, where the card number ends 7016, would be the set of all card numbers ending 7016. The K-space in this example will actually be exactly the partition [L4], hence S does indeed satisfy the specified policy: $S \vdash^{o}_{\text{AS}}$ L4 for all choices of o. From now on, we write just $S \vdash_A R$ to mean that it holds for all choices of ordering (or, equivalently, we can consider $\vdash_A$ to be shorthand for $\vdash^{EM}_A$, since EM is the strongest ordering).

Type 1 policies

Type 1 policies allow us to specify "low dependent" erasure (Section 2, Listing 1.3), where different amounts may be erased on different runs, but where the erasure condition is independent of the erasure subject itself.

For systems in Sys(V) the erasure condition is specified as a partition $P \in \text{PT}(T_V)$. This is paired with a function $f : P \to \text{Type-0}(V)$, which associates a Type 0 policy with each element of the partition. Since the domain of f is determined by the choice of P, we use a dependent type notation to specify the set of all Type 1 policies:

$$\text{Type-1}(V) = \langle P : \text{PT}(T_V), P \to \text{ER}(V) \rangle$$

Because we want to allow only low dependency - ie the erasure condition must be independent of the erasure subject - we require that P is *total for* V, by which we mean:

$$\forall X \in P.\Phi(X) = V$$

This means that knowing the value of the condition will not in itself rule out any possible subject values. To define policy satisfaction we use the components $X \in P$ to partition a system S into disjoint sub-systems $S \cap X$ and check both that each sub-system is defined over the whole subject domain V (again, to ensure low dependency) and that it satisfies the Type 0 policy for sub-domain X. So, for a Type 1 policy $\langle P, f \rangle \in \text{Type-1}(V)$, an attacker model A, and system $S \in \text{Sys}(V)$, satisfaction is defined thus:

$$S \vdash^o_A \langle P, f \rangle \text{ iff } \forall X \in P. S_X \in \text{Sys}(V) \wedge S_X \vdash^o_A f\ X$$

where $S_X = S \cap X$.

Example. Consider the example of Listing 1.3 extended with an erasure signal followed by a memory dump (as in our discussion of Type 0 policies above). Let S be the system model for the extended program. We specify a conditional erasure policy where the condition depends solely on the user choice. The erasure condition can be formalised as the partition $\text{Ch} \in \text{PT}(T)$ with two parts, one for traces where the user answers "Allow" (which we abbreviate to a) and one for traces where he doesn't: $\text{Ch} = \{Y, \overline{Y}\}$, where $Y = \{t \in T | \exists s, s_1, s_2.\ t = s.?a.s_1.\eta.s_2\}$ and $\overline{Y} = T \setminus Y$. For runs falling in the Y component, the intended visibility policy is L4, as in the Type 0 example above. For all other runs, the intended policy is All_{CC}, specifying complete erasure. The Type 1 policy is thus $\langle \text{Ch}, g \rangle$ where $g : \text{Ch} \to \text{ER}(\text{CC})$ is given by:

$$g(X) = \begin{cases} \text{L4} & \text{if } X = Y \\ \text{All} & \text{if } X = \overline{Y} \end{cases}$$

Intersecting Y and $\overline{Y}$, respectively, with the system model S gives disjoint sub-systems S_Y (all the runs in which the user enters "Allow" to permit retention of the last four digits) and $S_{\overline{Y}}$ (all the other runs). Since the user's erasure choice is input independently of the card number, it is easy to see that both sub-systems are in $\text{Sys}(\text{CC})$, that $S_Y \vdash_{\text{AS}} \text{L4}$, and $S_{\overline{Y}} \vdash_{\text{AS}} \text{All}$. Thus $S \vdash_{\text{AS}} \langle \text{Ch}, g \rangle$.

The following theorem establishes that our "explicitness" design principle is realised by Type 1 policies:

Theorem 1. *Let* $\langle P, f \rangle \in$ *Type-*$1(V)$ *and* $S \in$ *Sys*(V) *and* $A \in$ *ER*(T). *Let* $o \in \{U, L, EM, CA, WA\}$. *If* $S \vdash^o_A \langle P, f \rangle$ *then:*

$$[\bigwedge_{X \in P} (f\ X)] \preceq_o \mathcal{K}_S(A)$$

Type 2 policies

Type 2 policies are the most flexible policies we consider, allowing dependency on both the erasure subject and other properties of a run.

Recall the motivating example from Section 2 (Listing 1.4) in which credit card numbers in a particular set StealthCard $\subseteq$ CC are always erased, while the user is given some choice for other card numbers. In this example, the dependency of the policy on the erasure subject can be modelled by the partition $\text{HC} = \{\text{StealthCard}, \overline{\text{StealthCard}}\}$. For each of these two cases, we can specify sub-policies which apply only to card numbers in the corresponding subsets. Since these sub-policies do not involve any further

dependence on the erasure subject, they can both be formulated as Type 1 policies for their respective sub-domains. In general then, we define the Type 2 policies as follows:

$$\text{Type-2}(V) = \langle Q : \text{PT}(V), W : Q \to \text{Type-1}(W) \rangle$$

To define satisfaction for Type 2 policies, we use the components $W \in Q$ to partition a system S into sub-systems (unlike the analogous situation with Type 1 policies, we cannot intersect S directly with W; instead, we intersect with T_W). To ensure that the *only* dependency on the erasure subject is that described by Q, we require that each sub-system $S \cap T_W$ is defined over the whole of the subject sub-domain W. So, for a Type 2 policy $\langle Q, g \rangle \in \text{Type-2}(V)$, an attacker model A, and system $S \in \text{Sys}(V)$, satisfaction is defined thus:

$$S \vdash^o_A \langle Q, g \rangle \text{ iff } \forall W \in Q . S_W \in \text{Sys}(W) \wedge S_W \vdash^o_A g\ W$$

where $S_W = S \cap T_W$.

To state the appropriate analogue of Theorem 1 we need to form a conjunction of all the component parts of a Type 2 policy:

- In the worst case, the attacker will be able to observe which of the erasure cases specified by Q contains the subject, hence we should conjoin the corresponding equivalence relation $\mathcal{E}(Q)$.
- Each Type 1 sub-policy determines a worst case equivalence relation, as defined in Theorem 1. To conjoin these relations, we must first extend each one from its sub-domain to the whole domain, by appending a single additional equivalence class comprising all the "missing" elements: given $W \subseteq V$ and $R \in \text{ER}(W)$, define $R^\dagger \in \text{ER}(V)$ by $R^\dagger = R \cup \text{All}_{V \setminus W}$.

Theorem 2. *Let $\langle Q, g \rangle \in$ Type-$2(V)$ and $S \in$ Sys(V) and $A \in$ ER(T). For any Type 1 policy $\langle P, f \rangle$, let $R_{\langle P,f \rangle} = \bigwedge_{X \in P}(f\ X)$. Let $o \in \{U, L, EM, CA, WA\}$. If $S \vdash^o_A \langle Q, g \rangle$ then:*

$$[\ \mathcal{E}(Q) \wedge \bigwedge_{W \in Q} R^\dagger_{(g\ W)}] \preceq_o \mathcal{K}_S(A)$$

4.1 Varying the attacker model

The hierarchy deals with erasure policies independently of any particular attacker model. Here we make some brief remarks about modelling attackers. Let us take the example of the erasure notion studied in [16] where the systems are simple imperative programs involving IO on public and secret channels. Then the *implicit* attacker model in that work is unable to observe any IO events prior to the erasure point, and is able to observe just the public inputs and outputs thereafter. (We note that [16] also considers a policy *enforcement* mechanism which uses a stronger, state-based non-interference property.)

```
1 get(data);
2 get(key);
3 data := data XOR key
4 key := null;
5 erased();
6 output(data);
```

Listing 1.6. Key Erasure

Now consider the example of the one-time pad described in the introduction, codified in Listing 1.6. Let system S be the set of traces modelling the possible runs of the program and let the subject be the first input in each trace. For the simple erasure attacker AS (Section 4), unable to observe the key provided in line 2, the

K-space will be $\{V\} = [\text{All}]$, hence $S \vdash_{\text{AS}} \text{All}$. This is because the value of data in the output does not inform the attacker about the initial value. On the other hand, the attacker who can also observe the key learns everything about the data from its encrypted value.[4] So for this stronger attacker, using encryption to achieve erasure does not work, and indeed policy satisfaction fails for this particular system.

If the attacker is strengthened even further, we arrive at a point where *no* system will be able to satisfy the policy. Intuitively, if an attacker can see the erasure subject itself (or, more specifically, more of the erasure subject than the policy permits to be retained) no system will be able to satisfy the policy. In general, we say that a policy p with subject domain V (where p may be of any of Types 0,1,2) is *weakly o-compatible* with attacker model A iff there exists $S \in \text{Sys}(V)$ such that $S \vdash^o_A p$ (we call this weak compatibility because it assumes that all $S \in \text{Sys}(V)$ are of interest but in general there will be additional constraints on the admissible systems). Clearly, to be helpful as a sanity check on policies we need something a little more constructive than this. For the special case of Type 0 policies and the upper ordering we have the following characterisation:

Lemma 2. *R is weakly U-compatible with A iff* $\forall v \in V. \exists O \in [A]. [v]_R \subseteq \Phi(O)$.

Deriving analogues of this result (or at least sufficient conditions) of more general applicability remains a subject for further work.

Finally, we note that, while our main aim has been to specify erasure policies, by varying the attacker model appropriately, we can specify quite general information-flow properties, not just erasure policies. For example, by classifying events into High and Low and defining the attacker who sees only Low events, we can specify non-interference properties.

5 Related work

We consider related work both directly concerned with erasure and more generally with knowledge based approaches to information flow policies.

Erasure The information-flow perspective on erasure was introduced by Chong and Myers [7] and was studied in combination with confidentiality and declassification. Their semantics is based on an adaptation of two-run noninterference definitions, and does not have a clear attacker model. They describe conditional erasure policies where the condition is independent of the data to be erased. Although this appears similar to Type 1 policies (restricted to total erasure), it is more accurately viewed as a form of Type 0 policy in which the condition defines the point in the trace from which the attacker begins observation.

The present paper does not model the behaviour of the user who interacts with an erasing system. This was studied in [14] for one particular system and attacker model. We believe that it would be possible to extend the system model with a user-strategy parameter (see [27, 21, 21] which consider explicit models of user strategies). Neither do we consider here the verification or enforcement of erasure policies; for specific

[4] Note, however, that we cannot model the fact that certain functions are not (easily) invertible, so our attackers are always endowed with unbounded computational power.

systems and attacker models this has been studied in a programming language context in [16, 8, 9, 13, 20].

Knowledge based approaches Our use of knowledge sets was inspired by Askarov and Sabelfeld's *gradual release* definitions [2]. This provides a clear attacker-oriented perspective on information-flow properties based on what an attacker can deduce about a secret after making observations. A number of recent papers have followed this approach to provide semantics for richer information flow properties, e.g. [4, 5]. Our use of knowledge sets to build a K-space, thus generalising the use of equivalence relations/partitions, is new. The use of partitions in expressing a variety of information flow properties was studied in early work by Cohen [10]. The use of equivalence relations and more generally partial equivalence relations as models for information and information flow was studied in [18] and resp. [24].

Recent work [3] uses an epistemic temporal logic as a specification language for information flow policies. Formulae are interpreted over trace-based models of programs in a simple sequential while language (without input actions), together with an explicit observer defined via an observation function on traces. Our work looks very similar in spirit to [3], though this requires further investigation, and it appears that our modeling capabilities are comparable. The use of temporal logic in [3] is attractive, for example because of the possibility of using off the shelf model-checking tools. However, our policy language allows a more intuitive reading and clear representation of the information leakage.

Alur *et al* [1], study preservation of secrecy under refinement. The information flow model of that work bears a number of similarities with the present work. Differences include a more concrete treatment of traces, and a more abstract treatment of secrets. As here, equivalence relations are used to model an attacker's observational power, while knowledge models the ability of an attacker to determine the value of trace predicates. Their core definition of secrecy coincides with what we call secrecy of queries (viz, negation of "can answer"), although they do not consider counterparts to our other knowledge-based properties.

Abstract Non-Interference Abstract Non-Interference [15] has strong similarities with our use of K-spaces. In abstract non-interference, *upper closure operators* (uco's) are used to specify non-interference properties. The similarities with the current work become apparent when a uco is presented as a *Moore family*, which may be seen as a K-space closed under intersection.

[15] starts by defining the intuitive notion of *narrow* abstract non-interference (NANI) parameterized by two upper closure operators η (specifying what the attacker can observe of low inputs) and ρ (ditto low outputs). A weakness of NANI is that it suffers from "deceptive flows", whereby a program failing to satisfy NANI might still be non-interfering. From our perspective, the deceptive flows problem arises because η fails to distinguish between what an attacker can *observe* of low inputs and what he should be allowed to *deduce* about them (ie, everything). Since we specify the attacker model independently from the flow policy, the deceptive flows problem does not arise for us.

The deceptive flows problem is addressed in [15] by defining a more general notion of abstract non-interference (ANI) which introduces a third uco parameter ϕ. The definition of ANI adapts that of NANI by lifting the semantics of a program to an abstract

version in which low inputs are abstracted by η and high inputs by ϕ. A potential criticism of this approach is that an intuitive reading is not clear, since it is based on an abstraction of the original program semantics. On the other hand, being based on Abstract Interpretation [12, 11], abstract non-interference has the potential to leverage very well developed theory and static analysis algorithms for policy checking and enforcement. It would therefore be useful to explore the connections further and to attempt an analysis of the ANI definitions (see also additional variants in [19]) relating them to more intuitive properties based on knowledge sets. A starting point could be [17] which provides an alternative characterisation of NANI using equivalence relations.

Provenance A recent abstract model of *information provenance* [6] is built on an information-flow foundation and has a number of similarities with our model, including a focus on an observer model as an equivalence relation, and a knowledge-based approach described in terms of queries that an observer can answer. Provenance is primarily concerned with a providing *sufficient* information to answer provenance-related questions. In secrecy and erasure one is concerned with *not* providing more than a certain amount.

6 Conclusions and further work

We have presented a rich, knowledge-based abstract framework for erasure policy specification, taking into account both quantitative and conditional aspects of the problem. Our model includes an explicit representation of the attacker. The knowledge-based approach guarantees an intuitive understanding of what it means for an attacker to deduce some information about the secret, and for a policy to provide an upper bound to these deductions.

Our work so far suggests a number of possible extensions:

- Develop a logic defined on traces, both to support policy definition and to give the basis for an enforcement mechanism (as is done in [3]).
- Model multilevel erasure, based on the fact the attacker might perform observations up-to a certain level in the security lattice. It would be interesting to investigate different classes of such attackers and to analyse their properties.
- Generalise policy specifications to use K-spaces in place of equivalence relations. This would allow specification of disjunctive policies such as "reveal the key or the ciphertext, but not both". Non-ER policies may also be more appropriate for protection of facts, rather than queries, since ER's are effectively closed under complementation and so cannot reveal a fact without also revealing its negation (for example, we may be prepared to reveal "not HIV positive" to an insurance company, but not the negation of this fact).
- Other, more ambitious ideas, concern extending the scope of the approach along the following key *dimensions* (defined in the same spirit as [25]):

  What: Our model is possibilistic but it is well known that possibilistic security guarantees can be very weak when non-determinism is resolved probabilistically (see the example in Section 5 of [23]). A probabilistic approach would be more expressive and provide stronger guarantees.

When: Our policies support history-based erasure conditions but many scenarios require reasoning about the future ("erase this account in 3 weeks"). This would require a richer semantic setting in which time is modelled more explicitly.

Who: We do not explicitly model the user's behaviour but it is implicit in our possibilistic approach that the user behaves non-deterministically and, in particular, that later inputs are chosen independently of the erasure subject. Modelling user behaviour explicitly would allow us to relax this assumption (which is not realistic in all scenarios) and also to model active attackers.

References

1. Alur, R., Zdancewic, S.: Preserving secrecy under refinement. In: Proc. of the 33rd Internat. Colloq. on Automata, Languages and Programming (ICALP 06), volume 4052 of Lecture Notes in Computer Science. pp. 107–118. Springer-Verlag (2006)
2. Askarov, A., Sabelfeld, A.: Gradual release: Unifying declassification, encryption and key release policies. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy. pp. 207–221. SP '07, IEEE Computer Society, Washington, DC, USA (2007), http://dx.doi.org/10.1109/SP.2007.22
3. Balliu, M., Dam, M., Le Guernic, G.: Epistemic temporal logic for information flow security. In: ACM SIGPLAN Sixth Workshop on Programming Languages and Analysis for Security (June 2011)
4. Banerjee, A.: Expressive declassification policies and modular static enforcement. In: In Proc. IEEE Symp. on Security and Privacy. pp. 339–353 (2008)
5. Broberg, N., Sands, D.: Flow-sensitive semantics for dynamic information flow policies. In: ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security (PLAS 2009). ACM (June 15 2009)
6. Cheney, J.: A formal framework for provenance security. In: The 24th IEEE Computer Security Foundations Symposium (june 2011)
7. Chong, S., Myers, A.: Language-based information erasure. Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop pp. 241–254 (June 2005)
8. Chong, S.: Expressive and Enforceable Information Security Policies. Ph.D. thesis, Cornell University (Aug 2008)
9. Chong, S., Myers, A.C.: End-to-end enforcement of erasure and declassification. In: CSF. pp. 98–111. IEEE Computer Society (2008)
10. Cohen, E.S.: Information transmission in sequential programs. In: DeMillo, R.A., Dobkin, D.P., Jones, A.K., Lipton, R.J. (eds.) Foundations of Secure Computation, pp. 297–335. Academic Press (1978)
11. Cousot, P.: Semantic foundations of program analysis. In: Muchnick, S., Jones, N. (eds.) Program Flow Analysis: Theory and Applications, chap. 10, pp. 303–342. Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1981)
12. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. ACM Symp. on Principles of Programming Languages. pp. 238–252 (Jan 1977)
13. Del Tedesco, F., Russo, A., Sands, D.: Implementing erasure policies using taint analysis. In: Aura, T. (ed.) The 15th Nordic Conference in Secure IT Systems. LNCS, Springer Verlag (October 2010)
14. Del Tedesco, F., Sands, D.: A user model for information erasure. In: SECCO. pp. 16–30 (2009)
15. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: Parameterizing non-interference by abstract interpretation. In: Proc. ACM Symp. on Principles of Programming Languages. pp. 186–197 (Jan 2004)

16. Hunt, S., Sands, D.: Just forget it – the semantics and enforcement of information erasure. In: Programming Languages and Systems. 17th European Symposium on Programming, ESOP 2008. pp. 239–253. No. 4960 in LNCS, Springer Verlag (2008)
17. Hunt, S., Mastroeni, I.: The per model of abstract non-interference. In: Static Analysis, 12th International Symposium, SAS 2005, London, UK, September 7-9, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3672, pp. 171–185. Springer (2005)
18. Landauer, J., Redmond, T.: A lattice of information. In: Proc. IEEE Computer Security Foundations Workshop. pp. 65–70 (Jun 1993)
19. Mastroeni, I.: On the rôle of abstract non-interference in language-based security. In: APLAS. Lecture Notes in Computer Science, vol. 3780, pp. 418–433. Springer (2005)
20. Nanevski, A., Banerjee, A., Garg, D.: Verification of information flow and access control policies with dependent types. In: Proc. IEEE Symp. on Security and Privacy (2011)
21. O'Neill, K.R., Clarkson, M.R., Chong, S.: Information-flow security for interactive programs. In: CSFW '06: Proceedings of the 19th IEEE workshop on Computer Security Foundations. pp. 190–201. IEEE Computer Society, Washington, DC, USA (2006)
22. Plotkin, G.D.: A powerdomain construction. SIAM J. Comput. pp. 452–487 (1976)
23. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. In: Proc. European Symp. on Programming. LNCS, vol. 1576, pp. 40–58. Springer-Verlag (Mar 1999)
24. Sabelfeld, A., Sands, D.: A per model of secure information flow in sequential programs. Higher-Order and Symbolic Computation 14(1), 59–91 (March 2001)
25. Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. Journal of Computer Security 15(5), 517–548 (2009)
26. Wei, M.Y.C., Grupp, L.M., Spada, F.E., Swanson, S.: Reliably erasing data from flash-based solid state drives. In: 9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011. pp. 105–117. USENIX (2011), `http://www.usenix.org/events/fast11/tech/techAbstracts.html#Wei`
27. Wittbold, J.T., Johnson, D.M.: Information flow in nondeterministic systems. In: IEEE Symposium on Security and Privacy. pp. 144–161 (1990)

A Proofs

Lemma 3. *Let I be a non-empty index set. Let $\{W_i\}_{i\in I}$ be a family of non-empty sets such that $\bigcup_{i\in I} W_i = V$. Let $\{K_i\}_{i\in I}$ and $\{K'_i\}_{i\in I}$ be families of K-spaces, with each K_i, K'_i covering W_i. Then, for $o \in \{L, U, EM, CA, WA\}$:*

$$(\forall i \in I. K_i \preceq_o K'_i) \Rightarrow \bigcup_{i\in I} K_i \preceq_o \bigcup_{i\in I} K'_i$$

Proof. We show the two interesting cases, CA and WA.

- *case CA. Assume $\forall i \in I. K_i \preceq_{CA} K'_i$ and consider a query $Q \subseteq V$ such that $\bigcup_{i\in I} K'_i$ can answer Q. By definition this implies there exists a $j \in I$ such that $\exists X' \in K'_j$ and either $X' \subseteq Q$ or $X' \subseteq V \setminus Q$.*
  - *Suppose $X' \subseteq Q$, then $Q' = W_j \cap Q$ is a query K'_j can answer via X'. Since $K_j \preceq_{CA} K'_j$, K_j can answer Q' as well, therefore there must be a $X \in K_j$ such that either $X \subseteq Q'$ or $X \subseteq W_j \setminus Q'$. If $X \subseteq Q'$ then $\bigcup_{i\in I} K_i$ can answer Q via X in K_j. Otherwise $X \subseteq W_j \setminus Q'$, but this means $X \subseteq V \setminus Q$ therefore $\bigcup_{i\in I} K_i$ can answer Q via X in K_j.*
  - *Suppose $X' \subseteq V \setminus Q$, then $Q' = W_j \setminus Q$ is a query K'_j can answer via X'. For the same reason we explained previously, there must be a $X \in K_j$ such that either $X \subseteq Q'$ or $X \subseteq W_j \setminus Q'$. If $X \subseteq Q'$ then $X \subseteq V \setminus Q$ and $\bigcup_{i\in I} K_i$ can answer Q via X in K_j. Otherwise $X \subseteq W_j \setminus Q'$, but this means $X \subseteq Q$ therefore $\bigcup_{i\in I} K_i$ can answer Q via X in K_j.*
- *case WA. Assume $\forall i \in I. K_i \preceq_{WA} K'_i$ and consider a query $Q \subseteq V$ such that $\bigcup_{i\in I} K'_i$ will answer Q. By definition this implies that $\forall X' \in K'_j$, either $X' \subseteq Q$ or $X' \subseteq V \setminus Q$ for all K'_j in $\{K'_i\}_{i\in I}$. Let us consider one K'_j of the family and define $Q'_j = Q \cap W_j$. Then we have $\forall X' \in K'_j$, either $X' \subseteq Q'_j$ or $X' \subseteq W_j \setminus Q'_j$, therefore Q'_j is a query K'_j will answer. Since $K_j \preceq_{WA} K'_j$, K_j will answer Q'_j as well, therefore $X \subseteq Q'_j$ or $X \subseteq W_j \setminus Q'_j$ must hold for all $X \in K_j$. But this implies $X \subseteq Q$ or $X \subseteq V \setminus Q$ as well, and the statement holds for all K_j in $\{K_i\}_{i\in I}$, therefore $\{K_i\}_{i\in I}$ will answer Q.*

□

Lemma 4. *Let $\{P_i\}_{i\in I}$ be a non-empty family of relations in $ER(V)$ for some set V, and let $R = \bigcap_{i\in I} P_i$. Then $[R] \preceq_{EM} \bigcup_{i\in I} [P_i]$.*

Proof. Every element of $[R]$ is of the form $[v]_R$, every element of $\bigcup_{i\in I} [P_i]$ is of the form $[v]_{P_i}$ for some $i \in I$, and every choice of v and i generates such elements. It thus suffices to show that $[v]_R \subseteq [v]_{P_i}$ for all choices of v and i. This follows since R is a finer equivalence relation than each P_i. □

Lemma 5. *Let $P \in PT(T_V)$, $S \in Sys(V)$ and $A \in ER(T)$. Then $\bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A) \preceq_{EM} \mathcal{K}_S(A)$.*

Proof. We show the lower and the upper ordering separately.

1. *$\forall X \in [P]. \mathcal{K}_{S\cap X}(A) \preceq_L \mathcal{K}_S(A)$:*
   Let $Y \in \mathcal{K}_{S\cap X}(A)$. Then, for some $O \in [A]$, $Y = \Phi(O \cap S \cap X)$ and $O \cap S \cap X$ is non-empty. Let $Y' = \Phi(O \cap S)$. Then $O \cap S$ is non-empty (since $O \cap S \cap X$ is non-empty), hence $Y \subseteq Y' \in \mathcal{K}_S(A)$.

2. $\forall Y \in \mathcal{K}_S(A).\exists X \in [P].\exists Y' \in \mathcal{K}_{S\cap X}(A).Y' \subseteq Y$:
   Let $Y \in \mathcal{K}_S(A)$. Then, for some $O \in [A]$, $Y = \Phi(O \cap S)$ and $O \cap S$ is non-empty. Suppose, towards a contradiction, that $O \cap S \cap X = \emptyset$ for all $X \in [P]$, hence $O \cap S \cap \bigcup_{X\in[P]} X = \emptyset$; but $[P]$ partitions T, so $\bigcup_{X\in[P]} X = T \supseteq O \cap S$, hence $O \cap S = \emptyset$, a contradiction.
   So let $X \in [P]$ with $O \cap S \cap X$ non-empty, and let $Y' = \Phi(O \cap S \cap X)$. Then $Y' \subseteq Y$ and $Y' \in \mathcal{K}_{S\cap X}(A)$.

From 1 it follows that $\bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A) \preceq_L \mathcal{K}_S(A)$ and from 2 it follows that $\bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A) \preceq_U \mathcal{K}_S(A)$, thus:

$$\bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A) \preceq_{EM} \mathcal{K}_S(A)$$

A.1 Proof of Therorem 1

By Lemma 5, $\bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A) \preceq_{\text{EM}} \mathcal{K}_S(A)$.

By assumption of policy satisfaction, $[f\ X] \preceq_o \mathcal{K}_{S\cap X}(A)$ for all $X \in [P]$,with each $\mathcal{K}_{S\cap X}(A)$ covering V.

So, by Lemma 3:

$$\bigcup_{X\in[P]} [f\ X] \preceq_o \bigcup_{X\in[P]} \mathcal{K}_{S\cap X}(A)$$

It then suffices to show that $[\bigcap_{X\in[P]}(f\ X)] \preceq_{\text{EM}} \bigcup_{X\in[P]} [f\ X]$. This is immediate by Lemma 4. □

Lemma 6. *: Let $\{R_W\}_{W\in Q}$ be a partition-indexed family of equivalence relations such that $R_W \in ER(W)$ for each $W \in Q$. Then:*

1. $\bigwedge_{W\in Q} R^\dagger_W = \bigcup_{W\in Q} R_W$
2. $\bigwedge_{W\in Q} R^\dagger_W \subseteq \mathcal{E}(Q)$

Proof. 1. *Recall $R^\dagger_W = R_W \cup All_{V\setminus W}$.Then, for all W in the partition Q, $\forall (x, y) \in R^\dagger_W$ either $x \in W \wedge y \in W$ or $x \notin W \wedge y \notin W$. In fact, suppose $x \in W$ but $y \notin W$, then $(x, y) \notin R_W$ because $y \notin W$ and $(x, y) \notin All_{V\setminus W}$ because $x \notin V \setminus W$, a contradiction.*
   We now show $\bigwedge_{W\in Q} R^\dagger_W \subseteq \bigcup_{W\in Q} R_W$. Consider $(x, y) \in \bigwedge_{W\in Q} R^\dagger_W$. Then $\exists W \in Q.x \in W \wedge y \in W$ because of the previous result, therefore $(x, y) \in R_W \subseteq \bigcup_{W\in Q} R_W$.
   We now show $\bigwedge_{W\in Q} R^\dagger_W \supseteq \bigcup_{W\in Q} R_W$. Consider $(x, y) \in \bigcup_{W\in Q} R_W$. Since $\forall W, W' \in Q.W \cap W' = \emptyset$, $\exists W \in Q.(x, y) \in R_W$, therefore $(x, y) \in R^\dagger_W$. For all others $W' \in Q.W' \neq W$ we have $x \notin W' \wedge y \notin W'$, therefore $(x, y) \in R^\dagger_{W'}$. So we can conclude $(x, y) \in \bigwedge_{W\in Q} R^\dagger_W$
2. *Consider $(x, y) \in \bigwedge_{W\in Q} R^\dagger_W$. Since $\bigwedge_{W\in Q} R^\dagger_W = \bigcup_{W\in Q} R_W$ and since $\forall W, W' \in Q.W \cap W' = \emptyset$, $\exists! W \in Q.(x, y) \in R_W$. But $R_W \subseteq W \times W$ and $W \times W \subseteq \mathcal{E}(Q)$ by definition, therefore $(x, y) \in \mathcal{E}(Q)$.*

□

A.2 Proof of Therorem 2

Recall the definition of $T_W = \{t \in T | \Phi(t) \in W\}$.

Let P_Q be a partition of T_V defined as $P_Q = \bigcup_{W \in [Q]} T_W$.

By Lemma 5 we have $\bigcup_{T_W \in [P_Q]} \mathcal{K}_{S \cap T_W}(A) \preceq_{\text{EM}} \mathcal{K}_S(A)$.

We then have $[R_{(g\,W)}] \preceq_o \mathcal{K}_{S \cap T_W}(A)$ for all $T_W \in [P_Q]$ by assumption of policy satisfaction and Theorem 1 applied to all subsystems $S \cap T_W$.

By Lemma 6 we have $\mathcal{E}(Q) \wedge \bigwedge_{W \in Q} R^{\dagger}_{(g\,W)} = \bigwedge_{W \in Q} R^{\dagger}_{(g\,W)} = \bigcup_{W \in Q} R_{(g\,W)}$.

To conclude the proof we only need $\bigcup_{W \in Q} [R_{(g\,W)}] \preceq_o \bigcup_{T_W \in P_Q} \mathcal{K}_{S \cap T_W}(A)$, which holds by Lemma 3 . □