A Theory of Gray Security Policies

Donald Ray and Jay Ligatti(✉)

Department of Computer Science and Engineering,
University of South Florida, Tampa, USA
{dray3,ligatti}@cse.usf.edu

Abstract. This paper generalizes traditional models of security policies, from specifications of *whether* programs are secure, to specifications of *how* secure programs are. This is a generalization from qualitative, black-and-white policies to quantitative, gray policies. Included are generalizations from traditional definitions of safety and liveness policies to definitions of gray-safety and gray-liveness policies. These generalizations preserve key properties of safety and liveness, including that the intersection of safety and liveness is a unique allow-all policy and that every policy can be written as the conjunction of a single safety and a single liveness policy. It is argued that the generalization provides several benefits, including that it serves as a unifying framework for disparate approaches to security metrics, and that it separates—in a practically useful way—specifications of how secure systems are from specifications of how secure users require their systems to be.

1 Introduction

Computer-security policies have traditionally been modeled as predicates, partitioning the secure from the insecure system behaviors. Policies partition behaviors by specifying constraints like "only administrators may write to files", "packets destined for port 120 must be logged", or "all array accesses must be bounds-checked". These are qualitative, black-and-white constraints that can be used to decide whether a given system is secure.

This paper generalizes the qualitative, black-and-white model of policies to a quantitative, gray model. Instead of specifying *whether* systems are secure, gray policies specify *how* secure systems are. For example, a gray policy for array-bounds checking might consider that checking every array access makes a program 100 % secure and that each unchecked access decimates a program's current rating.

Gray policies are useful because users are often unwilling to pay the costs required to achieve 100 % security according to some policy. As is well understood, enforcement costs can be high, typically in the form of:

- performance overhead (e.g., due to increased runtime checks),
- code-size overhead (e.g., due to inlined monitoring code),
- decreased usability (e.g., due to authentication procedures), and

G. Pernul et al. (Eds.): ESORICS 2015, Part II, LNCS 9327, pp. 481–499, 2015.
DOI: 10.1007/978-3-319-24177-7_24

– consumption of other system resources (e.g., due to security checks draining batteries or security logs draining file-system space).

To make an analogy with the physical world, safes are not rated as secure or insecure, but rather by the estimated amount of time needed to penetrate them with a given set of tools. Such a quantitative rating enables consumers to weigh the security metric against other metrics, such as size, weight, price, and availability, when choosing a safe to buy. Importantly, a choice made in one context may differ from a choice made in another context, depending on the priorities of the purchaser and resources available.

In this paper's framework, a gray policy specifies a system's security rating, while a *silhouette judge* specifies a user's security requirements. Returning to the safe analogy, a silhouette judge is like a consumer's purchasing-decision algorithm that inputs a safe's security rating and, combining it with the safe's other attributes, outputs a buy or don't-buy decision.

Thus, this paper's framework separates the intuitively distinct specifications of how secure systems are (gray policies) from how secure users require their systems to be (silhouette judges). This separation enables users with different security requirements to use the same gray policy in different ways, by specifying different silhouette judges. For example, in the context of high-performance systems research users might require 0 % security (e.g., no array-bounds checking), while in the context of flight-navigation software users might require 100 % security.

There are additional benefits of the gray model over the black-and-white model. Gray policies enable users to compare the security of different systems when choosing which to use. In the black-and-white model, a user who can't afford to run a "secure" web browser has to choose between other browsers only known to be "insecure"; in the gray model, the same user could choose the most secure of the affordable browsers. As another potential benefit, consumers often base purchasing decisions on measurable attributes, so quantifying security could drive demand for security improvements, even ones that degrade performance by 10–20 % or more, thus countering the arguments of some developers that such security overheads are intolerable [32].

Overview of Related Work and Contributions

Of course, the idea to quantify security is not new (e.g., [3,14,18,31,35,48,49]).

However, the extensive research into general-purpose models of policies has considered them to be predicates and therefore black and white (e.g., [17,22,24,29,41,54]). Many interesting results have come from these qualitative models of policies, including definitions of safety and liveness properties, which are tied to particular classes of enforcement mechanisms, and proofs that every black-and-white property is the conjunction of one safety property and one liveness property.

This paper contributes a more general, quantitative model of policies and properties (Sect. 2). This model generalizes existing definitions of policies, properties, safety, liveness, hypersafety, and hyperliveness. It is shown that the new

model is indeed a generalization, in that every black-and-white policy is also a gray policy, every black-and-white safety property is also a gray safety property, etc.

It is also shown that the gray model preserves interesting properties of safety and liveness that were previously derived in black-and-white models (Sect. 3). Specifically, the intersection of gray safety and gray liveness properties is a unique allow-all property, every gray property can be written as the conjunction of a single gray-safety and a single gray-liveness property, and similarly for hypersafety and hyperliveness policies.

Section 4 shows how this paper's model of gray policies can serve as a unifying framework for many disparate approaches to security metrics, and how gray policies might be constructed from existing black-and-white policies.

Section 5 formalizes silhouette judges and shows how they work in tandem with gray policies, and Sect. 6 briefly discusses future work.

2 From Black-and-White to Gray Policies

Policies reason about systems, which execute *events*. Let E be a non-empty, countable set of events, with metavariable e ranging over individual events. Intuitively, E is the system API and might contain instructions for manipulating system resources.

A system *trace*, or *execution*, x, is a possibly infinite sequence of pairs of events called *exchanges*. The events in an exchange $\langle e, e' \rangle$ indicate (1) an event e the system attempts to execute and (2) an event e' that actually executes. For example, the trace

$$\langle sti(0, 0x9ABC), sti(0, 0x1ABC) \rangle \, \langle rdr(4, 0x8FFF), rdr(4, 0x0FFF) \rangle$$

indicates that the *target system* being reasoned about, for example an application program, attempted to store the immediate value 0 at memory address 0x9ABC, but 0 was instead written at address 0x1ABC, due to the involvement of a runtime mechanism such as a virtual-memory manager. The second exchange in the trace also shows involvement of a runtime mechanism, again decreasing the memory address being accessed by 2^{15}.

This model of traces as sequences of exchanges is general, in part because it clarifies the effects of runtime mechanisms; such clarification improves expressiveness [24,42]. In cases where policies require no runtime support, such as statically enforced policies, the first and second events in exchanges will be the same.

Some additional notation will be useful. A set of events E determines the set of possible exchanges $\mathcal{E}$. Given exchange set $\mathcal{E}$, $\mathcal{E}^*$ denotes the set of all finite executions (i.e., finite sequences of exchanges), $\mathcal{E}^\omega$ denotes the set of all infinite executions, and $\mathcal{E}^\infty$ denotes the set of all finite and infinite executions. Also, $x \preceq y$ and $y \succeq x$ mean that execution $x \in \mathcal{E}^*$ is a *prefix* of execution $y \in \mathcal{E}^\infty$. Finally, shorthand quantifications will be used in formulae; for example, $\exists x \preceq y : F$ means $\exists x \in \mathcal{E}^* : (x \preceq y \wedge F)$, while $\forall x \succeq y : F$ means $\forall x \in \mathcal{E}^\infty : (x \succeq y \Rightarrow F)$.

2.1 Policies and Properties

The black-and-white model defines *policies* P as predicates over target systems; the policy returns a yes-no response to a given target system, to indicate whether that system is secure [54]. A target system X is modeled as the set of executions it can produce, for example, all possible runs of an application program. Hence, on a system with exchange set $\mathcal{E}$, X is a subset of $\mathcal{E}^\infty$, so a *black-and-white policy* is a $P : 2^{\mathcal{E}^\infty} \to \{false, true\}$.

The gray model defines policies G as functions mapping target systems not to false/true values, but to a real number between 0 and 1, with greater numbers indicating higher security. Gray policies generalize black-and-white policies because false/true values in the black-and-white model can always be encoded as 0/1 values in the gray model. A *gray policy* is simply a $G : 2^{\mathcal{E}^\infty} \to \mathbb{R}_{[0,1]}$.

In the black-and-white model, *properties* are policies that place no constraints on the relationships between executions [54]. It can be determined whether a target system satisfies a property by examining each possible trace in isolation; if every trace is valid in isolation (according to some predicate p over individual traces), then the policy as a whole is satisfied. Formally, a policy P is a *black-and-white property* iff

$$\exists\,(p : \mathcal{E}^\infty \to \{false, true\}) : \forall X \subseteq \mathcal{E}^\infty : P\,(X) = (\forall x \in X : p\,(x)).$$

The gray model also considers a policy to be a property when the policy's value for a given a set of executions can be determined by examining each execution in isolation. While the black-and-white model determines the policy's value $P(X)$ as the *conjunction* of the values of $p(x)$, for all $x \in X$, the gray model determines the policy's value $G(X)$ as the *infimum* (inf) of the values of $g(x)$, for all $x \in X$. Here g, like p, is a function over individual traces. Formally, a policy G is a *gray property* iff

$$\exists\left(g : \mathcal{E}^\infty \to \mathbb{R}_{[0,1]}\right) : \forall X \subseteq \mathcal{E}^\infty : G\,(X) = \inf\{g\,(x) \mid x \in X\}.$$

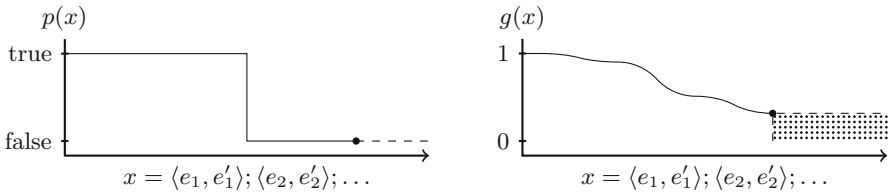
Gray properties generalize black-and-white properties because the conjunction of a set of false/true values always equals the infimum of a set of corresponding 0/1 values.[1]

It is often convenient to identify a property by the individual-trace function (p or g) it uses. There is no ambiguity in doing so, due to the one-to-one correspondence between a p or g function and the property it induces.

2.2 Safety and Liveness

Two subsets of black-and-white properties have been studied extensively: safety and liveness properties [1,38]. These sets are fundamentally intertwined with the sets of properties that can be enforced in practice [2,24,25,41,54].

[1] The use of the infimum precludes limiting the range of security values in the gray model to computable reals; computable reals are not closed under infimum operations [57].



(a) Black-and-white Safety (b) Gray Safety

Fig. 1. The security of traces as they proceed. The security level is according to (a) a black-and-white safety property and (b) a gray safety property. The dotted lines and shaded area represent the possible security values of the executions' extensions. In all cases, security levels are nonincreasing.

Safety. Black-and-white safety properties partition "secure" from "insecure" traces in such a way that every insecure trace has a finite insecure prefix that can never become secure [38]. Formally, property p is *black-and-white safety* iff

$$\forall x \in \mathcal{E}^\infty : (\neg p(x) \Rightarrow \exists x' \preceq x : \forall y \succeq x' : \neg p(y)).$$

An equivalent, perhaps more intuitive, definition of black-and-white safety is the set of properties that are both prefix- and omega-closed [24]. Prefix-closed means that all prefixes of secure traces are secure, while omega-closed means the converse, that if all prefixes of a trace x are secure then so must be x. Formally, property p is *black-and-white safety* iff

$$\forall x \in \mathcal{E}^\infty : p(x) = (\forall x' \preceq x : p(x')).$$

This formalization of black-and-white safety has an interesting similarity to the formalization of black-and-white properties; in both cases, an entity is secure exactly when all of its "simpler parts" are secure.

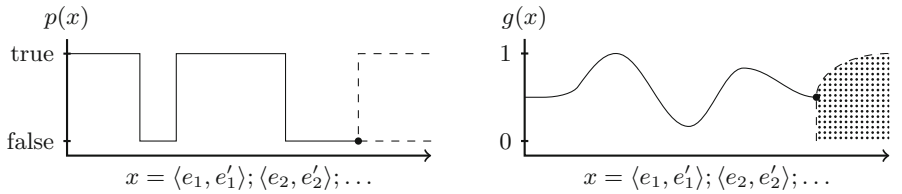
It can be seen from these definitions that black-and-white safety properties require traces to be as secure as their least-secure prefix; security cannot increase as traces proceed. Figure 1(a) plots the general shape of a trace's security as considered by a black-and-white safety property.

Gray safety properties also require traces to have nonincreasing security, as shown in Fig. 1(b). However, with gray safety, the requirement that traces be as secure as their least-secure prefix has to be modified—infinite traces may not have a least-secure prefix. To handle such cases the *infimum* is again used. Formally, property g is *gray safety* iff

$$\forall x \in \mathcal{E}^\infty : g(x) = \inf\{g(x') \mid x' \preceq x\}.$$

This formalization of gray safety retains the similarity, present in the black-and-white model, between the definitions of properties and safety.

As an example, the gray property described earlier, specifying that a trace's security level gets decimated with each unchecked array access, is a gray safety



(a) Black-and-white Liveness (b) Gray Liveness

Fig. 2. The security of traces as they proceed. The security level is according to (a) a black-and-white liveness property and (b) a gray liveness property. The dotted lines and shaded area represent the possible security values of the executions' extensions. In all cases, less-than-fully-secure traces have more-secure extensions.

property. Traces according to this policy begin as 100 % secure (before any exchanges occur) and can only proceed to lower security. In the limit, a trace containing an infinite number of unchecked array accesses has 0 % security, because the infimum of $\{1, 0.9, 0.81, 0.729, \ldots\}$ is 0.

Gray safety is a proper generalization of black-and-white safety. To understand why, note that black-and-white policies (properties) can be trivially converted to gray policies (properties) by partitioning target systems (traces) not as insecure or secure but as having security levels of 0 or 1. Then because a black-and-white safety property p is prefix- and omega-closed, traces are as secure as their least-secure prefix, so p converts to a gray safety property. Conversely, a black-and-white nonsafety property p' assigns the security of some trace to be different than its least-secure prefix, so p' converts to a gray nonsafety property.

Liveness. Black-and-white liveness properties require every finite trace to have a secure extension [1], as shown in Fig. 2(a). A canonical example is the termination property, which requires traces to be finite (so every finite trace x has a secure extension, namely x). Formally, property p is *black-and-white liveness* iff

$$\forall x \in \mathcal{E}^* : \exists y \succeq x : p(y).$$

Analogously, gray liveness properties require every finite trace to have a more-secure extension, with traces that are already 100 % secure exempted (because a fully secure trace cannot have a more-secure extension). Figure 2(b) illustrates the requirement that, according to a gray liveness property, every imperfectly secure trace has a more-secure extension.

To formalize gray liveness, a new operator $\overline{>}$ is defined that behaves exactly like a greater-than operator ($>$), except that $1 \overline{>} 1$. Then property g is *gray liveness* iff

$$\forall x \in \mathcal{E}^* : \exists y \succeq x : g(y) \overline{>} g(x).$$

For example, a gray liveness property could map trace x to a security value based on the number n of resources acquired but unreleased in x; the security level might be $1-0.01n$ when $0 \leq n \leq 100$ and 0 when $n > 100$. This property

gives traces a 1 % security penalty for every unreleased resource. It is a gray liveness property because every finite, imperfectly secure trace has a more-secure extension (in which acquired resources are released). It is interesting to compare the usefulness of, and information provided by, this gray property with its black-and-white version, which simply says that traces are secure iff all acquired resources eventually get released.

As with safety, gray liveness is a proper generalization of black-and-white liveness. A black-and-white liveness property p requires every finite, insecure trace to have a secure extension, so p converts to a gray liveness property. Conversely, a black-and-white nonliveness property p' forbids some finite, insecure trace from becoming secure, causing p' to convert to a gray nonliveness property.

2.3 Hypersafety and Hyperliveness

Just as black-and-white properties can be categorized as safety or liveness, the same can be done for black-and-white policies. Using the term "hyperproperty" to mean "policy", then, hyperproperties can be categorized as hypersafety or hyperliveness [17]. Intuitively, the definitions of hypersafety and hyperliveness raise the definitions of safety and liveness from the level of executions (properties) to the level of sets of executions (policies).

The definitions of safety and liveness rely on the the $\preceq$ and $\succeq$ operators to indicate *executions* being prefixed or extended; definitions of hypersafety and hyperliveness will need to raise these operators to the level of *sets of executions*. This raising is accomplished by defining a *terminating target system* X—that is, a set of finite executions—to *prefix* another target system Y, written $X \sqsubseteq Y$, iff every execution in X is a prefix of some execution in Y. Formally, given $X \subseteq \mathcal{E}^*$ and $Y \subseteq \mathcal{E}^\infty$, $X \sqsubseteq Y$ iff $\forall x \in X : \exists y \in Y : x \preceq y$ [17, Footnote 13]. The $Y \sqsupseteq X$ relation is defined symmetrically.

Hypersafety. Black-and-white hypersafety raises black-and-white safety from the level of traces (executions) to the level of target systems (sets of executions) by requiring that target systems are secure iff all their prefixes are secure. Hence, just as *property* p was defined to be black-and-white safety iff

$$\forall x \in \mathcal{E}^\infty : p(x) = (\forall x' \preceq x : p(x')),$$

policy P is *black-and-white hypersafety* iff

$$\forall X \subseteq \mathcal{E}^\infty : P(X) = (\forall X' \sqsubseteq X : P(X')).$$

Similarly, just as *property* g was defined to be gray safety iff

$$\forall x \in \mathcal{E}^\infty : g(x) = \inf\{g(x') \mid x' \preceq x\},$$

policy G is *gray hypersafety* iff

$$\forall X \subseteq \mathcal{E}^\infty : G(X) = \inf\{G(X') \mid X' \sqsubseteq X\}.$$

The reasoning that gray hypersafety is a proper generalization of black-and-white hypersafety follows the reasoning used to show that gray safety is a proper generalization of black-and-white safety.

Following [17], it is also possible to define (black-and-white and gray) k-hypersafety by restricting the set X' to have at most k elements. For example, policy G is *gray k-hypersafety* iff

$$\forall X \subseteq \mathcal{E}^\infty : G(X) = \inf\{G(X') \mid X' \sqsubseteq X, |X'| \leq k\}.$$

Hyperliveness. Black-and-white hyperliveness requires that all terminating target systems have secure extensions. Just as *property p* was defined to be black-and-white liveness iff

$$\forall x \in \mathcal{E}^* : \exists y \succeq x : p(y),$$

policy P is *black-and-white hyperliveness* iff

$$\forall X \subseteq \mathcal{E}^* : \exists Y \sqsupseteq X : P(Y).$$

Similarly, just as *property g* was defined to be gray liveness iff

$$\forall x \in \mathcal{E}^* : \exists y \succeq x : g(y) \overline{\geq} g(x),$$

policy G is *gray hyperliveness* iff

$$\forall X \subseteq \mathcal{E}^* : \exists Y \sqsupseteq X : G(Y) \overline{\geq} G(X).$$

Gray hyperliveness is a proper generalization of black-and-white hyperliveness by the same reasoning used to show that gray liveness properly generalizes black-and-white liveness.

2.4 Summary

Table 1 summarizes the gray definitions and compares each with its black-and-white counterpart.

3 Further Analysis of the Model

The generalization of black-and-white to gray policies preserves key properties of the black-and-white model.

3.1 Singleton Intersection of Safety and Liveness

In the black-and-white models, exactly one property is both safety and liveness: the "allow-all" property that considers every trace secure [1]. Similarly, exactly one policy is both hypersafety and hyperliveness: the policy that considers every target system secure [17]. The following theorems show that, analogously, exactly one property (policy) is both gray (hyper)safety and gray (hyper)liveness: the property (policy) that considers every trace (target system) perfectly secure.

Table 1. Summary of the generalization from black-and-white to gray policies. The black-and-white definitions are taken from [1,17,24,54]. As a reminder, the $\overline{>}$ operator behaves like greater-than ($>$), except that $1 \overline{>} 1$.

policy	◩	$P : 2^{\mathcal{E}^\infty} \to \{false, true\}$
	▨	$G : 2^{\mathcal{E}^\infty} \to \mathbb{R}_{[0,1]}$
property	◩	$\exists p : \forall X \subseteq \mathcal{E}^\infty : P(X) = (\forall x \in X : p(x))$
	▨	$\exists g : \forall X \subseteq \mathcal{E}^\infty : G(X) = \inf\{g(x) \mid x \in X\}$
safety	◩	$\forall x \in \mathcal{E}^\infty : p(x) = (\forall x' \preceq x : p(x'))$
	▨	$\forall x \in \mathcal{E}^\infty : g(x) = \inf\{g(x') \mid x' \preceq x\}$
liveness	◩	$\forall x \in \mathcal{E}^* : \exists y \succeq x : p(y)$
	▨	$\forall x \in \mathcal{E}^* : \exists y \succeq x : g(y) \overline{>} g(x)$
hypersafety	◩	$\forall X \subseteq \mathcal{E}^\infty : P(X) = (\forall X' \sqsubseteq X : P(X'))$
	▨	$\forall X \subseteq \mathcal{E}^\infty : G(X) = \inf\{G(X') \mid X' \sqsubseteq X\}$
hyperliveness	◩	$\forall X \subseteq \mathcal{E}^* : \exists Y \sqsupseteq X : P(Y)$
	▨	$\forall X \subseteq \mathcal{E}^* : \exists Y \sqsupseteq X : G(Y) \overline{>} G(X)$

Theorem 1. *The gray property $g(x) = 1$ is the only gray property that is both gray safety and gray liveness.*

Proof. First note that $g(x) = 1$ is trivially both a gray safety and a gray liveness property.

Now let g' be an arbitrary gray property that is both gray safety and gray liveness. For the sake of obtaining a contradiction, suppose there exists an execution x such that $g'(x) < 1$. If x is infinite, it must have a finite prefix whose security is also less than 1 because g' is gray safety; let x instead refer to that prefix. Because g' is gray liveness, there exists $y \succeq x$ such that $g'(y) \overline{>} g'(x)$, so because $g'(x) \neq 1$, it must be that $g'(y) > g'(x)$. Also, because g' is gray safety, $g'(y)$ must equal $\inf\{g'(y') \mid y' \preceq y\}$. However, x is a prefix of y, so by the definition of infimum, $g'(y) \leq g'(x)$, which contradicts the earlier result that $g'(y) > g'(x)$. Thus, for all x, $g'(x) = 1$, meaning that g' must be g. □

Theorem 2. *The gray policy $G(X) = 1$ is the only gray policy that is both gray hypersafety and gray hyperliveness.*

Proof. Analogous to that of Theorem 1. □

Figure 3 depicts the relationships between gray properties, black-and-white properties, and their subsets of safety and liveness properties. Notably, the gray sets subsume the black-and-white sets, and the intersection of safety and liveness is the black-and-white allow-all property.

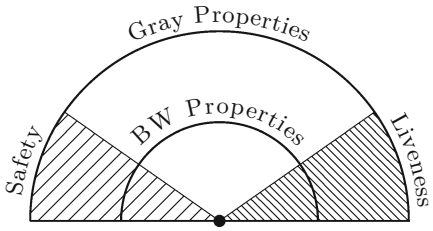


Fig. 3. Relationships between gray and black-and-white properties, and their subsets of safety and liveness properties. The central dot represents the intersection of safety and liveness, which only contains the property $g(x) = 1$.

3.2 Decomposition into Safety and Liveness

In the black-and-white models, every property p can be decomposed into properties p_s and p_l such that:

- p_s is a black-and-white safety property,
- p_l is a black-and-white liveness property, and
- $p(x) = (p_s(x) \wedge p_l(x))$.

In other words, every black-and-white property is the conjunction of a single safety and a single liveness property. This result appeared in [1], with alternative proofs appearing in [41,53]. A similar result has been shown for decomposing policies into hypersafety and hyperliveness [17].

Theorem 3 shows that the gray model preserves this decomposition result.

Theorem 3. *Every gray property g can be decomposed into g_s and g_l such that:*

- *g_s is a gray safety property,*
- *g_l is a gray liveness property, and*
- *$g(x) = min(g_s(x), g_l(x))$.*

Proof. Construct g_s and g_l as follows, where *sup* refers to the supremum function.

$$g_s(x) = \begin{cases} \inf\{g_s(x') \mid x' \preceq x\} & \text{if } x \in \mathcal{E}^\omega \\ g(x) & \text{if } x \in \mathcal{E}^* \text{ and } \forall x' \succeq x : g(x') \leq g(x) \\ \sup\{g(x') \mid x' \succeq x\} & \text{otherwise} \end{cases}$$

$$g_l(x) = \begin{cases} 1 & \text{if } x \in \mathcal{E}^* \text{ and } \forall x' \succeq x : g(x') \leq g(x) \\ g(x) & \text{otherwise} \end{cases}$$

To establish that g_s is gray safety, it must be shown that for all $x \in \mathcal{E}^\infty$, $g_s(x) = \inf\{g_s(x') \mid x' \preceq x\}$. By construction, this constraint holds for all $x \in \mathcal{E}^\omega$. For finite executions, g_s ensures that security never increases as traces proceed by giving every finite trace x a security value that's greater than or equal to all of x's extensions. Hence, the safety constraint holds for all finite executions as well.

To establish that g_l is gray liveness, it must be shown that for all $x \in \mathcal{E}^*$, $\exists y \succeq x : g_l(y) \sqsupseteq g_l(x)$. Let x be a finite execution. If all extensions x' of x have security less than or equal to x (according to g), then $g_l(x) = 1$ and the liveness constraint is satisfied by letting $y = x$. On the other hand, if some extension x' of x has security greater than x (according to g), then $g_l(x) = g(x)$ and the liveness constraint is satisfied by letting $y = x'$ (where $g_l(x')$ must be at least $g(x')$, which is greater than $g(x) = g_l(x)$).

It remains to establish that $g(x) = min(g_s(x), g_l(x))$. If x is a finite trace then this result immediately follows from the definitions of g_s and g_l. If x is an infinite trace, first observe that g_s assigns every prefix x' of x a security level of at least $g(x)$. Hence, g_s assigns $x \in \mathcal{E}^\omega$ a security level of at least $g(x)$, while g_l assigns $x \in \mathcal{E}^\omega$ a security level of $g(x)$, which completes the proof. □

As in the black-and-white models, the decomposition result in the gray model extends to policies, hypersafety, and hyperliveness.

Theorem 4. *Every gray policy G can be decomposed into G_s and G_ℓ such that:*

- *G_s is a gray hypersafety policy,*
- *G_ℓ is a gray hyperliveness policy, and*
- *$G(X) = min(G_s(X), G_\ell(X))$.*

Proof. Analogous to that of Theorem 3. □

4 Creating Gray Policies from Existing Metrics/Policies

Existing work on security metrics and on black-and-white policies can be used to create new gray policies.

4.1 Gray Policies Based on Security Metrics

The gray model serves as a unifying framework for disparate approaches to security metrics. The disparate approaches include:

- using greater values to indicate higher levels of security (e.g., [23]),
- using greater values to indicate lower levels of security (e.g., [46]),
- limiting security values to a bounded range (e.g., [14]),
- limiting security values to a range bounded only on the lower side (e.g., [40]), and
- placing no bounds on the range of security values (e.g., [9]).

In contrast to black and white models, all of these approaches can be encoded in the gray model.

Encoding these disparate approaches to security metrics in the gray model provides the benefit of consistency. In the gray model, security consistently ranges between 0 and 1, and for a fixed policy or property, greater security values consistently indicate higher security.

Table 2. Examples of functions that can be used to normalize security metrics to the gray model's range of $\mathbb{R}_{[0,1]}$. Variable x denotes the output of the security metric, constants A and B denote the metric's minimum and maximum values (when applicable), and constant C denotes a positive number (C affects how quickly the functions converge to absolute security or insecurity).

	Bounded	Lower bounded	Unbounded
Higher values represent higher security	$y = \frac{x - A}{B - A}$	$y = \frac{x - A}{x - A + C}$	$y = 0.5 + \frac{\tan^{-1}(C * x)}{\pi}$
Higher values represent lower security	$y = \frac{B - x}{B - A}$	$y = \frac{C}{x - A + C}$	$y = 0.5 + \frac{\tan^{-1}(-C * x)}{\pi}$

Table 2 shows several example functions that can be used to encode security metrics as gray policies. Every one of the more than forty metrics we've studied [3–10,12–16,18–21,23,27,28,30,31,33–37,39,40,45–47,49–52,55,56,58–61], in domains as varied as access control, noninterference, privacy, integrity, and network security, can be encoded as a gray policy or property by using one of the functions shown in Table 2.

The arctangent functions shown in Table 2 can be used to normalize metrics having an unbounded range because the arctangent's domain is all real numbers, and its output monotonically increases over the range $(-\frac{\pi}{2}, \frac{\pi}{2})$. The arccotangent function ($\cot^{-1}$), and many others, could be used instead.

4.2 Graying Black-and-White Policies

Gray policies can also be created from existing black-and-white policies.

For example, a gray policy $G(X)$ could be created by quantifying how well the given target system X obeys a particular black-and-white policy. This technique has already been used in this paper's examples: black-and-white policies might require all array accesses to be checked or all acquired resources to be released; these policies were grayed by penalizing target systems based on how far their traces deviate from ideal traces. A similar idea is used with cost-aware enforcement [25], where a cost, or penalty, can be assigned to certain exchanges.

Another approach to graying considers the overall security achieved by permitting some "insecure" executions to be run and/or denying some "secure" executions from being run [26].

Gray policies could be defined based on a similar idea: Given a black-and-white property of interest p, $G(X)$ might be defined as the product of:

– the probability that a randomly selected element of X satisfies p—*such a probability measures the* soundness *of* X *with respect to* p—and
– the probability that a randomly selected element of $\{x \mid p(x)\}$ is in X—*such a probability measures the* completeness *of* X *with respect to* p.

Following [44], these probabilities could be weighted by the likelihood of traces to actually be observed (due to nonuniform input distributions and target-system functionality, some traces may be observed much more frequently than others). Therefore, when calculating $G(X)$ in terms of the soundness and completeness probabilities defined above, one might choose traces not from uniform distributions, but instead with the more-likely-to-be-observed traces more likely to be chosen.

5 Silhouettes and Their Judges

The gray model separates specifications of how secure target systems are (gray policies) from specifications of how secure users require their systems to be (silhouette judges). In the safe analogy of Sect. 1, silhouette judges input a safe's security rating and output a buy or don't-buy decision. In other words, silhouette judges input a *silhouette*—a distillation of a safe's characteristics into a security value—and output a no/yes decision to indicate whether that silhouette is acceptably secure.

5.1 Silhouettes

Thus, silhouette judges, as their name implies, judge silhouettes, by outputting a no/yes (or false/true) to indicate whether a given silhouette is acceptably secure.

In the gray model, a *silhouette* represents the shape of a trace's (or target system's) security. For example, the plots shown in Figs. 1 and 2 illustrate silhouettes of traces—the plots abstract from the events of the underlying traces to provide only the shape of the security values achieved as the traces proceed.

Silhouettes can be formalized in many ways. For generality, the key requirement is to encode a trace's (or target system's) evolution of security values.

For example, the silhouette of a trace x according to property g can be formalized as a function s that takes a natural number n, or a special ∞ symbol, as input and returns the security (according to g) of x's n-length prefix, or the security of x itself if s's input is ∞. With this formalization, silhouettes are partial functions; e.g., the silhouette of the empty trace is undefined for all inputs $n > 0$.

With this formalization, *the silhouette of trace* x *according to gray property* g is the partial function $s_{x,g} : (\mathbb{N} \cup \{\infty\}) \to \mathbb{R}_{[0,1]}$, such that:

$$s_{x,g}(n) = \begin{cases} g(x) & \text{if } n = \infty \\ g(x') & \text{if } x' \text{ is the } n\text{-length prefix of } x \end{cases}$$

Because target systems may be infinite sets of infinite-length traces, silhouettes of target systems are more complicated than those of individual traces. Rather than mapping natural numbers to security values, target-system silhouettes could map real numbers to security values. In this case, the real number can encode which parts of the target system's traces to evaluate the security of.

For example, a silhouette for target system X could interpret an input like 0.192939..969109119... as identifying the set of traces containing the 1-length prefix of X's first execution (ordered lexicographically), the 2-length prefix of X's second execution, and so on, with each prefix length delimited by a 9 and written in base-7. Under this encoding, the target-system silhouette could interpret a 7 (8) appearing before the i^{th} 9 in an input real number as indicating exclusion of the (inclusion of the whole) i^{th} execution in X.

With such a formalization, the *silhouette of target system X according to gray policy G* is the partial function $S_{X,G} : \mathbb{R} \to \mathbb{R}_{[0,1]}$, such that:

$$S_{X,G}(r) = G(X'), \text{ where } r \text{ encodes } X' \text{ with respect to } X.$$

5.2 Silhouette Judges

Silhouette judges are simply predicates over silhouettes. A judge therefore acts as the final, black-and-white decision maker, determining whether a trace or target system is acceptably secure. Importantly, judges base their decisions on *silhouettes* of traces or target systems, not on the traces or target systems themselves, as is done in black-and-white models.

For example, a silhouette judge could forbid all trace silhouettes whose security ever drops below a certain minimum threshold. This sort of silhouette judge specifies a user's minimum security requirement, such as "traces must always be at least 80 % secure".

Another silhouette judge might forbid all silhouettes whose "final" security value (obtained by inputting ∞ to the given silhouette) is greater than 0. Such a judge might be used by high-performance systems researchers to require the complete insecurity of their executions.

More interesting judges can also be defined. For example, it may be reasonable to allow executions to occasionally behave less securely, provided they are usually more secure. Such a judge might be satisfied by exactly those silhouettes having a rolling average of security above a given threshold. Other judges could be satisfied by exactly those silhouettes that never dip below a desired threshold for more than k consecutive exchanges.

Theorem 5 states that combining a gray property g with a trace-silhouette judge j produces a unique black-and-white property p, but, on the other hand, every black-and-white property can be decomposed into uncountably many different gray-property, trace-silhouette-judge pairs. Theorem 6 states a similar result for black-and-white policies and gray-policy, trace-system-silhouette-judge pairs.

Theorem 5. *There is a one-to-uncountably-many correspondence between black-and-white properties p and pairs of gray properties and trace-silhouette judges (g, j) such that $\forall x \in \mathcal{E}^\infty : p(x) \Leftrightarrow j(s_{x,g})$.*

Proof. Every gray-property, trace-silhouette-judge pair (g, j) is equivalent to exactly one black-and-white property p; otherwise, there must be some execution x whose silhouette according to g, $s_{x,g}$, both satisfies and dissatisfies j, a contradiction.

It remains to show that every black-and-white property can be expressed by an uncountable number of gray-property, silhouette-judge pairs. Given black-and-white property p and arbitrary $r \in \mathbb{R}_{[0,1]}$, construct a gray property g_r and silhouette judge j_r as follows:

$$g_r(x) = \begin{cases} r & \text{if } p(x) \\ 0 & \text{otherwise} \end{cases}$$

$$j_r(s) \Leftrightarrow (s(\infty) = r)$$

By construction, $p(x) \Leftrightarrow j_r(s_{x,g_r})$. Because there are uncountably many values of r, there are uncountably many pairs of (g_r, j_r) equivalent to p. □

Theorem 6. *There is a one-to-uncountably-many correspondence between black-and-white policies P and pairs of gray policies and target-system-silhouette judges (G, J) such that $\forall X \subseteq \mathcal{E}^\infty : P(X) \Leftrightarrow J(S_{X,G})$.*

Proof. Analogous to that of Theorem 5. □

These theorems demonstrate the increased expressiveness of gray policies, properties, and silhouette judges, compared to black-and-white policies and properties.

6 Future Work

Several directions exist for future work.

One would be to design and evaluate programming languages, or other tools, for specifying gray policies and silhouette judges. As a part of this direction, it would be interesting to consider case studies, to learn which sorts of gray policies and silhouette judges seem to be more common, or practically useful.

Another direction would investigate generalizations of existing program-verification techniques, to transition from determining whether programs obey black-and-white policies to determining how well programs obey gray policies.

It would also be interesting to consider ways in which the gray security model could benefit from results known in the area of fuzzy set theory. Intuitively, gray policies are to black-and-white policies what fuzzy sets are to sets: A fuzzy set is an ordered pair (U, m), where U is a set and $m : U \to \mathbb{R}_{[0,1]}$ is a membership function that describes the degree to which each element of U is a member of the set [62]. Because of the similarity between gray policies and fuzzy sets, much

of the work on fuzzy set theory is expected to translate to gray policies. For example, the "very" operator takes a fuzzy set (U, m) and returns the fuzzy set (U, m^2); such an operation is a simple way to make gray policies stricter.

Further generalizations of gray policies may also be possible. For example, rather than the totally ordered set of $\mathbb{R}_{[0,1]}$, gray policies could have complete lattices as their codomains. Some alterations would need to be made to the gray model to handle such codomains, including replacing infimum (supremum) operations with meet (join) operations.

Yet another direction is in the area of enforceability theory. As other work has delineated the black-and-white properties enforceable by various mechanisms (e.g., [11,24,25,41,43,54]), the same could be done for gray properties and/or silhouette judges. This direction of research would explore whether, and how well, different mechanisms (static code analyzers or runtime monitors, possibly constrained in various ways) can enforce classes of gray properties and/or silhouette judges.

References

1. Alpern, B., Schneider, F.B.: Defining liveness. Inf. Process. Lett. **21**(4), 181–185 (1985)
2. Alpern, B., Schneider, F.B.: Recognizing safety and liveness. Distrib. Comput. **2**, 117–126 (1987)
3. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of the Computer Security Foundations Symposium, pp. 265–279, June 2012
4. An, X., Jutla, D., Cercone, N.: Privacy intrusion detection using dynamic bayesian networks. In: Proceedings of the International Conference on Electronic Commerce, pp. 208–215 (2006)
5. Andersson, C., Lundin, R.: On the fundamentals of anonymity metrics. In: Fischer-Hübner, S., Duquenoy, P., Zuccato, A., Martucci, L. (eds.) The Future of Identity in the Information Society. The International Federation for Information Processing, vol. 262, pp. 325–341. Springer, USA (2008)
6. Andrés, M.E., Palamidessi, C., van Rossum, P., Smith, G.: Computing the leakage of information-hiding systems. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 373–389. Springer, Heidelberg (2010)
7. Asnar, Y., Giorgini, P., Massacci, F., Zannone, N.: From trust to dependability through risk analysis. In: Proceedings of the Conference on Availability, Reliability and Security, pp. 19–26, April 2007
8. Au, M.H., Kapadia, A.: PERM: practical reputation-based blacklisting without TTPs. In: Proceedings of the Conference on Computer and Communications Security, pp. 929–940 (2012)
9. Au, M.H., Kapadia, A., Susilo, W.: BLACR: TTP-free blacklistable anonymous credentials with reputation. In: Proceedings of the Symposium on Network and Distributed System Security (2012)
10. Balzarotti, D., Monga, M., Sicari, S.: Assessing the risk of using vulnerable components. In: Proceedings of the Workshop on Quality of Protection, pp. 65–77 (2006)

11. Basin, D., Jugé, V., Klaedtke, F., Zălinescu, E.: Enforceable security policies revisited. ACM Trans. Inf. Syst. Secur. **16**(1), 3:1–3:26 (2013)
12. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. Electron. Notes Theor. Comput. Sci. **249**, 75–91 (2009). Proceedings of the Conference on Mathematical Foundations of Programming Semantics
13. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. In: Montanari, U., Sannella, D., Bruni, R. (eds.) TGC 2006. LNCS, vol. 4661, pp. 281–300. Springer, Heidelberg (2007)
14. Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P.A., Wagner, G.M., Reninger, A.S.: Fuzzy multi-level security: an experiment on quantified risk-adaptive access control. In: Proceedings of the Symposium on Security and Privacy, pp. 222–230, May 2007
15. Clark, K., Singleton, E., Tyree, S., Hale, J.: Strata-Gem: risk assessment through mission modeling. In: Proceedings of the Workshop on Quality of Protection, pp. 51–58 (2008)
16. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Quantifying information flow with beliefs. J. Comput. Secur. **17**(5), 655–701 (2009)
17. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6), 1157–1210 (2010)
18. Clarkson, M.R., Schneider, F.B.: Quantification of integrity. Math. Struct. Comput. Sci. **25**(2), 207–258 (2015)
19. Clauß, S.: A framework for quantification of linkability within a privacy-enhancing identity management system. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 191–205. Springer, Heidelberg (2006)
20. Clauß, S., Schiffner, S.: Structuring anonymity metrics. In: Proceedings of the Workshop on Digital Identity Management, pp. 55–62 (2006)
21. Deng, Y., Pang, J., Wu, P.: Measuring anonymity with relative entropy. In: Dimitrakos, T., Martinelli, F., Ryan, P.Y.A., Schneider, S. (eds.) FAST 2006. LNCS, vol. 4691, pp. 65–79. Springer, Heidelberg (2007)
22. Devriese, D., Piessens, F.: Noninterference through secure multi-execution. In: Proceedings of the Symposium on Security and Privacy, pp. 109–124 (2010)
23. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 54–68. Springer, Heidelberg (2003)
24. Dolzhenko, E., Ligatti, J., Reddy, S.: Modeling runtime enforcement with mandatory results automata. Int. J. Inf. Secur. **14**(1), 47–60 (2015)
25. Drábik, P., Martinelli, F., Morisset, C.: Cost-aware runtime enforcement of security policies. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 1–16. Springer, Heidelberg (2013)
26. Drábik, P., Martinelli, F., Morisset, C.: A quantitative approach for inexact enforcement of security policies. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. LNCS, vol. 7483, pp. 306–321. Springer, Heidelberg (2012)
27. Dwaikat, Z., Parisi-Presicce, F.: Risky trust: risk-based analysis of software systems. In: Proceedings of the Workshop on Software Engineering for Secure Systems, pp. 1–7 (2005)
28. Edman, M., Sivrikaya, F., Yener, B.: A combinatorial approach to measuring anonymity. In: Proceedings of the Conference on Intelligence and Security Informatics, pp. 356–363, May 2007
29. Fong, P.W.L.: Access control by tracking shallow execution history. In: Proceedings of the Symposium on Security and Privacy, pp. 43–55 (2004)

30. Frigault, M., Wang, L., Singhal, A., Jajodia, S.: Measuring network security using dynamic bayesian network. In: Proceedings of the Workshop on Quality of Protection, pp. 23–30 (2008)
31. Gervais, A., Shokri, R., Singla, A., Capkun, S., Lenders, V.: Quantifying web-search privacy. In: Proceedings of the Conference on Computer and Communications Security, pp. 966–977 (2014)
32. Göktas, E., Athanasopoulos, E., Bos, H., Portokalidis, G.: Out of control: overcoming control-flow integrity. In: Proceedings of the Symposium on Security and Privacy, pp. 575–589 (2014)
33. Goriac, I.: Measuring anonymity with plausibilistic entropy. In: Proceedings of the International Conference on Availability, Reliability and Security, pp. 151–160, September 2013
34. Gowadia, V., Farkas, C., Valtorta, M.: PAID: a probabilistic agent-based intrusion detection system. Comput. Secur. **24**(27), 529–545 (2005)
35. Halpern, J.Y., O'Neill, K.R.: Anonymity and information hiding in multiagent systems. J. Comput. Secur. **13**(3), 483–514 (2005)
36. Heumann, T., Trpe, S., Keller, J.: Quantifying the attack surface of a web application. In: Proceedings of Sicherheit, vol. 170, pp. 305–316 (2010)
37. Howard, M., Pincus, J., Wing, J.M.: Measuring relative attack surfaces. In: Lee, D.T., Shieh, S.P., Tygar, J.D. (eds.) Computer Security in the 21st Century, pp. 109–137. Springer, Heidelberg (2005)
38. Alford, M.W., Hommel, G., Schneider, F.B., Ansart, J.P., Lamport, L., Mullery, G.P., Zhou, T.H.: Distributed Systems: Methods and Tools for Specification. An Advanced Course. LNCS, vol. 190. Springer, Heidelberg (1985)
39. Lee, A.J., Yu, T.: Towards quantitative analysis of proofs of authorization: applications, framework, and techniques. In: Proceedings of the Computer Security Foundations Symposium, pp. 139–153, July 2010
40. Leversage, D.J., Byres, E.J.: Estimating a system's mean time-to-compromise. IEEE Secur. Priv. **6**(1), 52–60 (2008)
41. Ligatti, J., Lujo, B., Walker, D.: Run-time enforcement of nonsafety policies. ACM Trans. Inf. Syst. Secur. **12**(3), 1–41 (2009)
42. Ligatti, J., Reddy, S.: A theory of runtime enforcement, with results. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 87–100. Springer, Heidelberg (2010)
43. Mallios, Y., Bauer, L., Kaynar, D., Ligatti, J.: Enforcing more with less: formalizing target-aware run-time monitors. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 17–32. Springer, Heidelberg (2013)
44. Mallios, Y., Bauer, L., Kaynar, D., Martinelli, F., Morisset, C.: Probabilistic cost enforcement of security policies. In: Accorsi, R., Ranise, S. (eds.) STM 2013. LNCS, vol. 8203, pp. 144–159. Springer, Heidelberg (2013)
45. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Trans. Softw. Eng. **37**(3), 371–386 (2011)
46. Manadhata, P., Wing, J., Flynn, M., McQueen, M.: Measuring the attack surfaces of two FTP daemons. In: Proceedings of the Workshop on Quality of Protection, pp. 3–10 (2006)
47. Mardziel, P., Alvim, M.S., Hicks, M., Clarkson, M.R.: Quantifying information flow for dynamic secrets. In: Proceedings of the Symposium on Security and Privacy, pp. 540–555 (2014)
48. Martinelli, F., Matteucci, I., Morisset, C.: From qualitative to quantitative enforcement of security policy. In: Kotenko, I., Skormin, V. (eds.) MMM-ACNS 2012. LNCS, vol. 7531, pp. 22–35. Springer, Heidelberg (2012)

49. McQueen, M.A., Boyer, W.F., Flynn, M.A., Beitel, G.A.: Time-to-compromise model for cyber risk reduction estimation. In: Gollmann, D., Massacci, F., Yautsiukhin, A. (eds.) Quality of Protection. Advances in Information Security, vol. 23, pp. 49–64. Springer, Heidelberg (2006)
50. Molloy, I., Dickens, L., Morisset, C., Cheng, P.-C., Lobo, J., Russo, A.: Risk-based security decisions under uncertainty. In: Proceedings of the Conference on Data and Application Security and Privacy, pp. 157–168 (2012)
51. Ngo, T.M., Huisman, M.: Quantitative security analysis for programs with low input and noisy output. In: Jürjens, J., Piessens, F., Bielova, N. (eds.) ESSoS. LNCS, vol. 8364, pp. 77–94. Springer, Heidelberg (2014)
52. Pamula, J., Jajodia, S., Ammann, P., Swarup, V.: A weakest-adversary security metric for network configuration security analysis. In: Proceedings of the Workshop on Quality of Protection, pp. 31–38 (2006)
53. Schneider, F.B.: Decomposing Properties into Safety and Liveness using Predicate Logic. Technical report 87–874, Cornell University, October 1987
54. Schneider, F.B.: Enforceable security policies. ACM Trans. Inf. Syst. Secur. **3**(1), 30–50 (2000)
55. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 41–53. Springer, Heidelberg (2003)
56. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
57. Specker, E.: Nicht konstruktiv beweisbare sätze der analysis. J. Symbolic Logic **14**, 145–158 (1949)
58. Verslype, K., De Decker, B.: Measuring the user's anonymity when disclosing personal properties. In: Proceedings of the International Workshop on Security Measurements and Metrics, pp. 2:1–2:8 (2010)
59. Xi, L., Feng, D.: FARB: fast anonymous reputation-based blacklisting without TTPs. In: Proceedings of the Workshop on Privacy in the Electronic Society, pp. 139–148 (2014)
60. Xi, L., Shao, J., Yang, K., Feng, D.: ARBRA: anonymous reputation-based revocation with efficient authentication. In: Chow, S.S.M., Camenisch, J., Hui, L.C.K., Yiu, S.M. (eds.) ISC 2014. LNCS, vol. 8783, pp. 33–53. Springer, Heidelberg (2014)
61. Yu, K.Y., Yuen, T.H., Chow, S.S.M., Yiu, S.M., Hui, L.C.K.: PE(AR)^2: privacy-enhanced anonymous authentication with reputation and revocation. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 679–696. Springer, Heidelberg (2012)
62. Zadeh, L.A.: Fuzzy sets. Inf. Control **8**(3), 338–353 (1965)