Algorithmic Fairness and Secure Information Flow
Extended Abstract

Bernhard Beckert, Michael Kirsten, and Michael Schefczyk

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
beckert@kit.edu, kirsten@kit.edu, michael.schefczyk@kit.edu

Motivation. The concept of enforcing secure *information flow* is well studied in computer science in the context of information security: If secret information may "flow" through an algorithm or program in such a way that it can influence the program's public output, this is considered insecure information flow, as attackers could potentially observe (parts of) the secret. There is a wide spectrum of methods and tools to analyse whether a given program satisfies a given definition of secure information flow.

We argue that there is a strong correspondence between secure information flow and algorithmic fairness: if protected attributes such as race, gender, or age are treated as secret program inputs, then secure information flow means that these "secret" attributes cannot influence the result of a program.

Contribution. In our paper, we investigate the relationship between secure information flow and non-discrimination from both a computer-science and an ethical point of view. We discuss the roles of various notions of secure information flow when they are re-interpreted as non-discrimination. We consider notions such as *conditional* information flow to express *declassification* and *quantified* information flow to distinguish the amounts of potentially disclosed secrets. We also discuss formal methods and tools that have been developed for the analysis and verification of secure information flow and how they can be adapted to analysing notions of non-discrimination, focusing on formal methods for the static analysis of programs. We describe some of the available tools and evaluate their application to examples from the context of algorithmic fairness.

Information Flow. The basic definition of secure information flow can be given as follows: Consider a program p with two inputs h and l and an output o, i.e., $o = p(h, l)$. The input h represents a secret, while l and o are publicly observable. Then, the program p is said to contain insecure information flow if there are two secret inputs h, h' and a public input l such that $p(h, l) \neq p(h', l)$. That is, the two secrets can be distinguished by observing the outputs. In contrast, a program is secure if $p(h, l) = p(h', l)$ for all values h, h', l.

When insecure information flow is re-interpreted as (possible) discrimination, we say that the program p is free of discrimination if $p(h, l) = p(h', l)$ for all values h, h', l, where now h and h' are protected attributes defining a protected group. Thus, the basic version of secure information flow corresponds to counterfactual fairness, where an individual must be treated equally for different values of h as long as their other features l remain unchanged.

Black-box vs. White-box Analysis. In computer science, there are two kinds of analyses for secure information flow: (1) Black-box approaches where information flow is solely defined and analysed based on the the input/output behaviour of a program, and (2) white-box approaches where the inner working of the program is considered for the notion of information flow, its analysis, or both.

In our paper, we focus on the latter: White-box methods can analyse programs independently of any input data; the input distribution does not need to be known or measured to decide whether a program uses protected data only in justified ways. They allow a deeper understanding and may explain why or in what way a program is potentially discriminating. However, currently available white-box methods are mostly restricted to programs written in a procedural language, while machine-learning-based programs are mostly unamenable to white-box analysis. Moreover, static analysis can identify discrimination based on proxy attributes only if the correlation between the protected and the proxy attribute is known, as the correlation is not encoded in the program.

Conditional Information Flow and Declassification. The basic version of secure information flow as defined above is too restrictive for many applications. Protected inputs are typically used in some way or another, as otherwise it would not make sense to have them as input at all. For information-flow control, the concept of allowing limited flows is introduced, called *declassification*. The limit may be on what part of the secret information is revealed, where in the program and when (under what condition) it is used, or whose permission is needed.

Consider, for example, an attacker who tries to guess a password and uses a password checker to learn whether their guesses are (by chance) correct. Information about the secret password is leaked by each check; but that may be considered acceptable as long as the number of guesses is limited.

As a practical example of declassification in the context of algorithmic fairness, consider a program making credit decisions. Even if an applicant's age is a protected attribute, a credit decision may justifiably depend on whether or not the time span until the credit is paid back is shorter than the span until the applicant retires. Thus, the age does interfere with the outcome, and the information flow in its basic form is insecure. In that case, one may formally specify and check what limited information about the applicant's age is used: the program may only check whether the following condition holds: age + duration $\leq$ retirement.

The decision if, under which conditions, and to what extent a declassification is justified and does not lead to discrimination should be taken at an early stage in the program's development process – before the program is implemented. The decision can then be formalised in a tool-readable language, such that the actual implementation can be automatically checked and compared to the requirement.

Quantified Information Flow. In information-flow control, there is also the notion of quantifying how much information from the secret may interfere with the output, which can also be re-interpreted in the context of algorithmic fairness. For example, one may measure the number resp. proportion of secret input values that may leak to the public

output. Another interesting measure is the proportion to which the result depends on a protected attribute (and whether that proportion is different for different values of the attribute).

Formal Methods and Tools for Analysing Information Flow. Secure information flow relates program runs for different inputs and cannot simply be tested by considering individual runs separately. We therefore focus on static analysis methods, e.g., formal verification, that are based on semantic (logic-based) and syntactic analysis of the programs without actually running them.