How They Did It: An Analysis of Emission Defeat Devices in Modern Automobiles

Moritz Contag*, Guo Li†, Andre Pawlowski*, Felix Domke‡,
Kirill Levchenko†, Thorsten Holz*, and Stefan Savage†

* Ruhr-Universität Bochum, Germany, {moritz.contag, andre.pawlowski, thorsten.holz}@rub.de
† University of California, San Diego, {gul027, klevchen, savage}@cs.ucsd.edu
‡ tmbinc@elitedvb.net

***Abstract*—Modern vehicles are required to comply with a range of environmental regulations limiting the level of emissions for various greenhouse gases, toxins and particulate matter. To ensure compliance, regulators test vehicles in controlled settings and empirically measure their emissions at the tailpipe. However, the black box nature of this testing and the standardization of its forms have created an opportunity for evasion. Using modern electronic engine controllers, manufacturers can programmatically infer when a car is undergoing an emission test and alter the behavior of the vehicle to comply with emission standards, while exceeding them during normal driving in favor of improved performance. While the use of such a defeat device by Volkswagen has brought the issue of emissions cheating to the public's attention, there have been few details about the precise nature of the defeat device, how it came to be, and its effect on vehicle behavior.**

In this paper, we present our analysis of two families of software defeat devices for diesel engines: one used by the Volkswagen Group to pass emissions tests in the US and Europe, and a second that we have found in Fiat Chrysler Automobiles. To carry out this analysis, we developed new static analysis firmware forensics techniques necessary to automatically identify known defeat devices and confirm their function. We tested about 900 firmware images and were able to detect a potential defeat device in more than 400 firmware images spanning eight years. We describe the precise conditions used by the firmware to detect a test cycle and how it affects engine behavior. This work frames the technical challenges faced by regulators going forward and highlights the important research agenda in providing focused software assurance in the presence of adversarial manufacturers.

I. Introduction

On September 18, 2015, the US Environmental Protection Agency (EPA) issued a notice of violation to the Volkswagen Group, accusing one of the world's largest automakers of circumventing the EPA's emissions tests [18], setting into motion the most expensive emissions scandal in history.

At the heart of the scandal is Volkswagen's use of a *defeat device*, defined by the EPA as any device that "reduces the effectiveness of the emission control system under conditions which may reasonably be expected to be encountered in normal vehicle operation and use," with exceptions for starting the engine, emergency vehicles, and to prevent accidents [19].

The defeat device in Volkswagen vehicles used environmental parameters, including time and distance traveled, to detect a standard emissions test cycle: if the engine control unit determined that the vehicle was not under test, it would disable certain emission control measures, in some cases leading the vehicle to emit up to 40 times the allowed nitrogen oxides [15].

Defeat devices like Volkswagen's are possible because of how regulatory agencies test vehicles for compliance before they can be offered for sale. In most jurisdictions, including the US and Europe, emissions tests are performed on a chassis dynamometer, a fixture that holds the vehicle in place while allowing its tires to rotate freely. During the test, a vehicle is made to follow a precisely defined speed profile (i.e., vehicle speed as a function of time) that attempts to imitate real driving conditions. The conditions of the test, including the speed profile, are both standardized and public, ensuring that the testing can be performed in a transparent and fair way by an independent party. However, knowing the precise conditions of the test also makes it possible for manufacturers to intentionally alter the behavior of their vehicles during the test cycle, a practice colloquially called "cycle beating."

While Volkswagen's cheating was breathtaking in scope (a dozen vehicle models spanning at least six years), it has also highlighted the difficulty of monitoring manufacturers' emission compliance. Meeting modern emissions standards is one of the main challenges faced by car manufacturers as emission standards become more stringent. In many cases, technological limitations put compliance in conflict with consumer demands for performance, efficiency, or cost—creating a powerful incentive for car makers to evade the regulatory burden. At the same time, automobiles have grown in complexity: the modern automobile is a complex cyber-physical system made up of many electronic components, making it as much a software system as a mechanical one. A premium-class automobile, for example, can contain more than 70 electronic control units and 100 million lines of code [4]. As a part of this trend, nearly all aspects of engine operation are controlled by an Engine Control Unit (ECU), an embedded system creating a closed control loop between engine sensors and actuators. This allows manufacturers to precisely control all aspects of engine operation and thus drive significant improvements in performance, reliability, and fuel economy. The ECU is also responsible for ensuring that the vehicle complies with the emissions requirements imposed by governmental regulatory bodies. Indeed, while some emission control measures, like the catalytic converter or particulate filters, are passive, many others require *active* control by the ECU, which must sometimes sacrifice performance or efficiency for compliance. These tradeoffs are particularly challenging for diesel engines, which in their simplest form are noisier and emit more particulates and nitrogen oxides (NO_x) than gasoline engines [3].

Electronic engine control has also made it easier to circumvent emissions testing by implementing a defeat device in software. The black box nature of emissions testing makes it nearly impossible to discover such a software-based defeat device during a test, forcing regulators to rely on heavy fines to discourage cheating. Unfortunately, as the Volkswagen case illustrates, it can take many years to discover such a defeat device. Given the ultimate limitations of testing, we are led to consider whether we can detect defeat devices using *software verification* techniques. Unfortunately, verifying complex software systems is a difficult problem in its own right, more so for a *cyber-physical* system like a modern automobile. In our case, the setting is also *adversarial*—rather than trying to find bugs, we are looking for intentional attempts to alter a system's behavior under test conditions. This paper aims to be a first step in cyber-physical system verification in an adversarial setting with two case studies of automobile defeat devices and binary analysis techniques to identify verification-critical code elements across multiple software revisions.

We begin with two case studies of software defeat devices found in light diesel vehicles. The first set belongs to automobiles produced by the Volkswagen Group, which has publicly admitted to their use. The Volkswagen defeat device is arguably the most complex in automotive history. Unfortunately, there are few technical details available to the public about its operation, its effect on engine behavior, and how its design evolved over time; our paper closes this gap and we believe helps highlight the key challenges for regulators going forward. Unfortunately, Volkswagen is not alone in evading emissions testing. Fiat Chrysler Automobiles (FCA) is currently being investigated in Europe because recent road test data showed significantly higher emissions than in regulatory compliance tests [17]. In this paper, we identify and describe a timer-based defeat device used in the Fiat 500X automobile. We believe we are the first to publicly identify this defeat device.

Both the Volkswagen and Fiat vehicles use the EDC17 diesel ECU manufactured by Bosch. Using a combination of manual reverse engineering of binary firmware images and insights obtained from manufacturer technical documentation traded in the performance tuner community (i. e., car enthusiasts who modify their software systems to improve performance), we identify the defeat devices used, how they inferred when the vehicle was under test, and how that inference was used to change engine behavior. Notably, we find strong evidence that both defeat devices were *created by* Bosch and then *enabled by* Volkswagen and Fiat for their respective vehicles.

To conduct a larger study, we used static code analysis techniques to track the evolution of the defeat device across hundreds of versions of vehicle firmware. More precisely, we developed a static analysis system, called CurveDiff, to automatically discover the Volkswagen defeat device in a given firmware image and extract the parameters determining its operation. Overall, we analyzed 926 firmware images and successfully identified 406 potential defeat devices inside these images. Further, we automatically verified the effects on one particular subsystem.

In summary, our contributions are as follows:

- We provide a detailed technical analysis of defeat devices present in vehicles marketed by two independent automobile manufactures, Volkswagen Group and Fiat Chrysler Automobiles, whose effect is to circumvent emission tests in the US and Europe.
- We design and implement a static binary analysis tool called CurveDiff for identifying such defeat devices in a given firmware image, which enables us to track the evolution and behavior of circumvention code across a large number of firmware images.
- We use our tool to study the evolution of the defeat devices and its effect on engine behavior across eight years and over a dozen vehicle models.

However, more than these detailed technical contributions, we believe the broader impact of our work is to articulate the challenge of certifying regulatory compliance in the cyber-physical environment. Today's black box testing is costly and time consuming and, as these cases show, can be easily circumvented by defeat device software that "tests for the tester." The gap between black box testing and modern software assurance approaches drives a critical research agenda going forward that will only become more important as regulators are asked to oversee and evaluate increasingly complex vehicular systems (e. g., autonomous driving). We believe that concrete examples, such as those we describe in this paper, are key to ground this discussion and make clear the realistic difficulties faced by regulators.

The remainder of this paper is organized as follows. Section II provides the necessary technical background for the rest of the paper, followed by a discussion of the available data sets in Section III, and a detailed description of the defeat devices we found in Section IV. We explain how we implement this detection at scale in Section V followed by a summary of the results we find using this tool. Finally, we discuss the implications of our finding in Section VII and then conclude with Section VIII.

II. Technical Background

In the following, we provide a brief overview of the technical concepts needed to understand the rest of this paper.

A. Diesel Engines

The distinguishing difference between a gasoline and diesel engine is the manner in which combustion is initiated. In a gasoline engine, a mixture of air and fuel is drawn into the combustion cylinder and ignited by a spark. In a diesel engine, air is drawn into the combustion cylinder and, at a critical point in the compression cycle, fuel is injected into the cylinder, igniting in the compressed air. Thus, in a gasoline engine, fuel and air are mixed *before* being drawn into the cylinder and ignited, whereas in a diesel engine, fuel and air are mixed *at the time of ignition*, resulting in an imperfect and inhomogeneous mixture. This is responsible for many of the diesel engine's distinctive characteristics, including the black smoke and heavy knocking sound known as "diesel knock." The black smoke, made up of particulate matter, also called soot, results from the incomplete combustion of the fuel and is subject to strict limits in light-duty diesel vehicles. The second major pollutant in diesel exhaust are nitrogen oxides (NO and NO_2, abbreviated NO_x). Current emission standards impose tight limits on the amount of particulate matter and NO_x emitted and require special steps to limit their levels. The vehicles that are the subject of this work rely on the following emission control devices to achieve regulatory conformance.

EGR. Exhaust Gas Recirculation (EGR) is an emission control scheme where exhaust gas is recirculated back into the engine intake. EGR significantly reduces the amount of NO_x in the exhaust [12], [16]. Unfortunately, EGR also increases the amount of particulate matter in the exhaust, leading to a trade-off between NO_x and particulate matter.

NSC. A NO_x Storage Catalyst (NSC), also called a Lean NO_x Trap (LNT), works by oxidizing NO to NO_2 and then storing NO_2 in the catalyst itself. The storage capacity of the catalyst is limited, lasting from 30 to 300 seconds, after which it must be regenerated. To regenerate the catalyst, the engine switches to a rich fuel-air mixture for 2 to 10 seconds. During regeneration, the engine is less efficient, decreasing fuel economy [16]. A rich fuel-air mixture also increases particulate matter production, again trading off NO_x emissions for particulate emissions.

SCR. Selective Catalyst Reduction (SCR) is an alternative to NSC for reducing NO_x emissions that works by injecting urea into the exhaust stream. SCR is more effective than NSC (described above) and is usually used in 3-liter diesel engines and larger. The drawback of SCR is its increased complexity and the need to carry and replenish the urea fluid (als known by its trademark name AdBlue). Several Volkswagen vehicles implicated in the emission cheating scandal are reported to limit urea injection levels outside of a test cycle. Except for results reported in Table II, this paper does not cover defeat devices that manipulate SCR.

DPF. A Diesel Particulate Filter (DPF) traps particulates (soot), greatly reducing the amount of black smoke leaving the tailpipe. While the DPF is highly effective at trapping particulates, as the amount of particulates accumulates, the resistance to air flow increases also, increasing the load on the engine. To purge the DPF of accumulated deposits, it must undergo a regeneration cycle approximately every 500 km, lasting 10 to 15 minutes. DPF regeneration requires high exhaust temperatures that are usually only achieved at full load. If the vehicle is operated at full load, the DPF will regenerate on its own. Unfortunately, these conditions may not arise in normal urban driving, requiring the ECU to perform *active regeneration*. In this mode, the ECU adjusts engine operation to increase exhaust temperature to regenerate the DPF; however, if the vehicle is only driven for short distances, such a temperature may never be reached. At sufficiently high soot load, the vehicle will illuminate a special warning lamp, prompting the driver to drive the vehicle at increased speed to allow active regeneration to take place. If this does not happen, the DPF will require service [21]. Thus, while the DPF is highly effective at reducing particulate emissions, it imposes a performance penalty and can become a hassle for the owner who drives the vehicle for short distances. Moreover, according to the New York Attorney General's complaint [15], at normal load Volkswagen's DPF could only last 50,000 miles before needing replacement, far short of the 120,000 mile standard Volkswagen was required to meet, compelling Volkswagen to reduce wear on the DPF.

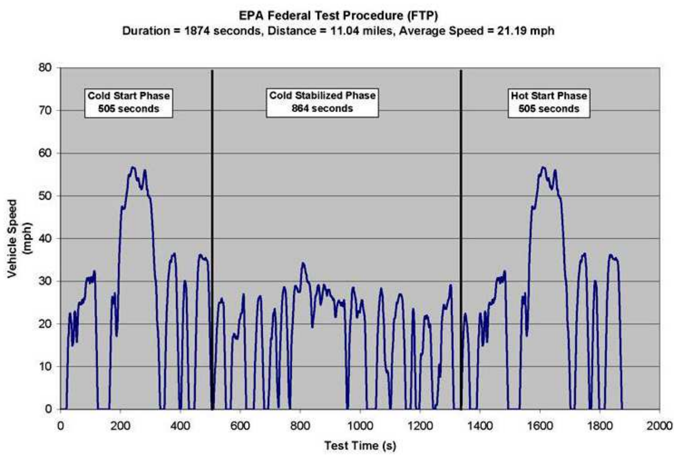


Fig. 1: FTP-75 (Federal Test Procedure) driving cycle depicting the speed over time. Image taken from EPA [20].

B. Emission Test Cycles and Emission Standards

An *emission test cycle* defines a protocol that enables repeatable and comparable measurements of exhaust emissions to evaluate emission compliance. The protocol specifies all conditions under which the engine is tested, including lab temperature and vehicle conditions. Most importantly, the test cycle defines the speed and load over time that is used to simulate a typical driving scenario. An example of a driving cycle is shown in Figure 1. This graph represents the FTP-75 (Federal Test Procedure) cycle that has been created by the EPA and is used for emission certification and fuel economy testing of light-duty vehicles in the US [7]. The cycle simulates an urban route with frequent stops, combined with both a cold and a hot start transient phase. The cycle lasts 1,877 seconds (about 31 min) and covers a distance of 11.04 miles (17.77 km) at an average speed of 21.2 mph (34.12 km/h).

Table IV in the Appendix lists the main test cycles used for exhaust emission tests of light-duty vehicles in different regions of the world. Besides urban test cycles such as FTP-75, there are also cycles that simulate driving patterns under different conditions.

To assess conformance, several of these tests are carried out on a chassis dynamometer, a fixture that holds a car in place while allowing its drive wheel to turn with varying resistance. Emissions are measured during the test and compared to an *emission standard* that defines the maximum pollutant levels that can be released during such a test. In the US, emissions standards are managed on a national level by the EPA. In addition, California has its own emissions standards defined and enforced by the California Air Resources Board (CARB). California standards are also used by a number of other states, together with California covering a significant fraction of the US market, making them a de facto second national standard. In Europe, the emission standards are called Euro 1 through Euro 6, where Euro 6 is the most recent standard in effect since September 2014.

C. Electronic Engine Control

In a typical modern car, there are 70-100 electronic control units [4], [8] that are responsible for tasks such as the human-machine interface as part of the infotainment system, a speed control unit, a telematic control unit, or brake control modules. Among these is the Engine Control Unit (ECU), which is responsible for the operation of the engine. The subject of this work is the Bosch EDC17 ECU used in many diesel light passenger vehicles, and in all of the vehicles implicated in the Volkswagen diesel emissions scandal. At its core, the ECU implements a closed control loop by periodically reading sensor values, evaluating a control function, and controlling actuators based on the control signal.

Sensors. To control engine behavior, the ECU relies on a multitude of sensors readings, including crankshaft position; air pressure and temperature at several points in the intake; intake air mass; fuel, oil, and coolant temperature; vehicle speed; exhaust oxygen content (lambda probe); as well as driver inputs such as the accelerator pedal position, brake pedal position, cruise control setting, and selected gear.

Control functions. Based on the sensor inputs, the ECU implements different functions to control and influence the combustion process by interpreting the input data. In a diesel engine, one of the most important control values is the fuel injection timing that defines when and for how long the fuel injectors remain open in the engine cycle. As noted earlier, injection timing affects engine power, fuel consumption, and the composition of the exhaust gas. The ECU also determines how much of the exhaust gas should be recirculated and how much urea should be injected into the exhaust to catalyze nitrogen oxides.

Actuators. The ECU uses the computer control signals to directly control several actuators, most notably the fuel injector valves and air system valves, including the EGR valve.

Communication. The ECU also communicates with other systems inside the car, for example to display the current engine speed RPM signal or light up diagnosis lamps. Furthermore, status information about the ECU is sent via an interface such as the On-Board-Diagnostics (OBD-II) system and the ECU can also communicate with other control units via the CAN bus.

D. Business Relationships

The EDC17 ECU is manufactured by Bosch and bought by automakers, including Volkswagen and Fiat, to control their diesel engines. The exact details of the business relationship between Bosch and its customers is not public; however, media reports, court filings [15], and the documentation we have obtained indicates the following basic structure: Bosch builds the ECU hardware and develops the software running on the ECU. Manufacturers then specialize an ECU for each vehicle model by calibrating characteristic firmware constants whose semantics are explained in the ECU documentation. We have found no evidence that automobile manufacturers write any of the code running on the ECU. All code we analyzed in this work was documented in documents copyrighted by Bosch and identified automakers as the intended customers.

E. Related Work

Unfortunately, there is little technical documentation about defeat devices that is publicly available. Domke and Lange were the first to present several technical insights into the defeat device used in a Volkswagen Sharan [9], [10]. We leverage these analysis results and adopted a similar methodology to identify defeat devices. The New York Attorney General's compaint against Volkswagen AG [15] contains several general insights into defeat devices, but does not provide any technical details. Fiat Chrysler Automobiles (FCA) is currently being investigated in Europe [17] and to the best of our knowledge, we are the first to document how this defeat device is implemented.

III. DATASET

In this paper, we focus on the EDC17 ECU manufactured by Bosch. This diesel engine ECU was used in the cars implicated in the Volkswagen emission scandal as well as the Fiat 500X. We rely on three data sources for our analysis of ECUs and affected vehicles which we describe below.

A. Function Sheets

Function sheets (called *Funktionsrahmen* in German) document the functional behavior of a particular release of the ECU firmware. The function sheets describe each software functional unit of the ECU using a formal block diagram language that precisely specifies its input/output behavior, along with some additional explanatory text. The block diagram and text documentation also names the variables and calibration constants used by the functional unit. Car makers tune the behavior of the ECU by changing these calibration constants. In the Bosch function sheets, scalar calibration constants are identified by the `_C` suffix, one-dimensional array constants by the `_CA` suffix, and higher-dimensional arrays by the `_MAP` suffix. Further, curve definitions use the suffix `_CUR`.

Function sheets are generally not available to the public, however, many make their way into the automobile performance tuning community. All of the function sheets used in this work have been obtained from such tuner sites. All figures throughout the paper are derived from these function sheets that are already publicly available.

Authenticity. Since we did not obtain the function sheets directly from the ECU manufacturer (Bosch), we cannot be absolutely certain of their authenticity. Nevertheless, all function sheets used in this work bear a "Robert Bosch GmbH" copyright and show no evidence of alteration by a third party. Indeed, we have not encountered any function sheets that show any signs of content tampering in the wild. We have also explicitly verified that key functional elements, like the Volkswagen "acoustic condition" described in Section IV-A, match the code in the firmware.

B. A2L and OLS Files

The automotive industry uses the ASAM MCD-2 MC [1] file format, commonly called A2L, to communicate elements of a firmware image that a car manufacturer must modify in the calibration process. Generally speaking, an `.a2l` file is comparable to a `.map` or `.pdb` file used by developers on the Linux or Windows platform, respectively. While all of these file types map debugging symbols to concrete addresses, `.a2l` files can also give contextual information beyond mere symbol names. The format is developed to "support ... automotive-specific processes and working methods" [1]. Consequently, additional metadata used to describe an address (i. e., an ECU variable) may include axis descriptions for lookup tables, information about the byte order, or unit conversion formulas. An example is given in Listing 1 in the Appendix.

Given that `.a2l` files contain lots of details and insights into a given ECU, they are typically only available for people working on engine development, calibration, and maintenance. However, car tuning enthusiasts also regularly get hold of these files and trade them at online forums. In order to understand the inner workings of certain ECU firmware images in more detail, we obtained access to such files. When we were not able to obtain a `.a2l` file for a given firmware image, we focused on binary code only and leveraged insights gained from similar ECUs to bootstrap our analysis.

In some cases, we relied on OLS files, an application format used by the WinOLS software used to change configuration values in firmware. The OLS format contains both a firmware image and elements of the A2L file annotating calibration constants.

Authenticity. As with function sheets, we did not obtain A2L files used in this work from Bosch or the car maker, and so cannot guarantee their authenticity with absolute certainty. Each A2L file is paired with a specific firmware image; we confirmed their match before using the A2L to extract values from the image. We used A2L to identify variables and constants in code extracted from the firmware. Examining the context in which a value thus served as a kind of sanity check.

C. Firmware images

We also obtained firmware images from various sources. Similar to `.a2l` files, firmware images are also circulated in the car tuning community. We obtained several images from the tuner community. We also obtained images from the *erWin* portal ("electronic repair and workshop information"), a platform operated by Volkswagen that provides access to official firmware images for car repair shops. The portal provides archives containing firmware updates up to a certain date. Every image is named after its software part number and revision, allowing us to uniquely identify it. The timestamp is roughly equivalent to the release date of the firmware.

Unfortunately, the images contain no additional metadata such as the actual model in which the firmware is deployed. We used online portals offered by aftermarket automobile part vendors to determine which vehicles a firmware image was used on.

Authenticity. Firmware data for VW, Audi, Seat and Skoda is obtained from the *erWin* portal, operated by Volkswagen. The newest image is dated October 11, 2016. We also obtained Volkswagen group images dated 2009–2010 from various online sources. We only included images for which *Freigabeschein* (street release certification) documents allowed us to obtain information about both release date and car model. We obtained the Fiat 500X OLS file from a tuning site. It was sold to us as an original (unmodified) image. Our main findings based on this OLS file align with the test results of the German KBA [22].

IV. Defeat Devices

A *defeat device* is a mechanism that causes a vehicle to behave differently during an emission test than on the road.[1] Conceptually, a defeat device has two components:

- **Monitor.** Determine if observed conditions *rule out* an emission test, and
- **Modify.** Alter vehicle behavior when not under test.

Defeat devices rely on any number of external or internal variables to detect that a test is taking place. From 1991 to 1995, for example, General Motors used the fact that air conditioning was turned on in its Cadillac automobiles to rule out a test cycle—at the time, emission testing was done with air conditioning turned off—making the air-fuel mixture richer to address an engine stalling problem, but also exceeding CO emission limits [14]. General Motors was fined $11 million and forced to recall all affected vehicles.

As the Cadillac example suggests, the monitoring element of a defeat device does not need to be perfect, so long as

[1]More precisely, the US Code of Federal Regulations defines a *defeat device* as "an auxiliary emission control device (AECD) that reduces the effectiveness of the emission control system under conditions which may reasonably be expected to be encountered in normal vehicle operation and use, unless: (1) Such conditions are substantially included in the Federal emission test procedure; (2) The need for the AECD is justified in terms of protecting the vehicle against damage or accident; (3) The AECD does not go beyond the requirements of engine starting; or (4) The AECD applies only for emergency vehicles ..." (40 CFR § 86.1803-01). European regulations follow a very similar definition.

Min	Max	Unit	Signal	Description
−50	140	°C	`InjCrv_tClntEngNs_mp`	Coolant temperature
−50	140	°C	`FuelT_t`	Fuel temperature
−50	140	°C	`Oil_tSwmp`	Oil temperature
795	—	hPa	`EnvP_p`	Atmospheric pressure
`true`			`StSys_stStrt`	Engine starting

TABLE I: Initial conditions activating the acoustic condition in the EDC17C54 firmware. parameters taken from firmware part number 03L906012F. If all conditions hold, the *set* signal to the outer (topmost) flip-flop in Figure 2 is asserted.

its error is one-sided. Like the Cadillac device, the defeat devices we found assume that the vehicle is under test unless some internal or external variable allows it to *rule out* an ongoing test. Then, when the monitoring element signals that the observed variables are not consistent with any known test cycle, the vehicle can switch to an operating regime favored by the manufacturer for real driving rather than the clean regime necessary to pass the emission test.

In the remainder of this section, we describe the defeat devices used by Volkswagen and Fiat to circumvent emission testing and their effect on vehicle behavior. Our description is based on function sheets for the ECU, reverse engineering of the firmware, and publicly available information, notably the Complaint filed by the State of New York against Volkswagen and its US subsidiaries [15].

A. The Volkswagen Device: Test Detection

The Volkswagen defeat device is a continually evolving family of devices. All instances are organized around a single condition monitoring block that determines if the vehicle is undergoing testing and points throughout emission-related ECU modules where the result of this determination can affect the behavior of the module. The monitoring element of the Volkswagen defeat device is encapsulated in a function block that computes the status of the *kundenspezifische Akustikbedingung*, which translates to "customer-specific acoustic condition." (Here, *customer* refers to the automaker, namely, Volkswagen.) The outcome of the computation is represented by the signal/variable `InjCrv_stNsCharCor` (`stNsCharCor` for short). This signal is then used at many points in the ECU to alter the behavior of the engine. Figure 2 shows the logic block responsible for computing the acoustic condition. (The Figure is taken from the function reference sheet created by Bosch.) The value `stNsCharCor` $= 0$ means that the ECU considers itself to be in normal driving mode, while `stNsCharCor` $= 1$ indicates testing (emmissions-compliant) mode.

Activating conditions. The state of the acoustic condition is stored in the top flip-flop in the figure ①. The *set* signal to the flip-flop is true if all of a set of five conditions are true. These conditions are shown in Table I. Note that the last condition, engine starting, is only true when the engine is starting and is false during normal operations. If the engine runs in normal mode (i. e., has not recently been started), has exceeded a velocity of, e. g., $9.5\ \mathrm{km/h}$ at some point, and pressure and temperature match the aforementioned boundaries, the function proceeds with the actual cycle checking. Otherwise, the engine stays in the same mode. The effect of this is that the acoustic condition can only be set if coolant temperature, fuel temperature, oil temperature, and atmospheric pressure are within the prescribed limits when the car starts ②. If any of the four parameters is outside the required range, an ongoing emissions test is ruled out and the acoustic condition is never activated. However, we note that these conditions are easily satisfied in both testing and real-world scenarios.

If the acoustic condition is set at startup, it may be canceled by meeting several conditions that rule out a test. We call these the *deactivating* conditions. If any of these conditions are met, the inner flip-flop is set ③. The output of the inner flip-flop asserts the *reset* signal of the outer flip-flop, setting the acoustic condition variable `stNsCharCor` to zero. There are four deactivating conditions any one of which, if true, sets the inner flip-flop that in turn sets `stNsCharCor` to zero, indicating the vehicle is in normal driving mode.

Deactivating conditions. There are four deactivating conditions ④. The first deactivates the acoustic condition if the engine has started and a configurable time period `InjCrv_tiNsAppVal_C` has elapsed since the accelerator pedal position first exceeded a configurable threshold `InjCrv_rNsAppVal_C`. The second deactivates the acoustic condition if the engine revolution counter exceeds a configurable threshold `InjCrv_ctNsStrtExtd_C`. The third deactivation condition, if the acoustic condition is inhibited, is never triggered.

Until about May 2007, there were only three deactivation conditions, as described above. Of the firmware images available to us, the fourth condition first appears in a firmware image dated May 2007 for EDC17CP04 P 617. It starts by computing a time and distance measurement. The time measurement, call it t, is computed by measuring the time since the vehicle first exceeded a configurable velocity `InjCrv_vThres_C`. The distance measurement is the distance in the current driving cycle, call it d.

Test cycle curves. The acoustic condition logic computes a pair of points d_{lower} and d_{upper} using two linearly interpolated curves. These curves, which define a function of t using a small number of points, are configurable by the manufacturer. In this case, there are seven pairs of curves, giving seven pairs of values d_{lower} and d_{upper} computed for the current value t. If d is ever less than d_{lower} or greater than d_{upper}, the flip-flop corresponding to the pair of curves is set and remains set indefinitely ⑤.

The output of this flip-flop indicates that the vehicle has strayed outside the prescribed time-distance profile defined by the pair of curves. Hence, the curves describe an upper and lower bound on the covered distance. The flip-flop allows the logic to remember this, and at any given time, the state of the flip-flops indicate whether the vehicle has so far stayed within the prescribed time-distance profile defined by the pair of curves. If all seven flip-flops are set, then the vehicle has strayed outside the profile of each of the curves at least once

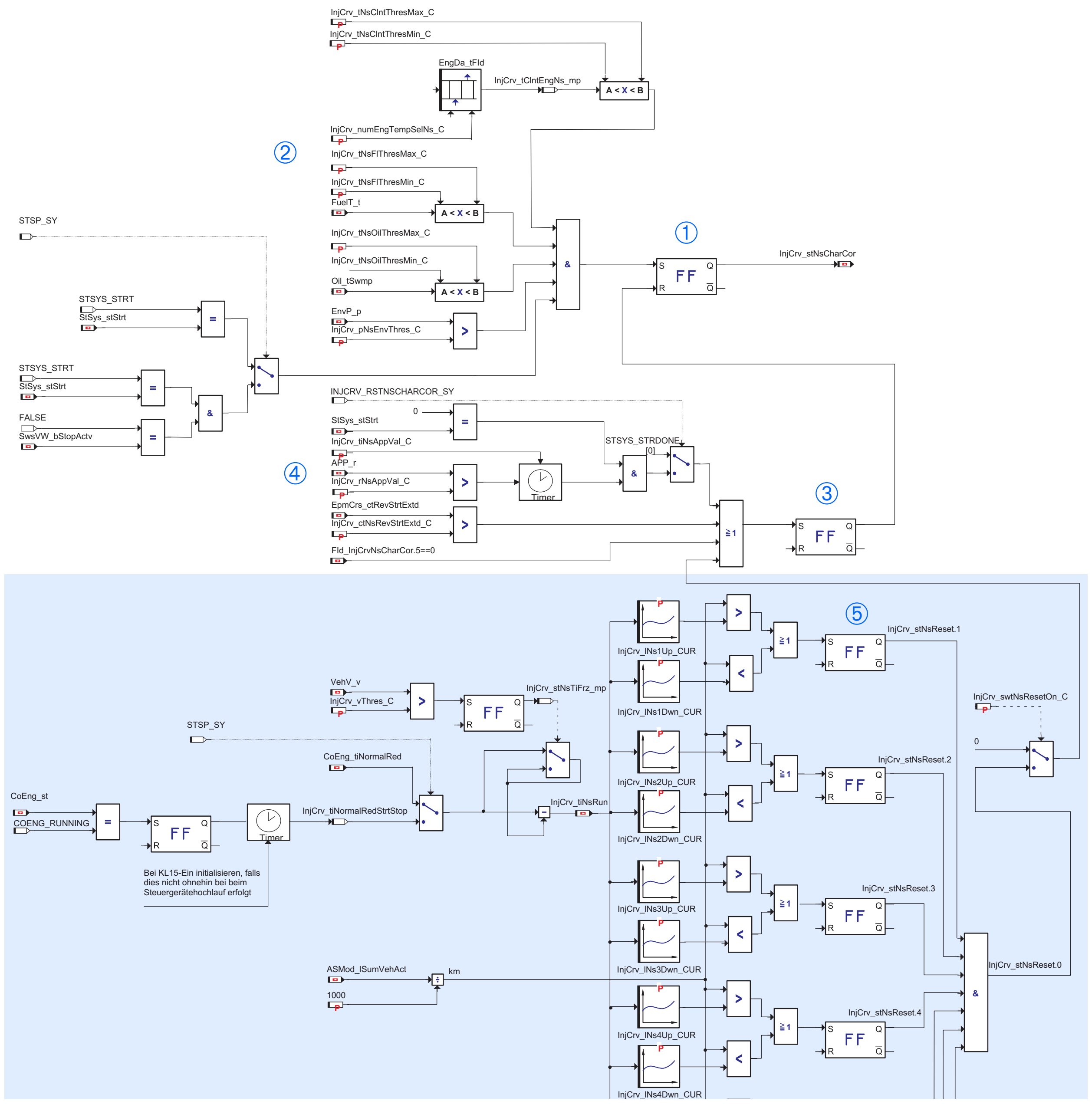


Fig. 2: Acoustic condition activation logic from function sheet EDC17C54 P 874 for, e. g., VW Passat, dated December 2009. The portion shaded light blue disables the so-called “customer-specific acoustic condition” if the distance traveled as a function of time falls outside of all 7 possible test cycle speed profiles. The highlighted portion does not appear in function sheets prior to EDC17C04 P 617, dated May 2007. Copyright Robert Bosch GmbH. Diagram cut at fourth test curve, continues up to seven below. Blue shading and numbers ① through ⑤ added by authors.

since start. If this happens, the acoustic condition is canceled.

In the firmware we examined, these curves are used to define the precise time-distance profile of known emission tests. In general, we note that the number of profiles has been increasing with time. As shown in Table II, the number of curves checked has increased from 0 in EDC17CP04 P 531 to 7 in EDC17CP44 P 859.

Figure 3 shows several curve pairs found in the firmware of an EDC17C54 ECU (software part number 03L906012, revision 7444; remaining curves found in Figure 14 in the Appendix). The area outside of the upper and lower boundaries d_{lower} and d_{upper} as defined by curves is shaded. If the computed time and distance value (t, d) ever enters this gray area, the test is considered to be ruled out, and the corresponding flip-flop is set. In addition to the boundaries, we have plotted the test cycles of known emission tests given in Table IV. Test cycles matching the profile are shown using heavy lines; all others using light lines. As our results show, several of the configured boundaries match a known test cycle quite closely. In particular, profile 1 matches the FTP-75 test cycle to within 4.2 km and profile 5 the HWFET test cycle to within 16.1 km (not fully shown in the figure).

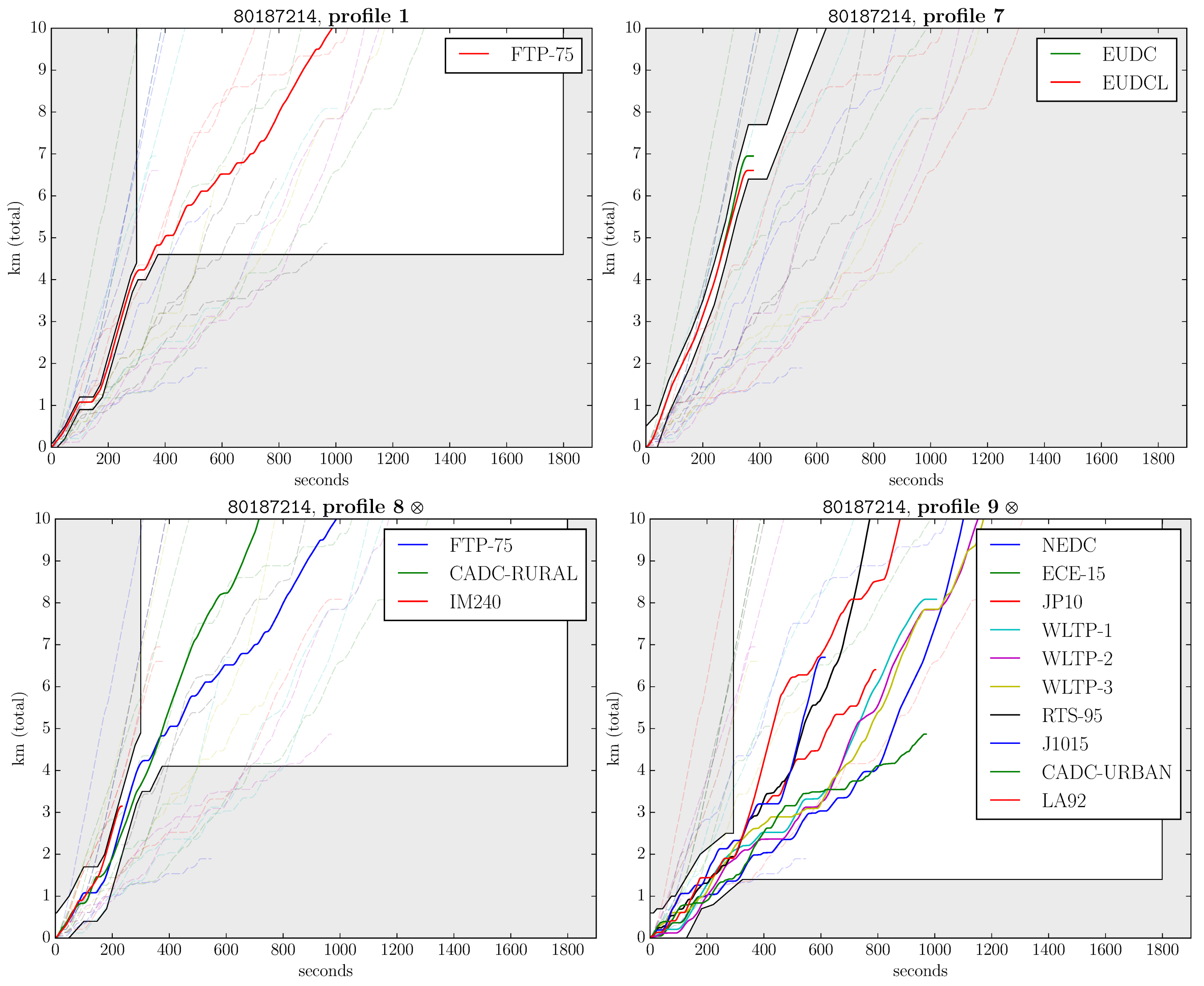


Fig. 3: Selection of curve checks testing against various emissions test cycles in the firmware for a VW Passat, released 12/2014 (EDC17C54, software part number 03L906012, revision 7444). The area in which the software reports the driving profile to match is colored white. As evident from the graphs, this area tends to follow a test cycle closely. The legend lists the known matching test cycles. ⊗ indicates an additional steering wheel check. Figure 14 in the Appendix depicts the remaining curves.

Steering wheel checks. Starting 2009, the EDC firmware used in Volkswagen automobiles increased the number of profiles check from 7 to 10. As noted, the profiles shown in Figure 3 were extracted from an EDC17C54 firmware image, VW part number 03L906012, that has 10 profiles, four of which are shown in the figure. (The acoustic condition illustrated in Figure 2, from an older EDC17C54 function sheet, shows only 7 profiles.)

Note that profiles 8 and 9 are considerably less precise than profiles 1 and 7. In fact, profile 9 matches a total of 10 known emissions test cycles. In addition to checking the time-distance relation shown in Figure 3, profiles 8, 9, and 10 also included a steering wheel angle check: in addition to deviating from a prescribed time-distance profile, the flip-flop corresponding to profiles 8, 9, and 10 would be reset if the steering wheel angle deviated by more than 20° from the neutral position. Unfortunately, we could not obtain a function sheet containing this improved acoustic condition. The following code is our reconstruction of the logic used to determine if a steering wheel angle deviation should cancel the acoustic condition.

```
if (-20 /* deg */ < steeringWheelAngle &&
    steeringWheelAngle < 20 /* deg */) {
    lastCheckTime = 0;
    cancelCondition = false;
} else {
    if (lastCheckTime < 1000000 /* microsec */) {
        lastCheckTime = lastCheckTime + dT;
        cancelCondition = false;
    } else
        cancelCondition = true;
}
```

In the updated firmware, the signal `cancelCondition` computed as above is applied as a third input to the ≥ 1 gates that lead into the flip-flops corresponding to each profile.

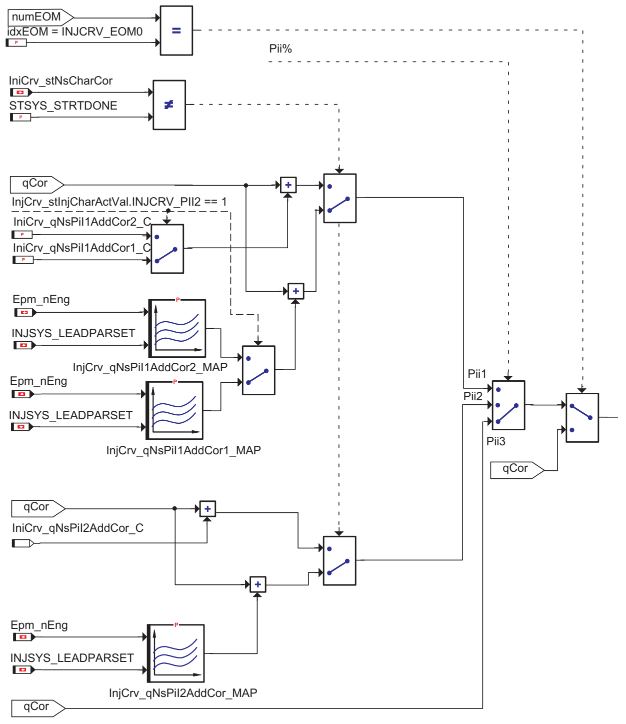


Fig. 4: Acoustic condition (signal `InjCrv_stNsCharCor`) used to modify the desired injection quantity correction `qCor`. From the EDC17C54 P 874 function sheet. Copyright Robert Bosch GmbH.

If the sophisticated test cycle detection logic encoded above sets the acoustic condition to one, a number of behavior changes take place in the vehicle. These changes are effected by using the value of the acoustic condition variable `stNsCharCor` to switch between configurable variables or parameters, allowing the vehicle to operate with one set of calibration values in the normal driving mode and using another set in testing mode, as if two distinct personalities took turns controlling the vehicle. Next, we describe the two contrasting personalities that emerge depending on the state of the acoustic condition.

B. The Volkswagen Device: Effect on Injection

As described in Section II-A, the operation of a diesel engine is controlled through fuel injection timing. The start and duration of injections not only affects engine power output but also exhaust emissions. Based on its naming and first use, the acoustic condition was introduced to alter engine injection behavior [15]. We identified several points in the firmware we analyzed manually where the acoustic condition can modify fuel injection behavior. Here, we describe how the acoustic condition can be used to adjust the quantity of injection.

Figure 4 shows how the fuel injection quantity (additive) correction (`qCor`) is modified by the acoustic condition. If the acoustic condition is true (under test), `qCor` is modified by adding a constant (`InjCrv_qNsPiI1AddCor{1,2}_C` or `InjCrv_qNsPiI2AddCor_C`). Otherwise, `qCor` is modified by adding a value computed based on engine speed (`Epm_nEng`). The function sheet describes this logic block as "*Berechnung zusätzlicher (kundenspezifischer) Korrekturen für die Voreinspritzungen*" (Calculation of additional (customer-specific) corrections for the pilot injections.).

C. The Volkswagen Device: Effect on EGR

As noted earlier, Exhaust Gas Recirculation (EGR) is a very effective means of reducing NO_x levels in the exhaust gas. Unfortunately, the beneficial effect on NO_x has the opposite effect on particulate matter: decreasing NO_x emissions by increasing the amount of exhaust gas recirculated increases the amount of soot in the exhaust. This, in turn, increases load on the Diesel Particulate Filter (DPF) used to reduce soot emissions. The acoustic condition can also be used to alter the amount of exhaust gas recirculated (see Figure 11 in the Appendix). The logic block shown in the figure is used to compute `mDesVal1Cor`, a correction value to the total desired air mass. The correction may be applied additively or multiplicatively, based on a configurable parameter, to the base amount to arrive at the desired air mass value (this calculation is not shown in the figure).

D. The Fiat 500X Device

The Volkswagen emission scandal brought attention not only to Volkswagen itself, but also to other automakers of diesel vehicles. Among them was Fiat Chrysler Automobiles (FCA), which on February 2, 2016 issued a press release stating: "FCA diesel vehicles do not have a mechanism to either detect that they are undergoing a bench test in a laboratory or to activate a function to operate emission controls only under laboratory testing. [...] [W]hen tested following the only testing cycle prescribed by European law (NEDC) [FCA diesel vehicles] perform within the regulatory limits and comply with the relevant regulatory requirements." [11]. On February 9, 2016, a week after FCA issued the press release, it was accused by German environmental protection organization Deutsche Umwelthilfe (DUH) of exceeding emission limits on their Fiat 500X cross-over SUV equipped with a 2-liter Fiat MultiJet II diesel engine. DUH used a chassis dynamometer for testing. As of this writing, FCA has not acknowledged that its car has a defeat device.

Like other vehicles in this study, the diesel engine of the Fiat 500X uses the Bosch EDC17 ECU. Its exhaust after-treatment system includes an NO_x Storage Catalyst (NSC) and a Diesel Particulate Filter (DPF). To investigate the claim, we obtained a Fiat 500X function sheet (EDC17C69 P 1264) and firmware image (55265162). We examined both for the presence of the Volkswagen defeat device, but found neither mention of the acoustic condition in the function sheet nor any evidence of curve-checking logic in the firmware image.

However, we found that Fiat 500X contained what amounts to a defeat device in the logic governing NSC regeneration. Unlike the Volkswagen defeat device, the FCA mechanism relies on time only, *reducing the frequency of NSC regenerations 26 minutes 40 seconds after engine start.* Recall that the primary role of the NSC (Section II-A) is to reduce NO_x emissions by trapping NO_2 in the catalyst during the loading

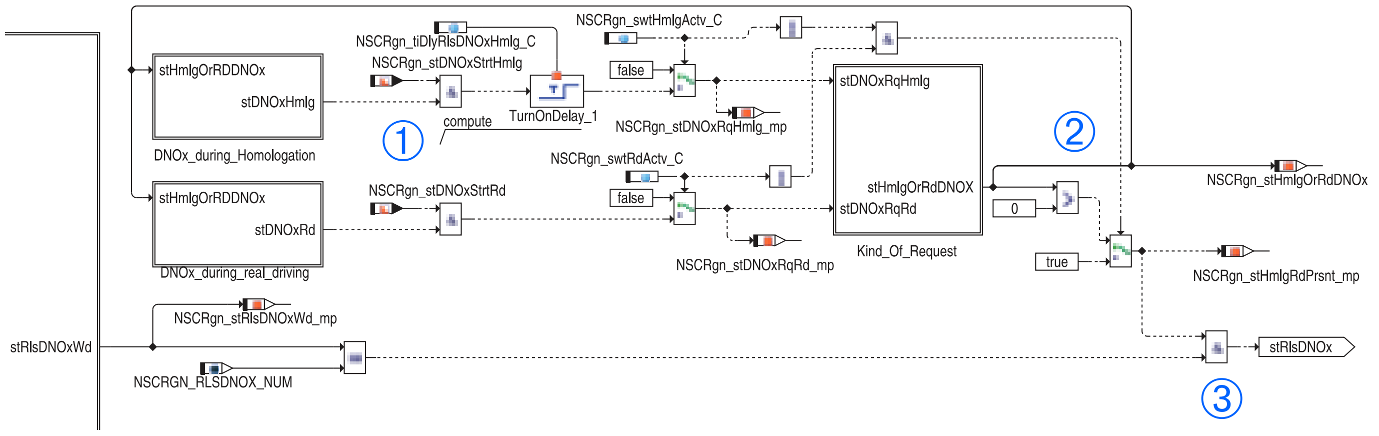


Fig. 5: NO_x regeneration release logic combining the start and release signals from the homologation and real driving logic to compute the single regeneration release signal **`stRlsDNOx`**. Demand and release conditions are computed separately for the homologation and real driving logic ①. The output of the **`Kind_of_Request`** block is non-zero if either of the homologation or real driving signals is **`true`** ②. The final release signal **`stRlsDNOx`** is only asserted ③ if either the homologation or real driving release signals is **`true`**. Blue numbers ① through ③ added by authors. From function sheet EDC17C69 P 1264 for Fiat 500X. Copyright Robert Bosch GmbH.

phase (lasting from 30 to 300 seconds) and purging it during the regeneration phase (lasting 2 to 10 seconds). Regeneration reduces fuel economy and increases the load on the DPF. By reducing the frequency of NSC regeneration, a manufacturer can improve fuel economy and increase DPF service life, at the cost of increased NO_x emissions.

In the Fiat 500X ECU, the logic controlling NSC regeneration is divided into *demand logic* and *release logic*. The former determines when NSC regeneration should take place, while the latter imposes constraints on when regeneration is allowed to start. For regeneration to start, the demand logic must request regeneration, asserting the **`NSCRgn_stDNOxStrt`** signal while the release signal **`NSCRgn_stRlsDNOx`** must be asserted by the release logic. (DNOx refers to NSC regeneration, which purges stored NO_x from the catalyst.) In the EDC17C69 function sheet we examined, both the demand and release logic was *duplicated* into two parallel blocks. The first pair of demand and release blocks applies to a "homologation cycle" while the second pair to "real driving." (*Homologation* refers to the process or act of granting approval by an official body, for example, of a vehicle for sale in a particular jurisdiction. The terms "homologation" and "real driving" are taken from the EDC17C69 function sheet.) Names of signals and logic blocks used in the homologation logic contain **`Hmlg`** in their name, while those used in the real driving logic contain **`Rd`** in the their name. The demand logic for the homologation and real driving blocks are very similar, using the total estimated NO_x load, catalyst temperature, and other variables to determine when to trigger regeneration. The homologation and real driving logic, however, uses different calibration parameters, allowing the manufacturer to supply completely different models for the test cycle and real driving.

Both homologation and real driving logic blocks can request a regeneration. Similarly, the release signal is also controlled by two parallel logic blocks. Figure 5 shows how the signals are joined. The homologation release signal is AND-combined

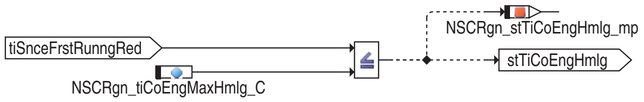


Fig. 6: The **`stTiCoEngHmlg`** signal logic required to set **`stDNOxHmlg`**, allowing NO_x regeneration to proceed under the "homologation" schedule. In the 55265162 Fiat 500X firmware image we examined, **`NSCRgn_tiCoEngMaxHmlg_C`** is set to 1600 seconds. Copyright Robert Bosch GmbH.

with the homologation demand signal, and likewise the real driving signal demand and release signals (marked ① in Figure 5). The homologation release signal is delayed by **`NSCRgn_tiDlyRlsDNOxHmlg_C`**, which is set to 300 seconds in the 55265162 firmware image. The resulting release signal out of the block is asserted if either the homologation or the real driving signal is **`true`**.

The logic controlling the homologation regeneration release signal is shown in Figure 12, and the corresponding logic block for real driving in Figure 13 in the Appendix. The important feature of the homologation release block is that all conditions defined by the blocks shown in the figure must be met, because their outputs are AND-combined to produce the output signal **`stDNOxHmlg`**. In particular, this means that the **`stTiCoEngHmlg`** output of the first sub-block must be **`true`**. The bottom of Figure 12 shows how this signal is computed: **`stTiCoEngHmlg`** is set if the running time since engine start, **`tiSnceFrstRunngRed`**, is less than or equal to the constant **`NSCRgn_tiCoEngMaxHmlg_C`**. In the Fiat 500X firmware image we examined, this constant was calibrated to 1600 seconds. Thus, *the homologation regeneration release signal* ***`stDNOxHmlg`*** *will be inhibited if the engine has been running longer than 1600 seconds.* In addition, **`stDNOxHmlg`** also requires that the total driving cycle fuel consumption be at most **`NSCRgn_volFlConsMaxHmlg_C`**, which is configured to 1.3 liters in our firmware image.

This means that regeneration requested by the homologation demand block *will only be allowed to start a regeneration during the first 1600 seconds (26 minutes 40 seconds) of engine operation.* After that, only NSC regeneration requested by the "real driving" logic will be allowed to start regeneration. We note that this coincides with the runtime of standardized emissions test cycles.

The logic blocks described above include several switches that may disable this dual path behavior. In the Fiat 500X firmware image we examined, we found that both paths were enabled (`NSCRgn_swt{Hmlg,Rd}HmlgActv_C = true`). The homologation release delay `NSCRgn_tiDlyRlsDNOxHmlg_C` was set to 300 seconds, which limited the frequency of homologation-requested regeneration to once every five minutes. We also examined the demand logic for homologation and real driving.

V. Detecting Defeat Devices

Based on the insights obtained in our case studies, we designed a static analysis tool that helps us to identify a defeat device in a given firmware image. We implemented a prototype of this approach in a tool called CurveDiff for EDC17 ECUs that enables us to track the evolution and behavior of such a device across a large number of firmware images. In the following, we discuss design considerations and the general workflow together with implementation details.

A. Design Considerations

Our method aims to *automatically* identify potential defeat devices which actively try to detect an ongoing emissions test based on the car's driving profile during the test cycle. More specifically, we try to identify code regions in a given firmware image that attempt to determine if the car currently follows one of the standardized test cycles and whose behavior influence the operation of the engine. We thus focus on the type of defeat devices implemented by Volkswagen since they represent more sophisticated defeat devices compared to the time-based ones implemented by FCA.

Our design decision to focus on test cycle detection is due to two important factors. First, this approach requires relatively little previous domain knowledge about firmware specifics and is thus rather unlikely to be subject to syntactical changes in the checking logic. In turn, this also means that we do not have to rely on additional data such as `.a2l` files, which may be hard to obtain for a given firmware image (even though it would significantly simplify the analysis). Second, this approach provides higher means of non-repudiation: Because we do not rely on accurately determining ECU variables but try to generically detect matches against well-known emissions test cycles, the fact that the software *actively* checks against the latter is hard to refute in general.

B. General Workflow

We use static code analysis to implement our approach because we cannot easily execute a given ECU firmware image in an emulator to perform a dynamic analysis. Furthermore, static analysis enables us to obtain high code coverage by analyzing each function individually. Our analysis framework called CurveDiff is based on the IDA Pro 6.9 [13] disassembler, which includes support for the Infineon TriCore processor used in Bosch's EDC17 ECU. The framework is fully automated and takes a binary firmware image as input. When analyzing a firmware image, we perform the following steps:

1) Generate and pre-process the IDA database,
2) Build core structures and lift to static single assignment (SSA) form,
3) Analyze curve function invocations,
4) Match curve checks against test cycles.

In the following sections, we describe each step in more detail and provide information about implementation details.

C. Preliminaries

The curve function `SrvX_IpoCurveS16` is a vital part of the defeat device used by Volkswagen. It is also a *core function* provided by the operating system itself and thus present in all firmware images using the same OS. Further, we found that it is widely used throughout the code of a firmware image. Basically, it returns the y coordinate for a given x coordinate on curve c, i. e., $y \leftarrow$ `SrvX_IpoCurveS16`(c, x). Since c might be represented by a few data points only, the function interpolates linearly.

Matching the current driving profile against predefined emissions test cycles is performed by posing two curve queries using `SrvX_IpoCurveS16`: one yields the upper boundary on y corresponding to the given x value, whereas the other yields the lower boundary. Specifically, the boundaries fit a known test cycle that the real driving profile (seconds since engine startup x and covered distance y) is checked against.

D. Pre-Processing

For our analysis, in order to resolve memory accesses, we need to obtain the *small data* regions (for global variables, via TriCore's system global register `a0`) and *literal data* regions (for read-only data, via register `a1`) as well as the function vector table (accessed via register `a9`), which stores data associated with a certain function. The system global registers are architecture and OS dependent and initialized during startup, as all functions operate on them to access the specific memory regions. Further, we need to obtain the address of the curve function, which can be easily detected by matching on parts of the function semantics (namely, linear interpolation of two curve points) and verifying the result using its call graph. Since this function is not customer-dependent but provided by the OS, it does not change significantly.

Note that there are a few things we need to consider. As the curve function may be wrapped, we need to detect such instances to avoid having to perform inter-procedural analyses later. In practice, wrappers can easily be detected using the function's call graph. In addition, we need to take peculiarities of the architecture into account: TriCore supports *scratch pad RAM* (SPRAM for short), which mirrors parts of the firmware's code in faster memory. As this is done on startup (i. e., at runtime), we need to extract the mapping of mirrored regions, as we otherwise might miss calls targeting this memory area.

E. Lifting to Static Single Assignment Form and Optimization

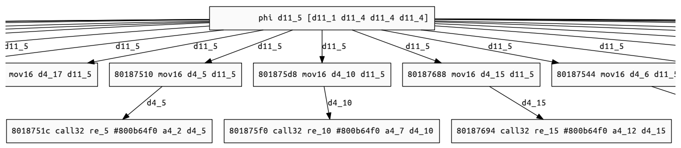
In order to facilitate a robust static analysis suitable for our task, we operate on an intermediate language (IL) in Static Single Assignment (SSA) form. SSA was introduced by Cytron et al. in 1991 [6] and describes the property of an IL in which there is only one single definition for each variable and each definition dominates its uses. This, in turn, enables the design of efficient data-flow analysis algorithms.

The TriCore assembly language is expressive enough to diminish the need for a full-fledged IL, e. g., side effects are rare and nearly all data flow is explicit. Hence, rather than developing a new IL from scratch, we modify the assembly representation slightly in order to conform with requirements assumed when transforming to SSA form. More precisely, for instructions containing an operand that is both read and written, we duplicate the operand such that *use* and *definition* are properly distinguished. Similarly, for instructions defining more than one variable, we add one single definition (a temporary register), and insert helper instructions that extract the correct definition from the temporary register, and store them into the target variable. For example, `call`s may, amongst others, return results in both registers `a2` and `d2`. As SSA form does not allow multiple definitions for one instructions, we introduce the temporary register `re` that stores the return values of the call. Right after the `call` instruction, we add artificial `cconv.w` instructions that read from `re` and store the corresponding part of the return value into `a2` and `d2`, respectively. Further, we encode other particularities of the TriCore calling convention explicitly. For example, we add uses of parameter-passing registers `a4` and `d4` to `call`s and, in a similar vein, uses of `a2` and `d2` to return instructions. We transform the resulting assembly into pruned SSA form [5] using liveness analysis. Finally, in order to coalesce memory access via system global registers `a0`, `a1`, and `a9`, we optimize each function using constant propagation.

F. Relating Curve Queries

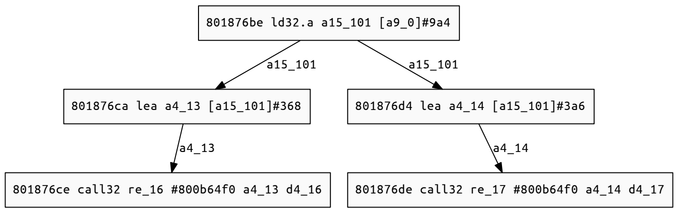
Having transformed all functions into an intermediate representation, each function is analyzed separately in order to construct a list of candidates potentially checking against emissions test cycles. To this end, we extract all invocations of the curve function and try to group them into pairs of two, where each call queries either the upper or lower boundary for a given data point. This allows us to programmatically extract the curves defining both boundaries and match them to well-known cycles in a later step. We define two such calls to the curve function as being *related*.

Section V-C explained how two calls to the curve function `SrvX_IpoCurveS16` are made in order to match the current driving profile against predefined emissions test cycles. This observation allows us to identify several important properties that must hold for two related curve queries:



phi d11_5 [d11_1 d11_4 d11_4 d11_4]
d11_5
d11_5
d11_5
d11_5
d11_5
d11_5
d11_5
d11_5
mov16 d4_17 d11_5
80187510 mov16 d4_5 d11_5
801875d8 mov16 d4_10 d11_5
80187688 mov16 d4_15 d11_5
80187544 mov16 d4_6 d11_
d4_5
d4_10
d4_15
8018751c call32 re_5 #800b64f0 a4_2 d4_5
801875f0 call32 re_10 #800b64f0 a4_7 d4_10
80187694 call32 re_15 #800b64f0 a4_12 d4_15

Fig. 7: Excerpt of a data-flow graph used to verify property **P-1**. Curve calls can be grouped by the origin of input coordinate x (passed via register `d4`, originating from register `d11`).



801876be ld32.a a15_101 [a9_0]#9a4
a15_101
a15_101
801876ca lea a4_13 [a15_101]#368
801876d4 lea a4_14 [a15_101]#3a6
a4_13
a4_14
801876ce call32 re_16 #800b64f0 a4_13 d4_16
801876de call32 re_17 #800b64f0 a4_14 d4_17

Fig. 8: Example for data-flow graph for verifying property **P-2**. Both calls to the curve function are passed a different target curve via register `a4`.

P-1 Both curve queries in a related check have to take the same variable as query point x (parameter `d4`). This requirement follows from the fact that both curves use the same axis. Concretely, x corresponds to the time since engine start.

P-2 Both curve queries have to operate on distinct curves (parameter `a4`). This is because both curves encode multiple possible driving profiles and allow for some deviation from the exact test cycles due to potential imprecisions during the emission testing.

P-3 The results y_{low}, y_{high} (register `d2`) of both curves have to be *related* in the sense that they implement a range check on the actual y value (i. e., the distance driven since engine start).

Properties P-1 and P-2. Effectively, property **P-1** allows us to group several calls to the curve function together, based on the value provided for parameter x. In order to achieve this, for each call, we backtrack register `d4` and build a data-flow graph, where nodes are instructions and edges connect a variable's (necessarily unique) definition and its uses. An example for the resulting graph can be seen in Figure 7. Evidently, all curve calls take $\texttt{d11}_5$ as parameter x. Similarly, we can backtrack register `a4` in order to find out the actual curves the functions operate on. Figure 8 visualizes this approach. Note that both leave nodes are connected via $\texttt{a15}_{101}$, which is the function vector entry holding all data associated with the current function. However, both calls still operate on different curves with offsets `0x368` and `0x3a6`, respectively.

Property P-3. Property **P-3** effectively states that the resulting ys of two distinct curve calls are *related* if they end up in the same "meaningful" expression; an expression is meaningful, for example, if it implements an interval check by comparing a given value with upper and lower boundaries as specified by the curves.

In order to check **P-3**, we begin by building a forest of data-flow graphs by tracking forwards the return values of all curve calls that lie in the same group, according to property **P-2**. Note that in the data-flow graphs, not all uses of an instruction are considered. Each connected component then either corresponds to the data-flow graph arising from one single curve call or it connects data-flow graphs of multiple curve calls together. While the first case does not provide any useful information, the latter case tells us that both curve calls are in fact related. Even though this fact already is useful as-is, we can further inspect *how* two calls are related.

Intuitively, the type of relation between two curve calls is described by the node where the data flows for each return value *meet*. We call these nodes *(forward) join nodes*. They can be computed by calculating the *lowest common ancestor* (LCA) [2] of the vertex-induced subgraph of every possible pair of curve calls. Figure 10 in the Appendix depicts a part of the (single) connected component that reveals the relations of all curve calls in the acoustic function. This statement implements an interval check that could further be confirmed by, e. g., symbolically executing the path up to the join node. Similarly, we can define *(backward) join nodes* as the LCAs in the reverse data-flow graph (more precisely, in the subgraphs induced by all pairs of leafs). Figure 10 contains an example for a backward join node, the *phi* node defining $\mathtt{d9}_3$. Unsurprisingly, this definition equals to the distance covered so far.

In order to cover cases where, e. g., the lower boundary check is control-dependent on the upper boundary check (i. e., might not be executed based on the result of the other check) but not directly data-dependent, we enrich the data-flow graphs by control-dependency edges, i. e., build a reduced program dependency graph. The introduced concepts, however, apply to this extension as well.

G. Matching Test Cycles

Given two related curve calls, we can extract the curves they operate on by backtracking the parameter register `a4`. Thus, we obtain a curve representing the upper boundary of matched driving profiles $c_\top$ and the lower boundary $c_\bot$. Roughly speaking, a specific driving profile is *matched* if its data points lie within said boundaries.

Note that we can perform a sanity check of the extracted curves before processing them further. Namely, we want a curve to be monotonically increasing because the covered distance obviously cannot decrease. This requirement is relaxed for the last data points of a boundary, as there are cases where $c_\top$ drops below $c_\bot$ to effectively reject all driving profiles after this point. In a similar vein, we can detect what we call *invalidated* checks. These are characterized by having all y values set to a constant value (`0x7fff` for $c_\bot$ and `0` for $c_\top$ in the firmware images we analyzed) such that the check rejects any driving profile. Using this method, the manufacturer can parameterize a profile check such that it is not actually used.

The reference test cycles as used for emissions testing are available either free of charge [20] or tied to a small subscription fee [7]. In most cases, the cycles are given in the

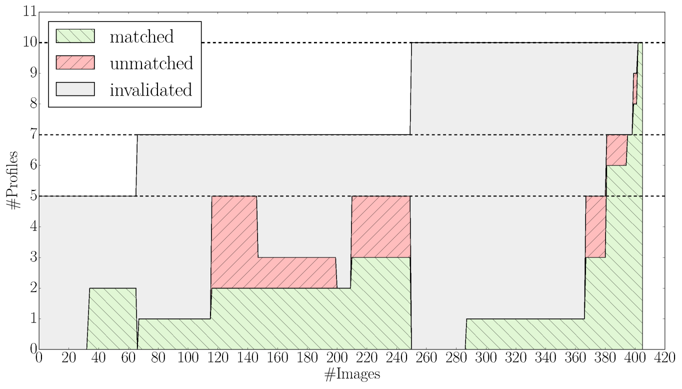


Fig. 9: Coverage of profiles in all firmware images in which a defeat device has been found. Matched profiles are highlighted using backward diagonal lines, unmatched ones use forward diagonal lines and invalidated curves are shaded gray.

form of two-dimensional data points, containing information about the elapsed time in seconds and the speed at this point (given either in mph or $\mathrm{km/h}$). In order to actually match test cycles to the boundaries extracted from the firmware images, both representations need to be normalized first. For the latter, we scale the y axis by factor 0.1 to obtain the distance in kilometers, and the x axis by 6.25 to obtain the engine runtime since startup in seconds (corresponding to unit `TimeRed` in the A2L file). As the test curves provide speed instead of the covered distance, we integrate them and convert from mph to km/h, if applicable. Finally, to match a curve, each data point has to lie in the interval as defined by $c_\top$ and $c_\bot$, respectively.

Still, some checks cut off the driving profile near the end of a particular test cycle, where the emissions effects would most likely not be picked up by an ongoing emissions test any more. The corresponding test cycle would not match by comparing all its data points due to the premature mode switch. To account for this, we do not check the interval for the last 10% of a given test cycle.

VI. Evaluation

Based on the prototype implementation of CurveDiff, we performed a larger study of Volkswagen firmware images to investigate which of them contain a defeat device. In the following, we present the evaluation results together with some highlights we found.

We analyzed 963 firmware images and configured the analysis system with a timeout of seven minutes to avoid long-running analysis tasks. 924 images were successfully analyzed according to the steps outlined in the previous section, while 20 tasks timed out and 19 tasks failed to be processed by IDA. In total, we found that 406 (44%) of the analyzed images contained a defeat device, out of which 333 contained at least one active (i. e., non-invalidated) profile.

Performance. The static analysis is fully automated and the fastest analysis task finished after 55 seconds, while several tasks also timed out (see above). The geometric mean for the analysis of all successful tasks is 105 seconds, hence we can analyze a given image on average in less than two minutes.

TABLE II: Acoustic condition logic and affected systems, based on function sheets. The *Model* column shows the ECU model (prefix EDC omitted). The *Version* column shows the ECU version for which the function sheet was generated. The *Date* column gives the date given in the function sheet. Column *N* shows the number of profiles checked by the acoustic condition or "—" if the acoustic condition logic block was not included in the function sheet. The *Affected Subsystems* column shows the subsystems where the acoustic condition was referenced, extracted from the variable cross-reference table in the function sheet.

Model	Version	Date	N	Affected Subsystems
16CP	P_397 A.V.0	2005-06-24	0	`InjCrv`, `Rail`
17CP04	P_531 2.F.0	2005-10-28	0	`InjCrv`
16CP	P_397 A.V.9	2006-03-02	0	`InjCrv`, `Rail`
17CP04	P_617 3.K.0	2006-11-06	0	`InjCrv`
17CP04	P_617 3.N.0	2006-12-22	0	`InjCrv`
17CP24	P_628 3.K.1	2007-03-29	—	`InjCrv`
17CP24	P_628 3.U.0	2007-05-02	—	`InjCrv`
17CP24	P_703 3.V.5	2007-07-12	—	`InjCrv`
17CP04	P_617 3.U.0	2007-05-14	5	`InjCrv`
17CP14	P_531 3.U.0	2007-05-24	5	`InjCrv`
17CP14	P_617 3.U.5	2007-08-30	5	`InjCrv`
17CP24	P_628 3.W.5	2007-09-18	—	`AirCtl`, `InjCrv`
17CP14	P_714 3.U.A	2007-10-12	5	`InjCrv`
17CP24	P_703 3.W.A	2007-11-05	—	`AirCtl`, `InjCrv`
17CP24	P_628 3.W.G	2008-02-12	5	`AirCtl`, `PFlt`, `InjCrv`
17CP24	P_703 3.W.G	2008-02-14	5	`AirCtl`, `PFlt`, `InjCrv`
17CP24	P_628 3.W.H	2008-03-04	5	`AirCtl`, `PFlt`, `InjCrv`
17CP14	P_804 4.F.0	2008-03-26	5	`InjCrv`, `Rail`
17CP24	P_703 3.W.K	2008-04-23	5	`AirCtl`, `PFlt`, `InjCrv`
17CP24	P_628 3.W.L	2008-05-17	5	`AirCtl`, `SCRFFC`, `PFlt`, `InjCrv`
17CP24	P_859 4.F.0	2008-05-30	5	`AirCtl`, `PFlt`, `InjCrv`, `Rail`
17CP24	P_628 3.W.M	2008-06-27	5	`AirCtl`, `SCRFFC`, `PFlt`, `InjCrv`
17CP44	P_804 4.P.0	2008-08-05	—	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `PCR`, `Rail`
17CP24	P_859 4.P.0	2008-09-18	—	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `PCR`, `PFlt`, `Rail`
17CP44	P_930 4.P.5	2008-11-13	—	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `PCR`, `PFlt`, `PFltPOp`, `Rail`
17CP44	P_804 5.A.0	2009-01-22	7	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `InjSys`, `PCR`, `PFltPOp`, `Rail`, `SmkLim`
17CP44	P_804 5.A.5	2009-02-04	7	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `InjSys`, `PCR`, `PFltPOp`, `Rail`, `SmkLim`
17CP44	P_859 5.A.0	2009-03-16	7	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `InjSys`, `PCR`, `PFlt`, `PFltPOp`, `Rail`, `SmkLim`
17CP44	P_859 5.F.5	2009-07-13	7	`AFS`, `AirCtl`, `ASMod`, `InjCrv`, `InjSys`, `PCR`, `PFlt`, `PFltPOp`, `Rail`, `SmkLim`

Compared to an analysis with a chassis dynamometer, such an approach is at least two orders of magnitude faster.

Results. Table III shows the results of our analysis. Results above the double line contain firmware from the dump obtained from the chiptuning scene (years 2009 and 2010). Dates and software part numbers are taken from the street release certification next to the firmware images. Results below the double line are based on firmware obtained via the *erWin* portal, which provides official firmware images for car shops (years 2012 to 2016). Dates are taken from the firmware's time stamp. For both data sources, the models have been matched by querying an online database for spare parts, which yields metadata for a given part number (in this case, the part numbers specify the ECU). This mapping may not be 100% accurate, as disclaimed by said sites as well. For part numbers where multiple model names were returned (due to varying model naming schemes in different regions), we chose the European name, as most firmware images check for European emissions test cycles. In cases where multiple firmware images in a month matched the same model, we denoted the number of images analyzed for a specific model in parentheses. Finally, for all the images released in the same month, we took the union of test cycles they check for to give an impression of the number and variety of matched cycles. Figure 9 depicts our coverage in terms of identified test cycles. For firmware images with 5 profiles, we were able to match a test cycle to each profile. However, for later images checking 5 and 7 profiles, respectively, we were unable to find a matching test cycle for some of the profiles.

Effects on EGR. Based on the results in Table III, we *automatically* identified a lower bound of firmware images in which the acoustic condition affects the `AirCtl` subsystem, responsible for calculating the amount of recirculated exhaust gas (EGR). We did not use A2L files for this, as we do not have matching files for all firmware images. We found that in *at least* 268 images (66%), the acoustic condition can affect EGR. Based on the parameters we extracted, we can confirm that in 247 (92%) of these images, the acoustic condition actually influences the choice of parameters. Note that the `AirCtl` detection can be improved upon as well as extended to other subsystems as listed in Table II to fully confirm further defeat devices in Table III.

We also manually analyzed some of the defeat devices detected by CurveDiff to verify our results, and in the following we highlight some of the findings.

Steering wheel check. We found that the 2014's EDC17C54 P1169 firmware image with part number 03L906012DE and revision 8401 has started checking the steering wheel angle in addition to the time-distance profiles, as described in Section IV-A. An automatic scan for the steering wheel check yielded three more images, namely 03L906012, revision 7444 (depicted in Figure 3); 03L906012DD, revision 8400; and 03L906012BP, revision 7445. The images seemingly have been released on December 3, 2014, 22:55 and are used in VW Passat cars according to an online database. This refinement of the defeat device is noteworthy given that at that point in time, the CARB had already started to investigate emission abnormalities in Volkswagen cars [15] (cf. facts 140, 141). As evident from Table III, these images are responsible for the largest set of test cycle matches across nearly 400 images, further highlighting the necessity of the check. Other firmware images released the same month (for Audi A4 and A6) do not contain this additional logic and only match a subset of the listed test cycles.

VII. Discussion

Our empirical evaluation results demonstrate that our approach to detect Volkswagen-style defeat devices is viable across a large number of firmware images. Nevertheless, there are certain open challenges and potential limitations of our approach that we discuss in the following.

TABLE III: Results for 363 of 406 firmware images in which CurveDiff detected a potential Volkswagen-style defeat device. For firmwares not listed in this table either the release date or the model are unknown. The number in parentheses depicts the number of firmware images analyzed for this model. The lower part of the table, below the double line, shows the result based on *erWin* data; the upper part is drawn from a chip tuning dump.

Matched emissions test cycles are listed as the union of matched cycles in all firmware images in that row. The last column shows whether firmware images in this row were found to contain additional steering wheel checks that guard individual curves. The affected models in that row are printed in bold. Note that model data in this table is retrieved from external (non-VW) sources. Further conditions may affect the defeat device's operation.

Rls. Date	**Models (number of images)**	**Matched Cycles (upper bound)**	**St.W.**
2009-01	Golf, Passat (2)	ECE-15, EUDC(L), NEDC	✗
2009-07	A3	ECE-15, FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2009-08	Passat Blue Motion	ECE-15, EUDC(L), NEDC	✗
2009-09	Golf (2), Passat (3)	ECE-15, EUDC(L), NEDC	✗
2009-10	Golf+, Passat	ECE-15, EUDC(L), NEDC	✗
2009-11	A3 (8), Golf Blue Motion, Golf (2), Passat	ECE-15, EUDC(L), NEDC	
2009-12	A3 (5), Golf Variant (2), Golf+ (2), Golf (7), Jetta (3), Passat (4)	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2010-01	Jetta, Passat (2)	ECE-15, EUDC(L), NEDC	✗
2010-03	A3 (2), Golf (3), Jetta, Passat (3), Q5 (4)	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2010-04	Jetta (2), Passat, Passat Coupe (4), Q5	ECE-15, EUDC(L), NEDC	✗
2012-05	A3 (19), A4, A6, Alhambra (4), Altea, Eos (2), Golf, Ibiza (4), Leon, Octavia (6), Q5 (2), Superb (2), TT, Tiguan, Yeti (4)	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2012-06	Amarok (8), CC, Eos (2), Golf (2), Jetta (2), Octavia (3), Q5 (2), Sharan (7), Tiguan, Touran (2)	ECE-15, EUDC(L), NEDC	✗
2012-07	A1 (3), Alhambra (4), Caddy (2), Sharan (8)	ECE-15, EUDC(L), NEDC	✗
2012-09	Golf (2), Passat, Yeti (6)	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2012-10	A3, Alhambra (2), Tiguan, Yeti	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2012-12	Eos (2), Golf Cabriolet, Tiguan (7), Touran, Yeti	ECE-15, NEDC	✗
2013-01	Leon, Passat	ECE-15, EUDC(L), FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2013-04	Amarok (6)	(deactivated)	✗
2013-05	Amarok (4)	ECE-15, EUDC(L), NEDC	✗
2013-06	Amarok (5), Superb (3), Tiguan	ECE-15, EUDC(L), NEDC	✗
2013-07	Octavia	ECE-15, EUDC(L), NEDC	✗
2013-08	Yeti (3)	ECE-15, NEDC	✗
2013-11	Superb (3)	ECE-15, EUDC(L), NEDC	✗
2013-12	Superb (2), Yeti (4)	ECE-15, EUDC(L), NEDC	✗
2014-01	Caddy (4)	ECE-15, NEDC	✗
2014-03	Amarok (16), Eos, Tiguan, Yeti	ECE-15, EUDC(L), NEDC	✗
2014-04	Q5, Superb (2)	ECE-15, EUDC(L), NEDC	✗
2014-06	Amarok (6), Tiguan (4)	ECE-15, EUDC(L), NEDC	✗
2014-09	Alhambra	ECE-15, EUDC(L), NEDC	✗
2014-10	Sharan	ECE-15, EUDC(L), NEDC	✗
2014-12	A4 (3), A6, **Passat** (4)	CADC-RURAL, CADC-URBAN, ECE-15, EUDC(L), FTP-75, HWFET, IM240, J1015, JP10, LA92, NEDC, RTS-95, SC03, US06, WLTP-1, WLTP-2, WLTP-3	✓
2015-01	Superb	ECE-15, NEDC	✗
2015-02	A3 (3)	ECE-15, FTP-75, HWFET, LA92, NEDC, SC03, US06	✗
2015-03	Alhambra (2)	ECE-15, EUDC(L), NEDC	✗
2015-05	Alhambra (6), Sharan (6)	ECE-15, EUDC(L), NEDC	✗
2015-07	Q3 (2)	ECE-15, NEDC	✗
2015-10	Altea (2), Yeti (3)	ECE-15, EUDC(L), NEDC	✗
2015-11	Superb	ECE-15, EUDC(L), NEDC	✗
2016-02	Altea	ECE-15, NEDC	✗
2016-03	A4, Exeo (4)	ECE-15, NEDC	✗
2016-04	A6, Exeo, Q3	ECE-15, NEDC	✗
2016-06	Altea (3), CC (3), Jetta, Leon (2), Superb, Tiguan (2)	ECE-15, EUDC(L), NEDC	✗
2016-07	Amarok, CC, Golf, Superb	ECE-15, NEDC	✗
2016-08	CC (3), Golf Cabriolet, Golf (2), Passat (2), Scirocco, Touran (3)	ECE-15, EUDC(L), NEDC	✗
2016-09	CC (14), Octavia (2), Passat (2), Tiguan (7)	ECE-15, EUDC(L), NEDC	✗
2016-10	Eos	ECE-15, NEDC	✗

Generally speaking, there are two approaches to distinguish regular street driving conditions from those (rather special) conditions exhibited during emission tests: *active* and *passive* detection. Active detection techniques take characteristics of the car during emission tests into account and hence are able to target specific tests. Most notably, the Volkswagen defeat device covered in this paper is able to detect an ongoing emission test based on the car's driving profile and comparing it to well-known test curves. Our approach is based on this insight and we propose a curve-agnostic method to detect that the firmware attempts to match a certain driving profile. CURVEDIFF can detect such defeat devices and we found many instance of such devices. However, a car manufacturer could also implement other active evasion approaches, for example by matching on the profile of related parameters such as speed or torque; another concrete example being the defeat device found in the Opel Zafira [9].

On the other hand, passive detection techniques cover test-agnostic methods that do not actively observe vehicle specifics to detect an ongoing emission test, but rather target general peculiarities of those tests. For example, emission tests are comparably short, which opens up the possibility to simply stay in a compliant mode for as long as the average emission test is carried out and switch to a more harmful emissions policy afterwards. The Fiat defeat device we discussed earlier belongs to this category. In principle, an ECU can leverage all available sensors in an attempt to fingerprint the testing environment, for example by measuring the temperature or the ambient pressure since both are also standardized. In addition to software-based methods, hardware-based approaches such as over-inflating tires for dynamometer tests also fall into this category. Our coverage of such passive defeat devices is limited since we focus on curve-based defeat devices. This is mostly due to the fact that the latter approach provides higher means of confidentiality. Still, tracking the data flow in the code and analyzing whether certain sensor conditions influence the Exhaust Gas Recirculation (EGR) or other subsystems related to emission control might enable the detection of such passive devices. As part of future work, we plan to study the viability of such an approach and evaluate if we can detect Fiat's defeat device in an automated manner.

We implemented our approach in a tool called CURVEDIFF. Given that we perform an intra-procedural analysis, we might miss certain ways how a defeat device can be implemented and an inter-procedural analysis could enhance the soundness of our implementation. Furthermore, our analysis can be extended to take more primitive building blocks such as timers and multiplexers into account to deepen the knowledge about the relation of various components in the detection logic.

VIII. CONCLUSION

As software control becomes a pervasive feature of complex systems, regulators in the automotive domain (as well as many others) will be faced with certifying software systems whose manufacturers have an immense financial incentive to cheat. In this paper, we described two families of defeat devices used in the Bosch EDC17 ECU to circumvent US emission tests. The first family of defeat devices was used by Volkswagen and lies at the heart of the Volkswagen diesel emissions scandal. The second device appears in the diesel Fiat 500X automobile sold in Europe, and has not beed documented previously. We also presented and evaluated an automated approach to detect defeat devices in a given firmware image based on the insights we obtained from manually analyzing the Volkswagen defeat device.

ACKNOWLEDGMENTS

Part of this work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 640110 – BASTION). This work was funded in part by the National Science Foundation through grant NSF-1646493.

REFERENCES

[1] Association for Standardisation of Automation and Measuring Systems (ASAM e.V.). ASAM MCD-2 MC. `https://wiki.asam.net/display/STANDARDS/ASAM+MCD-2+MC`.

[2] Michael A. Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest Common Ancestors in Trees and Directed Acyclic Graphs. In *Journal of Algorithms*, 2005.

[3] Robert J. Blaszczak. EPA Technical Bulletin: Nitrogen Oxides (NOx) – Why and How They are Controlled. `https://www3.epa.gov/ttncatc1/cica/other7_e.html`, 1999.

[4] Robert N. Charette. This Car Runs on Code. *IEEE Spectrum*, 46(3), 2009.

[5] Jong-Deok Choi, Ron Cytron, and Jeanne Ferrante. Automatic Construction of Sparse Data Flow Evaluation Graphs. In *ACM Symposium on Principles of Programming Languages (POPL)*, 1991.

[6] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph. In *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1991.

[7] DieselNet. Emission Test Cycles. `https://www.dieselnet.com/standards/cycles`.

[8] Christof Ebert and Capers Jones. Embedded Software: Facts, Figures, and Future. *Computer*, 42(4), 2009.

[9] Felix Domke. Software Defined Emissions, 33C3. `https://media.ccc.de/v/33c3-7904-software_defined_emissions`.

[10] Felix Domke and Daniel Lange. The exhaust emissions scandal ("Dieselgate"), 32C3. `https://media.ccc.de/v/32c3-7331-the_exhaust_emissions_scandal_dieselgate`.

[11] Fiat Chrysler Automobiles. FCA on Real Driving Emissions. `https://www.fcagroup.com/en-US/media_center/fca_press_release/2016/february/Pages/fca_on_real_driving_emissions.aspx`, 2016.

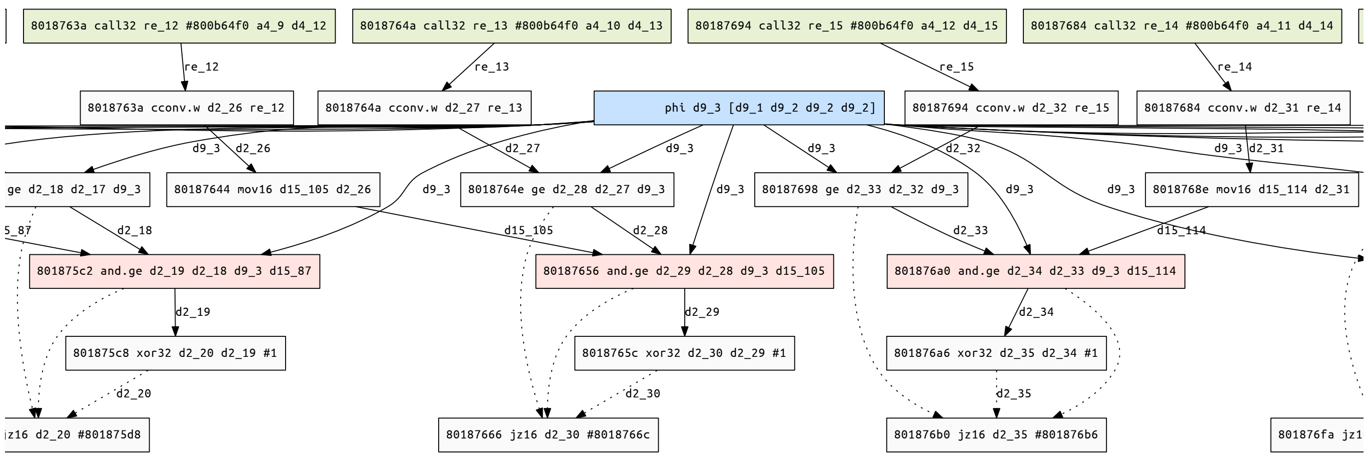
[12] Ulrich Flaig, Wilhelm Polach, and Gerhard Ziegler. Common Rail System (CR-System) for Passenger Car DI Diesel Engines; Experiences with Applications for Series Production Projects. In *SAE Technical Paper*. SAE International, 1999.

[13] Hex-Rays SA. Product Page for the Interactive Disassembler. `https://www.hex-rays.com/products/ida`.

[14] Laura Myers. GM Forced to Recall Cadilacs with Emission 'Defeat Device'. `http://www.apnewsarchive.com/1995/GM-Forced-to-Recall-Cadillacs-With-Emission-Defeat-Device-/id-4b030c7601a14dcc8208fcc1d1bd30cc`, 1995.

[15] New York State Office of the Attorney General. NY A.G. Schneiderman, Massachusetts A.G. Healey, Maryland A.G. Frosh Announce Suits Against Volkswagen, Audi And Porsche Alleging They Knowingly Sold Over 53,000 Illegally Polluting Cars And Suvs, Violating State Environmental Laws. `http://www.ag.ny.gov/press-release/ny-ag-schneiderman-massachusetts-ag-healey-maryland-ag-frosh-announce-suits-against`, 2016.

[16] Robert Bosch GmbH. *Diesel Engine Management*. John Wiley & Sons Ltd., fourth edition, 2005.



8018763a call32 re_12 #800b64f0 a4_9 d4_12
8018764a call32 re_13 #800b64f0 a4_10 d4_13
80187694 call32 re_15 #800b64f0 a4_12 d4_15
80187684 call32 re_14 #800b64f0 a4_11 d4_14
re_12
re_13
re_15
re_14
8018763a cconv.w d2_26 re_12
8018764a cconv.w d2_27 re_13
phi d9_3 [d9_1 d9_2 d9_2 d9_2]
80187694 cconv.w d2_32 re_15
80187684 cconv.w d2_31 re_14
d9_3
d2_26
d2_27
d9_3
d9_3
d2_32
d9_3
d2_31
ge d2_18 d2_17 d9_3
80187644 mov16 d15_105 d2_26
d9_3
8018764e ge d2_28 d2_27 d9_3
d9_3
80187698 ge d2_33 d2_32 d9_3
d9_3
d9_3
8018768e mov16 d15_114 d2_31
5_87
d2_18
d15_105
d2_28
d2_33
d15_114
801875c2 and.ge d2_19 d2_18 d9_3 d15_87
80187656 and.ge d2_29 d2_28 d9_3 d15_105
801876a0 and.ge d2_34 d2_33 d9_3 d15_114
d2_19
d2_29
d2_34
801875c8 xor32 d2_20 d2_19 #1
8018765c xor32 d2_30 d2_29 #1
801876a6 xor32 d2_35 d2_34 #1
d2_20
d2_30
d2_35
jz16 d2_20 #801875d8
80187666 jz16 d2_30 #8018766c
801876b0 jz16 d2_35 #801876b6
801876fa jz1

Fig. 10: A part of a connected component in the data-flow forest of the defeat device. It can be seen how the boundaries obtained via two calls to the `SrvX_IpoCurveS16` (at `0x800b64f0`) are compared against the covered distance (in $\mathtt{d9}_3$). Specifically, following the two leftmost curve calls (at `0x8018763a` and `0x8018764a`), we end up with the *forward join node* at `0x80187656` (**`and.ge d2, d2, d9, d15`**), implementing the interval check. Similarly, the ϕ node defining $\mathtt{d9}_3$ is a *backward join node*, whose definition equals to the distance covered so far. Continuous lines represent data flow of the labeled variable, whereas dotted lines show control dependencies.

TABLE IV: Overview of various test cycles used for emissions testing. The first segment details tests following US EPA and CARB legislation, the second segment is relevant to EU law, and the last segment shows international standards. Information follows [7], [20].

Abbreviation	**Full Name**
EPA IM-240	Inspection and Maintenance
FTP-75, EPA-75	Federal Test Procedure
EPA HWFET	Highway Fuel Economy Driving Schedule
SFTP SC03	Speed Correction Driving Schedule, SC03 SFTP
CARB LA92	“Unified” Dynamometer Driving Schedule, Unified Cycle
CADC-RURAL	Common Artemis Driving Cycles, Rural Road Cycle
CADC-URBAN	Common Artemis Driving Cycles, Urban Cycle
UN/ECE 15	ECE Elementary Urban Cycle
UN/EUDC	ECE Extra-Urban Driving Cycle
UN/EUDCL	ECE Extra-Urban Driving Cycle for Low-Powered Vehicles
NEDC	New European Driving Cycle
WLTP	Worldwide Harmonized Light Vehicles Test Procedure

[17] The Telegraph. Diesel emissions scandal: Fiat under investigation. `http://www.telegraph.co.uk/cars/news/diesel-emissions-scandal-fiat-under-investigation`, 2016.

[18] United States Environmental Protection Agency. Notice of Violation. `https://www.epa.gov/sites/production/files/2015-10/documents/vw-nov-caa-09-18-15.pdf`, 2015.

[19] US Code of Federal Regulations. 40 CFR §86.

[20] US Environmental Protection Agency (EPA). Dynamometer Drive Schedules. `https://www.epa.gov/vehicle-and-fuel-emissions-testing/dynamometer-drive-schedules`.

[21] Volkswagen of America, Inc. Self Study Program 826803: 2.0 Liter TDI Common Rail BIN5 ULEV Engine. `http://www.natef.org/natef/media/natefmedia/vw\%20files/2-0-tdi-ssp.pdf`, 2008.

[22] WirtschaftsWoche Online. Kommt der zweite Abgasskandal aus Italien? `http://www.wiwo.de/unternehmen/auto/fiat-500x-doblo-und-jeep-renegade-kommt-der-zweite-abgasskandal-aus-italien/14483066.html`, 2016.

Appendix

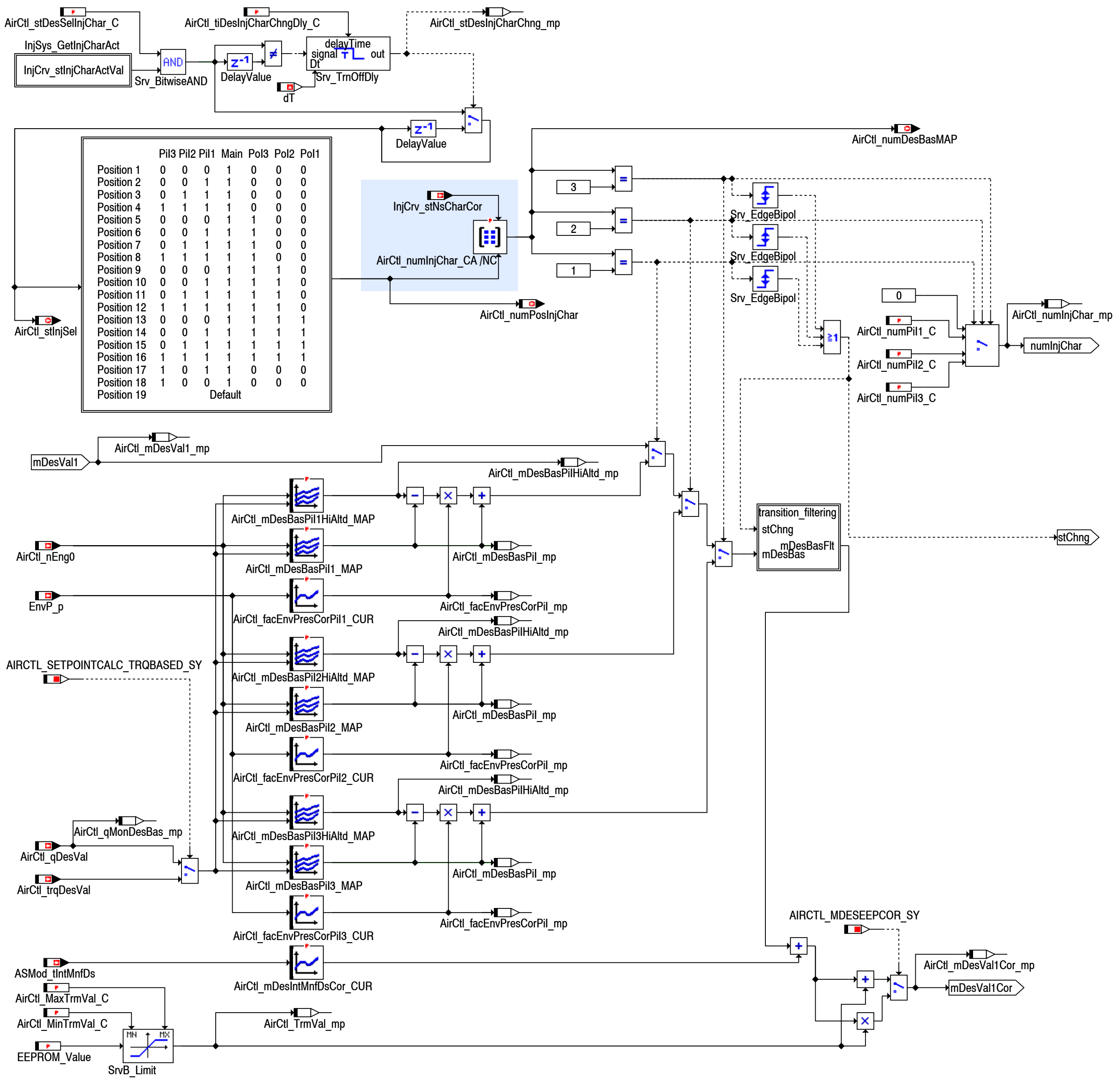
```
/begin MEASUREMENT
  InjCrv_stNsCharCor
  "Status der Akustikbedingung"
  UBYTE
  OneToOne
  1
  100
  0.00
  255.0

  FORMAT "%5.1"
  ECU_ADDRESS 0xC0000CDD
/end MEASUREMENT
-------------------------------------------------------------
/begin CHARACTERISTIC
  AirCtl_numInjChar_CA
  "Abgasstrategie fÃ r AirCtl und VswCtl"
  MAP
  0x801C5A34
  Map_Xu8Yu8Wu8
  255.0
  OneToOne
  0.00
  255.0

  FORMAT "%5.1"
  EXTENDED_LIMITS 0.00 255.0

  /begin AXIS_DESCR
    STD_AXIS
    InjCrv_stNsCharCor
    OneToOne
    ...
```

Listing 1: Excerpt from an A2L file, depicting metadata given for the acoustic condition **`InjCrv_stNsCharCor`** as well as the array **`AirCtl_numInjChar_CA`**. In the latter case, it is evident how the x axis is indexed by the acoustic condition. Akin to regular symbol files, the **`ECU_ADDRRESS`** entry identifies the address of the variable in the firmware image.



	Pil3	Pil2	Pil1	Main	Pol3	Pol2	Pol1
Position 1	0	0	0	1	0	0	0
Position 2	0	0	1	1	0	0	0
Position 3	0	1	1	1	0	0	0
Position 4	1	1	1	1	0	0	0
Position 5	0	0	0	1	1	0	0
Position 6	0	0	1	1	1	0	0
Position 7	0	1	1	1	1	0	0
Position 8	1	1	1	1	1	0	0
Position 9	0	0	0	1	1	1	0
Position 10	0	0	1	1	1	1	0
Position 11	0	1	1	1	1	1	0
Position 12	1	1	1	1	1	1	0
Position 13	0	0	0	1	1	1	1
Position 14	0	0	1	1	1	1	1
Position 15	0	1	1	1	1	1	1
Position 16	1	1	1	1	1	1	1
Position 17	1	0	1	1	0	0	0
Position 18	1	0	0	1	0	0	0
Position 19				Default			

Fig. 11: Acoustic condition (signal `InjCrv_stNsCharCor`) used to modify the desired air mass correction `mDesVal1Cor`, which modifies the desired air mass from which the amount of air recirculated is computed. `AirCtl_numInjChar_CA` is a two-dimensional array. The acoustic condition is used to select either row. From the EDC17C54 P 874 function sheet. Shading added by the authors. Copyright Robert Bosch GmbH.

Figure 5444 NSCRgn_RlsLogic/NSCRgnRlsLogic/RegenerationReleaseLogic/ReleaseLogicDNOx/DNOx_during_Homologation [NSCRgn_RlsLogic.NSCRgn-RlsLogic.RegenerationReleaseLogic.ReleaseLogicDNOx.DNOx_during_Homologation]

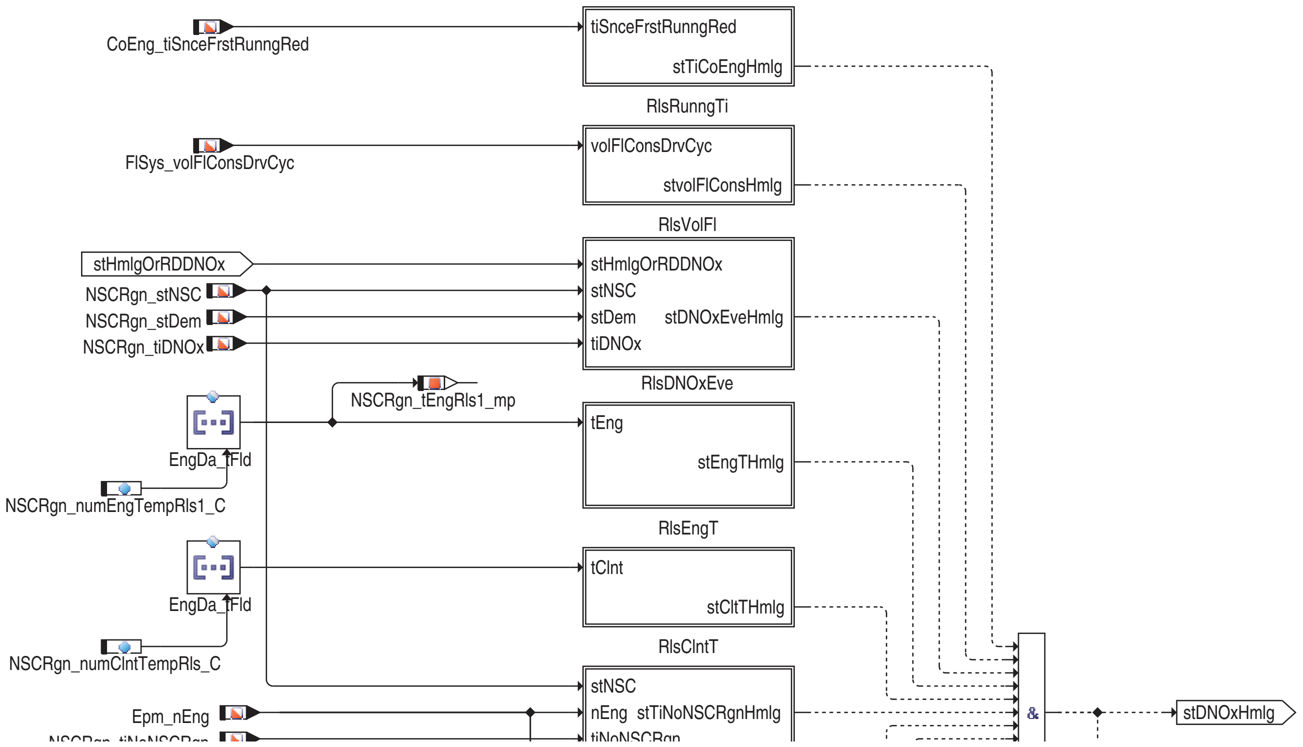


Fig. 12: Part of the NO$_x$ regeneration release logic "during homologation cycle" from function sheet EDC17C69 P 1264 for Fiat 500X. The homologation release signal requires multiple signals to be asserted, including `stTiCoEngHmlg` (Section IV-D). It is only asserted if engine running time does not exceed `NSCRgn_tiCoEngMaxHmlg_C`, set to 1600 seconds in the 55265162 Fiat 500X firmware image. Copyright Robert Bosch GmbH.

Figure 5456 NSCRgn_RlsLogic/NSCRgnRlsLogic/RegenerationReleaseLogic/ReleaseLogicDNOx/DNOx_during_real_driving [NSCRgn_RlsLogic.NSCRgnRls-Logic.RegenerationReleaseLogic.ReleaseLogicDNOx.DNOx_during_real_driving]

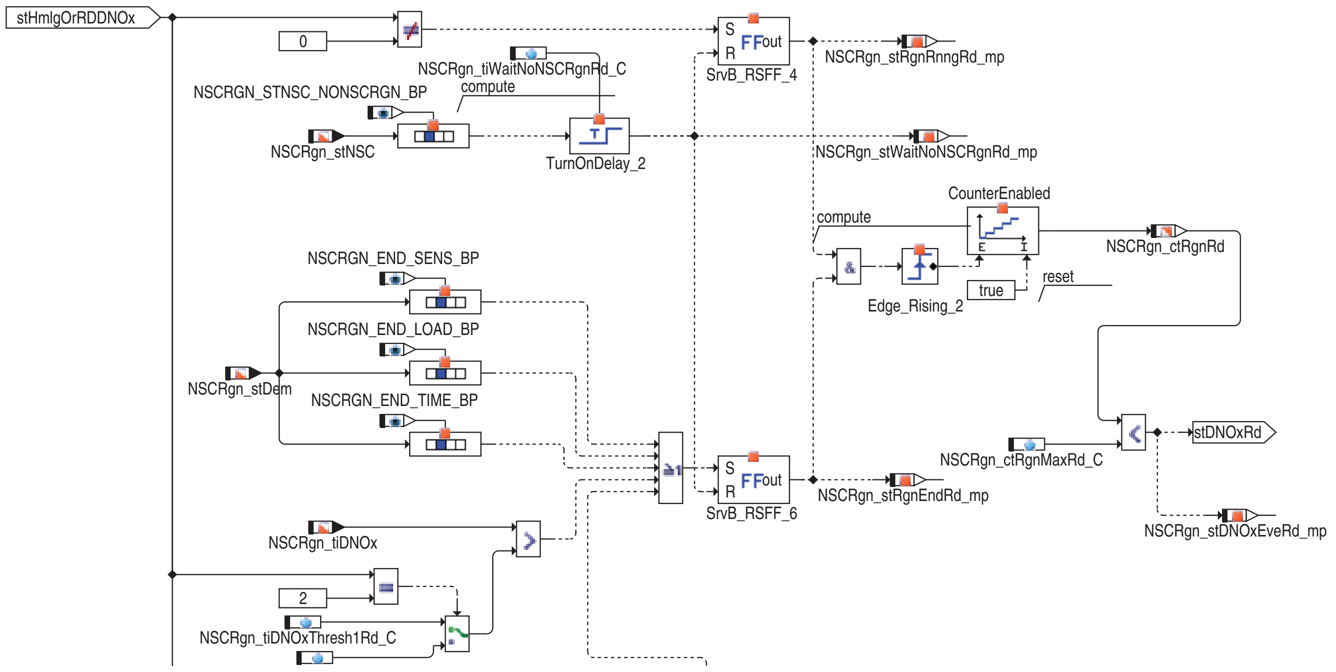


Fig. 13: Part of the NO$_x$ regeneration release logic "during_real_driving" from function sheet EDC17C69 P 1264 for Fiat 500X. First element controls release based on engine running time. A parallel logic block controls release "during real driving." Copyright Robert Bosch GmbH.

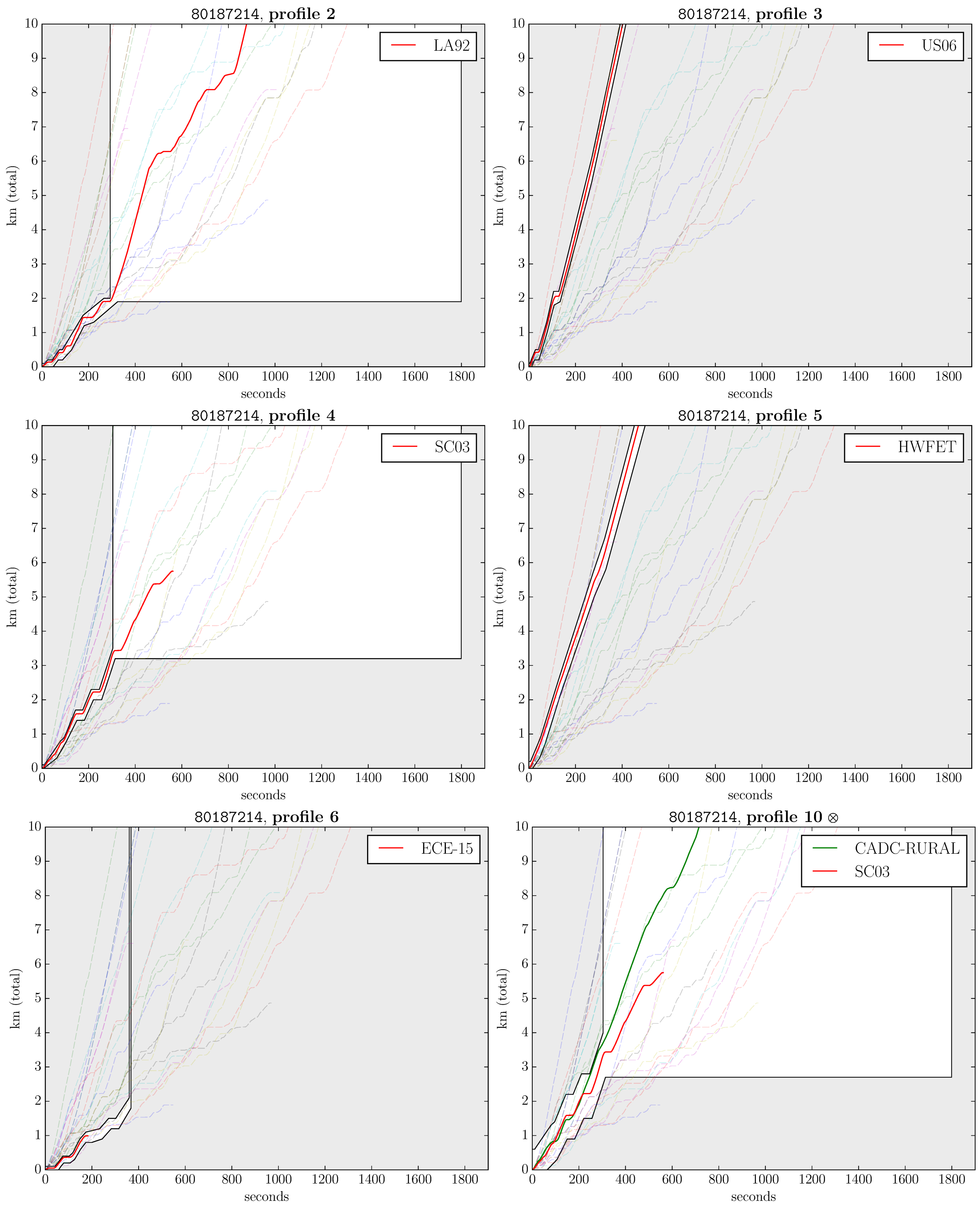


Fig. 14: Remaining curve checks testing against various emissions test cycles in the firmware for a VW Passat, released 12/2014 (EDC17C54, software part number 03L906012, revision 7444), completing Figure 3. The area in which the software reports the driving profile to match is colored white. The legend lists the known matching test cycles, ⊗ indicates an additional steering wheel check.