Graphical User Interfaces for a Qualitative and a Quantitative Side-Channel Analysis Tool

Technical Report

January 2022

Martin Edlund, Heiko Mantel, Alexandra Weber, Tim Weißmantel

Technical University of Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Modeling and Analysis of Information Systems

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1119 - 236615297.

CROSSING

Graphical User Interfaces for a Qualitative and a Quantitative Side-Channel Analysis Tool

Martin Edlund, Heiko Mantel, Alexandra Weber, and Tim Weißmantel

Department of Computer Science, TU Darmstadt, Germany
{edlund,mantel,weber,weissmantel}@mais.informatik.tu-darmstadt.de

Abstract. The tools Side-Channel Finder AVR (SCF-AVR) and CacheAudit 0.3 (CA-0.3) support a proactive treatment of side channels in different contexts. SCF-AVR is a qualitative analysis tool. It detects potential timing side channels in AVR assembly programs. CA-0.3 is a quantitative analysis tool. It computes upper bounds on the cache-side-channel leakage of x86 assembly programs, including programs that use floating-point instructions or follow a circuit-based implementation technique. Both tools have proven useful to analyze real-world implementations from cryptographic libraries. So far, the tools were controlled using their command-line interfaces and manually created configuration files. In this report, we present graphical user interfaces that we developed for the two analysis tools to simplify the application of the tools to case studies.

1 Introduction

Side channels are channels through which programs communicate unintentionally. This communication happens through execution characteristics of the program that depend on information processed by the program. For instance, the execution time of a program that performs modular exponentiation might depend on the Hamming weight of the exponent [13]. Similarly, the interaction of an AES implementation with a cache might depend on the secret AES key [20]. An attacker who can, e.g., measure the execution time or probe a shared cache might learn secret information through such channels. In fact, side channels have been exploited to recover cryptographic keys in numerous cases, including, e.g., [13, 15, 20, 25].

To enable a proactive detection and assessment of potential side channels, tools for the analysis of programs with respect to side channels are being developed. There are tools that target code at different levels of abstraction, ranging, e.g., from high-level Java [14] to Java bytecode [16], and from the LLVM intermediate representation [3] to assembly-level languages, like x86 assembly [10] and AVR assembly [7]. The tools focus on different types of side channels (e.g., timing side channels like the tools [7, 16] or cache side channels like the tools [3, 10, 21]). The analysis tools have different goals ranging from the detection of potential side channels like [3, 7, 21], to the computation of leakage bounds like [10, 16], to experimental or approximate assessment like [6, 18]. In this report, we describe new graphical user interfaces for two existing open-source side-channel-analysis tools: Side-Channel Finder AVR [7] and CacheAudit 0.3 [17].

Side-Channel Finder AVR (SCF-AVR) is a static analysis tool for the detection of potential timing side channels in AVR assembly programs. It is based on a security type system that is sound with respect to a timing-sensitive non-interference property. It is a qualitative analysis tool with the goal to detect potential timing side channels or verify the absence of timing side channels. The tool has been applied to verify the absence of timing side channels in multiple crypto implementations from the library μNaCl and to detected a timing side channel in a leaky implementation of string comparison. So far, all analyses with Side-Channel Finder AVR had to be configured through manually written JSON files and the target program had to be compiled into an analyzeable format manually. Triggering the analysis and viewing the results was done via the tool's command-line interface. Examples of a configuration and results are shown in Figure 1.

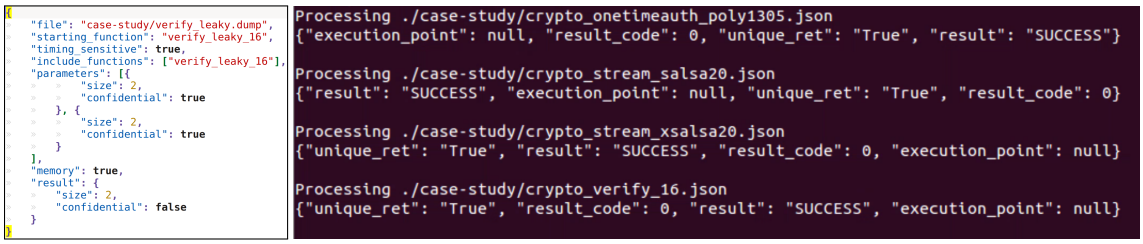

Fig. 1. Sample configuration (JSON) and output (CLI) of analyses with SCF-AVR

CacheAudit 0.3 (CA-0.3) performs static analysis of x86 binaries to compute reliable upper bounds on the leakage through cache side channels. It supports binaries that use floating-point instructions and circuit-based binaries. The tool is implemented in the CacheAudit framework [10]. Like other analysis implementations in the framework ([8, 9, 19, 2, 23]), CA-0.3 is based on a combination of abstract interpretation and information theory. It has been used to quantify the side-channel leakage across multiple crypto implementations, including implementations of AES, DES, 3DES and Camellia. Similarly to the case of SCF-AVR, analyses with CA-0.3 so far had to be configured using configuration files or parameters of the command-line interface. Examples of a configuration file and of an analysis output are shown in Figure 2.

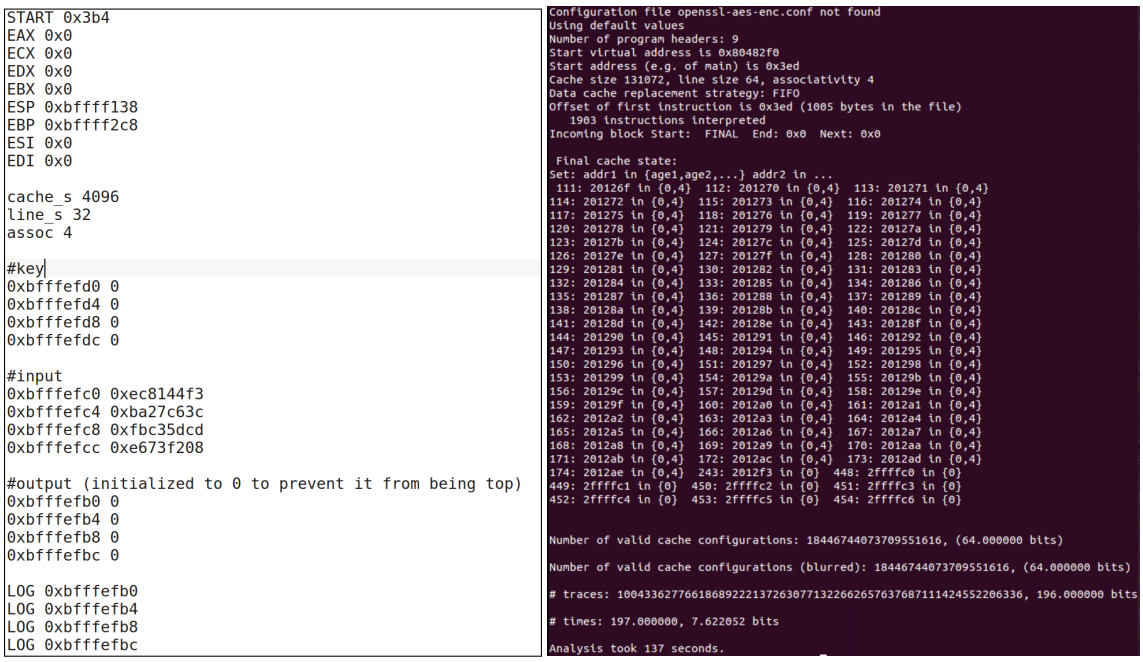

Fig. 2. Sample configuration and output of analyses with CacheAudit

In this report, we present graphical user interfaces for both, the qualitative tool Side-Channel Finder for AVR and the quantitative tool CacheAudit 0.3. The interfaces make it easier to trigger analyses and process analysis reports. They also include convenience features, e.g., for managing and reusing results of prior analyses. We apply the interfaces to analyze multiple case studies that were previously analyzed using command-line interfaces only. Using our GUI for SCF-AVR, we analyze a leaky string comparison, the secure string comparison from the crypto library μNaCl, and implementations of Salsa20, xSalsa20 and Poly1305 from μNaCl. With our GUI for CA-0.3,

we analyze implementations of AES, DES, 3DES and Camellia from the library mbedTLS and functions from a software for Quantum Key Distribution (QKD).

2 Using the Interfaces

In this section, we describe how to configure and perform analyses using our graphical user interfaces for the tools SCF-AVR and CA-0.3.

2.1 Using the Interface for SCF-AVR

SCF-AVR detects potential timing side channels in AVR assembly programs with respect to a security policy. The policy specifies which inputs to the target program are considered secret and which outputs of the target program (regular channel, timing side channel) are visible to an attacker. The GUI for SCF-AVR consists of two tabs. In the first tab, the *Policy Editor*, a user can set up the analysis and specify his security policy. In the second tab, called *Analysis*, the user can run an analysis and inspect the analysis results.

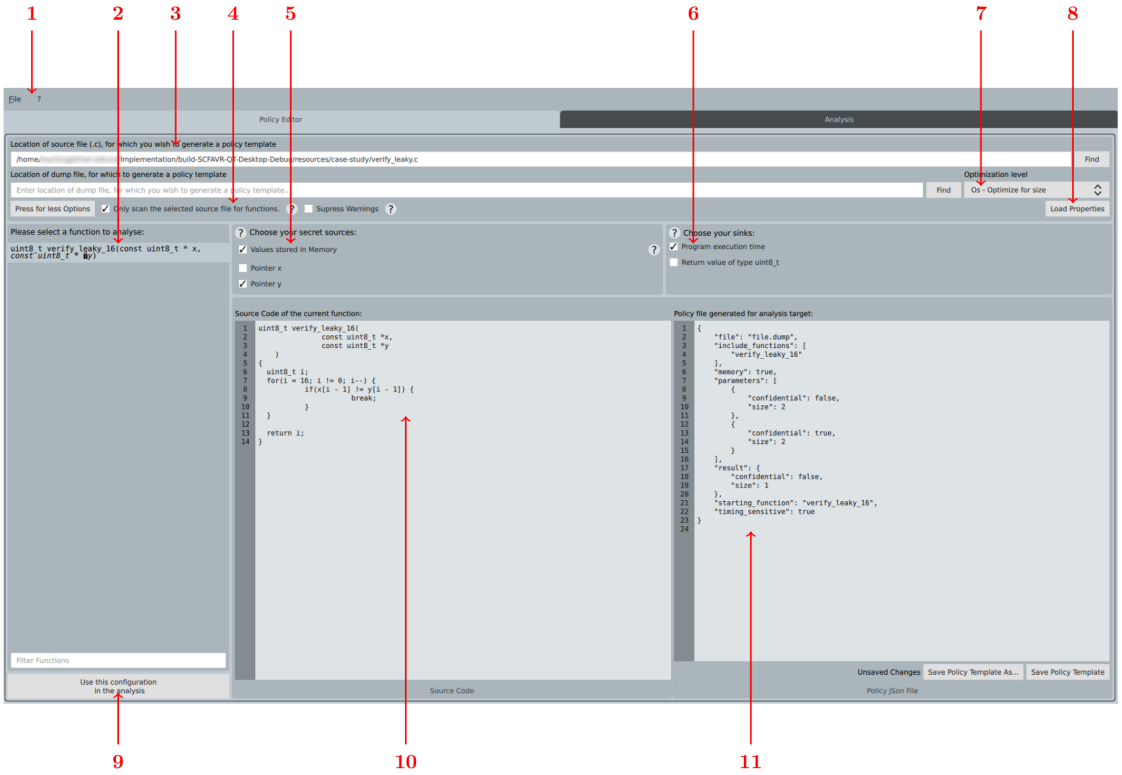

Fig. 3. Policy Editor in the SCF-AVR GUI

Policy Editor. Figure 3 shows the Policy Editor of the SCF-AVR GUI. In the Policy Editor, a user can select to reproduce an existing case study in the File menu (1) or configure a new analysis. To configure a new analysis, the user can specify the target software to analyze by selecting either a file with C source code or a file with a dump of AVR assembly code (3). All analyses are performed at the assembly level, but using the GUI, the user can easily obtain an analyzeable dump from his source code by choosing the desired optimization level for the compilation (7). Once the user presses

the Load-Properties Button (8), the functions that occur in the target software are presented and he can choose which function he would like to analyze (2). Using additional options, the user can refine the list of available functions (4).

Once the user has selected a target function, he can configure his security policy by selecting which inputs he considers secret (5) and which outputs he considers attacker-visible (6). To support the selection, the GUI displays the source code of the selected target function (10) and a preview of the configuration file that will be generated for the analysis based on the user's selection (11). The size of the input parameters that has to be specified in the configuration file is computed automatically for the user. The user can save the configuration file to make his analysis easily reproducible. He can then move on to the analysis with the specified policy (9).

Analysis Tab. Figure 4 shows the Analysis Tab of the SCF-AVR GUI. The target software and configuration file for the analysis are automatically transferred from the Policy Editor and can also be changed directly in the Analysis Tab in case an existing configuration file is used (1–6). The user can start the analysis using the Run-Analysis Button (7).

Once the analysis completes, the GUI presents the analysis result to the user (8). In Figure 4, a potential side channel was detected. In addition to the overall result, the GUI displays the line in the assembly code in which the potential leakage was detected (9) and the corresponding section of the source code (10). The user can hover over individual lines of the assembly code to highlight the corresponding lines of source code or vice versa to investigate the source of the potential leakage.

The overall GUI is based on open-source software. Information about the relevant licenses is available by clicking the question mark in the menu bar (1).

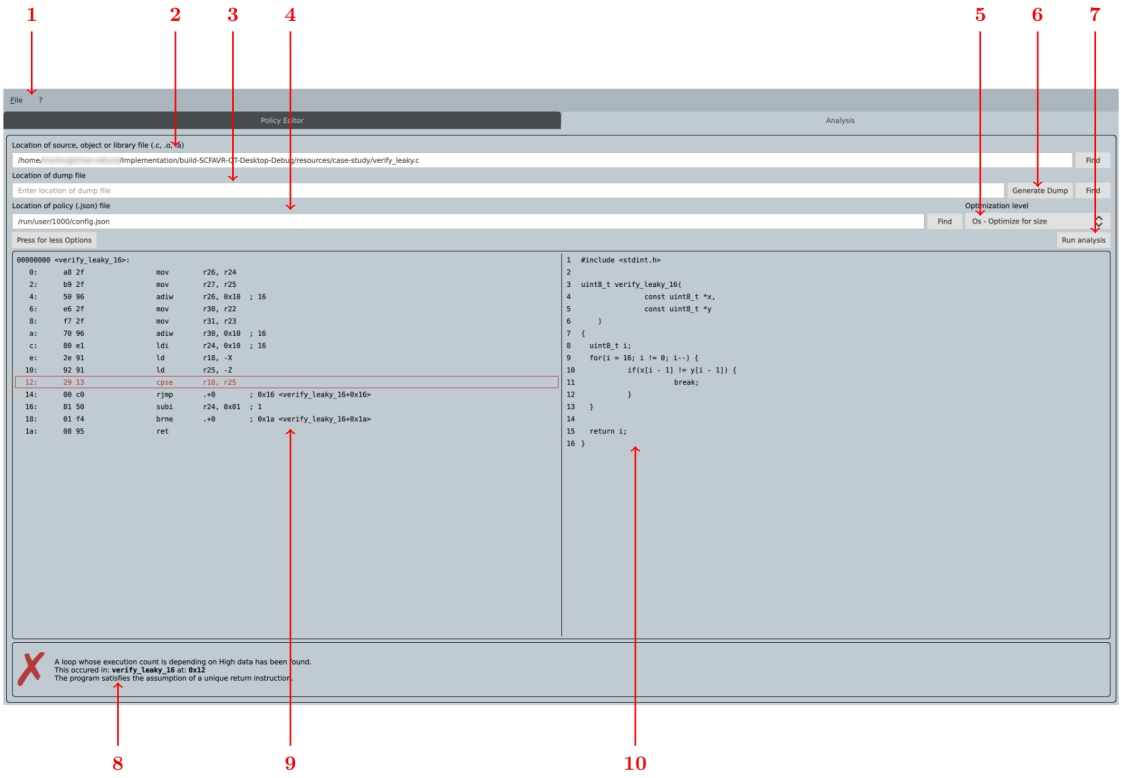

Fig. 4. Analysis Tab in the SCF-AVR GUI

2.2 Using the Interface for CA-0.3

CA-0.3 computes upper bounds on the cache-side-channel leakage of x86 binaries. The GUI for CA-0.3 consists of three tabs: a tab called *Analysis* for running analyses, a tab called *Options* and a tab called *Advanced Options*. The two latter tabs allow the user to configure the analysis.

Analysis Tab. Figure 5 shows the Analysis Tab of the CA-0.3 GUI. The GUI supports the user in reproducing prior analyses and managing libraries of analysis configurations and results. The user can select a library folder (11), browse the analyses in the folder (1) and manage the analyses in the folder (12) directly in the GUI. After loading an analysis from the library or choosing to create a new analysis, the user can configure the analysis using the different tabs (2–4). In the analysis tab, he can select the target binary (5) and choose a target function to analyze. As in the SCF-AVR GUI, he can select the target function from a list that is generated by the GUI (7). Optionally, the user can also load an existing analysis configuration file (6). When a fresh analysis is generated without an existing configuration file, the configurations are set to default values.

The user can start the analysis with the Run-Analysis Button (8). While the analysis with SCF-AVR, which is based on a type system, is very fast, the analysis with CA-0.3, which is based on abstract interpretation, might take significant amounts of time, depending on the complexity of the analyzed software. During the analysis, the user can view information about the analysis progress in the GUI (14) and adjust the level of detail for this information dynamically (13). He can also use a button to cancel the ongoing analysis (9).

After the analysis is concluded, the analysis results, i.e., the leakage bounds for the target function with respect to different cache-side-channel attacker models are displayed and the user can save them to the library for future reference.

Like for SCF-AVR, the GUI is based on open-source software. Information about the licenses for each component is available under (10).

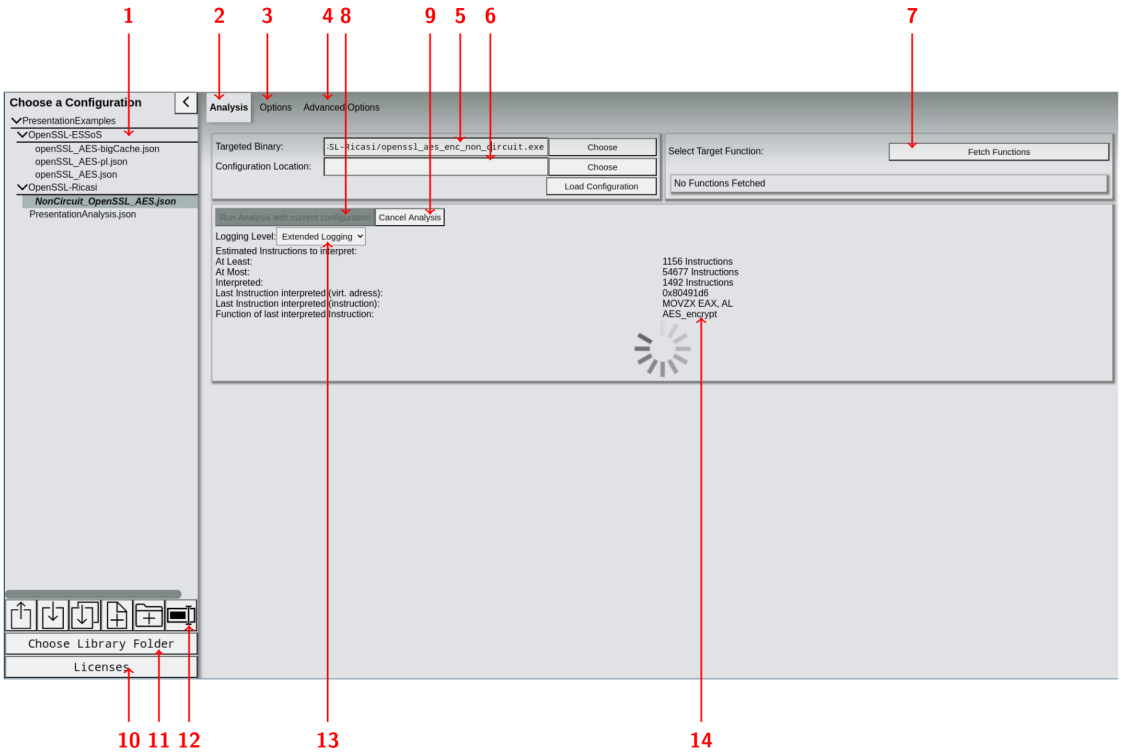

Fig. 5. Analysis Tab in the CA-0.3 GUI

Options Tab. The Options Tab of the CA-0.3 GUI, shown in Figure 6, allows the user to adapt basic configurations of the analysis. In particular, it allows the user to modify the start address of the function to be analyzed manually (1) and to adapt the cache configuration with respect to which the analysis is performed. The user can choose between a joint cache for instructions and data, two separate caches for instruction and data, and data cache only (2). He can then specify the parameters of the cache, namely the cache size, line size, associativity, and replacement strategy for the data cache (3) and, in case of split caches, for the instruction cache (4). He can also choose from a list of available abstract domains for each cache to specify the level of granularity at which the state of the respective cache will be tracked during the analysis. Finally, the user can specify the extent to which loop unrolling is applied during the analysis (5).

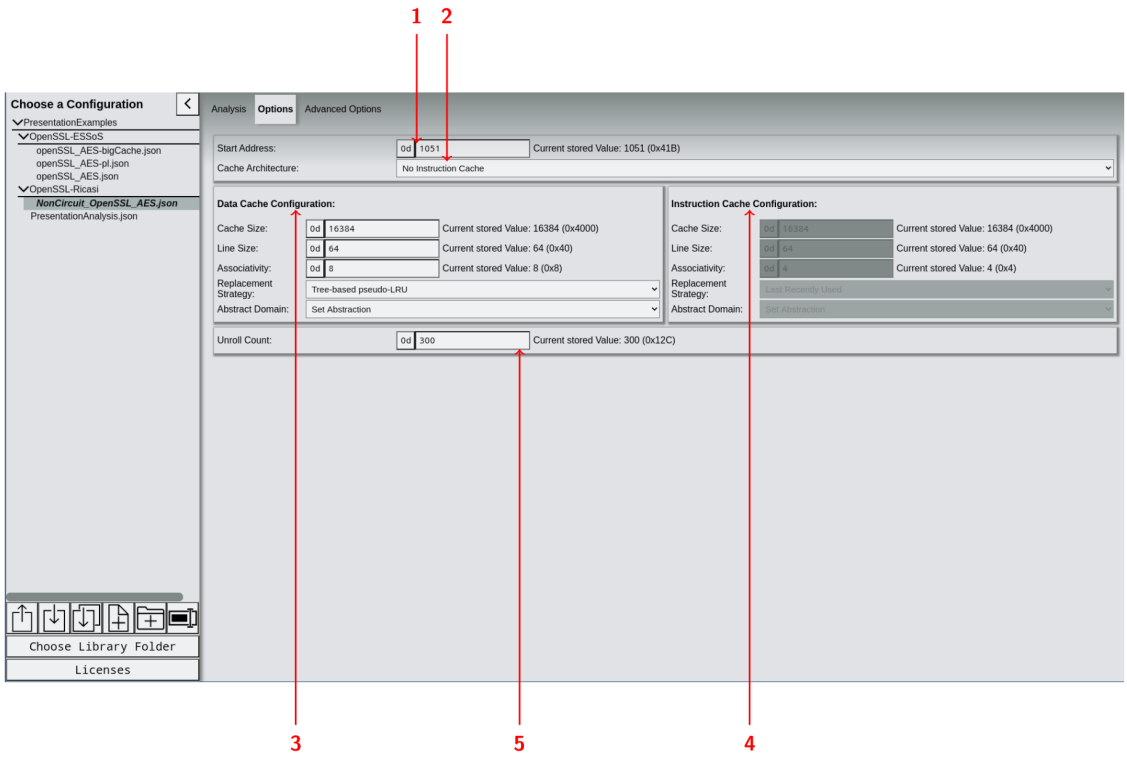

Fig. 6. Options Tab in the CA-0.3 GUI

Advanced Options Tab. The Advanced Options Tab, shown in Figure 7, offers additional configuration options to the user. The user can specify general analysis options like an end address and instruction base address (1). He can select which processor flags to track during the analysis or use the option to determine the minimum set of flags that are used in the target program automatically (3). He can enable consistency checks during the analysis (4) and specify which types of cache-side-channel attacker models the analysis should consider (5). Finally, the user can view the initial values that will be assigned to memory entries, registers, entries of the FPU stack and processor flags for the analysis (2).

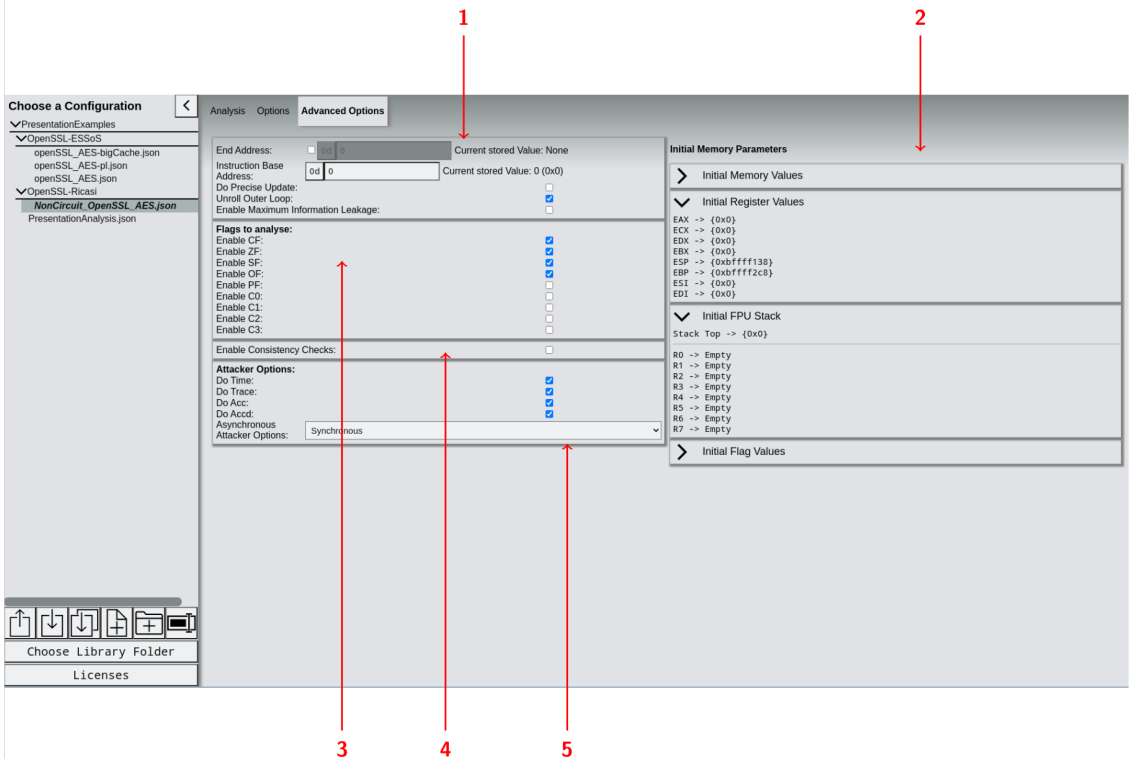

Fig. 7. Advanced Options Tab in the CA-0.3 GUI

3 Implementation

This section describes the structure of the GUI implementations and how they interface with the analysis tools. The GUI for the Python tool SCF-AVR is implemented using QML. The GUI for OCaml tool CA0.3 is implemented using ReasonML.

3.1 Implementation for SCF-AVR

SCF-AVR is implemented in Python. Our GUI is implemented in C++ with the Qt framework. Figure 8 provides an overview of the implementation.

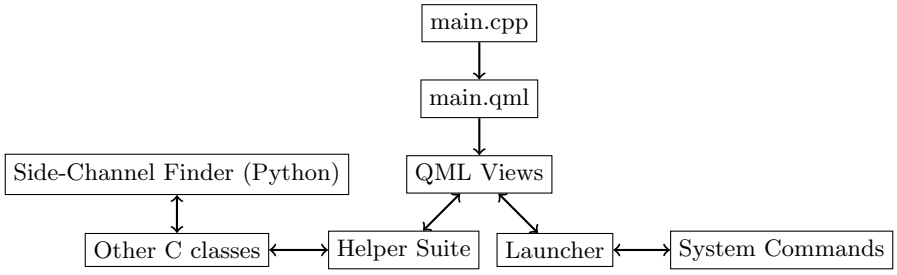

Fig. 8. Overview of the implementation of the SCF-AVR GUI

The UI elements are described in the Qt modeling language (QML). The file `main.qml` contains all elements of the UI. Larger elements and repeated elements are moved to their own QML files

and then imported into the main QML file. Notable examples are the components that contain the two main tabs of the GUI: the FileLocationView, which lets the user choose files to run an analysis on and presents its results, as well as the ConfigView, which allows for the generation and editing of JSON Configuration files. The file `main.cpp` loads `main.qml` and contains code for changing Qt settings, such as registering C++ types for use in QML or setting general configurations.

There are two C++ classes that can be directly accessed from QML: The Launcher and HelperSuite classes. The Launcher is responsible for calling system commands. The Helper Suite interacts with other C++ classes that control the Side-Channel Finder tool, running in python.

The HelperSuite has two types of functions, blocking and non-blocking. Blocking functions are those that return a value for which the UI will wait. This means it will be interrupted until the result of the function is available, thus blocking the UI. These functions should be used if the computation is relatively light and will not impede the use of the UI. For heavier computation (such as running the analysis) a non-blocking function should be used. These involve launching a separate thread which is given the parameters needed for executing the function. In this project an object of the class Worker is placed on the thread. Worker contains all needed functionality for actually performing the computation. The starting of the computation and the reporting of the results is done using the message-passing Qt Signal system. A signal is sent to the HelperSuite from QML, which sends a Signal to the worker thread to start the computation. Signals return nothing, and will therefore not block the UI. When the computation on the worker thread is done, it will send a signal containing the result to the HelperSuite which in turn informs QML of this. The Worker is then slated for deletion by the HelperSuite. The HelperSuite also contains an ObjDumpParser that is created by the Worker during Analysis to parse the generated object dump in order to give information on the location of a potentially identified leak in the AVR code.

3.2 Implementation for CA-0.3

The tool CA-0.3 is implemented in the CacheAudit framework in OCaml. To smoothly interface with the OCaml implementation, the graphical user interface is implemented in ReasonML. The implementation is complied to JavaScript code that uses the React library. This JavaScript code is then executed in an Electron browser. An overview of the technology involved is given in Figure 9.

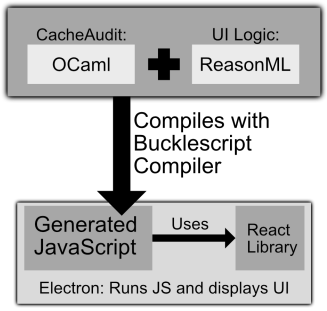

Fig. 9. Overview of technology for the CA-0.3 GUI

To compile to JavaScript, we use the Bucklescript compiler. To make CA-0.3 compatible with this compiler, we had to replace some library modules (Stdint, Big_Int, Float) used by CA-0.3 with alternatives. Furthermore, we made some changes to the file I/O and logging functionality to make the tool compatible with the GUI implementation.

The GUI implementation itself consists of multiple processes: a renderer process and two worker processes. The renderer process controls the overall program. It loads all UI control elements and displays the main window to the user. The worker processes runs in two hidden Electron browser

windows in the background: one process runs CA-0.3, and one runs tasks related to file-system interaction. This split avoids that the renderer process is blocked while a cache-side-channel analysis is running or while more complex file-system I/O is performed.

To communicate between the renderer and the workers, we use Electrons inter-process communication (IPC). Repeated listeners are set up along with a channel and a callback, since communication is not possible directly between windows. Each message sent from the renderer to the workers contains a token so that the reply can be uniquely identified. By using the `reductive` package for Reason, we provide a global state accessible from the entire UI. It is defined by a function for changing the state, a store containing the default values of the state and several selectors that allow one to retrieve a specific value from the global state. To use the global state, we wrap it as a React Component around all components that need it.

4 Design Principles

Principles for designing tools for security experts have been proposed by Jaferian, Botta, Hawkey and Beznosov in [12]. While the authors focus on systems for security monitoring, the principles were still a helpful inspiration during the implementation of our GUIs for SCF-AVR and CA-0.3. Below, we describe how selected principles have impacted our design choices in the implementation.

Knowledge sharing The guidelines suggest to enable security practitioners to share knowledge within their community. In our GUIs, we support the export and import of analysis configurations and results so that they can be shared between experts who are performing side-channel analyses. In particular, we expect the feature for the management of library folders in the GUI for CA-0.3 to simplify the sharing and reproduction of results among users of CA-0.3.

Multiple presentation formats The guidelines suggest to offer different presentation or visualization techniques for data to support security practitioners. In our GUI for SCF-AVR, we present the overall analysis result in a textual way, accompanied by an icon. In case a potential vulnerability has been detected, we additionally offer two alternative representations of the potential vulnerability: one at the level of assembly code and one at the level of C code.

Different levels of abstraction In addition to multiple presentation formats, the guidelines suggest to break down information into multiple levels of abstraction. Both our GUIs allow the user to work at multiple levels of abstraction. In the SCF-AVR GUI, the user can work with a target software at the level of C source code. Options for working with lower-level code or selecting a specific optimization-level for the compilation are hidden by default and can be activated with a button. The GUI also provides the option to suppress warnings about the availability of assembly functions if they are not relevant to the user. In the CA-0.3 GUI, the user can work in the Analysis Tab directly, using default configurations for his analysis, or configure the analysis at more fine-grained levels. The options for configuration are split into two levels of abstraction (the Options Tab with options likely to be required more frequently and the Advanced Options Tab with options that need manual adjustments only in special cases). Both GUIs are also compatible with the original, very low level of abstraction for configuring analyses through manually created configuration files.

5 Application to Case Studies

We tested our GUIs by applying them to multiple case studies that have been performed with SCF-AVR and CA-0.3 using their command-line interfaces in prior work.

5.1 Application of the SCF-AVR GUI

We applied our GUI for SCF-AVR to analyze a leaky string comparison and four implementations from the μNaCl crypto library that were already considered in [7].

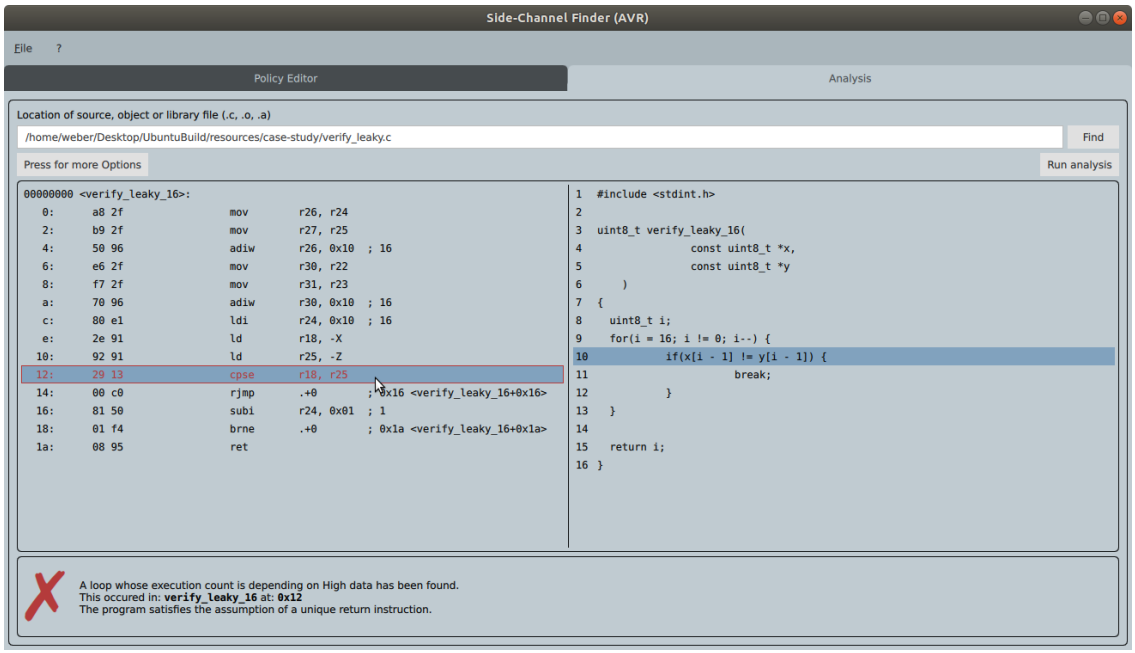

Fig. 10. Application of the SCF-AVR GUI to a leaky string comparison

The result for the leaky string comparison is shown in Figure 10. The detected vulnerability is the secret-dependent abortion of the comparison upon the first mismatch, as discussed in [7]. The GUI highlights the vulnerability in the assembly code on the left-hand side of the Analysis Tab. When we hover over the line that contains the vulnerability, the GUI highlights the corresponding lines of the C code on the right-hand side of the Analysis Tab as shown in Figure 10.

The results of analyzing the four implementations from the crypto library μNaCl (string comparison, Salsa20, xSalsa20, Poly1305) are shown in Figure 11. Since SCF-AVR verifies that all four implementations are secure with respect to timing side channels (see also [7]), the GUI reports the successful analyses in all four cases.

5.2 Application of the CA-0.3 GUI

We applied our GUI for CA-0.3 to analyze block-cipher implementations from the cryptographic library mbedTLS 2.16.5. More concretely, we analyzed the implementations of AES, DES, 3DES and Camellia that were already considered in [17]. The results of the analyses are shown in Figure 12. We were able to conveniently configure the analysis to the same cache setup as in [17] and obtained the same leakage bounds. With the library-management feature of our UI, we organized our analyses for easy future access (shown at the left side of the screenshots in Figure 12).

We also applied our GUI to two examples of functions from a software for Quantum Key Distribution (QKD), which were analyzed previously as part of the case study in [23]. The results are shown in Figure 13. The GUI supports the easy reproduction of the leakage bounds discussed for these functions in [23]. In the left-hand side screenshot in Figure 13, we see the 4-bit leakage bound with respect to the trace-based attacker model. This is the leakgae bound that lead to the detection of a vulnerability in the QKD software in [23]. In the right-hand side screenshot, we see the leakage bounds for the example of a function that is free from cache-side-channel leakage with respect to the four attacker models considered.

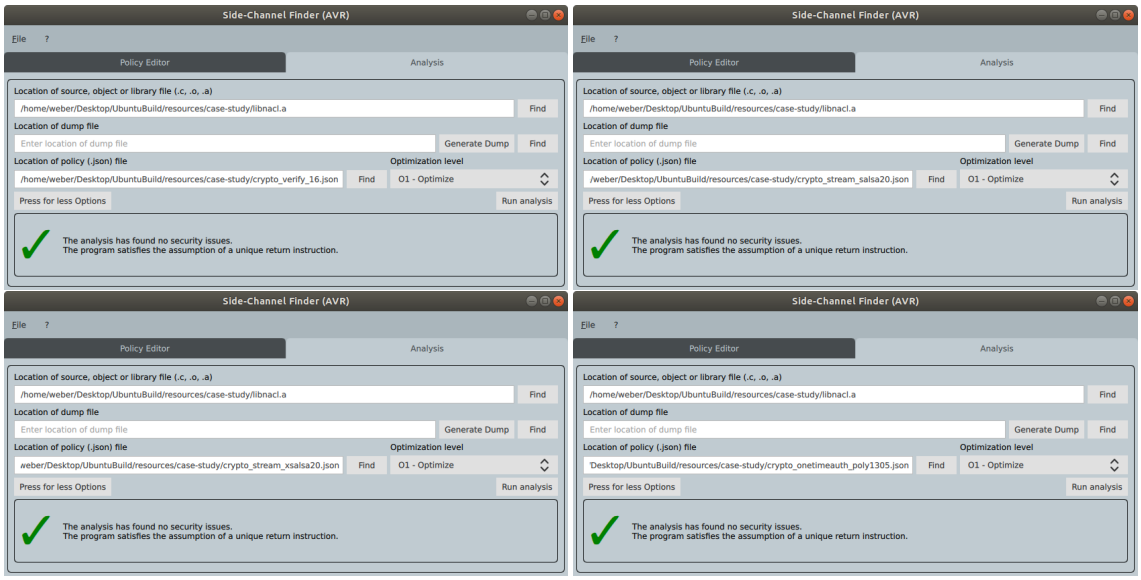

Fig. 11. Application of the SCF-AVR GUI to implementations from the crypto library μNaCl

6 Related Work

In addition to the analysis tools Side-Channel Finder AVR and CacheAudit 0.3, there are numerous side-channel analysis tools that target different types of side channels at different levels of abstraction and with different goals. Most closely related are the other tools from the CacheAudit framework [10]: the original analysis tools CacheAudit 0.1 [8] and CacheAudit 0.2 [10], the variant CacheAudit 0.2b with extended abstract domains for the analysis across different AES implementations [19], the variant CacheAudit 0.2c with extended abstract semantics for the analysis of lattice-based crypto [2], a version of CacheAudit for the analysis of more powerful attacker models [9], and CacheAudit-FPU with an abstract domain and abstract semantics for the analysis of binaries with floating-point instructions [23]. CacheAudit 0.3 [17] extends CacheAudit-FPU by support for the analysis of circuit-based implementations with large binaries.

Tools in the CacheAudit framework. All tools in the CacheAudit framework compute upper bounds on cache-side-channel leakage. Most variants focus on access-based attacker models, which can observe a snapshot of the cache state after the target program has executed, a trace-based attacker model, which can observe the trace of cache hits and misses, and a time-based attacker model, which can observe the duration of cache hits and misses. The variant from [9] considers additional attacker models, which can observe the trace of memory-access addresses at different granularities.

Other cache-side-channel analysis tools. Other tools that target cache side channels include CacheD [22], CacheS [21], CaSym [3], Abacus [1] and CHALICE [4]. CacheD and CacheS detect cache side channels at the binary level. They focus on an attacker model that can observe the trace of memory-access addresses at cache-line granularity and, in case of CacheS, also the outcome of branching decisions. CaSym detects cache side channels with respect to an access-based attacker model, which can observe snapshots of the cache state at some points during the execution, and which works at the level of the LLVM intermediate representation. Abacus and CHALICE quantify cache side channels at the binary level. The former focuses on a memory-access-trace attacker model at cache-line granularity, which can also observe branching decisions. The latter focuses on attacker models that are functions of the trace of cache hits and misses or of the number of memory blocks accessed in each cache set.

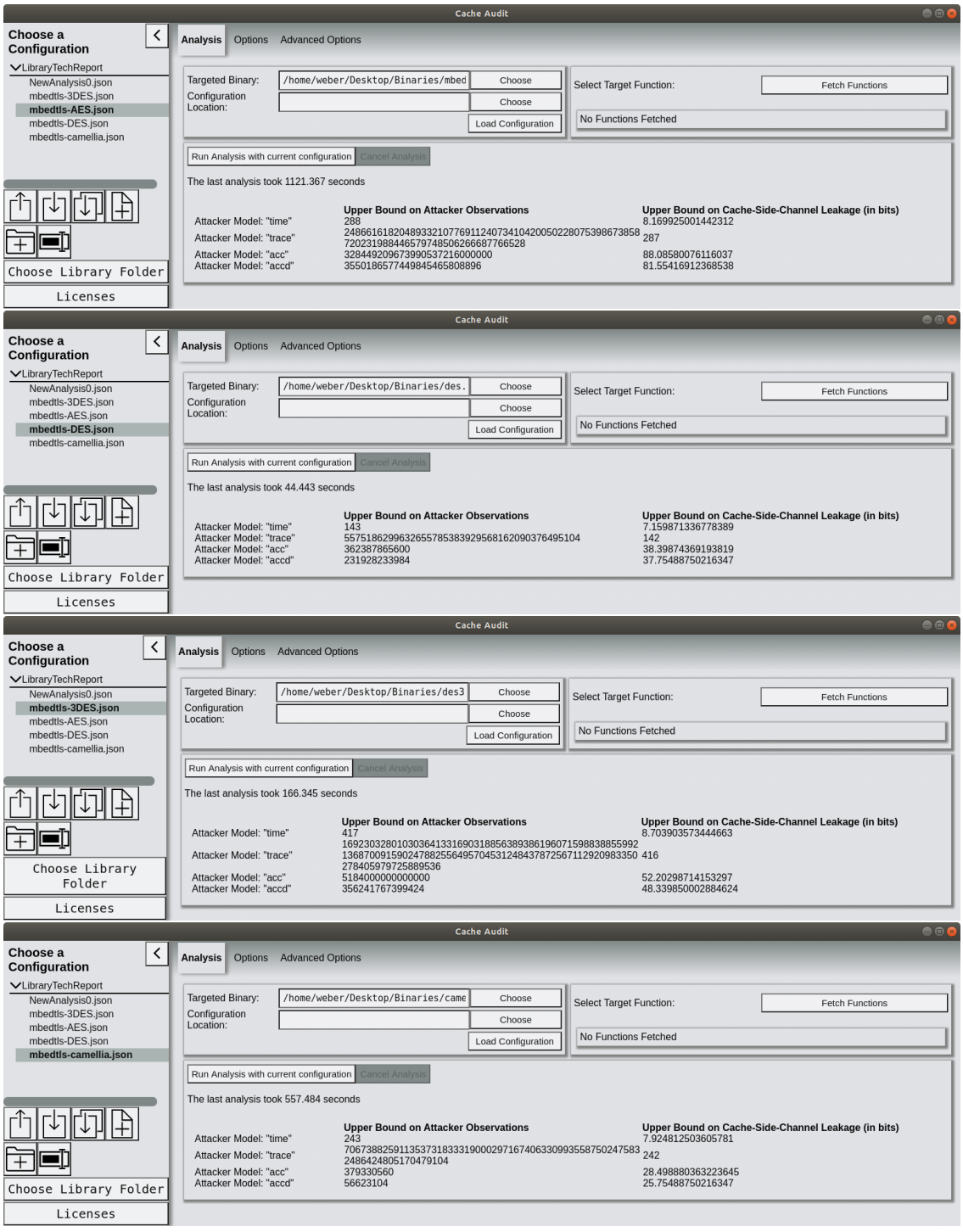

Fig. 12. Application of the CA-0.3 GUI to selected block-cipher implementations

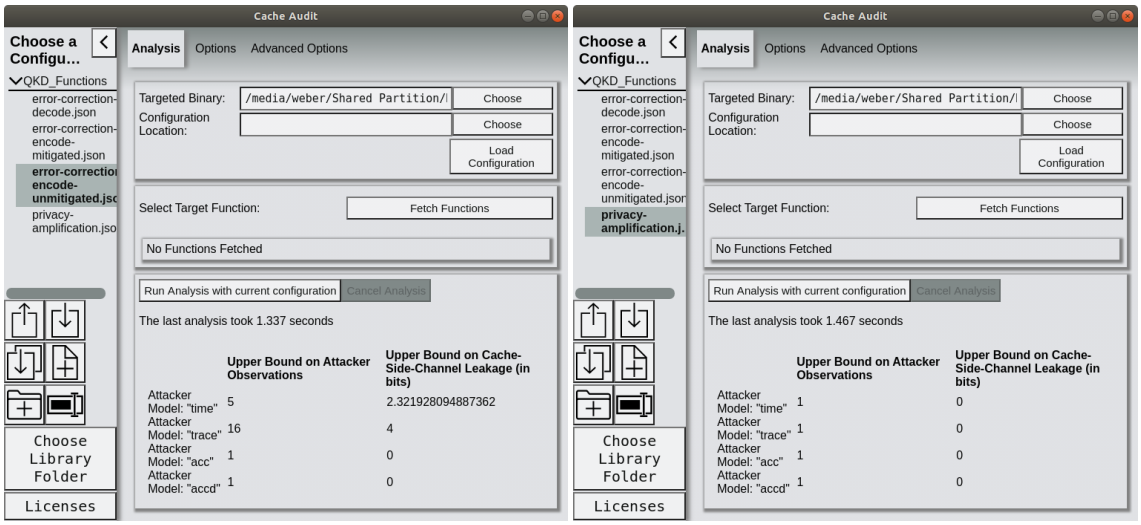

Fig. 13. Application of the CA-0.3 GUI to selected functions from a QKD software

Other timing-side-channel analysis tools. Tools that target timing side channels include [16] and SPASCA [14]. While Side-Channel Finder AVR aims at detecting timing side channels at the level of AVR assembly, the tool from [16] quantifies side channels, including timing side channels, at the level of Java bytecode. SPASCA detects potential timing side channels at the Java source-code level. It is implemented as an Eclipse plugin and supports crypto developers by highlighting the detected potential leaks directly in the development environment.

Tools based on experimental data. While some of the above-mentioned tools (CacheD, Abacus, CHALICE) utilize dynamically measured execution traces to optimize their analyses, there are also approaches that are completely based on experimental data. These include, e.g., LeakWatch [6], DATA [24] and distinguishing experiments [18]. LeakWatch approximates the leakage of Java programs (with annotations that specify the attacker observation) based on sample measurements collected from repeated executions. DATA detects side channels with respect to an attacker model that can observe address traces that include the sequence of instruction pointers and the full addresses of instruction operands. It records traces across multiple sample executions, detects differences in the traces and classifies the differences to (1) reduce false positives and (2) determine the type of information that might leak. The latter can be seen as an implicit quantification of the leakage. Distinguishing experiments quantitatively assess the leakage of implementations based on the probability that an attacker can successfully distinguish between secret inputs based on sample observations through a side channel, e.g., a software-based energy side channel as in [18].

Design of user interfaces for security analysis tools. The design of user interfaces for security-related applications has been investigated from the end-user perspective, e.g., in [11] for permission-granting on Android. The perspective of security experts has been considered, e.g., in [5] and [12]. Both focus mainly on security-monitoring tools, but multiple of the proposed guidelines are independent of the concrete type of security tool. For instance, both [5] and [12] suggest to support working at multiple levels of abstraction and sharing knowledge in a community. Such guidelines were a helpful inspiration for our side-channel analysis GUIs.

7 Conclusion

In this report, we presented graphical user interfaces for two side-channel analysis tools: the tool SCF-AVR [7] for detecting timing side channels in AVR assembly programs, and the tool

CA-0.3 [17] for computing upper bounds on the leakage of x86 binaries through cache side channels. Our interfaces allow security experts who use the tools for side-channel analysis to create analysis configurations, perform analyses and manage both, configurations and results, more conveniently. The interfaces are inspired by guidelines for designing security-management tools in order to support side-channel analysts to share knowledge and to work with multiple presentation formats and at multiple levels of abstraction. We successfully applied the GUIs to prior case studies that had originally been analyzed using command-line interfaces.

Acknowledgments. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1119 - 236615297.

References

1. Q. Bao, Z. Wang, X. Li, J. R. Larus, and D. Wu. Abacus: Precise Side-Channel Analysis. In *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 797–809. IEEE Computer Society, 2021.
2. N. Bindel, J. Buchmann, J. Krämer, H. Mantel, J. Schickel, and A. Weber. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. In *Postproceedings of the 10th International Symposium on Foundations & Practice of Security (FPS)*, LNCS 10723, pages 225–241. Springer, 2017.
3. R. L. Brotzman, S. L. Liu, D. Zhang, G. Tan, and M. T. Kandemir. CaSym: Cache Aware Symbolic Execution for Side Channel Detection and Mitigation. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P)*, pages 505–521. IEEE Computer Society, 2018.
4. S. Chattopadhyay, M. Beck, A. Rezine, and A. Zeller. Quantifying the Information Leakage in Cache Attacks via Symbolic Execution. *ACM Transactions on Embedded Computing Systems*, 18(1):1–27, 2019.
5. S. Chiasson, R. Biddle, and A. Somayaji. Even Experts Deserve Usable Security: Design guidelines for security management systems. In *2007 SOUPS Workshop on Usable IT Security Management (USM)*, pages 1–4, 2007.
6. T. Chothia, Y. Kawamoto, and C. Novakovic. LeakWatch: Estimating Information Leakage from Java Programs. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, LNCS 8713, pages 219–236. Springer, 2014.
7. F. Dewald, H. Mantel, and A. Weber. AVR Processors as a Platform for Language-Based Security. In *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS)*, LNCS 10492, pages 427–445. Springer, 2017.
8. G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, pages 431–446. USENIX Association, 2013.
9. G. Doychev and B. Köpf. Rigorous Analysis of Software Countermeasures against Cache Attacks. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementations (PLDI)*, pages 406–421. ACM, 2017.
10. G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. *ACM Transactions on Information and System Security (TISSEC)*, 18(1):4:1–4:32, 2015.
11. P. Gerber, M. Volkamer, and K. Renaud. The simpler, the better? Presenting the COPING Android permission-granting interface for better privacy-related decisions. *Journal of Information Security and Applications (JISA)*, 34(1):8–26, 2017.
12. P. Jaferian, D. Botta, K. Hawkey, and K. Beznosov. Design guidelines for IT security management tools. In *2008 SOUPS Workshop on Usable IT Security Management (USM)*, pages 1–6, 2008.
13. P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference (CRYPTO)*, LNCS 1109, pages 104–113. Springer, 1996.
14. X. Li, H. Mantel, J. Schickel, M. Tasch, I. Toteva, and A. Weber. SPASCA: Secure-Programming Assistant and Side-Channel Analyzer. Technical Report TUD-CS-2017-0303, TU Darmstadt, 2017.
15. M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P)*. IEEE Computer Society, 2021. To appear.

16. P. Malacaria, M. Khouzani, C. S. Pasareanu, Q.-S. Phan, and K. S. Luckow. Symbolic Side-Channel Analysis for Probabilistic Programs. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF)*, pages 313–327. IEEE Computer Society, 2018.
17. H. Mantel, L. Scheidel, T. Schneider, A. Weber, C. Weinert, and T. Weißmantel. Ricasi: Rigorous cache side channel mitigation via selective circuit compilation. In *Proceedings of the 19th International Conference on Cryptology and Network Security (CANS)*, LNCS 12579, pages 505–525. Springer, 2020.
18. H. Mantel, J. Schickel, A. Weber, and F. Weber. How Secure is Green IT? The Case of Software-Based Energy Side Channels. In *Proceedings of the 23rd European Symposium on Research in Computer Security (ESORICS)*, LNCS 11098, pages 218–239. Springer, 2018.
19. H. Mantel, A. Weber, and B. Köpf. A Systematic Study of Cache Side Channels across AES Implementations. In *Proceedings of the 9th International Symposium on Engineering Secure Software and Systems (ESSoS)*, LNCS 10379, pages 213–230. Springer, 2017.
20. D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In *Proceedings of the 6th Cryptographer's Track at the RSA Conference (CT-RSA)*, LNCS 3860, pages 1–20. Springer, 2006.
21. S. Wang, Y. Bao, X. Liu, P. Wang, D. Zhang, and D. Wu. Identifying Cache-Based Side Channels through Secret-Augmented Abstract Interpretation. In *USENIX Security*, pages 657–674. USENIX Association, 2019.
22. S. Wang, P. Wang, X. Liu, D. Zhang, and D. Wu. CacheD: Identifying Cache-Based Timing Channels in Production Software. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 235–252. USENIX Association, 2017.
23. A. Weber, O. Nikiforov, A. Sauer, J. Schickel, G. Alber, H. Mantel, and T. Walther. Cache-Side-Channel Quantification and Mitigation for Quantum Cryptography. In *Proceedings of the 26th European Symposium on Research in Computer Security (ESORICS)*, pages 235–256. Springer, 2021.
24. S. Weiser, A. Zankl, R. Spreitzer, K. Miller, S. Mangard, and G. Sigl. DATA – Differential Address Trace Analysis: Finding Address-based Side-Channels in Binaries. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, pages 603–620. USENIX Association, 2018.
25. Y. Yarom and K. Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*, pages 719–732. USENIX Association, 2014.