NOTICE: This is the author's version of a work that was accepted to *Signal Processing: Image Communication* in 2013. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version has been published in *Signal Processing: Image Communication*, vol. 28, no. 10 (special issue on Recent Advances on MPEG Codec Configuration Framework), pp. 1315-1334, 2013, Elsevier. DOI: 10.1016/j.image.2013.08.015.

Secure Computing with the MPEG RVC Framework

Junaid Jameel Ahmad[a], Shujun Li[b], Richard Thavot[c], Marco Mattavelli[c]

[a] *University of Konstanz, Germany*
[b] *University of Surrey, UK*
[c] *École Polytechnique Fédérale de Lausanne (EPFL), Switzerland*

Abstract

Recently, ISO/IEC standardized a dataflow-programming framework called Reconfigurable Video Coding (RVC) for the specification of video codecs. The RVC framework aims at providing the specification of a system at a high abstraction level so that the functionality (or behavior) of the system become independent of implementation details. The idea is to specify a system so that only intrinsic features of the algorithms are explicitly expressed, whereas implementation choices can then be made only once specific target platforms have been chosen. With this system design approach, one abstract design can be used to automatically create implementations towards multiple target platforms. In this paper, we report our investigations on applying the methodology standardized by the MPEG RVC framework to develop secure computing in the domains of cryptography and multimedia security, leading to the conclusion that the RVC framework can successfully be applied as a general-purpose framework to other fields beyond multimedia coding. This paper also highlights the challenges we faced in conducting our study, and how our study helped the RVC and the secure computing communities benefited from each other. Our investigations started with the development of a Crypto Tools Library (CTL) based on RVC, which covers a number of widely used ciphers and cryptographic hash functions such as AES, Triple DES, ARC4 and SHA-2. Performance benchmarking results on the RVC-based AES and SHA-2 implementations in both C and Java revealed that the automatically generated implementations can achieve a comparable performance to some manually-written reference implementations. We also demonstrated that the RVC framework can easily produce implementations with multi-core support without any change to the RVC code. A security protocol for mutual authentication was also implemented to demonstrate how one can build heterogeneous systems easily with RVC. By combining CTL with Video Tool Library (a standard library defined by the RVC standard), a non-standard RVC-based H.264/AVC encoder and a non-standard RVC-based JPEG codec, we further demonstrated the benefits of using RVC to develop different kinds of multimedia security applications, which include joint multimedia encryption-compression schemes, digital watermarking and image steganography in JPEG compressed domain. Our study has shown that RVC can be used as a general-purpose implementation-independent development framework for diverse data-driven applications with different complexities.

Keywords: Reconfigurable Video Coding (RVC), Secure Computing, Crypto Tools Library (CTL), Video Tool library (VTL), Multimedia Security, Cryptography.

1. Introduction

Security and privacy are very important features of nowadays communications and information systems. Designing and implementing such features on the wide variety and heterogeneity of processing platforms

requires crossing the boundaries of multiple systems (e.g., general-purpose PCs/laptops, smart-phones, tablets, network routers/repeaters, cellular base-stations, wireless sensor nodes and many others) and implies to be able of porting applications that include video, audio and data processing components interleaved with security processing components on heterogeneous processing systems. This poses several challenges to the system designers (including security designers) since portability, reusability and top-down system designs still are elusive features on current system design practice.

In order to meet similar challenges for the implementation-independent[1] development of video/graphics codecs, ISO/IEC has recently standardized a framework initially called MPEG RVC (Reconfigurable Video Coding) [2, 3] and recently being renamed to RMC, where the M stands for "Media" to cover the inclusion of 3-D graphics and audio codecs. The MPEG RVC framework is based on the asynchronous dataflow programming paradigm [4] and adopts a computation model defined by the formal language used to express the dataflow program [5]. The RVC standard framework intends to provide systems specifications with a number of distinctive features such as *modularity*, *reusability*, *reconfiguration*, *implementation independence*, *code analyzability*. Modularity and reusability help to simplify the design of complicated programs by having functionally separated and reusable computational blocks; reconfigurability makes reconfiguration of complicated programs easier by offering an interface to configure and replace computational blocks; code analyzability allows automatic analysis of both the source code and the functional behavior of each computational block so that code conversion and program optimization can be done in a more systematic manner. The automated code analysis enables to conduct a fully-/semi-automated design-space exploitation to find critical paths and/or parallel data-flows, which suggests different optimization refactorings (merging or splitting) of different computational blocks [6], and/or to achieve concurrency by mapping different computational blocks to different computing resources [7]. In contrast to the traditional sequential programming paradigm, the dataflow programming paradigm is ideally suited for such optimizations thanks to its data-driven nature.

Just like the video codecs and other signal processing systems, cryptosystems such as ciphers and security protocols are also data-driven in nature so it is quite natural to specify and design them as dataflow systems.

In this paper, we extend our recently published work on secure computing with the RVC framework in two conference papers [1, 8] and report some new results we obtained after the above two papers were published. More specifically, this paper presents a global view of our investigations on how the RVC framework can be effectively used to address the above-mentioned system design challenges for secure computing on diverse platforms.

To support development of different kinds of secure computing applications with the RVC framework, we first developed a general-purpose library of cryptographic primitives called Crypto Tools Library (CTL), which is an open and implementation-independent cryptographic library used in all RVC-based secure computing applications we developed.The availability of these cryptographic primitives as reusable components in a general-purpose library allows developers and researchers to build more complicated applications involving cryptography with ease. The cryptosystems included in CTL also allowed us to carry out some run-time performance benchmarking studies on both single-core and multi-core systems, which led to the positive conclusion that the automatic code generation step of the RVC framework does not compromise the run-time performance of the synthesized implementations. In addition, the CTL can also serve as a simpler, but still sufficiently rich test bed for further improvement of supporting tools of the RVC framework.

Based on CTL, we developed a number of secure computing applications to investigate the potential of RVC for secure computing as a new programming environment. These applications include: an hash tree, a security protocol for mutual authentication which involves both hardware and software components, and a number of multimedia security applications. All applications have been functionally validated by generating real implementations executable on various targeted platforms.

[1] In [1] we used the term "platform-independent", where the word "platform" has a broader meaning than what it implies in programming languages. Basically, it denotes any computing environment that can execute/interpret code or compile code to produce executable programs, which includes both hardware and software platforms and also hybrid hardware-software systems. Since the word "platform" may cause some confusion, in this paper we switch to the more accurate term "implementation-independent".

The rest of the paper is organized as follows. In Sec. 2, we give a brief overview of related work, focusing on a comparison 1) between the RVC and other existing dataflow solutions, 2) between CTL and other existing cryptographic libraries and systems. Section 3 gives a brief overview of the dataflow programming languages specified in RVC standard and available supporting tools. Section 4 discusses our motivation behind the reported work and challenges we met in applying the RVC framework to secure computing, and how our study helped to evolve the RVC framework and related supporting tools, which then made development of secure computing more easily with RVC. In Sec. 5, we formally introduce the CTL by presenting its design principles, list of currently available cryptosystems/utilities and some system examples of cryptosystems. In addition, this section also reports the performance benchmarking study of AES on some single-core platforms (two general-purpose PCs and an embedded system). In Sec. 6 two cryptographic applications are presented: 1) a hash tree application, which is specifically designed to study the performance benchmarking of SHA-256 in CTL and to demonstrate the seamless support of the MPEG RVC framework for multi-core systems, 2) a mutual authentication protocol demonstrating how RVC supports hardware/software co-design. Then in Sec. 7, we present four multimedia security applications covering joint multimedia encryption-compression, digital watermarking and image steganography are described. In Sec. 8, the paper draws some concluding remarks on the usage of the MPEG RVC framework for secure computing applications and give some directions for future work.

2. Related Work

The basic concepts of dataflow can be traced back to the 1960s [4] and the past half century has witnessed the development of many dataflow model of computations and development tools (of which some have also been used widely in industry), but few formal dataflow languages capable of expressing a wide range of model of computation. The RVC framework whose standardization work started quite recently, in fact the first edition was released as late as in December 2009 [9] tried to build on the most recent and advanced dataflow developments. A summary of advantages of RVC over other solutions is given in Table 1. We emphasize that this comparison focuses on the features relevant to achieve the goals of this paper, so it should not be considered as an exhaustive overview of all pros and cons of the evaluated solutions[2].

In addition to the dataflow programming and cryptographic development frameworks listed in Table 1, many cryptographic libraries have been developed over the years (e.g., Crypto++ [29], Cryptlib [30], OpenSSL cryptographic library [31], Qilin [32], Bouncy Castle [33], PureNoise CryptoLib [34], sphlib [35]), but very few can support multiple programming languages or some other features we evaluated in Table 1. Some libraries do support more than one programming language, but often in the form of separate sets of source code and separate programming interfaces/APIs (e.g. [33]) or available as commercial software only (e.g. [25, 26]). There is also a large body of optimized implementations of cryptosystems in the literature [36, 37, 38, 39, 40, 41, 42], which normally depend even more on the specifics of the target platforms (e.g., the processor architecture and/or special instruction sets [43, 37, 38, 44]).

Despite the fact that the RVC possesses so many distinctive features in comparison to other existing alternative solutions, the wide usage of the RVC framework for different kinds of data-driven systems, especially multimedia (video, audio, image and graphics) codecs [45, 46, 47, 48, 49, 50, 51], also encouraged us to use the RVC framework to conduct our study on the design and development of secure computing applications in the dataflow programming paradigm.

Hence, as the first step of our efforts, we used the RVC framework to develop a general-purpose library we call Crypto Tools Library (CTL) as the foundation of further development of more complex secure computing applications such as security protocols and multimedia security systems. The development and performance benchmarking aspects of SHA-2 of the CTL were reported in [1].

In addition, in [8], we highlighted some challenges being faced by developers while building multimedia security applications in imperative languages (e.g., C/C++, Java, etc.) and discussed how those challenges can be addressed by developing multimedia security applications in the RVC framework.

[2]This table is slightly updated edition of Table 1 presented in [1].

Table 1: Comparison of RVC framework with other candidate solutions. Candidates with similar characteristics are grouped together. These categories include 1) high-level specification languages for hardware programming languages, 2) frameworks for hardware/software co-design, 3) commercial products, and 4) other cryptographic libraries. The columns in the table represent the following features: A) high-level (abstract) modeling and simulation; B) platform independence; C) code analyzability (i.e., semi-automated design-space exploitation); D) hardware code generation; E) software code generation; F) hardware/software co-design; G) supported target languages; H) open-source or free implementations; I) international standard.

Cat.	Candidate	A	B	C	D	E	F	G	H	I
	RVC	✓	✓	✓	✓	✓	✓	C, C++, Java, LLVM, Verilog, VHDL, XLIM, PROMELA	✓	✓
1	Handel-C [10]	-	-	-	✓	-	-	VHDL	-	-
	ImpulseC [11]	-	-	-	✓	-	✓	VHDL	-	-
	Spark [12]	-	-	-	✓	-	✓	VHDL	-	-
2	BlueSpec [13]	✓	-	✓	✓	✓	-	C, Verilog	-	-
	Daedalus [14]	✓	✓	✓	✓	✓	✓	C, C++, VHDL	✓	-
	Koski [15]	✓	✓	✓	✓	✓	✓	C, XML, VHDL	-	-
	PeaCE [16]	✓	✓	✓	✓	✓	✓	C, C++, VHDL	✓	-
3	CoWare [17]	✓	✓	-	✓	✓	✓	C, VHDL	-	-
	Esterel [18]	-	✓	-	✓	✓	-	C, VHDL	✓	-
	LabVIEW [19]	✓	✓	✓	-	-	-	-	-	-
	Matlab [20, 21]	✓	✓	✓	✓	✓	-	C, C++, Verilog, VHDL	-	-
	Synopsys System Studio [22]	✓	✓	✓	✓	✓	✓	C++, SystemC, SystemVerilog	-	-
4	CAO [23, 24]	✓	✓	-	-	✓	-	C, x86-64 assembly, ARM	-	-
	Cryptol [25, 26]	✓	✓	✓	✓	✓	-	C, C++, Haskell, VHDL, Verilog	-	-
	Charm [27, 28]	✓	-	-	-	-	-	-	✓	-

This paper present a global view of our study on the secure computing with the RVC framework and presents some new performance benchmarking results and applications, which were not reported in [1, 8]. More specifically, this paper presents our perfromance benchmarking results of AES on different single-core platforms (general-purpose PCs, JVMs and an embedded system) and some improved performance benchmarking results of hash tree application on a quad-core machine. In addition, we also report two new applications: 1) a mutual authentication cryptographic protocol highlighting hardware/software co-design supported by the RVC framework, 2) an image steganography application in JPEG compressed-domain. In the future work, we also give a discussion on the potential of using the RVC framework for privacy-preserving applications in particular those based on garbled circuits [52], which will allow us to demonstrate how design-space exploitation and parallelism optimization can be done in the RVC framework.

3. Reconfigurable Video Coding (RVC)

Based on the dataflow programming paradigm, the ISO/IEC standardized the Reconfigurable Video Coding (RVC) framework in 2009 [9] after its working group JTC 1/SG 29/WG 11 (MPEG) had been working on it for around three years. The main objective of developing and standardizing the RVC framework was to provide an effective solution to the technical challenges incurred in developing video codecs. One of those was to make the video codecs more reconfigurable, meaning that codecs with different configurations (e.g., different video coding standards, different profiles and/or levels, different system requirements) can

be built on the basis of a unified set of implementation-independent building blocks. In order to achieve this goal, the RVC standard defines a framework that covers different normative steps for the standard specification of a processing system. In addition, it also provides the appropriate starting point for all non-normative steps needed to support the whole development life-cycle for systems' implementation. More precisely, supporting tools [53, 54, 55] have also been developed to ease the development, simulation of the applications and code generation towards different target programming languages. These supporting tools make the RVC framework not only a standard, but also a real development environment ready to be used for the design and development of systems.

In essence, the RVC framework allows developers to work with a single implementation-independent design at a higher level of abstraction while still being able to generate multiple editions of the same design that target different platforms (e.g., general-purpose PCs, embedded systems, FPGAs, etc.). It is deserved mentioning here that RVC can also support platform-dependent elements in the form of native actors (FUs containing native functions/procedures [56]) without compromising the platform-independent core of the framework (i.e. native construct is ignored by the abstract model without influencing the I/O behavior of anything in an FU network). In addition, the RVC framework also supports heterogeneous systems by enabling the partitioning of any single abstract design into different parts running as software and/or hardware components. But in order to make these heterogeneous components to talk with each other, a communication interfacing mechanism is required. In [57], authors proposed to interface the communication channel between any two heterogeneous components using "wrappers" (see Fig. 1). A wrapper is a channel "driver" sitting between the abstract FU and the platform-dependent I/O channel. To interface the communication between two heterogeneous components connected via a channel, two wrappers are needed, where first wrapper transforms the data from the format of source component (e.g., a software module running on a PC) to the format of the communication channel and the second wrapper transform the data from the format of channel to the format of the destination component (a hardware module running on an FPGA). These wrapper FUs are implemented as "native actors", meaning that wrappers FUs have to be implemented separately for each target platform using the specifics (e.g., system calls etc.) available at that target platform. Moreover, the RVC framework is based on two formal languages (RVC-CAL and FNL) that allow fully automated code analysis to facilitate design-space exploration for implementations running on multi-core and many-core systems [6, 46, 7].

As we mentioned in Sec. 1, any application domains with a data-driven nature (such as signal processing systems, communication systems) can take advantage of the distinctive features of the RVC framework. In this paper, we show that RVC dataflow programming paradigm is also suited for cryptosystems and many other secure computing systems.

3.1. How does the RVC Framework Work?

Figure 2 illustrates how an application is designed and how target implementations are generated with the RVC framework. At the design stage, different FUs (if not available in some existing libraries) are first written in RVC-CAL to describe their functional and I/O behavior, and then an FU network is specified to build the functionality of a whole application. The FU network can be composed by simply connecting all FUs (by selecting from existing libraries or the newly written FUs) to form a directed graph which is represented as an XDF file following the standardized FNL language. The connections between the FUs can be done graphically by means of a supporting tool called Graphiti Editor [55], which translates the graphical FU network description into a textual description written in FNL. The FUs and the FU network are instantiated to form an abstract model. This abstract model can be simulated to test its functionality without the need to assume the execution on any specific platform. Currently two tools support the simulation of RVC applications: ORCC [54] and OpenDF [53].

Note that, in Fig. 2 we show only two libraries to illustrate the concept. But in principle, users can simultaneously use as many libraries as they need. For example, the development of multimedia security applications presented later in Sec. 7, involved the following four libraries: 1) the standard video tool library (defined in the MPEG-C Part 4 [3]) for the H.264/AVC video decoder, 2) a non-standard tool library for the H.264/AVC video encoder, 3) a non-standard tool library for the JPEG encoder and decoder, 4) the

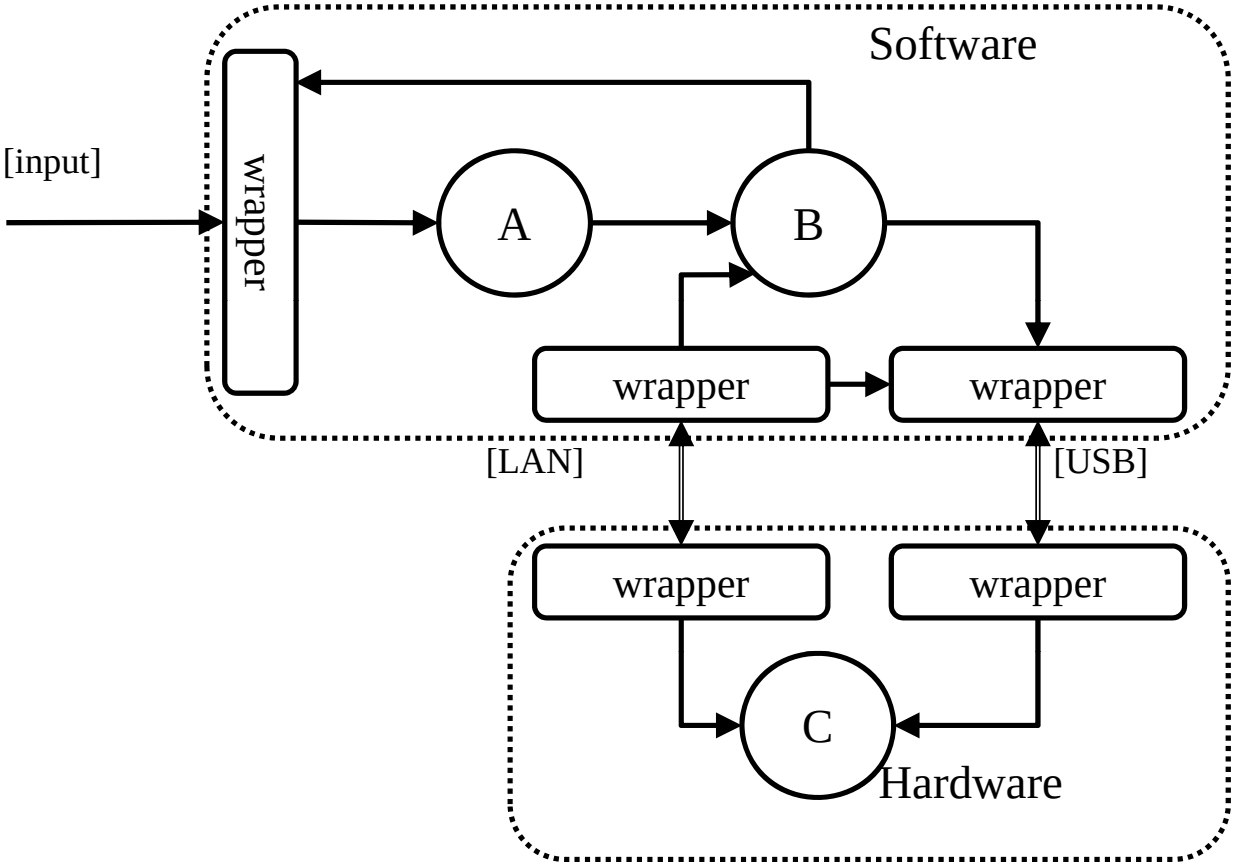


Figure 1: The structure of a typical heterogeneous system with wrappers.

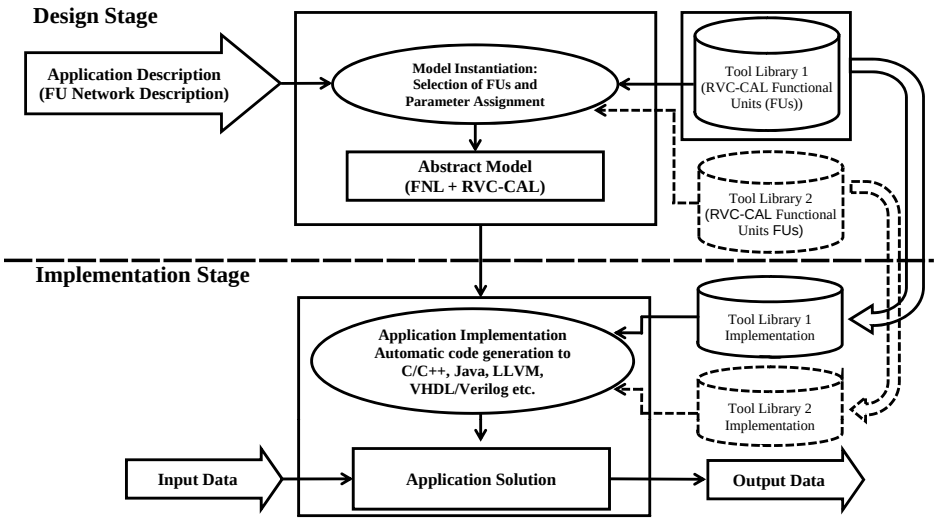


Figure 2: Process of application implementation generation in the RVC framework.

non-standard Crypto Tools Library (CTL). For the development of cryptographic applications, we only used the CTL since no multimedia data is involved.

At the implementation stage, the source code written in other target programming languages can be generated from the abstract model *automatically*. ORCC contains code generation backends for C, C++, Java, LLVM, Verilog, XLIM, and PROMELA. In addition, OpenDF also includes a Verilog HDL code generation backend. ORCC is currently more widely used in the RVC community and it is also the choice of our work reported in this paper.

4. Secure Computing with RVC

In this section, we first discuss the motivation behind the idea of secure computing in the RVC framework. Then, we highlight the key differences between secure computing applications and multimedia codecs. Then, we focus on identifying how our study can be helpful in evolving the RVC (standard, tools etc.) and development in the secure computing domain. In a nutshell, this section bridges the gap between the state-of-the-art (of RVC and secure computing domains) and the contributions being delivered by this study.

4.1. Why Secure Computing with RVC?

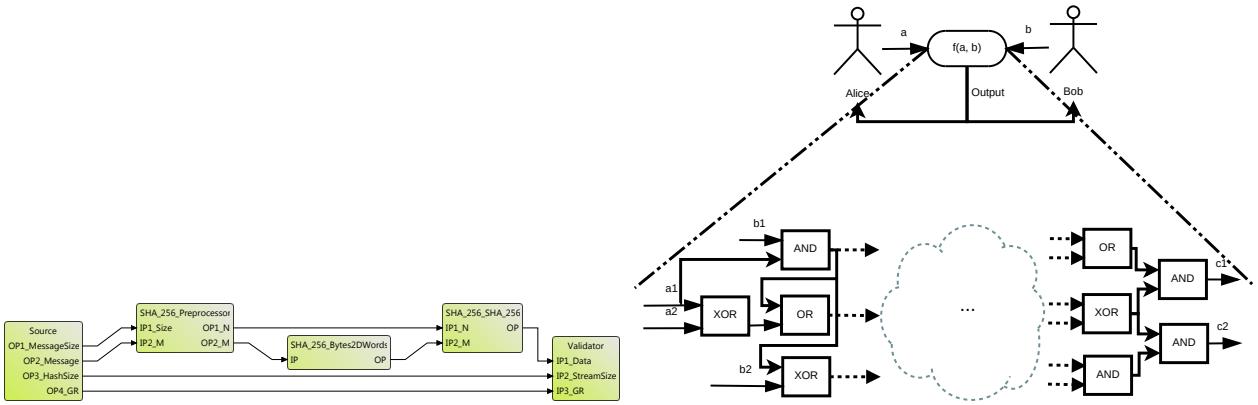
As mentioned in Sec. 1, the challenge of re-programming applications (caused by the diversity of computing hardware/software platforms and communication infrastructures) also affects the system design and development of secure computing applications (i.e., the same secure computing application have to be re-programmed for different target programming languages). The practice of re-programming the same algorithms in different programming languages not only wastes human and computational resources, but also makes it more difficult to synchronize different components working under different platforms thus may potentially reduce the overall performance of the whole system and increases the maintenance costs.

In addition, most secure computing systems are data-driven (just like signal processing systems e.g., video/image/graphic codecs), hence very suitable to be programmed as dataflow systems. Since the RVC framework possesses the richest features set among existing solutions (see Sec. 2), we decided to investigate if the RVC framework indeed offers a good enough environment for the development of implementation-independent secure computing systems. Based on this study, we identified the areas where the RVC framework is not yet up to the mark and helped to evolve RVC by suggesting/implementing solutions to overcome its limitations.

4.2. Secure Computing Systems vs. Multimedia Codecs

Although both secure computing systems and multimedia codecs are structurally data-driven in nature, we noticed some important structural differences, which can be highlighted with the help of a side-by-side comparison of the FU networks of a SHA-2 (a typical cryptographic hash function we developed) [58] and MPEG-4 Part-2 Simple-Profile decoder [59] as shown in Fig. 3a and Fig. 4, respectively. In the following, we list the key differences between these two example applications.

- The most obvious difference between these two system is the number of FUs. It can be seen that cryptographic primitives are normally very small like consisting of just a few FUs (we also implemented the same functionality of SHA-2 within a single FU) whilst multimedia codecs are often bigger in size (from tens to hundreds of FUs). Moreover, multimedia security applications can be slightly bigger than multimedia codecs as they use both multimedia codecs and security modules constructed using the basic cryptographic primitives. In addition, there exist some security protocols (e.g., Garbled Circuit (GC) protocols used in the privacy preserving applications [52]), which could consist of thousands to millions or even billions of FUs. The privacy preserving applications allow un-trusting two parties, Alice and Bob, to jointly evaluate a function $f(a, b)$ without revealing their inputs to each other. The typical structure of a privacy preserving application implementing the secure function $f(a, b)$ as a garbled circuit is shown in Fig. 3b. The GC protocols implement the function $f(a, b)$ as a circuit (or FU network) of basic logic gates, where the security of the GC protocols lies in computing each logical operation securely via its own dynamically computed garbled truth table (often computed via cryptographic hash operations) [52]. Although FUs (or gates) comprising the GC circuit are simpler



(a) Flattened (complete) FU network of SHA-256 hash function.

(b) Typical structure of privacy preserving applications implementing garbled circuit protocols.

Figure 3: Two kinds of secure computing applications.

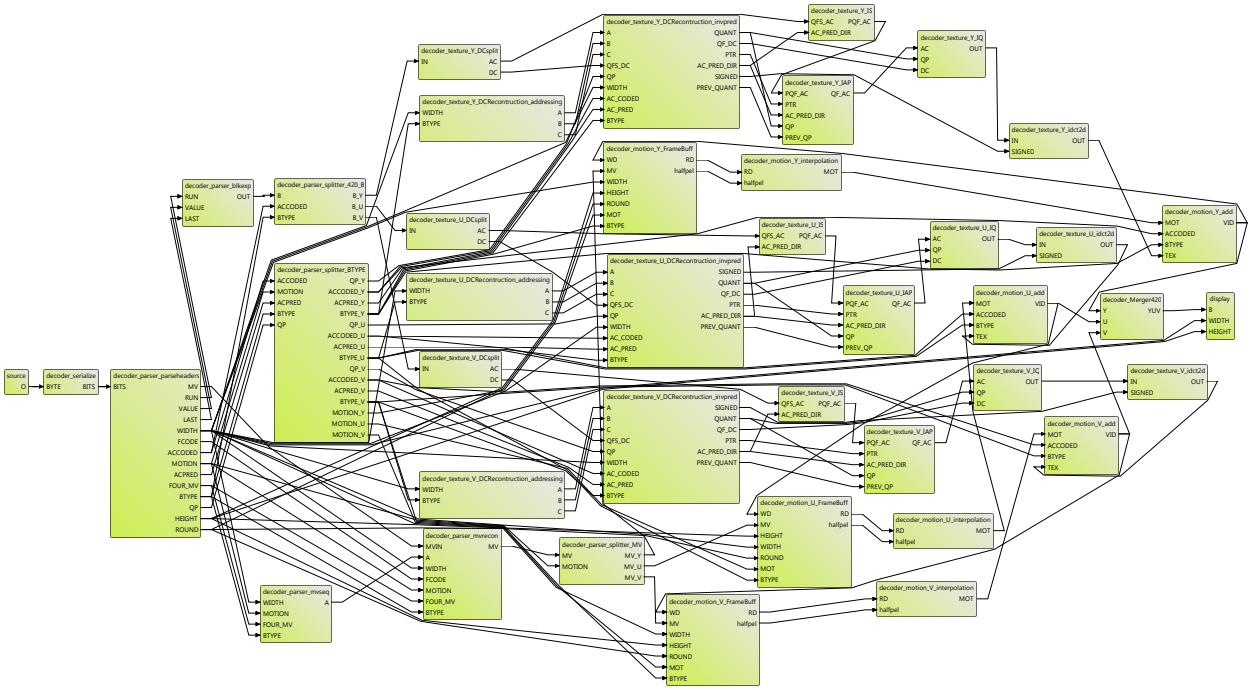


Figure 4: Flattened (complete) FU network of MPEG-4 Part-2 Simple-Profile decoder.

(performing simple logical operations based on their garbled truth tables), the number of FUs needed to implement a given function $f(a, b)$ can be enormously large and the inter-connections of these FUs make the overall topology of the circuit very complicated. As a result, these kinds of systems become far more complicated than multimedia codecs and provide a new dimension for researchers to study the development and automated optimization/parallelization of these systems in the RVC framework (see Sec. 8 for our efforts in this direction), and potentially help in evolving the RVC standard and supporting tools to become more general-purpose. In a nutshell, unlike multimedia codecs secure computing applications can vary a lot in their sizes, and provide the opportunity to apply and evaluate the RVC framework to applications with two extreme scenarios – the very simple

ones and the very complicated ones.

- FUs in secure computing applications are often simpler and less complex than most FUs in multimedia codecs, which are comparatively more complex in terms of number of ports, actions, finite state machine etc.

- As we mentioned in Sec. 2, the dataflow programming models also costs some overheads in scheduling of FUs and flow of tokens through the data channels between FUs (i.e., FIFOs). Secure computing applications with simpler and smaller FU networks may be more sensitive to these overheads than multimedia codecs.

- Except multimedia streaming application and Distributed Video Codecs (DVC), which are still to be studied in the RVC framework, most multimedia codecs are not required to be implemented as heterogeneous systems as both encoder and decoder can work independently of each other[3]. On the other hand, security protocols nearly always involve multiple parties and work in a distributed manner, which requires these protocols to be implemented as heterogeneous systems. The distributed nature of security protocols may demand some new features to be supported by the RVC standard and the support tools. For example, the hPIN/hTAN system [60][4] requires some build-in features to support the count-down timers, event handlers, and some other components implementing the human computer interactions. Although native functions and wrappers can be helpful to support some of these features, new mechanisms may be devised to support newly emerging needs of applications.

4.3. How helpful Secure Computing is to RVC?

Although RVC was successfully used for the development of multimedia codecs, while applying the RVC framework towards the development of secure computing applications, their slightly different nature from multimedia codecs (as discussed in Sec. 4.2) poses a number of challenges. In this section, we highlight how those challenges gave us the opportunity to help in evolving RVC.

4.3.1. Evolution of the RVC Standard

Although the RVC languages (RVC-CAL and FNL) are general-purpose enough to inspire us to use the RVC framework for the development of secure computing applications, they lacked some features that are required to enhance its support for RVC applications in general. This gave us an opportunity to evolve RVC and make it even more general-purpose. Some of the notable contributions, in which we were also involved, are reported in [62, 63, 64, 65]. It deserves mentioning that these contributions were neither solely driven by us nor targeted any specifics of secure computing applications (as the RVC standard only accepts features, which could also be useful for the development of multimedia codecs). Rather, these contributions are quite general in nature and can be beneficial to RVC applications from all domains (multimedia, secure computing etc.).

4.3.2. Evolution of RVC Supporting Tools

Although existing supporting tools of the RVC framework have provided a ready-to-use environment (for development, automated analysis, design-space exploitation etc.) for multimedia codecs, similar to the RVC standard these tools were also developed with *only* multimedia codecs in mind. Moreover, at the start of our study these tools were at the initial stage of their development. This gave us the opportunity to work closely with the developers of these tools[5] and we helped in testing and suggesting improvements to these from very beginning. By testing these tools with secure computing applications[6], we were continuously

[3]Like all other MPEG video coding standards, the RVC standard also focuses only on video decoders. But the same principles can be applied towards the development of encoders too.

[4]In [61], we present the hPIN/hTAN system as an example of a heterogeneous system.

[5]The ORCC should be mentioned exclusively as it is our main choice for the development of secure computing applications.

[6]As highlighted in Sec. 4.2, the nature and complexity of secure computing applications are slightly different from multimedia codecs.

suggesting limitations in their existing components, hence helped in making the tools as much general-purpose as possible. Some notable limitations were: 1) previously ORCC did not support large integers and floating point numbers but they are needed to implement some secure computing applications, 2) although different modules (Graphiti-Editor, backends, simulator etc.) of ORCC are fully compatible with the RVC standard but they did not support Garbled Circuit protocols with huge XDF networks (see Fig. 3b) [66], hence demands to adapt ORCC to handle the slightly different nature of secure computing systems. In addition, we are currently evaluating TURNUS [67] to assist in automated optimization/parallelization of garble circuit protocols (see Sec. 8 for details). Moreover, our suggestions to the further development of RVC supporting tools help provide insights to further refine the RVC-CAL languages, which led to several new constructs in the new editions of RVC standard (e.g., unit, native, import, I/O system actors etc.).

4.3.3. Run-time Performance

Although the RVC framework has a number of distinctive and useful features, one straightforward concern of using it for secure computing is if the high-level abstract nature of RVC and the automated code generation process will compromise the overall run-time performance at the implementation level so much that the benefit does not outweighs the costs. Since the target implementations will be automatically generated from the abstract RVC descriptions, one may expect that the run-time performance will be worse compared with implementations written directly in the target programming languages. But if the run-time performance of these target implementations is found to be comparable to non-RVC implementations, then the advantages of using the RVC framework can compensate for the compromise of run-time performance.

Except the MPEG contribution [68] investigating frame decoding rates of multimedia decoders, there were no studies reporting run-time performance of RVC applications on single-core and multi-core computing platforms. Hence, one of the goals of this study is to benchmark the run-time performance of secure computing systems (implemented with the RVC framework) on both single-core and multi-core platforms. Based on the benchmarking results, it can be evaluated if the RVC can be acceptable as a really useful general-purpose development framework. Our performance benchmarking studies on some selected cryptosystems and multimedia security applications are presented in Sec. 5 and [61], respectively. Some key results of our performance benchmarking studies were also reported to MPEG in [69, 70, 71] so that the RVC standard and supporting tools can be evolved (if needed) accordingly.

4.4. How helpful RVC is to Secure Computing?

In this section, we highlight the benefits one can achieve by adapting the RVC framework for the development of secure computing applications and makes it a unique environment for them. These benefits are listed below:

- **Preservation of Structure:** Quite often, the very first step researchers or cryptographers take to design a secure computing system is to draw the system as a flowchart, which visually reflects how the data flows from the input side through all the system components to produce the output. This is also what we see as schematic flow diagrams in the explanation of most ciphers and security protocols (e.g., [72, 73, 74, 58]). This is not surprising as many secure computing systems are designed and used for data security and/or integrity, where data are the center of the whole system. After the flowchart and the I/O behavior of each component are finalized, one can start thinking about how to carry out the programming needed to realize the system. The data-driven nature of secure computing systems suggests that the dataflow programming paradigm can better preserve the structure of their original design than the traditional sequential/imperative programming diagram.

- **Fast development/prototyping:** Most imperative programming languages (like C/C++, Java, etc.) are not very helpful in maintaining modular and reusable components, thus do not help much in reconfiguration and maintenance. By adapting the dataflow programming paradigm the components of secure computing applications are inherently *modular*, *reusable*, and easily *reconfigurable*. These properties are key to make the development/prototyping and reconfiguration of secure computing applications faster, more user-friendly and easier. Moreover, the faster/easier reconfiguration also

makes the maintenance far easier. Therefore, it can be implied that RVC (as a typical dataflow programming framework) provides a faster development/prototyping environment than the development environments based on imperative programming languages.

- **Multiple target languages:** The implementation independence of RVC further allows developer of secure computing systems to concentrate on the functional side before targeting an implementation on a particular platform. Hence, secure computing applications can be programmed only once, but be automatically translated into source code for multiple programming languages (C, C++, Java, LLVM, Verilog, VHDL, XLIM, and PROMELA at the time for this writing[7]).
- **Adequate run-time performance:** Since secure computing applications are highly abstract programs, the run-time performance of automatically synthesized implementations poses a big concern about if the RVC can be truly adapted as a general-purpose development framework. As we mentioned earlier, one of the goals of this study is to investigate the run-time performance of RVC implementations against their corresponding non-RVC solutions and our studies have shown that the run-time performance of RVC implementations is indeed adequate comparable to non-RVC reference implementations.
- **Hardware/Software codesign:** As we highlighted in Table. 1, none of the evaluated cryptographic solutions (Category 4) supports the development of heterogeneous systems (hardware/software codesign). But because of the distributed nature of many security protocols, the support for the development for heterogeneous system becomes very important. The RVC framework is also beneficial in this regard, as it allows the development of secure computing applications with heterogenous components involving software, hardware, and various I/O devices/channels [57].
- **Automatic code analyzability and optimization:** Unlike imperative languages, secure computing applications developed in the RVC framework can go through (semi-)automated design-space exploitation at the algorithmic level, which can help to optimize the algorithmic structure of the secure computing application by refactoring (automatically identified) computational blocks, and/or automatically suggesting strategies to parallelize different computational blocks of secure computing applications on multi-/many-core computing resources. Our study on the automated analysis and optimization of secure computing applications is currently in progress and is briefly discussed as future work in Sec. 8.

5. Crypto Tools Library (CTL)

Crypto Tools Library (CTL) is a collection of RVC-CAL actors and FU networks for cryptographic primitives such as block ciphers, stream ciphers, cryptographic hash functions and PRNGs. Being an open project, the source code and documentation of CTL is available at `http://www.hooklee.com/default.asp?t=CTL`.

As mentioned in Sec. 2, most existing cryptographic libraries are developed based on a single programming language (mostly C/C++ or Java) that can hardly be converted to source code written in other languages. In contrast, CTL is an implementation-independent solution, whose source code is written in RVC-CAL and FNL that can be automatically translated into multiple programming languages including C, C++, Java, LLVM, Verilog, VHDL, XLIM, and PROMELA. More programming languages can be supported by developing new code generation tools for RVC applications.

In the following, we present the design principles being followed in the development of CTL, the list of cryptosystems currently available in CTL, some cryptosystems examples from the CTL (highlights the advantages we claimed in Sec. 4), and the run-time performance benchmarking of AES on single-core platforms.

[7]More code generation backends are going to be made in the future, especially OpenCL, which can be used to support GPUs in RVC.

5.1. Design Principles

The CTL is developed by strictly following the specifications/standards defining the implemented cryptosystems. However, when it is possible, the CTL FUs are designed to exploit inherent parallelism in the implemented cryptosystems. For instance, for block ciphers based on multiple rounds, the round number is also transmitted among different FUs so that encryption/decryption of different blocks can be parallelized.

The CTL is designed so that different cryptosystems can share common FUs. We believe that this can better support code reusability and ease reconfigurability of the CTL cryptosystems. In addition, CTL includes *complete* solutions (e.g., both encipher and decipher) of the implemented cryptosystems.

5.2. Cryptosystems Covered

CTL contains some standardized and frequently used cryptosystems. In the following, we list the cryptosystems currently implemented in CTL. The correctness of all cryptosystems has been validated using the test vectors given in the respective standards or official documents (for non-standards cryptosystems).

- Block Ciphers:
  - AES-128/192/256 [72],
  - DES [73] and Triple DES [73, 74],
  - Blowfish [75],
  - Modes of operations: CBC, CFB, OFB, CTR.
- Stream Ciphers: ARC4 [76] and Rabbit [77].
- Cryptographic hash functions: SHA-1, SHA-2 (SHA-224, SHA-256) [58].
- Keyed-hash message authentication code (HMAC): SHA-1, SHA-2 (SHA-224, SHA-256) [78, 79].
- PSNRs: 32-bit and 64-bit LCG and LFSR-based PRNG [76].

CTL also includes some common utility FUs (e.g., multiplexing/demultiplexing of sequences of tokens, conversion of bytes to bits and vice versa etc.) that are shared among different cryptosystems and can also find applications in non-cryptography systems. Currently implemented utilities are listed below.

- `XOR_1b` and `XOR_8b`: bitwise and bytewise XOR of two token sequences;
- `Mux2` and `Mux8`: merging 2 and 8 sequences of tokens into a single one;
- `Demux2` and `Demux8`: splitting a token sequence into 2 and 8 sub-sequences;
- `Any2Bits`: converting n-bit tokens into binary (i.e., 1-bit) tokens;
- `Bit2Any`: converting binary tokens into n-bit tokens;
- `Smaller2Bigger`: converting n_2/n_1 input tokens of bit size n_1 into one output token of bit size $n_2 > n_1$;
- `Bigger2Smaller`: converting each input token of bit size n_1 into n_1/n_2 output tokens of bit size $n_1 > n_2$.

In each RVC-CAL file of CTL FUs and test beds, there is a header comments section providing detailed information about that RVC-CAL file: FU name, FU interface (input ports, output ports, FU parameters), how to use the CAL file, reference to corresponding standard document, and so forth. Furthermore, under each folder there is also a readme file containing a list of all files in the corresponding folder.

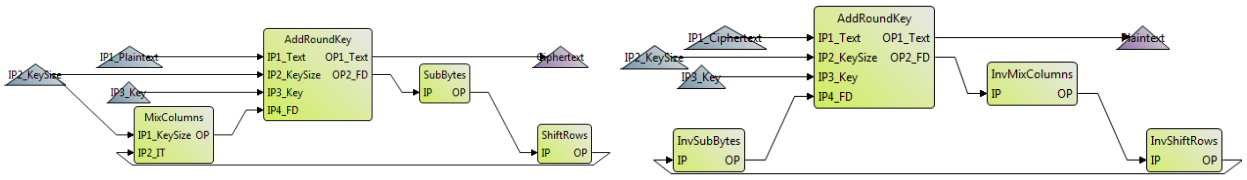
5.3. Cryptosystem Examples

With the objective of giving a feel of how CTL cryptosystems look like, in this subsection, we present AES, DES, and Blow fish block ciphers as examples of cryptosystems from CTL. We present the XDF networks for encipher and decipher along with a brief description about the associated FUs.

5.3.1. AES

CTL includes two different implementations of AES: 1) one for the educational purpose, which has been implemented by strictly following the AES standard [72]; 2) a look-up-tables (LUTs) based optimized implementation following the Rijndael's optimized reference implementation [80]. In the following, we present both of them.

Standard Implementation. Figure 5 shows encipher and decipher FU networks of the standard AES implementation in the CTL. Both have three input ports and one output tokens, all of type byte. Thus, AES always consumes 16 byte tokens as plaintext/ciphertext and produces 16 byte tokens as ciphertext/plaintext. The key size and key are always read at the beginning of the encryption/decryption process and remains the same until it is not changed. All the four basic operations are implemented as separate RVC-CAL FUs. The key expansion function (i.e., key scheduler) is implemented as part of the `AddRoundKey` FU since it is not used in other FUs. It should be noted that both AES encipher and decipher have similar structure. However, for AES decipher the four basic components are connected in a reversed order.



(a) AES Encipher (b) AES Decipher

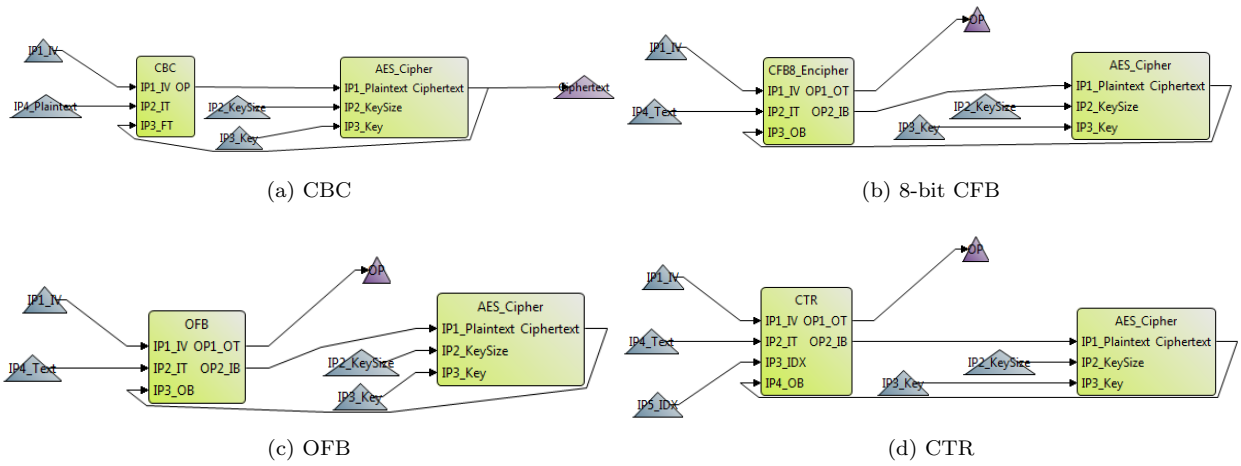
Figure 5: AES encipher and decipher in CTL.

To enhance the parallelism of the AES encipher and decipher, we transmit a token representing the round index together each plaintext/ciphertext block. This helps in parallel processing multiple blocks. Hence, each data block in AES consists of 17 tokens (one round number + 16 data tokens).

The AES encipher and decipher shown above are running in the simplest Electronic Code Book (ECB) mode. Since block ciphers running in ECB mode have the potential risks of known/chosen plaintext attack and chosen-ciphertext attack, block ciphers are often run in other modes involving feedback of ciphertext and/or use of counters. Each mode of operation is implemented as a single RVC-CAL FU, which can be connected with the AES network running in ECB mode to make it work under the expected mode of operation.

Figure 6 shows the AES encipher running at four other modes of operation where `AES_Cipher` FU in each sub-figure encapsulates AES ECB encipher of Fig. 5a. It should be noted that, changing the mode from ECB to another required target mode is just a matter of connecting the target mode's FU with the basic FU network of AES.

Look-up-tables based Optimized Implementation. The main objective for developing such implementation was to evaluate the run-time performance of AES when it is implemented in a way similar to an optimized sequential program. This optimized AES implementation follows the look-up-tables based optimization algorithm used in the Rijndael's optimized reference implementation [80]. We implemented `CTL-LUT` encipher and decipher as single FU each. Both FUs contain three actions: 1) `readKeyInfo` – receives the key size; 2) `keyExpansion` – receives the key and performs the key scheduling/expansion; 3) `encrypt/decrypt` – receives the plaintext/ciphertext, performs encryption/decryption operation and produces ciphertext/plaintext.



(a) CBC (b) 8-bit CFB

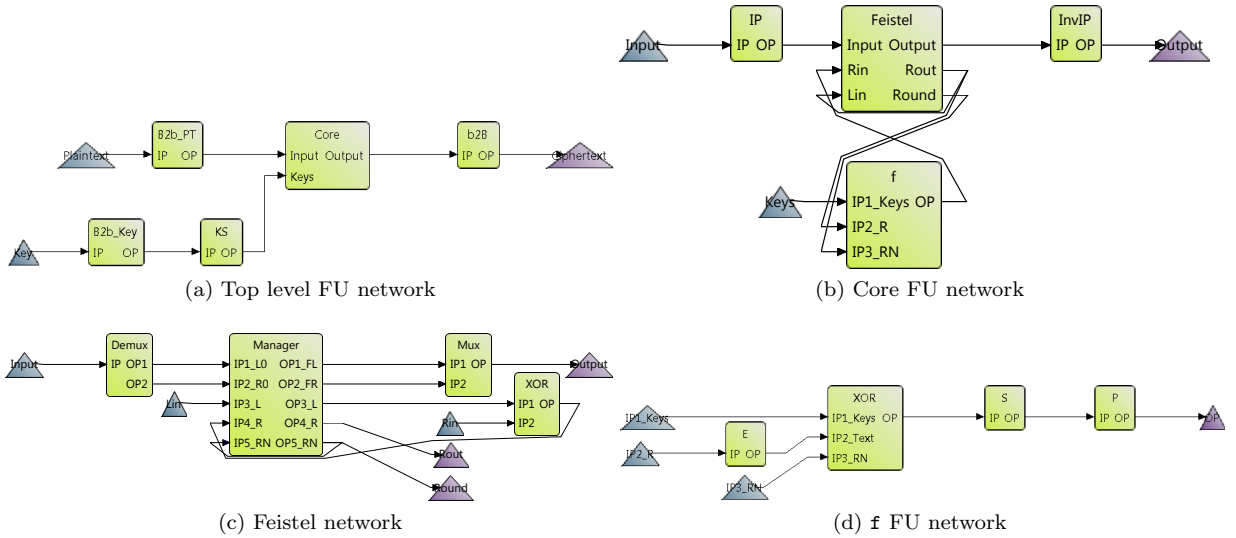
(c) OFB (d) CTR

Figure 6: AES encipher running at different modes of operation.

Similar to AES standard implementation, both encipher and decipher FUs of this optimized AES implementation can also be used with any operational mode FU without any problem.

5.3.2. DES

Different from AES, DES is more bit-oriented. The current DES implementation in the CTL is based on bit tokens. Figure 7a shows the top level FU network of the DES encipher in the CTL, where the two `B2b` FUs (instances of `Any2Bits`) are used to convert byte tokens to bit tokens and the `b2B` FU (instance of `Bits2Any`) is used to convert bit tokens into byte tokens. The `KS` FU is key scheduler generating round keys. DES is a Feistel cipher that has an identical structure for the encipher and the decipher (except the key scheduler), so the top level network of the DES decipher is the same as the DES encipher except that `KS` FU is reconfigured to send keys in a reversed order.



(a) Top level FU network (b) Core FU network

(c) Feistel network (d) `f` FU network

Figure 7: DES encipher implementation in CTL.

The `Core` FU network in Fig. 7b is composed of an initial permutation (`IP`) FU, a Feistel cipher network,

an inverse permutation (`InvIP`) FU and the round function (`f`) FU, as shown in Fig. 7b. The Feistel network is shown in Fig. 7c, where the `Manager` FU controls the dataflow inside the Feistel network in different rounds and the `f` FU network implements the core round function, which is shown in Fig. 7d. As shown in Fig. 8, the `S` FU in the `f` FU network is further composed of a demuxer FU, eight parallel 6×4-bit S-boxes and a muxer FU.

5.3.3. Blowfish

Just like DES, Blowfish [75] is also a Feistel cipher but with a different round function `f`. Figure 9 shows the top level FU network of the Blowfish encipher in the CTL. In this implementation of Blowfish, we have *reused* the Feistel network of Fig. 7c, which we previously used to implement the DES block cipher. The processes of building of Blowfish sub-keys and S-boxes have been grouped with the implementation of the round operation in `f` FU. After 16 iterations through the Feistel network, the data block streams through the `Final_XOR` FU, which XORs last two sub-keys with the data block to generate the ciphertext.

Similar to our DES implementation, Blowfish encipher and decipher also have an identical structure. An instance parameter is used to reconfigure `f` FU to use the sub-keys in either sequential order (for encipher) or reversed order (for decipher).

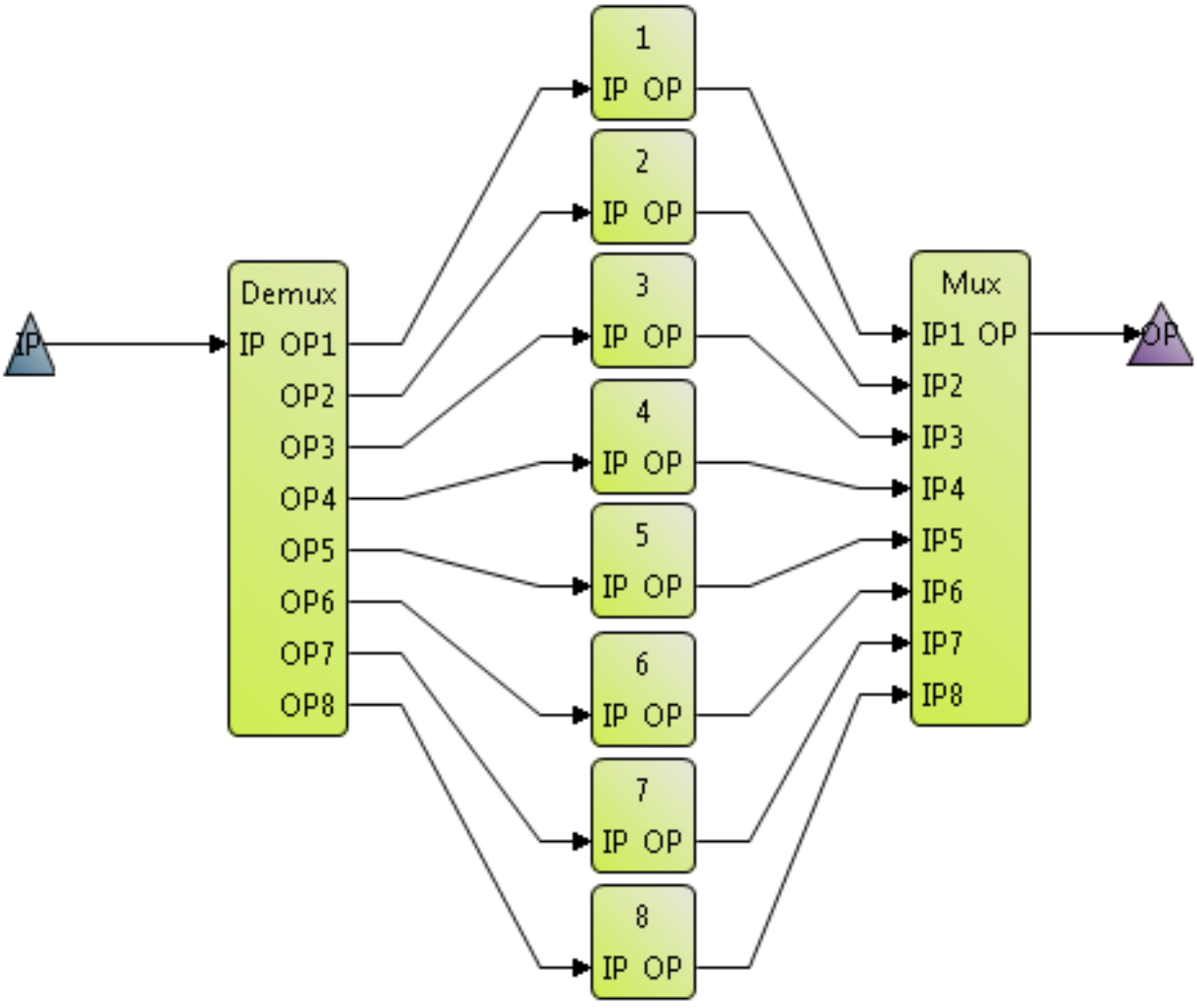


Figure 8: The eight S-boxes in the core `f` FU network of DES encipher and decipher in CTL.

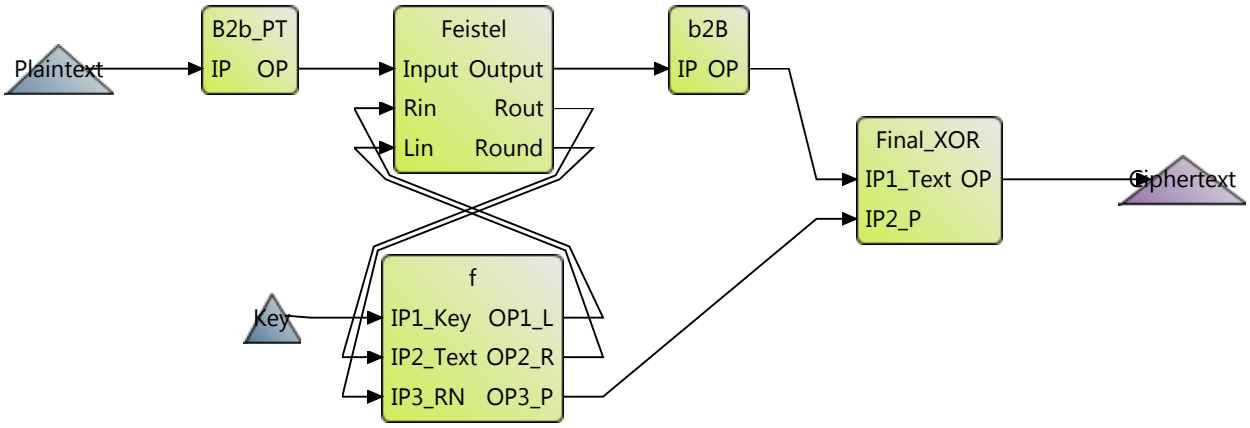


Figure 9: Blowfish encipher implementation in CTL.

5.4. Run-time Performance Benchmarking

Previous work has demonstrated that the RVC framework can outperform other sequential programming languages in terms of implementing highly complex and highly parallelizable systems such as video codecs [45]. However, there were still doubts about if the high-level abstraction of RVC-CAL and the automated code generation process may compromise the overall performance to some extent at the platform level. In this subsection, we clarify those doubts by conducting run-time performance benchmarking of AES and SHA-256 in CTL against selected non-RVC reference implementations. The results showed that the automatically generated RVC implementations in C and Java can both achieve a comparable run-time performance to these non-RVC reference implementations.

In this subsection, we present the performance benchmarking of AES[8] against some reference implementations on varying single-core platforms (two general-purpose PCs, a resource-constrained embedded device and two Java virtual machines). In this subsection, we first give the details of our experimental setup required to reproduce our results. Then, we present the performance benchmarking results.

5.4.1. Tested AES implementations

As shown in Sec. 5.3.1, currently CTL contains two different implementations of AES targeting two different objectives. One implementation is created for educational/understanding viewpoint and has been implemented by strictly following the AES standard [72]. The second implementation has been created to achieve better run-time performance and has been implemented by following the look-up-tables based optimized algorithm used in the Rijndael's optimized reference implementation [80]. Both of these AES implementations are benchmarked in this study. In the rest of this subsection, we will respectively use "CTL-STD" and "CTL-LUT" as the short names for these two AES implementations in CTL.

Along with AES ECB encipher and decipher, AES running in CTR mode (shown in Fig. 6d) is also included for this benchmarking study because the CTR mode has the benefit of being able to encrypt multiple blocks in parallel, so it can be considered as a better candidate for benchmarking cryptosystems implemented in RVC-CAL. Hence, the following three CTL implementations of AES-128 were benchmarked in our study:

- AES-128 CTR Cipher
- AES-128 ECB Encipher
- AES-128 ECB Decipher

[8]The benchmarking results of SHA-2 are not included in this paper as they have already been reported in Sec. 5 of [1].

5.4.2. Reference implementations

To benchmark the performance of the ORCC generated C code of AES-128, some reference implementations are needed to compare with. For this study, the following three implementations are selected:

- Rijndael reference implementation ver. 2.2 [81]
- AES implementation available at www.X-N2O.com [82]
- Rijndael optimized reference implementation ver. 3.0 [80]

In the rest of this subsection, "Ref. 2.2", "Ref. 3.0" and "X-N2O" are the short names used to refer to the three reference implementations of AES-128. Not all of these three implementations support CTR mode, so the code has been manually modified to add CTR support.

We select these three implementations because to make this performance benchmarking study judicial, we need the reference implementations that follow the same implementation style and optimizations as our CTL implementations. For instance, Ref.2.2 and X-N2O implementations are similar to our CTL-STD implementation because they do not contain any optimizations and are implemented by following the AES standard. Similarly, Ref. 3.0 implementation is similar to our CTL-LUT implementation as both are optimized by using pre-computed look-up tables.

Similarly, we also benchmarked the ORCC generated Java code of AES-128 against the AES implementation available as part of the Java Cryptography Architecture (JCA) [83].

5.4.3. Platforms

Our experiments were run on two PCs and one embedded system. This was done to represent two typical configurations of today's PCs, one new desktop and one old laptop are selected. For embedded systems, we selected a resource constrained wireless sensor mote for our study. For both PCs, we have conducted this performance evaluation under Windows and Linux operating systems whilst our embedded system runs a stripped-down version of Linux operating system. The concrete configurations of these platforms are given Table 2.

M1 has a dual-core CPU so the performance benchmarking may be less accurate due to the internal scheduling of CPU instructions. So we switched the dual-core support off in M1's BIOS setup. Furthermore, the multi-task nature of operating systems may also influence the benchmarking results, so we ran all our tests under their Safe Mode shell to minimize such effects.

To generate the executables running under different operating systems, we selected Microsoft Visual Studio 2008 as the C compiler for Windows XP/7, GCC 4.3.2 for Linux Debian Live and arm-linux-gcc 3.4.1 for Imote2-Linux on M3. For Java programs, we used Eclipse SDK 3.6.1.

5.4.4. Run-time performance metric

For each executable running under a specific OS, a continuous encryption/decryption process was run over the same test vector of 4096 bytes. We measured the total number of CPU cycles consumed for the whole encryption/decryption process and divided it by 4096 to get the run-time performance in cycles/byte. Since we only care about the core part of the encipher/decipher, the time consumed in initial inputs and final output is not counted.

On M1 and M2, the CPU cycles were measured using `RDTSC` and `CPUID` instructions of Intel processors [86]. For M3, we used `Cycle CouNT (CCNT)` register available in Intel ARM XScale processors [85]. However, the measured CPU cycles may vary depending on the availability of needed data/instructions in the data/instruction cache. To solve this problem, we follow the suggestion given in [86] to run the same executables for 100 times and use the averaged value of CPU cycles as the final measurement.

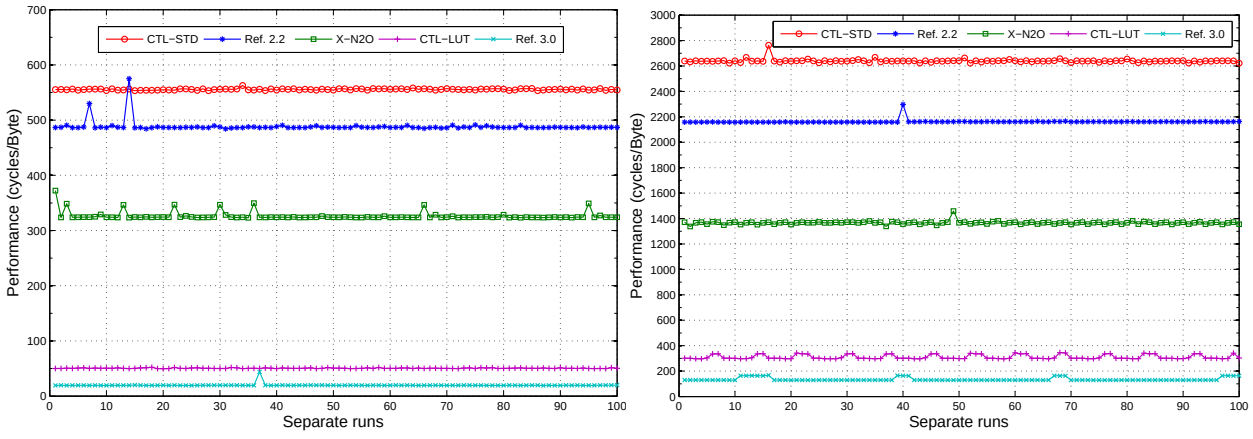
5.4.5. Benchmarking Results

In this subsection, we present the results of our performance benchmarking study on the CTL implementations of AES-128 and the corresponding reference implementations on all three machines.

Table 2: Configurations of Testing Platforms.

Platform	**Hardware and Operating System Details**
Machine 1 (M1): Desktop PC	– Model: HP Compaq 8000 Elite Convertible Minitower – CPU: Intel Pentium Dual-Core CPU E5400 2.70GHz – Memory: 2GB RAM – OS1: Windows 7 Professional (32-bit Edition) – OS2: Linux Debian Live (rescue image kernel version 2.6.26-2-686) – Windows 7 Java(TM) SE Runtime Environment: build 1.6.0_26-b03
Machine 2 (M2): Laptop PC	– Model: Samsung Q25 – CPU: Intel Pentium M 1.3GHz – Memory: 504MB RAM – OS1: Windows XP Professional SP 2 – OS2: Linux Debian Live (rescue image kernel version 2.6.26-2-686) – Windows XP Java(TM) SE Runtime Environment: build 1.6.0_27-b07
Machine 3 (M3): Embedded System	– Model: Imote2 Wireless Sensor Mote [84] – CPU: Intel ARM XScale PXA271 CPU 415.33MHZ [85] – Memory: 32MB SRAM – OS1: Imote2-Linux (Kernel version 2.6.29.1)
Machine 4 (M4): Quad-core Desktop PC	– Model: HP Centurion – CPU: Intel(R) Core(TM)2 Quad CPU Q9550 2.83GHz – Memory: 8GB RAM – OS1: Windows Vista Business with Service Pack 2 (64-bit Edition) – OS2: Ubuntu Linux (Kernel version: 2.6.27.11)

AES-128 CTR Cipher. As mentioned in the previous subsection, in order to compensate the cache effects, each executable was run 100 times. Figure 10 shows the results obtained under Windows and Linux operating systems on M1 and M3, respectively.



(a) On general-purpose PC – M1.

(b) On embedded system – M3.

Figure 10: Separate Runs of AES-128 CTR cipher under Windows and Linux OS.

The results in these plots show some abrupt fluctuations in the performance curve, but they do not occur very often. Most of the times, the CPU cycles counting remains consistent. Therefore, the CPU

cycles counting method based on RDTSC and CPUID instructions is stable enough as a metric of the run-time performance for the evaluated implementations.

The average performance results for all implementations on PCs and embedded system are given in Tables 3a and 3b. It can be observed that, under both Windows 7/XP and Linux operating systems, the CTL-STD and CTL-LUT implementations have a performance comparable to Ref. 2.2 and X-N2O, and Ref. 3.0, respectively.

It deserves noticing that all algorithms perform better on M1 than M2 or M3. This can be easily explained by the more powerful CPU and larger memory available on M1. On the contrary, all algorithms consume higher number of CPU cycles on M3 because of its limited resources.

(a) Averaged performance benchmarking of AES-128 C implementations on PCs.

CTR Cipher	M1					M2				
	CTL-STD	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**	**CTL-STD**	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**
Win XP/7	555.6	488.4	326.4	50.5	19.8	637.2	1292.0	393.7	65.4	24.4
Linux	3979.6	1632.4	353.1	71.6	37.0	4711.8	1991.3	581.7	77.3	41.5

ECB Encipher	M1					M2				
	CTL-STD	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**	**CTL-STD**	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**
Win XP/7	418.8	484.5	296.3	21.8	18.8	472.2	1296.2	374.1	27.2	23.5
Linux	3088.1	1599.4	1439.6	36.3	34.7	4205.2	1948.3	1558.1	41.7	37.8

ECB Decipher	M1					M2				
	CTL-STD	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**	**CTL-STD**	**Ref. 2.2**	**X-N2O**	**CTL-LUT**	**Ref. 3.0**
Win XP/7	719.1	668.9	544.4	21.8	19.3	1769.9	1899.9	1572.1	26.6	23.8
Linux	4333.6	2229.4	1761.8	37.9	35.2	5630.2	1463.7	2641.2	42.0	37.9

(b) Averaged performance benchmarking of AES-128 C implementations on an embedded system.

	CTL-STD	**Ref.2.2**	**X-N2O**	**CTL-LUT**	**Ref.3.0**
CTR Cipher	2639.8	2161.8	1366.1	311.1	135.0
ECB Encipher	2058.1	2167.6	1364.2	147.6	132.1
ECB Decipher	4706.4	4231.9	2982.7	147.7	129.3

(c) Averaged performance benchmarking of AES-128 Java implementations on PCs.

	M1		M2	
	CTL-LUT	**JCA**	**CTL-LUT**	**JCA**
CTR Cipher	3926.5	1391.8	4632	1581.2
ECB Encipher	2820.3	2820.3	3731.5	1362
ECB Decipher	2791.2	779.5	3798.1	850.6

AES-128 ECB Encipher. Tables 3a and 3b also contain the performance benchmarking results on AES-128 ECB encipher. One can see that the results of CTL-STD implementation in ECB mode are similar to the corresponding CTL-STD implementation in CTR mode. In addition, unlike for the AES CTR cipher, the performance of CTL-LUT encipher implementation in ECB mode is quite close to the corresponding Ref.3.0 implementation. This sudden performance gain in the ECB mode can be explained by the design of ECB and CTR ciphers in the RVC framework. In CTR mode, the encryption process was jointly performed

by CTR and AES Cipher FUs while at any given time only one of them processing data and other one waiting for the data (i.e., when the CTR FU processes data, AES Cipher waits for the data and vice versa). However, in ECB mode the whole encryption process is encapsulated within a single FU and saves a considerable time that could be depleted in waiting for the dataflows from other FUs. Based on these results it can safely established that: 1) if the needed optimization are implemented within the same FU, the run-time performance similar to reference implementations can be achieved; 2) on single-core machines, the introduction of dataflow networks between FUs affects the run-time performance. In other words, these results suggest that the run-time performance of the RVC applications on single-core machines is inversely related to the number of intermediate FUs (and FIFOs). However, very small RVC applications (like cryptosystems) can be implemented within a single FU. With the increase in the algorithmic complexity of applications, it become difficult to implement in fewer FUs, which in a way also creates an opportunity to achieve better run-time performance by parallelizing the FUs on multi-core machines. In the next section, this point is further highlighted while presenting the performance benchmarking study on a quad-core machine.

AES-128 ECB Decipher. Tables 3a and 3b also give the performance benchmarking results on AES-128 ECB decipher. It can be seen that CTL-STD's decipher implementation also has a performance similar to Ref. 2.2 and X-N2O. In addition, for CTL-LUT's decipher implementation, we obtained the results similar to the results of AES-128 ECB encipher.

Benchmarking of AES Java Implementations. Since the C implementations of CTL-LUT achieved a reasonable performance, the Java implementation of CTL-LUT was also benchmarked against the AES's Java implementation available as part of the Java Cryptography Architecture (JCA) [83]. Tables 3c provides the results of the performance benchmarking, which was conducted under the Windows 7/XP Java run-time environment of M1 and M2. These results confirms that the ORCC Java backend does not generate very efficient code at this moment and requires a lot of improvements.

6. Cryptographic Applications

In this section, we present two cryptographic applications implemented in the RVC framework. The first application, HashTree application, is developed to demonstrate the multi-core support of the RVC framework and the actual run-time performance of automatically generated C implementations on a quad-core machine. The second application, the SKID3 mutual authentication protocol [87], is given as an example application showcasing hardware/software co-design in the RVC framework.

6.1. Application 1: HashTree

Figure 11 shows the FU network of the HashTree application, which implements the following functionality using five hash operations: given an input signal $x = x_1 \parallel x_2 \parallel x_3 \parallel x_4$ consisting of four blocks x_i of the same size, hash each block $h_i = H(x_i)$ and then output $H(h_1 \parallel h_2 \parallel h_3 \parallel h_4)$. In our implementation of HashTree, we instantiated H with SHA-256.

We benchmarked the HashTree application on the quad-core machine (M4) shown in Table 2 to evaluate the amount of performance gain achieved by partitioning the whole FU network into several parts and mapping them to different CPU cores. In the given HashTree application, the partitioning and mapping were both done manually, but they can be automated for large applications thanks to the code analyzability of RVC-CAL. The C backend of the RVC supporting tool ORCC [54] allows multi-core mapping without any change to the RVC and FNL code, so different parts of an FUs network can be easily allocated to different CPU cores. To show the effect of the multi-core mapping more clearly, we benchmarked three configurations of the HashTree application.

- **HashTree with Single SHA-256**: This implementation of HashTree computes the same operation (i.e., $H(H(x_1) \parallel H(x_2) \parallel H(x_3) \parallel H(x_4))$) as of HashTree application of Fig. 11. All computations are

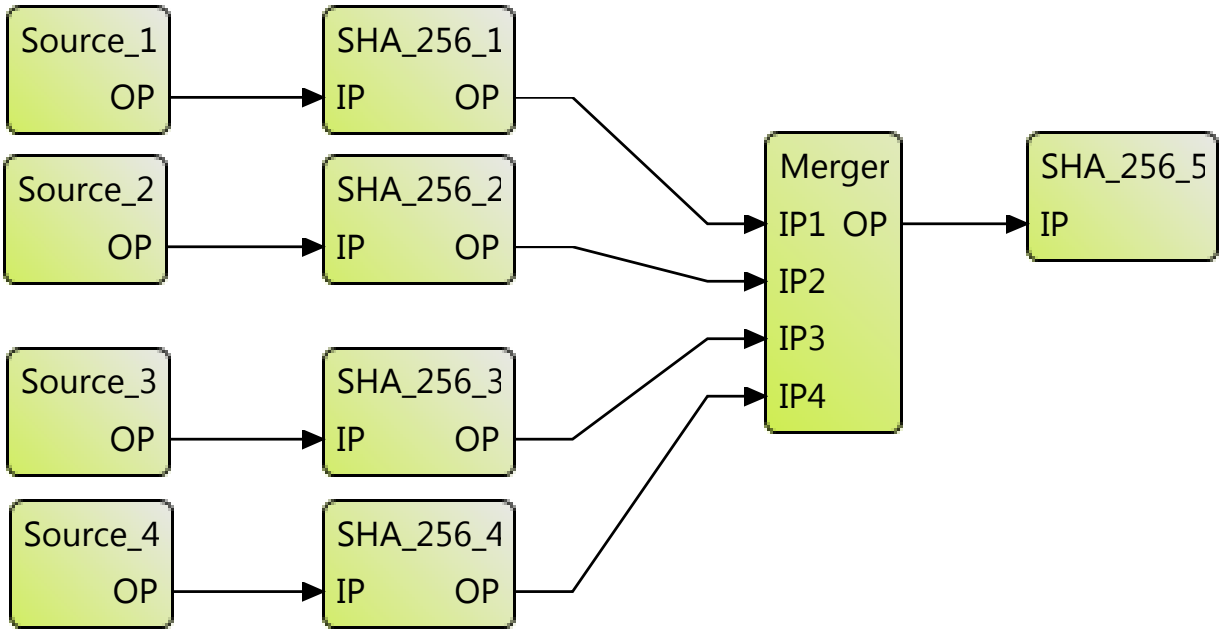


Figure 11: HashTree Application.

performed by means of a single SHA-256 FU while it was running on a single CPU core. This configuration is used as a reference point to evaluate the performance gain of the following two configurations of the HashTree application[9].

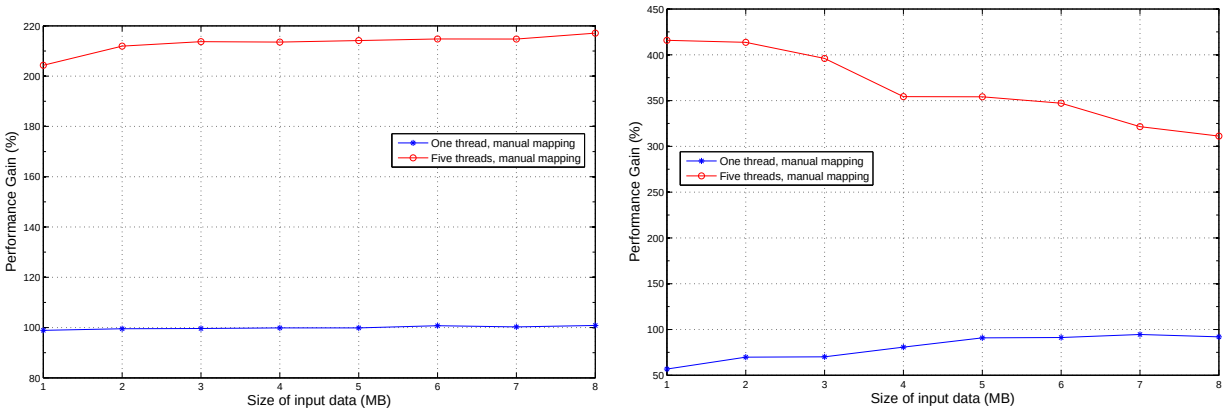
- **HashTree with one thread**: In this configuration of HashTree, all five SHA-256 instances are bounded to run in a single thread on a specific CPU core of the quad core machine. This configuration is expected to have similar run-time performance to the above simpler SHA-256 application.
- **HashTree with five threads**: In this configuration of HashTree, the five SHA-256 instances are programmatically configured to run as five separate threads which are mapped to the four CPU cores as follows: each of the first four threads is mapped to a different CPU core. The 5th thread is only launched after the other four are finished and is mapped to the first CPU core. We expect that this configuration can run faster than the simpler SHA-256 application and the one thread configuration.

It should be noted that thread creation and CPU core mapping also consume some CPU time, which is the cost one has to pay to achieve concurrency. Therefore, in order to make the study judicial, we also count the times spent on thread creation and thread mapping. The benchmarking results are shown in Fig. 12. One can see that the performance gain is between 200% to 400% when five threads are used and are very similar to the results we reported in [1].

6.2. Application 2: SKID3 – A Mutual Authentication Protocol

Heterogeneous systems become ubiquitous in today's digital world because of the use of many different hardware devices and software systems working on different platforms. Since an RVC specification can be considered as implementation-independent and the code generation step normally only generates codes of

[9] In [1], a similar performance benchmarking study of the HashTree application was presented. However in that study, the reference configuration used to evaluate the performance gain of the following two configurations, was computing $H(x_1 \parallel x_2 \parallel x_3 \parallel x_4)$ with a single SHA-256 operation. Recently, it was noticed that the performance benchmarking would become more slightly judicial if the reference configuration also computes *exactly* the same HashTree operation. Hence, this paper presents the performance benchmarking results of HashTree application with this more judicial reference configuration.



(a) Windows (b) Linux

Figure 12: The performance gain obtained from the benchmarked configurations of the HashTree application.

a whole application for a single targeted (hardware or software) platform, other mechanisms are needed to handle heterogeneous systems. To this end, the concept of "wrapper" [57] has been proposed to bridge implementation-independent FUs in the RVC framework with physical I/O devices/interfaces/channels (e.g., a device attached to USB port, a host connected via LAN/WLAN, a web server, etc.). This opens the door for partitioning a whole RVC application into different components for different platforms and then linking them to form a heterogeneous system.

To demonstrate how wrappers can be used to enhance the RVC framework, we developed a second cryptographic application involving a hardware component and a software component. This application implements a simple mutual authentication protocol called SKID3 [87], where one party is a hardware device and the other is a program running on a computer. Normally one party runs as a server (S) and the other runs as a client (C). The protocol works as follows:

- **Step 1**: $C \rightarrow S$: (CID, r_C),
- **Step 2**:
  a) $S \rightarrow C$: $(r_S, H_1 = \text{HMAC}(K_C, r_S||r_C||\text{SID}))$,
  b) C checks if the received H_1 is equal to $H_1' = \text{HMAC}(K_C, r_S||r_C||\text{SID})$. If it does, then the server authentication is done.
- **Step 3**:
  a) $C \rightarrow S$: $H_2 = \text{HMAC}(K_C, r_C||r_S||\text{CID}))$,
  b) S checks if the received H_2 is equal to $H_2' = \text{HMAC}(K_C, r_C||r_S||\text{CID})$. If it does, then the client authentication is done.

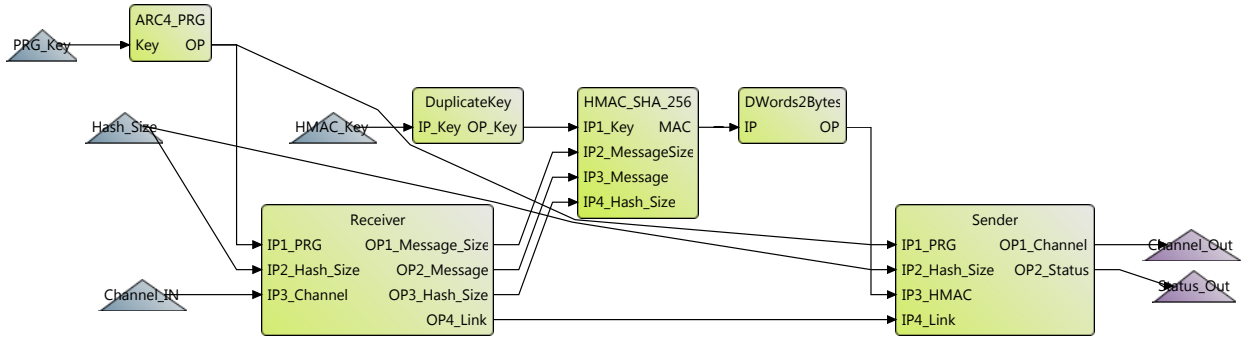
In the above protocol, K_C is the key for HMAC operations, CID and SID are the identification names (IDs) of C and S, respectively. Similarly, r_C and r_S are the nonces (random numbers) generated by C and S, respectively.

6.2.1. SKID3 as an RVC Application

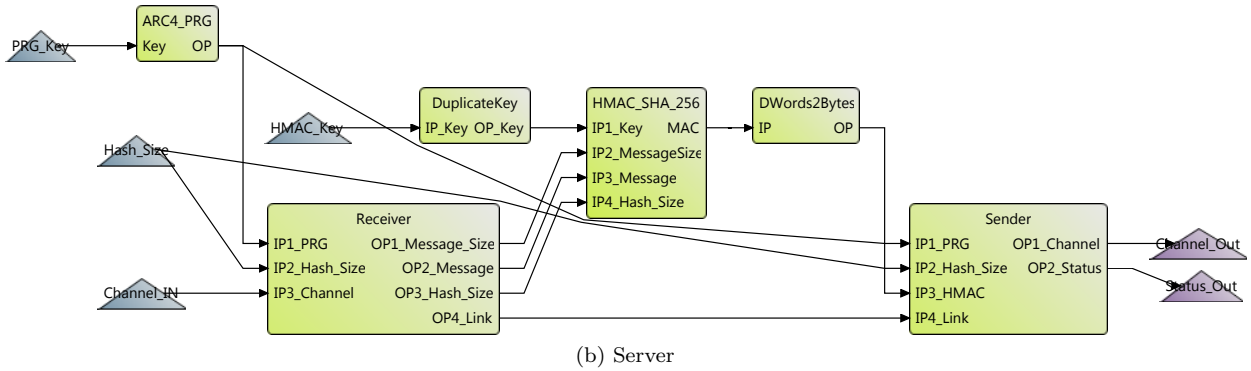
Figure 13 presents the designs of the SKID3's parties (the client and the server) in the RVC framework. Except the receiver and the sender FUs, the designs of both parties are exactly the same. The `ARC4_PRG` FU is used to generate the nonces, the `HMAC_SHA_256` FU performs the SHA-256 based HMAC, and the `DWords2Bytes` FU converts the HMAC output generated as double words to a stream of bytes.

The `Client_Receiver` FU receives data from the server (r_S and H_1) via the communication channel and forwards these data to `Client_Sender`. It also prepares data for HMAC operations (H_1' and H_2). The `Client_Sender` FU and also the HMACs H_1' and H_2), authenticates the server by comparing H_1' and H_1 and sends H_2 to the server.

The `Server_Receiver` FU receives the data from the client (CID, r_C and H_2) via the communication channel and forwards these data to `Server_Sender`. It also prepares data for HMAC operations (H_1 and H_2'). The `Server_Sender` FU authenticates the client by comparing H_2' and H_2.



(a) Client



(b) Server

Figure 13: The designs of the SKID3 client and server.

6.2.2. Hardware/Sofware Co-design and Interfacing

Figure 14 presents how the communication between the client and serve components of the SKID3 protocol can be realized with the help of `TCP_Interface_Client` and `TCP_Interface_Server`, respectively. We can see that, both client and server components send/receive data to/from their TCP interfaces, which implements the standard TCP/IP protocol. Since these TCP interfacing FUs are implemented by following the concepts of "native" and "wrappers" (described earlier in Sec. 3), these two FUs are dependent on the TCP/IP system calls available on the target platforms where client and server are eventually deployed.

In different security application scenarios, the SKID3 parties can be running either as software or hardware modules. Considering this, we evaluated the following two configurations of the SKID3 protocol:

1. the server running as an HW module and the client running an SW module;
2. the server running as an SW module and the client running an HW module.

The hardware components in both configurations were evaluated on the following two different hardware devices: Xilinx's Virtex-5 XC5VLX110T FPGA [88] (with Xilinx's XUPV5 LX110T board [89]), and Texas Instrument's TMS320C6678 DSP [90] (with TMDSEVM6678LE board [91]).

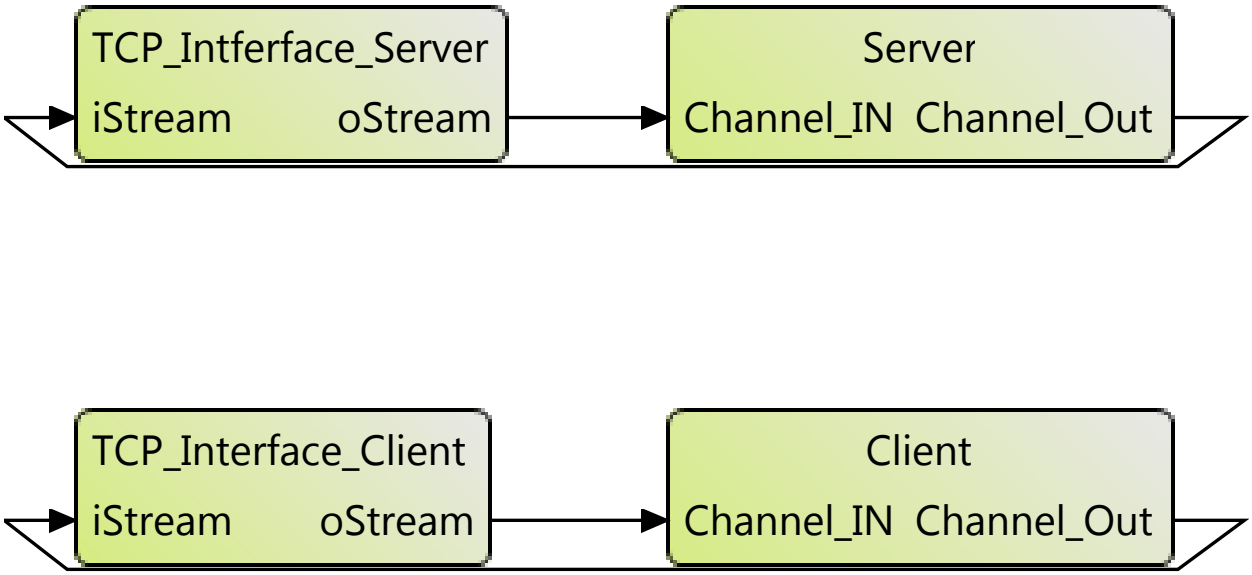


Figure 14: The interfacing between SKID3's client and server via their TCP/IP protocol.

For the FPGA, the Verilog backend of ORCC [54] was used to generate the Verilog code of the SKID3's client and server. Hence, we evaluated both aforesaid mentioned deployment configurations by hosting the server and the client on the FPGA device, respectively. The communication between the server and the client was provided via the TCP protocol between Ethernet interfaces of the two devices. Inside the FPGA, the TCP protocol (from the Lightweight IP (lwIP) library [92]) was implemented by a MicroBlaze^{TM}processor [93].

Similarly, for the DSP device, C backend of ORCC [54] was used to generate the C code of the SKID3's client and server and deployed them on the DSP device to evaluated both of the aforesaid mentioned configurations. The communication between the server and the client was provided via the TCP protocol between Ethernet interfaces of the two devices. The TCP protocol was implemented with a standard TCP/IP networking stack library.

7. Multimedia Security Applications

In [8], we introduced how the RVC framework can be efficiently used for the development of multimedia security applications. Furthermore, to augment our discussion we also presented the following three multimedia security applications: a joint encryption-encoding system for H.264/AVC videos, a joint encryption-encoding system for JPEG images, and compressed-domain JPEG image watermarking. In this paper, we do not revisit our discussion on these three multimedia security applications (as they are already covered in [8]) and present a multimedia steganographic application developed in the RVC framework.

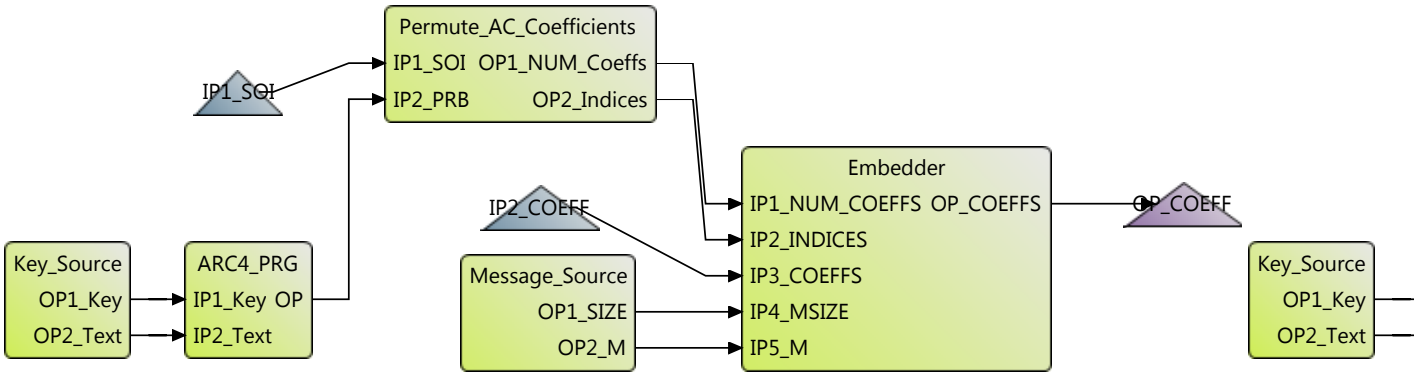
7.1. Compressed-Domain JPEG Image Steganography

In this subsection, we present an image steganography scheme working in JPEG compressed-domain called F5 [94]. In this scheme, the secret message is embedded only into the non-zero coefficients of the whole image using a matrix embedding scheme. Before the matrix embedding operation, the non-zero AC coefficients are permuted using some pseudo random generator so that the attacker cannot find the sequence in which the AC coefficients were used by the underlying matrix embedding scheme. In [94], the matrix embedding scheme used in F5 is based on the $(1, n, k)$ Hamming distance code, where $n = 2^k - 1$. Hence, to embed every k bits of the message, the embedding scheme changes at most one element in each set of n non-zero AC coefficients. The parameters k and n for each steganographic operation are computed as a

function of the size of the secret message being embedded and the number of non-zero AC coefficients such that the message just fits into the carrier image.

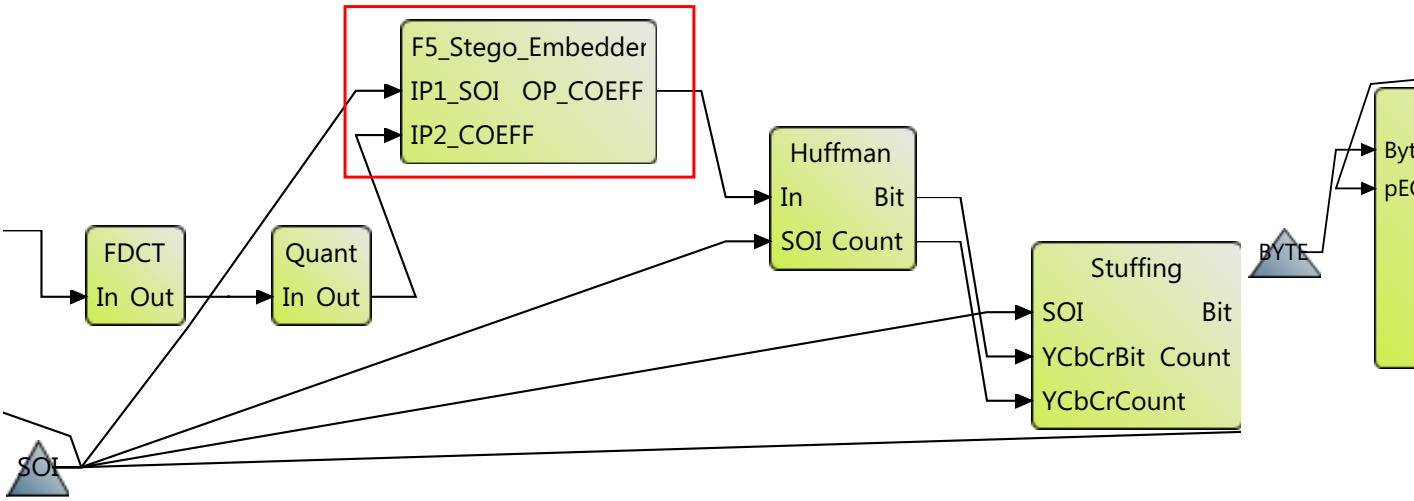
We implemented the steganographic embedder and detector FU networks of this scheme, which work with the RVC JPEG codec available in the ORCC Applications project [50]. Figures 15a and 15b show our implementations of the steganographic embedder and detector FU networks, where the `Permute_AC_Coefficients` FU permutes the indices of the AC coefficients using the pseudo random generator of `ARC4`. The permuted sequence of AC coefficients is sent to the embedder/detector, which perform its matrix embedding/detection operation on the non-zero AC coefficients.

It is worth mentioning that nsF5 [95], an improved version of `F5`, suggests using a matrix embedding scheme based on either the wet paper codes (WPC) or syndrome trellis codes (STC). Hence, our implementation of F5 can be easily reconfigured to nsF5 by only implementing WPC or STC based matrix embedder and detector schemes for the FUs `Embedder` and `Detector`, respectively.



(a) Embedder

Figure 15: The F5 steganographic embedder and detector FU networks.



(a) Embedder

Figure 16: The F5 steganographic embedder and detector with the RVC JPEG decoder.

7.1.1. Steganographic Embedder in JPEG Encoder

Figure 16a shows the F5 steganographic embedder FU network incorporated into the RVC JPEG encoder. Since this steganographic scheme works on quantized AC coefficients, we inserted the steganographic embedder FU network after DCT coefficients are quantized, but before they are Huffman coded. Hence, after the message is embedded, the whole DCT stego image is forwarded to the remainder of the encoder.

7.1.2. Steganographic Detector in JPEG Decoder

Figure 16b shows the F5 steganographic detector FU network incorporated in the RVC JPEG decoder. Similar to the steganographic embedder FU network, the detector FU network has been placed so that it can scan the quantized DCT coefficients before they go through the de-quantization step. The detector FU network works very similarly to the embedder FU network: it reconstructs the same permutation of the AC coefficients and performs the reverse of the matrix embedding operation to detect the message bits. After extracting the message bits, the whole DCT image is forwarded to the rest of the decoder.

8. Conclusion and Future Work

This paper provides a new solution for implementation-independent secure computing applications, which allows such systems to be developed in implementation-independent environment and implementations can be automatically generated towards target platforms. Our solution is based on the MPEG RVC dataflow programming framework which supports many desired system development features including hardware/software co-design. In addition, the performance benchmarking of secure computing applications on single-core and multi-core platforms confirmed that RVC applications obtain an adequate performance. Moreover, as system design examples, this paper also presents secure computing applications in the domains of applied cryptography and multimedia security.

In our future work, we plan to continue our research on the following directions.

Privacy-Preserving Applications. Concerning privacy preserving applications, as mentioned in Sec. 3, through design-space exploitation the RVC framework supports automated optimizations (e.g., refactoring) and parallelization of FUs in RVC based solutions. However, at the time of this writing, these features have only been evaluated for video codecs with up to a few hundred FUs and there are still some doubts about if these features can be extended towards far more complex applications (e.g., systems involving many thousands or even more FUs). There are some cryptographic protocols (e.g., garbled circuit protocols used in the privacy preserving applications [52]), which are quite complex in nature and can be used to study and explore the potentials of the RVC framework in this aspect. In addition, the run-time performance and circuit-oriented optimizations (e.g., simplification of circuit's topology, refactoring of selected FUs) of the garbled circuit protocols have become very hot topics within the cryptographic research community and some frameworks have already been proposed [96, 97, 98]. The most recent framework "Might Be Evil" [96] dramatically improves over previous frameworks (TASTY [97] and Fairplay [98]) in terms of run-time performance, circuit-oriented optimizations and the ease of development. However, all of these frameworks only allow the garbled circuit protocols to be implemented towards a single implementation and they do not offer any support for automatic parallelization of complex garbled circuits.

Currently, we are working to use the RVC approach to design and develop a new framework for implementation-independent and parallelizable garbled circuits. Although our work on this framework is not completed yet, we are using the concepts and techniques used in previous frameworks and extending them to include the support for multiple programming languages and automated parallelization (using design-space exploitation) targeted towards multi-core and many-core (e.g., GPUs, FPGAs) systems. On one side, this activity will give the cryptographic community another new (and hopefully also much better) framework providing implementation-independent and parallelizable garbled circuits. On the other hand, it will also help us to evaluate and improve the RVC algorithms (e.g., automatic design space exploitation for the garbled circuits) to support complex systems beyond video codecs.

More Cryptographic Primitives. The CTL can be enriched by including more cryptographic primitives (especially public-key cryptography), which will allow creation of more secure computing applications. Another direction is to develop optimized versions of CTL cryptosystems. For instance, bit slicing can be used to optimize parallelism in many block ciphers [37, 44].

More Security Protocols. In this paper, we have presented SKID3 as an exemplar application of hardware/software co-design. As the next step, we plan to implement the hPIN/hTAN e-banking security protocol [60], which uses SKID3 as a building block. The hPIN/hTAN is a typical (but small-scale) heterogeneous system involving a hardware token, a web browser plugin on the user's computer, and a web service running on the remote e-banking server. We have already implemented an hPIN/hTAN prototype system without using RVC, so the new RVC-based implementation can be benchmarked against the existing system.

Acknowledgments

Junaid Jameel Ahmad was supported by the Zukunftskolleg of the University of Konstanz, Germany, which is part of the "Excellence Initiative" Program of the DFG (German Research Foundation).

References

[1] J. J. Ahmad, S. Li, A.-R. Sadeghi, T. Schneider, CTL: A Platform-Independent Crypto Tools Library Based on Dataflow Programming Paradigm, in: Proceedings of 16th International Conference Financial Cryptography and Data Security (FC 2012), Vol. 7397 of Lecture Notes in Computer Science, Springer, 2012, pp. 299–313, an extended edition is available at http://eprint.iacr.org/2011/697.

[2] ISO/IEC, Information technology – MPEG systems technologies – Part 4: Codec configuration representation, ISO/IEC 23001-4, 2nd Edition (December 2011).

[3] ISO/IEC, Information technology – MPEG video technologies – Part 4: Video tool library, ISO/IEC 23002-4 (January 2010).

[4] W. R. Sutherland, The on-line graphical specification of computer procedures, Ph.D. thesis, MIT (1966).

[5] J. Eker, J. W. Janneck, CAL language report: Specification of the CAL actor language, Technical Memo UCB/ERL M03/48, Electronics Research Laboratory, UC Berkeley (2003).

[6] C. Lucarz, M. Mattavelli, J. Dubois, A co-design platform for algorithm/architecture design exploration, in: Proc. 2008 IEEE International Conference on Multimedia and Expo (ICME 2008), IEEE, 2008, pp. 1069–1072. doi:10.1109/ICME.2008.4607623.

[7] J. Boutellier, V. M. Gomez, O. Silvén, C. Lucarz, M. Mattavelli, Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework, in: Proc. 2009 Conference on Design and Architectures for Signal and Image Processing (DASIP 2009), 2009. doi:10.1007/978-90-481-9965-5_11.

[8] J. J. Ahmad, S. Li, I. Amer, M. Mattavelli, Building multimedia security applications in the MPEG Reconfigurable Video Coding (RVC) framework, in: Proceedings of 2011 ACM SIGMM Multimedia and Security Workshop (MM&Sec 2011), ACM, 2011, pp. 121–130. doi:10.1145/2037252.2037275.

[9] ISO/IEC, Information technology – MPEG systems technologies – Part 4: Codec configuration representation, ISO/IEC 23001-4 (December 2009).

[10] E. Khan, M. W. El-Kharashi, F. Gebali, M. Abd-El-Barr, Applying the Handel-C design flow in designing an HMAC-hash unit on FPGAs, IEE Proc. Computers and Digital Techniques 153 (5) (2006) 323–334. doi:10.1049/ip-cdt:20050192.

[11] A. Antola, M. Fracassi, P. Gotti, C. Sandionigi, M. Santambrogio, A novel hardware/software codesign methodology based on dynamic reconfiguration with Impulse C and CoDeveloper, in: Proc. 2007 3rd Southern Conference on Programmable Logic (SPL 2007), IEEE, 2007, pp. 221–224. doi:10.1109/SPL.2007.371754.

[12] S. Gupta, N. Dutt, R. Gupta, A. Nicolau, SPARK: A high-level synthesis framework for applying parallelizing compiler transformations, in: Proc. 2003 16th International Conference on VLSI Design (VLSI Design 2003), IEEE, 2003. doi:10.1109/ICVD.2003.1183177.

[13] R. Nikhil, Tutorial – BlueSpec System Verilog: Efficient, correct RTL from high-level specifications, in: Proc. 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2004), IEEE, 2004, pp. 69–70. doi:10.1109/MEMCOD.2004.1459818.

[14] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, E. F. Deprettere, A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs, in: Proc. 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS 2007), ACM, 2007, pp. 9–14. doi:10.1145/1289816.1289823.

[15] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, K. Kuusilinna, UML-based multiprocessor SoC design framework, ACM Trans. on Embedded Compututer Systems 5 (2006) 281–320. doi:10.1145/1151074.1151077.

[16] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, Y.-P. Joo, PeaCE: A hardware-software codesign environment for multimedia embedded systems, ACM Trans. on Design Automation of Electronic Syststems 12 (3) (2007) Article 24. doi:10.1145/1255456.1255461.
[17] K. V. Rompaey, D. Verkest, I. Bolsens, H. D. Man, CoWare – a design environment for heterogeneous hardware/software systems, Design Automation for Embedded Systems 1 (4) (1996) 357–386.
[18] Esterel Synchronous Language, http://www-sop.inria.fr/esterel.org/files/.
[19] LabVIEW, http://www.ni.com/labview/whatis/.
[20] Mathworks Simulink Coder, http://www.mathworks.com/products/simulink-coder/.
[21] Mathworks Simulink: Simulation and Model-Based Design, www.mathworks.com/products/matlab-coder.
[22] Synopsys Studio, http://www.synopsys.com/SYSTEMS/BLOCKDESIGN/DIGITALSIGNALPROCESSING/Pages/SystemStudio.aspx.
[23] A. Moss, D. Page, Bridging the gap between symbolic and efficient AES implementations, in: Proc. 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM 2010), ACM, 2010, pp. 101–110. doi:10.1145/1706356.1706376.
[24] CAO and qhasm compiler tools, EU Project CACE deliverable D1.3, Revision 1.1, http://www.cace-project.eu/downloads/deliverables-y3/32_CACE_D1.3_CAO_and_qhasm_compiler_tools_Jan11.pdf (2011).
[25] J. R. Lewis, B. Martin, Cryptol: High assurance, retargetable crypto development and validation, in: Proc. 2003 IEEE Military Communication Conference (MILCOM 2003), IEEE, 2003, pp. 820–825. doi:10.1109/MILCOM.2003.1290218.
[26] Cryptol: The language of cryptography, Case Study, http://corp.galois.com/downloads/cryptography/Cryptol_Casestudy.pdf (2008).
[27] J. A. Akinyele, M. D. Green, A. D. Rubin, Charm: A framework for rapidly prototyping cryptosystems, Cryptology ePrint Archive: Report 2011/617, http://eprint.iacr.org/2011/617 (2011).
[28] Charm: A tool for rapid cryptographic prototyping, http://www.charm-crypto.com.
[29] W. Dai, Crypto++ library, http://www.cryptopp.com.
[30] P. Gutmann, Cryptlib, http://www.cs.auckland.ac.nz/~pgut001/cryptlib.
[31] The OpenSSL Project, OpenSSL cryptographic library, http://www.openssl.org/docs/crypto/crypto.html.
[32] T. Moran, The Qilin Crypto SDK: An open-source Java SDK for rapid prototyping of cryptographic protocols, http://qilin.seas.harvard.edu/.
[33] The Legion of the Bouncy Castle, Bouncy Castle Crypto APIs, http://www.bouncycastle.org.
[34] PureNoise Ltd Vaduz, PureNoise CryptoLib, http://cryptolib.com/crypto.
[35] T. Pornin, sphlib 3.0, http://www.saphir2.com/sphlib.
[36] G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, S. Marchesin, Efficient software implementation of AES on 32-bit platforms, in: Cryptographic Hardware and Embedded Systems – CHES 2002, Vol. 2523 of Lecture Notes in Computer Science, Springer, 2002, pp. 159–171. doi:10.1007/3-540-36400-5_13.
[37] M. Matsui, J. Nakajima, On the power of bitslice implementation on Intel Core2 processor, in: Cryptographic Hardware and Embedded Systems – CHES 2007, Vol. 4727 of Lecture Notes in Computer Science, Springer, 2007, pp. 121–134. doi:10.1007/978-3-540-74735-2_9.
[38] S. Tillich, C. Herbst, Boosting AES performance on a tiny processor core, in: Topics in Cryptology – CT-RSA 2008, Vol. 4964 of Lecture Notes in Computer Science, Springer, 2008, pp. 170–186. doi:10.1007/978-3-540-79263-5_11.
[39] D. J. Bernstein, P. Schwabe, New AES software speed records, in: Progress in Cryptology – INDOCRYPT 2008, Vol. 5365 of Lecture Notes in Computer Science, Springer, 2008, pp. 322–336. doi:10.1007/978-3-540-89754-5_25.
[40] D. Canright, D. A. Osvik, A more compact AES, in: Selected Areas in Cryptography (SAC 2009), Vol. 5867 of Lecture Notes in Computer Science, Springer, 2009, pp. 157–169. doi:10.1007/978-3-642-05445-7_10.
[41] R. Manley, D. Gregg, A program generator for intel AES-NI instructions, in: Progress in Cryptology – INDOCRYPT 2010, Vol. 6498 of Lecture Notes in Computer Science, Springer, 2010, pp. 311–327. doi:10.1007/978-3-642-17401-8_22.
[42] D. A. Osvik, J. W. Bos, D. Stefan, D. Canright, Fast software AES encryption, in: Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers, Vol. 6147 of Lecture Notes in Computer Science, Springer, 2010, pp. 75–93. doi:10.1007/978-3-642-13858-4_5.
[43] S. Tillich, J. Großschädl, Instruction set extensions for efficient AES implementation on 32-bit processors, in: Cryptographic Hardware and Embedded Systems – CHES 2006, Vol. 4249 of Lecture Notes in Computer Science, Springer, 2006, pp. 270–284. doi:DOI:10.1007/11894063_22.
[44] P. Grabher, J. Großschädl, D. Page, Light-weight instruction set extensions for bit-sliced cryptography, in: Cryptographic Hardware and Embedded Systems – CHES 2008, Vol. 5154 of Lecture Notes in Computer Science, Springer, 2008, pp. 331–345. doi:10.1007/978-3-540-85053-3_21.
[45] S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, M. Raulet, Overview of the MPEG Reconfigurable Video Coding framework, J. Signal Processing Systems 63 (2) (2011) 251–263. doi:10.1007/s11265-009-0399-3.
[46] I. Amer, C. Lucarz, G. Roquier, M. Mattavelli, M. Raulet, J. Nezan, O. Déforges, Reconfigurable Video Coding on multicore: An overview of its main objectives, IEEE Signal Processing Magazine 26 (6) (2009) 113–123. doi:10.1109/MSP.2009.934107.
[47] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, M. Raulet, Synthesizing hardware from dataflow programs: An MPEG-4 Simple Profile decoder case study, J. Signal Processing Systems 63 (2) (2011) 241–249. doi:10.1007/s11265-009-0397-5.
[48] H. Aman-Allah, K. Maarouf, E. Hanna, I. Amer, M. Mattavelli, CAL dataflow components for an MPEG RVC AVC baseline encoder, J. Signal Processing Systems 63 (2) (2011) 227–239. doi:10.1007/s11265-009-0396-6.
[49] S. Lee, T. Lim, E. Jang, J. H. Lee, SeungwookLee, MPEG Reconfigurable Graphics Coding framework: Overview and

applications, in: Proc. 2011 IEEE Visual Communications and Image Processing Conference (VCIP 2011), IEEE, 2011. doi:10.1109/VCIP.2011.6116047.
[50] RVC implementation of JPEG codec, http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/JPEG/.
[51] H. I. A. A. Ali, M. N. I. Patoary, Design and implementation of an audio codec (AMR-WB) using dataflow programming language CAL in the OpenDF environment, TR: IDE1009, Halmstad University, Sweden (2010).
[52] T. Schneider, Engineering secure two-party computation protocols – advances in design, optimization, and applications of efficient secure function evaluation, Ph.D. thesis, Ruhr-Universität Bochum (2011).
[53] Open Data Flow (OpenDF), http://sourceforge.net/projects/opendf.
[54] Open RVC-CAL Compiler (ORCC), http://sourceforge.net/projects/orcc.
[55] Graphiti, http://graphiti-editor.sf.net.
[56] M. Wipliez, M. Raulet, J. Nezan, Proposition to add support of native functions and procedures to rvc-cal, ISO/IEC JTC1/SC29/WG11, MPEG2011/m20076, 96th MPEG Meeting, Geneva, Switzerland (March 2011).
[57] R. Thavot, R. Mosqueron, J. Dubois, M. Mattavelli, Hardware synthesis of complex standard interfaces using CAL dataflow descriptions, in: Proc. 2009 Conference on Design and Architectures for Signal and Image Processing (DASIP 2009), ECSI, 2009.
URL http://infoscience.epfl.ch/record/146591
[58] NIST, Secure Hash Standard (SHS), FIPS PUB 180-3 (2008).
[59] RVC implementation of MPEG-4 Part-2 decoder, http://orc-apps.svn.sourceforge.net/viewvc/orc-apps/trunk/RVC/.
[60] S. Li, A.-R. Sadeghi, S. Heisrat, R. Schmitz, J. J. Ahmad, hPIN/hTAN: A lightweight and low-cost e-banking solution against untrusted computers, in: Financial Cryptography and Data Security (FC 2011), Vol. 7035 of Lecture Notes in Computer Science, Springer, 2011, pp. 235–249. doi:10.1007/978-3-642-27576-0_19.
[61] J. J. Ahmad, Secure Computing with the MPEG RVC Framework, Ph.D. thesis, University of Konstanz, Germany, available at http://kops.ub.uni-konstanz.de/handle/urn:nbn:de:bsz:352-221317. (December 2012).
[62] M. Raulet, S. Li, Text of ISO/IEC 23001-4:201X/PDAM 1 cal language extensions, ISO/IEC JTC1/SC29/WG11, MPEG2011/w12370, 98th MPEG Meeting, Geneva, Switzerland (November 2011).
[63] J. W. Janneck, M. Mattavelli, M. Wipliez, M. Raulet, S. Li, J. Eker, C. V. Platen, G. Roquier, I. Amer, C. Lucarz, P. Faure, J. J. Ahmad, A proposal of RVC-CAL extensions for improved support of I/O processing, ISO/IEC JTC1/SC29/WG11, MPEG2010/m18460, 94th MPEG Meeting, Guangzhou, China (October 2010).
[64] J. W. Janneck, M. Raulet, M. Wiplietz, J. Eker, G. Roquier, I. Amer, C. Lucarz, S. Li, J. J. Ahmad, M. Mattavelli, Writing dataflow networks components (FUs) with different models of computations using RVC-CAL: a tutorial, ISO/IEC JTC1/SC29/WG11, MPEG2010/m18458, 94th MPEG Meeting, Guangzhou, China (October 2010).
[65] S. Li, Defect report on ISO/IEC 23001-4, ISO/IEC JTC1/SC29/WG11, MPEG2010/w11469, 93rd MPEG Meeting, Geneva, Switzerland (July 2010).
[66] J. J. Ahmad, S. Li, Evaluating the support of handling huge XDF networks by the Open RVC-CAL Compiler (ORCC) and the Graphiti-Editor, ISO/IEC JTC1/SC29/WG11, MPEG2011/m21249, 97th MPEG Meeting, Torino, Italy (July 2011).
[67] S. C. Brunet, A. Elguindy, E. Bezati, R. Thavot, G. Roquier, M. Mattavelli, J. W. Janneck, Methods to Explore Design Space for MPEG RVC Codec Specifications, EURASIP Journal on Signal Processing Image Communication (SPIC).
[68] D. de Saint Jorre, J. Gorin, J. Nezan, M. Raulet, N. Siret, M. Wipliez, H. Yviquel, Report on performance of generated code (C, LLVM, and VHDL) from RVC descriptions, ISO/IEC JTC1/SC29/WG11, MPEG2011/m20074, 96th MPEG Meeting, Geneva, Switzerland (March 2011).
[69] J. J. Ahmad, S. Li, A comparative study on the performance of C code automatically generated by ORCC from RVC-CAL code, ISO/IEC JTC1/SC29/WG11, MPEG2011/m19383, 95th MPEG Meeting, Daegu, Korea (January 2011).
[70] J. J. Ahmad, S. Li, Extending the comparative study on the performance of C code automatically generated by orcc from RVC-CAL code to a resource-constrained embedded system, ISO/IEC JTC1/SC29/WG11, MPEG2011/m21289, 97th MPEG Meeting, Torino, Italy (July 2011).
[71] J. J. Ahmad, S. Li, Performance benchmarking of C code automatically generated by ORCC from RVC-CAL code on a quad-core machine, ISO/IEC JTC1/SC29/WG11, MPEG2011/m21250, 97th MPEG Meeting, Torino, Italy (July 2011).
[72] NIST, Specification for the Advanced Encryption Standard (AES), FIPS PUB 197 (2001).
[73] NIST, Data Encryption Standard (DES), FIPS PUB 46-3 (1999).
[74] NIST, Recommendation for the Triple Data Encryption Algorithm (TDEA) block cipher, Special Publication 800-67, Version 1.1 (2008).
[75] B. Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish), in: Fast Software Encryption (FSE'94), Vol. 809 of Lecture Notes in Computer Science, Springer, 1994, pp. 191–204. doi:10.1007/3-540-58108-1_24.
[76] B. Schneier, Applied Cryptography: Protocols, algorithms, and source code in C, 2nd Edition, John Wiley & Sons, Inc., New York, 1996.
[77] Cryptico A/S, Lightweight IP (lwIP) application examples, White Paper, Version 1.4, available online at http://www.cryptico.com/DWSDownload.asp?File=Files%2FFiler%2FWP%5FRabbit%5FPerformance%2Epdf (2005).
[78] NIST, The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198 (2002).
[79] IETF, HMAC: Keyed-Hashing for Message Authentication, RFC 2104 (1997).
[80] V. Rijmen, A. Bosselaers, P. Barreto, Optimised ANSI C code for the cipher (now AES), Rijndael Reference Implementation, Version 3.0, public domain software (2000).
[81] P. Barreto, V. Rijmen, Rijndeal reference implementation (+ KATs and MCTs) v2.2, Public domain software (1999).
[82] X-N2O, AES explained, http://www.x-n2o.com/aes-explained.
[83] Oracle®, Java™ Cryptography Architecture (JCA) Reference Guide, http://download.oracle.com/javase/6/docs/

technotes/guides/security/crypto/CryptoSpec.html.
[84] Intel® Mote 2 engineering platform, not available at Intel web site anymore, a copy available at http://ubi.cs.washington.edu/files/imote2/docs/imote2-ds-rev2.0.pdf.
[85] Intel Corporation, Intel XScale® core developer's manual (2004).
[86] Intel Corporation, Using the RDTSC instruction for performance monitoring, not available at Intel web site anymore, a copy still available at http://www.ccsl.carleton.ca/~jamuir/rdtscpm1.pdf (1997).
[87] A. Bosselaers, B. Preneel (Eds.), SKID, Vol. 1007 of Lecture Notes in Computer Science, Springer, 1995, Ch. 6, pp. 169–178. doi:10.1007/3-540-60640-8_8.
[88] XILINX, Virtex-5 family overview, http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.
[89] XILINX, Xilinx university program xupv5-lx110t development system, http://www.xilinx.com/univ/xupv5-lx110t.htm.
[90] T. Instruments, Tms320c6678 multicore fixed and floating-point digital signal processor, Data manual, available online at http://www.ti.com/lit/ds/symlink/tms320c6678.pdf.
[91] T. Instruments, Tmdsevm6678le evaluation modules, http://www.ti.com/tool/tmdxevm6678?DCMP=multicore&HQS=c6678-bhp-tf.
[92] S. MacMahon, N. Zang, A. Sarangi, Rabbit stream cipher, performance evaluation, XILINX Application Note: Embedding Processing, XA1026 (v3.1), available online at http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf (2011).
[93] XILINX, Microblaze soft processor core, http://www.xilinx.com/tools/microblaze.htm.
[94] A. Westfeld, F5–a steganographic algorithm: High capacity despite better steganalysis, in: Information Hiding: 4th International Workshop, IH 2001 Pittsburgh, PA, USA, April 25–27, 2001 Proceedings, Vol. 2137 of Lecture Notes in Computer Science, Spring, 2001, pp. 289–302. doi:10.1007/3-540-45496-9_21.
[95] J. Fridrich, J. Kodovsky, Simulator for nsF5 embedding, http://dde.binghamton.edu/download/nsf5simulator/.
[96] Y. Huang, D. Evans, J. Katz, L. Malka, Faster secure two-party computation using garbled circuits, in: Proceedings of 20th USENIX Security Symposium, USENIX Association, 2011.
[97] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, I. Wehrenberg, TASTY: Tool for Automating Secure Two-partY computations, in: Proceedings of 17th ACM Conference on Computer and Communications Security (CCS 2010), ACM, 2010, pp. 451–462. doi:10.1145/1866307.1866358.
[98] A. Ben-david, N. Nisan, B. Pinkas, FairplayMP – a system for secure multi-party computation, in: Proceedings of 15th ACM Conference on Computer and Communications Security (CCS 2008), ACM, 2008, pp. 257–266. doi:10.1145/1455770.1455804.