SoK: Instruction Set Extensions for Cryptographers

Hao Cheng[1], Johann Großschädl[1], Ben Marshall[2], Daniel Page[3] and Markku-Juhani O. Saarinen[4]

[1] DCS and SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg.
{hao.cheng,johann.groszschaedl}@uni.lu
[2] PQShield Ltd, Oxford, UK.
ben.marshall@pqshield.com
[3] Department of Computer Science, University of Bristol, Bristol, UK.
daniel.page@bristol.ac.uk
[4] SoC Hub Research Centre, Tampere University, Tampere, Finland.
markku-juhani.saarinen@tuni.fi

Abstract. Framed within the general context of cyber-security, standard cryptographic constructions often represent an enabling technology for associated solutions. Alongside or in combination with their design, therefore, the implementation of such constructions is an important challenge: beyond delivering artefacts that are usable in practice, implementation can impact many quality metrics (such as efficiency and security) which determine fitness-for-purpose. A rich design space of implementation techniques can be drawn on in order to address this challenge, but threat- and opportunity-driven innovation based on clear understanding and empirical evidence remains vital.

In at least some use-cases, software-based implementation of cryptography is important, e.g., because it delivers an attractive trade-off or is mandated for some reason. Such an implementation is heavily influenced both by 1) the Instruction Set Architecture (ISA) it is expressed using, and 2) the micro-architecture it is executed using. For example, the extent to which a general-purpose ISA can support more domain-specific requirements of a cryptographic construction will influence how the latter is mapped to the former (i.e., which implementation techniques are viable) *and* behavioural properties of doing so (e.g., the execution latency stemming from use of a given implementation technique).

This paper attempts to systematise the topic of cryptographic Instruction Set Extensions (ISEs), which represent an approach to provision of a platform where such support is more explicit and extensive. At a high level, the goal is to improve understanding of what is an extensive and somewhat inter-disciplinary body of literature (e.g., spanning academia and industry, hardware and software, as well as cryptographic and non-cryptographic publication venues). We argue that doing so will help to maximise the quality of subsequent work on this and associated topics.

Keywords: ISA, ISE, cryptographic engineering

1 Introduction

1.1 The micro-processor customisation design space

ISAs. The concept of an ISA is fundamentally important within the context of computer systems. Acting as an interface between hardware, i.e., some compliant micro-architectural implementation, and software executed by it, the ISA will usually define components such

as 1) the accessible state, including general- and special-purpose registers and memory, 2) the encoding and semantics of instructions that act on said state, and 3) an execution model for said instructions. The design of an ISA is, to some extent, a creative process driven by a technical ethos informing design decisions and trade-offs. However, the large design space, potential lifespan, and implications that features in an ISA have for hardware and software, demand the process is *also* driven by robust empirical evidence. This is often achieved through characterisation of software workloads, in order to motivate and evaluate features in the design. To appeal to the largest possible market such workloads are usually of a *general*-purpose composition (e.g., representative benchmark kernels and suites thereof, or complete applications), with evaluation with respect to pertinent quality metrics (e.g., instruction throughput) usually focusing on optimisation for the average-case. Although this approach is rational, it *may* be viewed as conservative in the sense it neglects opportunities related to *special*-purpose or *domain-specific* workloads.

The ISA is, and arguably [DB18] should remain an interface. This approach allows specialisation of a micro-architecture to satisfy any pertinent quality metric, market, or use-case, and thus, to some extent at least, workload. Beyond this fact, however, micro-processor customisation [IL07] captures broader specialisation of *both* the implementation *and* interface for a given domain. In short, a customisable micro-processor may be either

- parameterisable, in the sense that a template base ISA and micro-architecture are instantiated based on user-selected options, and/or
- extensible, in the sense that a base ISA and micro-architecture can be extended with user-defined functionality.

ISEs. ISEs have become a popular extensibility mechanism. In concept, the ISE design process typically mirrors that for ISAs apart from a focus on some specific domain. That is, one would 1) perform workload characterisation to identify functionality that could yield an improvement with respect to pertinent quality metrics (over use of the base ISA alone), 2) extend the base ISA with suitable state and instructions, thereby exposing them to ISE-aware software, then, finally, 3) implement additional components in a base micro-architecture, thereby allowing said software to be executed.

Other implementation options for a given workload include at least software-only (i.e., using the base ISA alone) and hardware-only (i.e., using a dedicated IP core) extremes. ISEs are attractive specifically because they represent a hybrid that sits between such extremes, suggesting they inherit characteristics of both; in a general sense, therefore, they relate to the field of hardware/software co-design [DEWW01]. Whether or not they are the *right* option is heavily dependant on the context, but, for example, an ISE-supported solution will often be more compact and performant than a software-only option, while also being more flexible and more efficient in terms of performance gain per additional logic gate than a hardware-only option. Such characteristics can be important for *both* high-end, performance-oriented *and* low-end, constrained platforms.

Although isolated examples[1] existed previously, media processing, e.g., decoding image and video formats such as JPEG and MPEG, was (arguably) the domain which popularised ISE-supported solutions more broadly. Market demand for efficiency in relation to such workloads led to development of ISEs [SS05] for most contemporary base ISAs, including AMD 3DNow! [OFW99], DEC Motion Video Instructions (MVI) [CCM97], Hewlett Packard Multimedia Acceleration eXtensions (MAX and MAX-2) [Lee95, Lee96, LH96], Intel MultiMedia eXtensions (MMX) [PW96], Motorola Altivec [DDHS00], MIPS Digital Media eXtension (MDMX) and MIPS-3D, and Sun Visual Instruction Set (VIS) [KMPZ95].

[1] One such example is support for computation of population count (i.e., Hamming weight), exemplified by the x86 `popcnt` [Int22b, Pages 4-399–4-400] instruction. Motivation for what is a fairly niche operation has an interesting (and in part alleged) history: see, e.g., https://groups.google.com/g/comp.arch/c/UXEi7G6WHuU/m/Z2z7fC7Xhr8J.

Subsequent examples highlight a subtlety, in the sense that the definition of an *extension of* versus an *addition to* the base ISA is imprecise: several of the ISEs listed above would arguably fall into the latter category. For example, what was initially Intel MMX has evolved into several generations of Streaming SIMD Extensions (SSE) [TH99] and then Advanced Vector Extensions (AVX). Such designs addressed market demand for efficient numerical computation, offering support via a vector (or SIMD) versus scalar programming model. Although support for them is detectable using the `cpuid` [Int22a, Chapter 20] feature identification mechanism, Intel-based micro-architectures rarely *remove* features: the ISEs effectively act as additions to the base ISA therefore. In contrast, ISAs such as RISC-V emphasise modularity and extensibility as first-class, by-design goals (see, e.g., [RV19, Section 26]). RV32I [RV19, Section 2], for example, defines a minimal base ISA plus a suite of (orthogonal) extensions which can be categorised as 1) standard extensions curated and ratified by RISC-V International; selected examples aim to support additional functionality (e.g., floating-point, via the standard F [RV19, Section 11] and D [RV19, Section 12] extensions), or satisfy specific optimisation goals (e.g., code density, via the standard C [RV19, Section 16] extension), plus 2) non-standard (or custom) extensions which are user-defined. Support for so-called "custom instructions" [CP20] within ARMv8-M could be viewed as having a similar, although more limited remit.

ASIPs. The difference between extension of a base ISA which does or does not have by-design support for extensibility is important. For example, where such support *is* evident, a vendor can decide whether or not a given a set of orthogonal domains are relevant to their intended market, then either include or exclude ISEs as need be.

This is taken to an extreme by the concept of an Application Specific Instruction Processor (ASIP), definitions of which can vary but usually includes three features. First, both the base ISA and micro-architecture are explicitly designed to support customisation. Second, tasks relating to customisation of these artefacts are often supported by an integrated suite of dedicated, (semi-)automatic tooling. Tensilica Xtensa [Gon00] is an exemplar of both points. The platform includes an 80-instruction base ISA and associated base micro-architecture which are both parameterisable (e.g., with respect to endianness, register file size, cache geometry, bus width, etc.), and extensible (via extensions to the base ISA, i.e., ISEs) by-design; the latter is supported by a proprietary language, namely Tensilica Instruction Extension (TIE), and some associated tooling. Moreover, third, trade-offs for an ASIP can be aggressive with respect to specialisation. Put simply, an ASIP might *only* be used within a specific domain or for a specific task. This contrasts with a non-ASIP, which must support general-purpose workloads and use, e.g., ISEs to support special-purpose niches.

1.2 Toward principled, effective cryptographic ISE development

Cryptographic ISEs. The domain of cryptography shares various high-level characteristics with that of media processing. For example, cryptographic workloads typically 1) involve computationally intensive, somewhat niche functionality, 2) need to satisfy a range of efficiency-related quality metrics such as throughput, latency, memory footprint, and power and energy consumption, but, at the same time, *also* 3) form a central target in what is a complex, evolving attack surface. This latter fact demands (see, e.g., [RKL+04, RRKH04, BMT16]) security be viewed an important, additional quality metric. A rich body of literature, capturing the field of cryptographic engineering, has explored techniques which attempt to address associated implementation challenges (including those relating to micro-processor design specifically: see, e.g., [Lee03]); this forms a significant design space of options.

Nahum et al. [NOOS95] argue that, although cryptographic hardware can be a viable option, various factors such as flexibility (or agility) mean that "*we need cryptographic*

software". Their work is among, if not the first to propose an ISE-supported solution as a means of addressing that need in balance with other metrics. This suggests the field of cryptographic ISEs [BGM09, HV11, RI16] spans at least a 25 year period, with support for Advanced Encryption Standard (AES) [NIST01] an important exemplar. For example, one can readily identify a significant body of literature [NIK04, TG05, TGS05, BBFR06, MD07, TG06, TG07a, TG07b, Elb07, Elb08, BBGR09, SCS+09, JSG10, BOS11, APRJ11, BEM+15, EMOC20, Saa20], patents [GFG], and broad support in deployed base ISAs: examples include x86 [Int22a, Section 12.13] (see also [Gue09, DGK19]), POWER [Pow18, Section 6.11.1], ARMv8-A [Arm20, Section A2.3], SPARC [SPA16, Sections 7.3+7.4], and RISC-V [RVK22, RVK21, RIS24] (see also [MPP20, MNP+21]).

Remit and organisation. Nahum et al. [NOOS95] pondered whether, in 1995, it was "*practical to add instructions to a [RISC] processor*"; they identify both technological and economic dimensions to this question. Whether or not the answer was positive then, we argue that two features of the contemporary technology landscape mean it certainly *is* positive now. First, open ISAs such as RISC-V [Wat16], and associated ecosystems (including open-source implementations, tooling, and crucially, community) have matured to the point of being competitive. Hill et al. [HCP+16] explore advantages and disadvantages of this approach, but, fundamentally, it has enhanced accessibility and so innovation where it would previously, in a proprietary context, be far more difficult. Second, concrete realisation of such implementations is now viable with relatively short, relatively low-cost design cycles for a given ISE. At least two facts evidence this claim, namely

- the availability and feature-sets of reconfigurable fabrics: this fact leads to viability of customisable FPGA-based soft *and* hard (e.g., supported by eFPGAs[2] and similar technologies) implementations, and
- an evolution of business models: this fact leads to viability of customisable ASIC-based hard micro-processor implementations, e.g., as the result of new vendors[3] and initiatives[4] which have lowered the barrier to entry versus traditional fabrication.

So, based on the argument that cryptographic ISEs are both practical and effective (cf. [FLO18]), and mirroring [IL07, Section 1.4] for example, we suggest that the field is entering a "golden age" in which maximising the quality of associated work is vital. This paper attempts to assist in doing so, acting as a Systematization of Knowledge (SoK) for the field of cryptographic ISEs. We organise the content as follows:

- In Section 2 and Section 3 we attempt to fix some notation and terminology, and, in doing so, refine the technical scope of subsequent sections.
- In Section 4 we survey existing cryptographic ISEs. A specific aim of doing so is the identification of high-level features which motivate their use, and thus promote *re*use of existing solutions and/or techniques in future work.

Throughout said content, we focus on two overarching aims. First, high quality cryptographic engineering is inherently interdisciplinary. In line with advice from Paar [Paa02, Point 5] that we "*educate ourselves about other fields*" and also "*entice people from other disciplines to work on problems in applied cryptography*", we therefore aim to consolidate experience around cryptographic ISEs and so increase future understanding and accessibility of the field. The paper title is specifically motivated by this point, and so is written in a similar spirit as, e.g., [GPS08, BGHZ11]. Second, consider that AES[5] was instrumental in popularising a standardisation process model where "*algorithm and implementation*

[2] See, e.g., https://flex-logix.com
[3] See, e.g., https://www.sifive.com
[4] See, e.g., https://efabless.com/chipignite
[5] See, e.g., https://www.govinfo.gov/content/pkg/FR-1997-09-12/pdf/97-24214.pdf

characteristics" play a role in evaluation of candidates. Although seldom *explicitly* ruled out, many such processes fix (a set of) non-customisable evaluation platforms which *implicitly* ignore or undervalue the role ISEs can and eventually *do* play. We argue this gap is problematic with respect to comprehensive evaluation, so aim to help shift best-practice to include consideration of ISEs. Finally, note that we often try to articulate points using examples which involve existing ISE designs. Although we critique such designs, for example to highlight a feature that, in our opinion, could be deemed a positive or negative, we categorically do *not* aim to criticise them.

2 Notation

Let $x_{(b)}$ denote x expressed in radix- or base-b. If the base is omitted, it is safe to assume use of decimal (i.e., that $b = 10$). Let $x \leftarrow y$ denote assignment of y to x. Let $\neg$, $\wedge$, $\vee$, and $\oplus$ denote the Boolean NOT, AND, (inclusive) OR, and (exclusive OR, or) XOR operators respectively, and $x \ll y$ and $x \lll y$ (resp. $x \gg y$ and $x \ggg y$) denote left-shift and left-rotate (resp. right-shift and right-rotate) of x by y bits.

Let $\mathsf{MEM}[i]^b$ denote a b-byte access to some byte-addressable memory, using the effective address i; where $b = 1$, the access granularity may be omitted. Let $\mathsf{GPR}[i]$, for $0 \leq i < r$, denote the i-th, w-bit entry in the r-entry general-purpose register file.

3 Terminology

Concept 1. An ISE comprises 1) a specification of an **interface** between hardware and software, plus 2) a hardware-based **implementation** within some micro-architecture.

Concept 2. The terms **base ISA** and **extended ISA** (resp. **base micro-architecture** and **extended micro-architecture**) are used to refer to the ISA (resp. micro-architecture) excluding and including the ISE interface (resp. implementation) respectively.

As is the case with an ISA, the ISE interface allows diversity with respect to the associated implementation. Versus an ISA, however, it is common to consider the ISE interface and implementation simultaneously, i.e., as part of a single design. As a result, the term ISE is often overloaded to mean either the interface and/or the implementation depending on the context.

Concept 3. A **bespoke ISA** is explicitly domain-specific by nature, and so distinct from some **generic ISA** being extended by a domain-specific ISE.

There is clearly overlap between the two approaches, and, up to a point, the end result could be deemed similar. However, we distinguish between them in part to limit our scope through a focus on the latter: doing so renders some examples of the former (e.g., ASIPs designs such as CryptoManiac [WWA01], Cryptonite [BHO04], MCCP [GBG+11], CIARP [NRE+12], Cryptoraptor [SC14], CRISC [DR14], CDSP [YHMH19], plus related tools such as [MA07], and work such as [GBG+03, Fou07, YYDZ08, BSP13]) out of scope. Beyond this fact, however, focusing on the extension of a generic, ideally well established base ISA affords (at least) two advantages. First, the use a generic base ISA makes it possible to leverage both experience and infrastructure (e.g., libraries, tool-chains, etc.) which already exist; in essence, it offers a lower barrier to entry. Second, the opportunity to design a bespoke ISA is a rare occurrence. Focusing on a generic base ISA therefore maximises the potential for technology transfer, i.e., there is more chance that an ISE design will actually be deployed and used, and therefore has some impact.

Concept 4. A base ISA is deemed **ISE-aware** if it explicitly facilitates ISEs, or **ISE-oblivious** otherwise.

Of course, ASIPs take the concept of ISE-awareness to an extreme. However, due to the overlap, we simply treat ASIP-based ISEs as any other ISE: doing so renders specific use of related technologies (e.g., Garp [CHW00], Tensilica Xtensa [Gon00], Synopsys Processor Designer [NST10], etc.) out of scope. Note that being ISE-oblivious does not mean one *cannot* extend the base ISA, simply that doing so may be harder due to the lack of facilitation.

Concept 5. An ISE is termed **algorithm-specific** when explicitly designed to support a specific algorithm, or **algorithm-agnostic** otherwise.

Concept 6. A given algorithm-specific ISE is said to offer **single-algorithm** (resp. **multi-algorithm**) support if the specificity involved relates to $n = 1$ (resp. a set of $n > 1$) algorithm(s).

An ISE is, by definition, intended to support domain-specific workloads. In the context of cryptography, however, specialisation *within* the domain, i.e., on a per-algorithm basis, can be reasonable. Consider, for example, an ISE that supports use of an S-box in block ciphers: it might support any block cipher which uses an S-box (algorithm-agnostic), a specific family of block ciphers (algorithm-specific, multi-algorithm), or a specific block cipher (algorithm-specific, single-algorithm). In part, adopting a per-algorithm approach can be justified by the important role that standardisation play in cryptography. This means a relatively small set of (standard) algorithms have strong practical relevance (e.g., NIST FIPS-197 [NIST01] and Federal Information Processing Standard (FIPS)-180 [NIST15] for AES and SHA256, and SP 800-38d [NIST07] for the GCM mode of operation). Selection between them is further guided by their inclusion in standard parameter sets (e.g., the cipher suite `TLS_AES_128_GCM_SHA256` as specified by Transport Layer Security (TLS) [Res18, Section B.4], then realised by OpenSSL[6]).

Note that multi-algorithm support may occur implicitly, if, for example, an algorithm takes advantage of an existing ISE by using a platform-aware design approach. For example, AES New Instructions (AES-NI) can be used beyond AES itself: Bos, Özen, and Stam [BOS11] explore application within block cipher based hash function constructions, Saarinen[7] uses it to implement of the SM4 block cipher, the Grøstl hash function [GKM+11] uses the S-box, and the YAES [BV14] authenticated encryption scheme uses a full round. In a sense, therefore, the definitions relate more to the explicit, by-design support offered by a given ISE.

Concept 7. An ISE is termed **discoverable** if the base ISA includes a mechanism by which (non-)support for it can be programmatically determined.

Concept 8. In the same way as instructions in a given base ISA, those in an ISE are termed **stateful** if they depend on some form of state; otherwise, they are termed **non-stateful** (or **stateless**).

Note that the state here refers to something, e.g., control or status data, *other than* the input operands; this means the action of instructions which are non-stateful are analogous to pure functions. Either way, one could further classify stateful ISEs into cases where 1) new state is *added* by the ISE versus 2) existing state is defined by the ISA and simply *used* by the ISE.

Choquin and Piry [CP20, Page 2] use a definitional framework to classify accelerator technologies. We use it as a starting point to to differentiate ISEs from related technologies, and to classify their implementation. First, the ISE implementation must be integrated with a base micro-architecture; this renders decoupled, memory mapped peripherals and co-processors (e.g., Celator [FPP08], CCproc [TSP09], FastCrypto [SA10],

[6] https://www.openssl.org
[7] https://github.com/mjosaarinen/sm4ni

and Falcon [KLA+19]) out of scope. Second, where such integration is evident, there are (at least) two sub-classes:

Concept 9. A **tightly-integrated ISE** implementation exists "within" the base micro-architecture, and so shares characteristics with the base ISA and micro-architecture:

- direct access to state defined by the base ISA; bandwidth and throughput limited by, e.g., ports on general-purpose register file,
- instructions executed internally by the base micro-architecture, e.g., interleaved with instructions from the base ISA,
- instruction execution has lower-latency, e.g., < 3 cycles.

Concept 10. A **loosely-integrated ISE** implementation exists "alongside" the base micro-architecture, with interaction between the two occurring via an integration interface between the two:

- indirect access to state defined by the base ISA; bandwidth and throughput limited by the integration interface,
- instructions executed externally to the base micro-architecture,
- instruction execution has higher-latency, e.g., ≥ 3 cycles.

Concept 11. The ISE implementation is usually captured in a set of hardware components then integrated into a base micro-architecture. The term used to describe said components varies, but **Application-specific Functional Unit (AFU)** is a common example.

Discussion: ISE versus non-ISE? Beyond classification based on the above, precisely differentiating an ISE from a non-ISE can be difficult and perhaps even counterproductive. One approach would be to demand ISE-based instructions are "similar to" ISA-based instructions, in the sense the former have no special-purpose features, e.g., in terms of implementation constraints, execution semantics, etc., relative to the latter. However, consider two examples:

- Steinegger and Primas [SP21] present an ISE for Ascon, based on RISC-V; [SP21, Figure 1] details implementation of a dedicated AFU for Ascon-p, i.e., the Ascon permutation, which is tightly-integrated with the base micro-architecture. The ISE itself includes one instruction, which essentially supports computation of an entire Ascon round via the AFU. As a result, the AFU assumes the state is read from (resp. written to) a set of 10 general-purpose registers; access to those registers is hard-wired, and must avoid pipeline hazards (which are not resolved by existing forwarding logic).
- Kumar and Paar [KP04] present an ISE for arithmetic in $\mathbb{F}_{2^{163}}$, based on AVR; [KP04, Figures 1+2] detail implementation of a dedicated AFU for said arithmetic, which is tightly-integrated with the base micro-architecture. For example, although the AFU *appears* tightly-integrated with other components, it 1) operates asynchronously: once an operation is initiated, polling is used to test for completion (i.e., during which time other instructions can be executed, although "*the software has to take special care not to call the custom hardware until the multi-cycle operation is completed*"), and 2) has a dedicated memory interface used to load and store 163-bit input and output words.

On one hand, the instructions specified in both designs clearly have special-case features that are (arguably) more aligned with those of a co-processor than they are an ISE. On the other hand, however, provided those features are acceptable, both designs are still viable and may be effective whether deemed an ISE or not: [SP21, Table 1 + Section 4] highlight that a factor of 1.1 area overhead affords a very significant, factor of 50 improvement in execution latency for Ascon, for example. We therefore argue it is more productive to define the term ISE as broadly as possible, e.g., as "*any mechanism accessible via execution of an instruction not included in the base ISA*", by emphasising the *interface*. Doing

so places a stricter demand on appropriate evaluation that includes the *implementation*, allowing informed comparison and selection to match the requirements of a given use-case.

Discussion: RISC versus CISC? It seems reasonable to say that although the terms Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC) can be understood intuitively, there is room for interpretation with respect to their definition. That is, when classifying a design as either RISC or CISC, it is often useful to consider a set of indicative characteristics and/or an underlying design ethos, rather than a terse, rigid definition. For example, Patterson and Séquin [PS81, PS82] offer a set of characteristics stemming from the seminal Berkeley RISC project: these can be summarised as 1) single-cycle instruction execution, 2) uniformity with respect to instruction encoding, 3) limited addressing modes (e.g., memory access only via dedicated load and store instructions), and 4) support for high-level programming languages. Likewise, Waterman [Wat16, Chapter 3] states that the ethos guiding RISC-V "*was to make an ISA suitable for nearly any computing device*".

On one hand, a base ISA plus ISE is (still) an extended ISA, so can clearly be classified using the same approach. On the other hand, however, it is crucial to avoid interpreting "extended" as "more complex" and thus CISC-like. That is, although cryptographic ISE designs exist which *are* complex, complexity is not an *implication* of being domain-specific: designs *also* exist which align with the ISA design ethos, e.g., by retaining RISC-like characteristics.

4 A rigorous, systematic survey of cryptographic ISEs

Analysis by Nahum et al. [NOOS95, Section 4] highlights "*three basic problems*" which they proposed to address via an ISE, namely "*operations on sub-wordsize units, operations on super-wordsize units, and operations of groups other than that of integers*". Regazzoni and Ienne [RI16] highlight ISEs focused on providing increased performance (see [RI16, Section III]) and resilience against side-channel attack (see [RI16, Section IV]).

In this section, we update and expand existing studies such as [BGM09] and [RI16] via a rigorous, systematic survey of cryptographic ISEs. At a high level, we follow Regazzoni and Ienne [RI16] by considering a classification of ISEs which includes major classes for 1) computation-oriented support, 2) storage-oriented support, and 3) security-oriented (or enhancing) support. However, we present the content using a set of fine(r) grained minor classes, since doing so allows us to highlight similar motivation, design approaches, etc. We adopt a uniform structure for each class, by first providing an accessible explanation of the underlying concept, and then, second, précising concrete example ISEs which realise said concept. This (as any) approach naturally has advantages *and* disadvantages. For example, on one hand it offers a best-effort attempt to usefully group similar ISEs together, but, on the other hand, it means a given ISE may span several classes even where highlighted in one.

4.1 Class $\mathcal{C}_1$: special-purpose data-types and computation

4.1.1 Concept

Section 1 cites a focus on general-purpose workloads in existing base ISAs, which typically means support for integer (and perhaps floating-point) data-types plus arithmetic and logic (or ALU-like) operations on them. In contrast, cryptographic workloads make often use of data-types and representations (e.g., stemming from richer Mathematical structures) which lack native support in the base ISA for associated operations. This fact provides perhaps the most compelling motivation, which has led to a broad class of ISEs that support special-purpose alternatives focused on cryptography.

Note that some significant special cases are covered elsewhere, i.e., as dedicated classes. For example, permutation-style operations are covered in Section 4.3, and substitution-style operations (supporting application of an S-box) are covered in Section 4.4.

4.1.2 Examples

(Somewhat) algorithm-specific operations. The "killer application" of ISEs relates to their ability to support domain-specific functionality. Based on this, a wide range of work explores ISEs which support strongly algorithm-specific operations, i.e., operations included in the specification of one algorithm but of limited or even no use beyond that. The bulk of such work relates to algorithms for symmetric cryptography, i.e., block and stream ciphers, and hash functions, where common instances include support for algorithm-specific permutation and/or substitution steps.

A survey of instances would include (at least) AES, as already discussed in Section 1 and [MNP+21, Section 2.3] (with further evaluation in, e.g., [GM23]); Ascon [SP21, AO21]; ChaCha [MPP21]; DES [OE08, OE10, EMOC20]; NTRU [ITAO20]; PRESENT [GP12, VSI+19, TGSD20, EAD+22]; PRINCE [EAD+22]; QARMA [DBB+23]; SEA [GP12]; SNEIK [Saa19]; SHA-1 [BEM+15]; SHA256 [EMOC20]; SHA-3 and/or Keccak [WSWH15, EES+16, ESE+16]; SM4 [Saa20]; XTEA [GP12]; candidates in the NIST SHA-3 process [CBG12a, CBG12b] (e.g., BLAKE, Grøstl, JH, Keccak, and Skein), and candidates in the NIST Light-Weight Cryptography (LWC) process [CGM+22] (e.g., Ascon, Elephant, GIFT-COFB, Grain-128AEADv2, PHOTON-Beetle, Romulus, Sparkle, TinyJAMBU, and Xoodyak). Some instances represent multi-algorithm ISEs: [EKP+13] (supporting at least CLEFIA, SERPENT, PRESENT, and AES); [TGSD20] (supporting at least GIFT, MANTIS, Midori, PRINCE, Skinny, and Twine), and [BLCN20] (supporting at least DES, AES, IDEA, A5-1, SM3, SM4, MD5, and SHA256, although with no clear instruction semantics).

Bit-manipulation. Particularly within the context of algorithms for symmetric cryptography, associated implementations often depend on support for bit-manipulation. One could define operations of this type as directly manipulating bits in some word, rather than the word itself (resp. bits which represent some value rather than the value itself).

Aiming to accelerate, e.g., implementations of the S-boxes for AES and Twofish specifically, instructions in the ISE described by Majzoub and Diab [MD07, Table 2] support somewhat more general-purpose bit-manipulation. For example, `ANDALL` supports a "3-operand AND", e.g., $r \leftarrow x \wedge y \wedge z$, whereas `BWAX` supports an "AND plus XOR reduce", i.e., $t \leftarrow x \wedge y; r \leftarrow t_7 \oplus t_6 \oplus \cdots \oplus t_0$.

More specific, although still general-purpose examples include left- and right-rotation. Although such operations are similar to left- and right-shift, and could be viewed as special-form permutations, they are not uniformly supported across ISAs. RISC-V ISAs, for example, include shift but not rotate instructions. In this and other cases which lack native support, the operation must be synthesised using instructions that exist in the base ISA. Dinu [Din17, Section 5.2.3], for example, carefully explores how to do so on various 8-, 16-, and 32-bit platforms. Even when synthesised instances are as efficient as possible, their use would typically imply some overhead versus a dedicated instruction. As such, many proposed ISEs support for left- and/or right-rotate: RISC-V ISAs, for example, do so via the standard B (bit manipulation) [RVB21, Section 1.3] extension. The designs of Altınay and Örs [AO21] and Burke, McDonald, and Austin [BMA00, Section 5] represent further examples; mirroring functionality offered by the ARM "flexible second operand" mechanism, the latter also consider rotate-plus-XOR variants.

Arithmetic in $\mathbb{F}_q[\mathbf{x}]$ for small q. Given the degree m binary polynomials $x, y \in \mathbb{F}_2[\mathbf{x}]$, computation of the degree $2 \cdot m$ product $r = x \times y$ is often described as *carry-less*

multiplication: the computation is similar to integer multiplication, but with partial products combined via addition in $\mathbb{F}_2[\mathbf{x}]$, i.e., without carries, rather than $\mathbb{Z}$, i.e., with carries. As a result, it is also reasonable to consider carry-less multiply-accumulate, i.e., computation of $r = x \times y + z$.

Use of carry-less multiplication in cryptography and beyond (e.g., certain error correcting codes) means an ISE-supported solution is attractive. Such support typically takes the form of an instruction for carry-less multiplication of m-bit operands (i.e., degree $m - 1$ polynomials), which can then be leveraged in higher-level implementation strategies for a larger m (see, e.g., [HLL17]). Support of this type was first proposed by Nahum et al. [NOOS95, Section 4] to facilitate computation in $\mathbb{F}_{2^m}$ and therefore ECC-based Diffie-Hellman [DH76] key exchange. There is now broad support in deployed base ISAs: examples include x86 (as `pclmulqdq` [Int22b, Pages 4-242–4-244]), ARMv8-A (as `pmull` [Arm20, Section C6.3.189]), and RISC-V (as `clmul` [RVB21, Section 2.6]).

Fiskiran and Lee [FL04] explore ISEs of this type within the context of PLX [LF05]. As well as exploring, e.g., how the multi-word result is dealt with, and if increased data-path width or super-scalar execution is more effective, they make use of the `shuffle` instruction to accelerate squaring in $\mathbb{F}_2[\mathbf{x}]$ (which involves "spacing out" coefficients in a given x by interleaving them with zeros). Bartolini et al. [BBGM04, BBGM08] do more or less likewise, but within the context of Intel XScale, i.e., the ARMv5TE base ISA.

Different trade-offs between latency and hardware cost of carry-less multiplication have been proposed. Tillich and Großschädl [TG04] presented a multiply-step instruction for binary-polynomial multiplication, called `mulgfs`, which is essentially a polynomial variant of the integer multiply-step instruction `mulscc` of the SPARC V8 architecture. The `mulgfs` instruction generates a 32-bit partial product and adds (i.e., `xor`s) it to a 64-bit accumulator held in two registers, one of which is the dedicated multiply/divide register `%y`. By taking advantage of this instruction, the product of two binary polynomials of degree 31 can be computed in 32 clock cycles. Tillich and Großschädl also proposed the `mulgfs2` instruction, which is a variant of `mulgfs` that generates and adds a 33-bit partial product, thereby reducing the execution time to 16 cycles. Puttmann, Shokrollahi and Porrmann [PSP08] analysed and compared the resource-efficiency (i.e., performance in relation to hardware cost) of three different approaches to support binary-polynomial multiplication on a 32-bit RISC processor. The first approach consists of a set of three custom instructions, two that combine a shift with an `xor` and one for address arithmetic. The the second and third approach are based on the `mulgfs` instruction from [TG04] (implemented as an ALU extension) and the carry-less carry-less multiply instruction, respectively, the latter executed on a dedicated binary-polynomial multiplier with a 2-cycle latency. It was concluded that the implementation of `mulgfs` offers the best trade-off between performance improvement and hardware cost.

Focusing specifically on the implementation, Savaš, Tenca, and Koç [STK00] define a dual-field adder as "*a full adder which is capable of performing addition both with carry and without carry*". One could generalise this definition to consider an n-field adder which supports n cases, using *multi*-field adder to refer to support for $n > 1$. Use of such adders to combine partial products can, e.g., avoid the need for separate carry-based and carry-less multiplier data-paths: instead, a single, unified data-path can cater for both cases. This approach is explored by Großschädl and Kamendje, who consider a 2-field ($\mathbb{Z}$ and $\mathbb{F}_2[\mathbf{x}]$) multiply [GK03b] and multiply-accumulate [GK03c] data-path and associated ISE as motivated by Elliptic Curve Cryptography (ECC) over prime and binary fields; Tillich and Großschädl [TG05] apply the same design to field arithmetic in AES. Großschädl and Savaş [GS04] and Liao et al. [LWDZ15] both adopt a broadly similar approach, while Vejda, Page, and Großschädl [VPG07] consider a 3-field ($\mathbb{Z}$, $\mathbb{F}_2[\mathbf{x}]$, and $\mathbb{F}_3[\mathbf{x}]$) alternative as motivated by cryptographic pairings over elliptic curves.

Arithmetic in $\mathbb{F}_q$ for small q. Burke, McDonald, and Austin [BMA00, Section 5] support IDEA via a `modmul` instruction which computes a multiplication modulo $2^{16}+1$, a special-form modulus which affords efficient modular reduction.

Großschädl, Kumar, and Paar [GKP04] present an ISE for MIPS32 which supports computation in Optimal Extension Fields (OEFs), i.e., $\mathbb{F}_{p^k}$ for a pseudo-Mersenne prime $p = 2^n - c$. More specifically, the ISE supports the sub-field $\mathbb{F}_p$: given support in MIPS32 for (32×32)-bit integer multiplication and multiply-accumulate, two additional instructions, namely `maddh` (a variant of `maddu`) and `subc` (a conditional subtraction), allow the efficient reduction modulo p of a 64-bit product held within special-purpose registers `hi` and `lo`. In part due to the form of p, and approach to modular reduction it affords (cf. [Cra]), the ISE can be viewed as general-purpose in the sense it is not specialised for any particular n or c bar the requirement that $p < 2^{32}$.

Alkim et al. [AEL+20] present an ISE for RISC-V, which supports computation in $\mathbb{F}_q$ for small, prime q. Examples include 12-bit $q = 3329$ and 14-bit $q = 12289$ motivated by use in Kyber and NewHope. The ISE interface [AEL+20, Figure 5] includes instructions for addition, subtraction, and multiplication in $\mathbb{F}_q$, plus reduction modulo q. The ISE implementation [AEL+20, Figure 4] is set within the context of the VexRiscv micro-architecture; standard integer arithmetic is performed in the execute stage, with Barrett reduction [Bar86] applied to the result during the memory access and write-back stages.

Kuo, Garcia-Herrero, and Maestro [KGM21] present an ISE for RISC-V which supports computation in $\mathbb{F}_{2^m}$ for small m, e.g., $m = 8$ motivated by use in AES. In their words, it represents an intermediate solution "*between the RISC-V base ISA and the scalar cryptographic K extension*", offering greater efficiency versus the former and greater flexibility versus the latter. The ISE interface [KGM21, Figure 1] consists of three main instructions 1) a `clmul` instruction for carry-less multiplication per the above, and 2) a `ffwidth` instruction which configures the irreducible polynomial (storing it in "*internal registers*"), 3) a `ffred` instruction which performs a reduction modulo said irreducible polynomial. The ISE implementation [KGM21, Figure 2] is set within the context of the SweRV-EL2 micro-architecture, essentially acting as an AFU within the execute stage. Kuo et al. [KGRM23] further extend this idea to that m is up to the machine word size, i.e., 32 in their case (32-bit SweRV-EL2 core). A new instruction `clmulh` is added, which returns the higher half of the result of carry-less multiplication. Plus `clmul`, `ffwidth`, and `ffred`, four instructions altogether form an algorithm-agnostic ISE. Besides, an algorithm-specific ISE variation is also proposed, which contains four instructions for respectively the field addition, multiplication, squaring, and inversion. Each different algorithm supported by this ISE needs a dedicated multiplier/inversion/square, e.g., three of which are required for respectively AES, Reed-Solomon error correcting code, and Classic McEliece post-quantum KEM (as shown in [KGRM23, Section 5]).

Cui and Balasch [CB23] present an ISE for RISC-V which supports computation in $\mathbb{F}_{2^8}$, which is motivated by inefficiency of masked implementation of, e.g., AES; the ISE implementation is set within the context of the VexRiscv micro-architecture. Beyond the application outside unmasked AES, an interesting feature of the ISE is support for both scalar and vectorised, or SWAR-like cases. Specifically, the latter packs four 8-bit field elements into a 32-bit word (rather than one). By using this capability to exploit Instruction Level Parallelism (ILP) which is evident in the masked software implementations, execution latency can be reduced at the cost of higher area.

Computation on matrix-like structures. Elbirt [Elb07] presents an ISE for SPARCv8 which supports arithmetic in $\mathbb{F}_2[\mathbf{x}]/p(\mathbf{x})$. More specifically, it supports the computation of a field multiplication $r(\mathbf{x}) = a(\mathbf{x}) \times c(\mathbf{x}) \pmod{p(\mathbf{x})}$ involving a constant polynomial c and a degree-8 irreducible polynomial p. Expressing the action of c and p as an (8×8)-entry matrix, the field multiplication is expressed as a matrix-vector multiplication over $\mathbb{F}_2$ (with

a then representing the vector). The AES `MixColumns` round function is a motivating use-case: within that context, $p = \mathbf{x}^8 + \mathbf{x}^4 + \mathbf{x}^3 + \mathbf{x} + 1$ and $c \in \{1, \mathbf{x}, \mathbf{x}+1\}$. However, the ISE is more general-purpose in the sense that the matrix stemming from c and p is parameterisable. The (abstract) analysis in [Elb07, Section III.A] considers storage of the matrix in Static RAM (SRAM), with the (concrete) evaluation in [Elb07, Section IV] considers use of an Field Programmable Gate Array (FPGA) block Random Access Memory (RAM). The ISE could therefore be viewed as a more general mechanism for matrix-vector multiplication over $\mathbb{F}_2$.

Targeting "*a generic solution that would apply to any (lightweight) cipher*", Engels et al. [EKP+13] present what they term a Non-linear/Linear Unit (NLU) which allows evaluation of 1) "*operations expressed in binary matrix multiply-and-add form*" in linear mode, or 2) "*operations expressed in their [A]lgebraic [N]ormal [F]orm (ANF)*" in non-linear mode; [EKP+13, Table 1] details a 4 instruction ISE that exposes the NLU to software; although based on an assumed word size of $w = 8$ bits, the underlying principles generalise to other choices of w. `NND` shifts some data x into the configuration register c. `NNL` takes an 8-bit input x, then computes $r_i = c(x_i)$ to form the output r; c is determined by the configuration register, with x_i and r_i represent 4-bit sub-words of x and r (meaning $i \in 0, 1$). `NMU` (resp. `NMA`) takes an 8-bit input x, then computes the output $r = c \times x$ (resp. $r = c \times x + y$). The operand c is determined by the configuration register, with the computation best described as a matrix-vector multiplication over $\mathbb{F}_2$ (making it functionally analogous to the approach of Elbirt [Elb07]). [EKP+13, Figure 1] details the high-level NLU data-path implementation, which highlights a depth-d First-In First-Out (FIFO) buffer allowing accumulation-like use of `NMU` and `NMA`; [EKP+13, Figure 2] and [EKP+13, Figure 3] detail low-level components used to support linear and non-linear modes. The design and implementation is evaluated, and shown effective for a range of block ciphers including PRESENT, CLEFIA, SERPENT, and AES. Although NLU is originally proposed for 8-bit platform (e.g., AVR microcontrollers), Engels et al. state that "*its modular architecture allows it to be used in 16, 32, 64 and even 4-bit CPUs*". Uzuner and Kavun [UK24] validate (part of) the statement by extending the design of NLU on the 32-bit RISC-V, where the new ISE is named NLU-V.

Tehrani et al. present ISEs for RISC-V which support computation of 1) bit-wise matrix-vector multiplication [TGSD20, Section III.C] and 2) nibble-wise matrix-vector multiplication [TGSD20, Section III.D], as used within a suite of light-weight, 64-bit block ciphers. The former is special-purpose, supporting the diffusion layer in PRINCE, whereas the latter is general-purpose, supporting the same layer in, e.g., Midori, Twine, Skinny, and MANTIS. [TGSD20, Figure 3] highlights some central features of the nibble-wise matrix-vector multiplication ISE. Specifically, note that the (compressed) 256-bit matrix operand is specified by the content of 8 special-purpose registers; the 64-bit vector operand is specified by the content of 2 general-purpose registers. [TGSD20, Figure 4] illustrates the associated data-path, which acts to decompress the matrix and compute the result. Two instructions are required, which produce the more- and less-significant 32-bit halves of the result respectively.

Computation on bit-sliced representations. As introduced by Biham [Bih97], bit-slicing is based on 1) a non-standard *representation* of data, and 2) a non-standard *implementation* of functions, which operate on said representations. It essentially describes a given cryptographic primitive as a "software circuit" comprising a sequence of bit-wise instructions (e.g., NOT, AND, and OR). Although not a general-purpose technique, when applicable, use of bit-slicing can offer advantages that include data-oblivious execution latency and hence immunity from cache-based side-channel attacks (see, e.g., [KS09]). In the design of Serpent [BAK98, Page 232], there is a suggestion for accelerating bit-sliced implementations via a "*BITSLICE instruction*" or ISE; the suggestion was later investigated in detail by

Grabher, Großschädl, and Page [GGP08]. In both cases, the idea is to compress a sub-circuit, i.e., the sequence of bit-wise instructions representing an n-input Boolean function, into a Look-Up Table (LUT): the LUT is first configured with a truth table for the function, then accessed to apply said function. Note that similar functionality is offered by, e.g., the x86 `vpternlogd` [Int22b, 5-616–5-618] instruction.

Kiaei et al. [KMD+20] present SKIVA, an extension of the SPARC-V compliant LEON3 core designed to support implementations based on the use of (aggregated) bit-slicing. More specifically, an ISE [KMD+20, Table 1] is used to efficiently support 1) conversion to and from bit-sliced representation, 2) higher-order masking (via an instruction that aligns aggregated slices), and 3) data-redundant computations. Kiaei, Conroy, and Schaumont [KCS23] later port the SKIVA design concept to RISC-V.

Computation in post-quantum constructions. In recent years, Post-Quantum Cryptography (PQC) has been one of the fastest-growing areas in cryptography and, meanwhile, has motivated many related ISE designs. Among various PQC algorithms, the ones based on hard lattice problems have received the most ISE design proposals, partly driven by the fact that at present three out of four standard PQC algorithms selected by NIST are lattice-based [AAC+22], i.e., Kyber, Dilithium, and Falcon. In optimised software implementations of lattice-based cryptosystems, hashing (based on Keccak-f[1600]) and polynomial arithmetic are often identified as the two major bottlenecks. About the latter, the polynomial multiplication is the most performance-critical, and the Number Theoretic Transform (NTT) is a widely-used method to accelerate its computation. To further improve the practical performance of NTT implementation, a number of optimisation methods are studied and cover the different aspects, e.g., various NTT butterflies, modular arithmetic, NTT layer merging, etc. Given NTT is a core and bottleneck operation, it is reasonable that almost all existing ISE proposals for lattice-based cryptosystems involve the custom instructions for NTT and/or the underlying arithmetic of NTT.

Targeting three lattice-based algorithms NewHope, Kyber, and Saber, Fritzmann, Sigl, and Sepúlveda [FSS20b] develop and integrate a set of tightly-coupled hardware accelerators into a 32-bit RISC-V CV32E40P core, and provide an associated ISE. The accelerators cover all the performance-bottleneck subroutines, e.g., NTT, modular arithmetic, Keccak-f[1600] permutation, and binomial sampling. Inside the NTT accelerator [FSS20b, Figure 2], several units are designed and target the different internal operations of NTT: 1) a modular arithmetic unit to support NTT butterfly and packed modular arithmetic operations; 2) a twiddle update unit to accelerate the computation of twiddle factors; 3) an address unit to accelerate NTT layer merging.

Nannipieri et al. [NDZ+21] present an ISE to accelerate Kyber and Dilithium on 64-bit RISC-V, which focuses on particularly the NTT in both algorithms. For each algorithm, there are five custom instructions for respectively the modular multiplication, modular reduction, twiddle factor setting, Cooley-Tukey (CT) butterfly [CT65] (used in forward-NTT), and Gentleman-Sande (GS) butterfly [GS66] (used in inverse-NTT). In the ISE implementation, a CAV6 core is used as the base core, and two PQ ALUs (for respectively Kyber and Dilithium) are added to support the custom instructions. Given this strategy uses two distinct PQ ALUs, Miteloudi et al. [MBB+23] explore the design of a *unified* PQ ALU for the same two algorithms to further reduce the hardware overhead. In detail, in the multiplication unit [MBB+23, Figure 5], the integer multiplication circuit is shared, whereas two reduction circuits are separated for two different moduli. Additionally, an associated RV32 ISE (named PQVALUE) is presented, and for each algorithm there are also five instructions: 1) three for modular operations (i.e., addition, subtraction, and multiplication) and 2) two for NTT butterflies (i.e., CT and GS). Instead of packing two polynomial coefficients in one register, each of their single-cycle butterfly instructions accepts three operands and outputs two results, which 1) overwrites the source registers

and 2) is only possible when the register file has three read ports and two write ports (e.g., their experimental platform PULPino SoC meets this condition). If the required number of ports is not available, one can instead use three modular arithmetic instructions to compute a NTT butterfly (see detail in [MBB+23, Figure 8]).

Li et al. [LQYW24] propose an ISE for 32-bit RISC-V to accelerate the GS butterfly, which is used in the inverse-NTT of Kyber. Due to the speed-up by ISE, the GS butterfly becomes a more efficient option than CT butterfly and therefore is used for also the forward-NTT in their case. The dedicated instruction for GS butterfly is called `butterfly` (see, e.g., [LQYW24, Figure 2]), which accepts two 16-bit polynomial coefficients (packed in one 32-bit register) and a twiddle factor as inputs, and outputs two 16-bit results (again packed). Notably, the associated coefficient reduction utilises an optimised version of the k^2-reduction [LN16], which, per their experiments, has a lower area overhead than other alternatives such as Montgomery and Barrett reduction. Furthermore, with the same base core (Hummingbrid E203), Li et al. [LTQ+24] present an ISE also for Dilithium. Based on also the k^2-reduction but using a different set of parameters, three custom instructions (for respectively modular addition, subtraction, multiplication) are designed to facilitate the CT/GS butterfly. Another major custom instruction is named `keccak` that performs a single round permutation of Keccak-f[1600], and it relies on a dedicated Keccak accelerator and an extra 1600-bit *inner register file* to store the whole state.

Based on a 32-bit RISC-V Ibex core, Geweher, Luza, and Moraes [GLM24] propose an ISE named Xkyber to assist the Kyber polynomial operations. Since they follow the wider RISC-V design principles, in all their custom instructions, the Kyber modulus p is not input as an source operand so that they can keep "$2\ rs + 1\ rd$" instruction encoding format. In addition, to assist the Barret reduction which is used in coefficient multiplication instruction, a special 5039 constant multiplier is developed, which is then further reused by a coefficient compression instruction. Remaining four instructions are designed to aid the coefficient addition and subtraction, and two CBD samplings, respectively.

In addition to NTT-based cases, Fritzmann, Sigl, and Sepúlveda[FSS20a] design and implement several accelerators and an associated ISE for LAC, in whose polynomial multiplications one polynomial is general but the other is ternary (i.e., coefficients are in $\{-1, 0, 1\}$). The coefficient-wise multiplication thus can be simplified to addition or subtraction, and based on this a dedicated ternary multiplier is presented [FSS20a, Figure 2]. The multiplier accepts the polynomials with a degree up to 255, decided by the trade-off between area and performance, which means before using it the splitting needs to be performed for LAC polynomials. Each of the remaining bottleneck subroutines, namely polynomial generation, BCH decoder, and modular reduction, is also assisted with an accelerator, yielding totally four custom instructions (one instruction per accelerator).

Motivated by the fact that the *hybrid* approach (i.e., PQC plus classical cryptography) is becoming a standard method to integrate PQC into real-world applications, Oberhansl et al. [OFP+24] explore the design of a uniform accelerator that can assist both pre-quantum X25519 and post-quantum Saber. The accelerator is based on the schoolbook multiplication, thus named Extended Schoolbook Multiplier (XSMUL), and is integrated into a 32-bit CV32E40P RISC-V core. The XSMUL works with 256-bit input/output (either 16 16-bit polynomial coefficients or a 256-bit integer), relying on a large set of fixed registers to store them. An ISE related to XSMUL is presented, e.g., the `pq.xsmul` instruction can choose different operations to be performed by XSMUL such as polynomial multiplication, $\mathbb{F}_{2^{255}-19}$ multiplication, etc.

Apart from the above, we note that some ISE designs for post-quantum cryptography are discussed elsewhere in this paper, e.g., [AEL+20, LMP22, KGRM23, YSZ+24, AOP+24].

4.2 Class $\mathcal{C}_2$: increased input and/or output bandwidth

4.2.1 Concept

Consider a base ISA with an n-address instruction format: it allows instructions to specify upto n general-purpose register addresses, such that $n = n_s + n_d$ for n_s sources and n_d destinations (or targets). Although the case where $n = 2 + 1 = 3$ is common, support for n-address instruction formats more generally, e.g., cases where $n > 3$, involves a non-trivial trade-off between a variety of factors. For example, increasing n can increase data-flow bandwidth which, in turn, facilitates *richer* forms of computation as a result of access to more data. Crucially, however, increasing n can also 1) increase encoding pressure (in the sense that more bits are required to encode the associated register addresses, and hence instruction), and 2) increase data-flow complexity (in the sense that either n register file ports or multi-cycle execution must be supported). In order to realise the clear advantage therefore, any challenges related to the associated *dis*advantages must be addressed.

Lee, Yang, and Shi [LYS04] observe that cryptographic workloads can be well positioned to take advantage of this potential, because they are naturally specified in terms of Multi-word Operands, Multi-word Results (MOMR) operations. When those operations are mapped more directly onto instructions, their execution will typically be more efficient than otherwise.

4.2.2 Examples

If n-address instruction formats are attractive, either per the above or in general, an approach to support them must exist in the base ISA or be specified as part of an ISE. On one hand, some candidate approaches could be viewed as tackling the challenge indirectly. For example, reducing encoding pressure could indirectly render an n-address instruction format viable. One could imagine 1) reducing the number, or restrict access to a subset of registers (cf. ARM Thumb or RV32E), thereby reducing the number of bits required to encode each register address, or 2) using a variable-length encoding (cf. x86), thereby increasing the effective number of bits available for a given format. On the other hand, other candidate approaches could be viewed as tackling the challenge directly. For example:

1. One could make all register addresses *explicit*. For example, the XS1 `lmul` [May09, Page 146] instruction uses $n = 4 + 2 = 6$, i.e., an 6-address instruction format [May09, Page 246].
2. One could make some register addresses *implicit*. For example, the x86 `mul` [Int22b, Page 4-146–4-147] instruction suggests $n = 1 + 0 = 1$, due to use of `eax` as an implicit source and both `eax` and `edx` as implicit destinations.

   Steinegger and Primas [SP21] present an ISE for RISC-V, which supports Ascon by adopting this approach. The ISE implementation [SP21, Figure 1] is set within the context of the RI5CY micro-architecture; an Ascon-specific AFU operates in the decode stage, implicitly accessing 10 out of the 32 general-purpose registers.
3. One could make some register addresses *overloaded*. For example, the ARMv8-M `umaal` [Arm22, Section C2.4.267] instruction encoding suggests $n = 2 + 2 = 4$, even though the semantics show that 4 sources are used: the two destinations have an overloaded role as additional sources, which characterises the instruction as destructive (wrt. the content in those sources) as a result.
4. One could make some register addresses *derived*. For example, Lee and Choi [LC08] propose the Register File Extension for Multi-word and Long-word Operation (RFEMLO) where a group of $n = 2^l$ contiguous register addresses are derived from a single register address i plus a level l, i.e., $(i, l) \mapsto \langle i, i+1, i+2, \ldots, i+2^l-1 \rangle$. Note that any register with special-case semantics may plausibly complicate such an approach. For example, various RISC-like ISAs fix the 0-th general-purpose register to 0, e.g., to

support synthesis of pseudo-instructions. It may be difficult, therefore, to (selectively) include or exclude that register in or from a group as need be.

Gao et al. [GGM+21] present an ISE for RISC-V, which supports masked software implementations by adopting this approach. The ISE implementation [GGM+21, Figure 3] is set within the context of the SCARV micro-architecture; a masking-specific AFU operates in the execute stage, deriving $n = 4 + 2 = 6$ addresses using a pair-based scheme [GGM+21, Page 7], i.e., an instance of RFEMLO where $l = 1$.

5. One could make some register addresses *distributed* across multiple instructions. For example, the RISC-V `mul` and `mulh` instructions [RV19, Section 7.1] use $n = 2 + 1 = 3$. However, the 2 destinations that capture a $(2 \cdot w)$-bit product are distributed across two instructions: by reusing the same 2 sources, `mul` and `mulh` compute and then write-back the w LSBs and MSBs of said product espectively.

   Many instances of ISEs for carry-less multiplication adopt this approach. For example, Fiskiran and Lee [FL04] present an ISE is set within the context of the PLX micro-architecture; their case-2 variant defines two instructions, namely `bfmul.lo` and `bfmul.hi`, which respectively write the LSBs and MSBs of a computed product into a general-purpose register.

4.3 Class $\mathcal{C}_3$: mismatched word size and data-path widths

4.3.1 Concept

Given a base ISA which specifies a w-bit word size, the first two problems identified by Nahum et al. [NOOS95, Section 4] relate to the use of operations whose natural word size is either less-than (i.e., "*sub-wordsize*") or greater-than (i.e., "*super-wordsize*") w. Put another way, such operations would be more naturally processed by a data-path, and so natural word size of some $w' \neq w$ bits. For example, the state (or block size) of a block cipher design might be 64-bit (e.g., PRESENT [BKL+07]) or 128-bit (e.g., AES [NIST01]) and so $w' > w = 32$; the computation performed on that state might be 1-bit (e.g., DES [NIST99]), 4-bit (e.g., PRESENT [BKL+07]), 8-bit (e.g., AES [NIST01]), or 16-bit (e.g., IDEA [LM90]) oriented and so $w' < w = 32$. In the case of AES, for example, the choice is explicitly rationalised: although still a compromise, specifying 8-bit oriented computation supports versatility [DR02, Section 5.1.4], i.e., "*the ability to be implemented efficiently on different platforms*". In other cases (e.g., Speck [BSS+13] and Simon [BSS+13], RC5 [Riv94] and RC6 [RRSY98]), this approach is taken further by allowing specification of the natural word size as a parameter.

In the $w' > w$ case, it is common to represent a w'-bit operand using a sequence of w-bit words; associated challenges include efficiently dealing with any interaction, e.g., carries, between those words. In the $w' < w$ case, it is common to harness existing support for Single Instruction Multiple Data (SIMD) or SIMD Within A Register (SWAR) (i.e., packed) operations. However, non-orthogonality in that support is a common challenge: in early work of this type Acar [Aca97, Section 5.4.1] noted, for example, that MMX "*lacks certain instructions such as* 16*-bit and* 32*-bit unsigned multiply and multiply-add*" which made harnessing it in cryptographic use-cases more difficult. Arguably this issue has improved over time, but, even now, support for $w' < 8$ is not common.

4.3.2 Examples

Permutation for $w' = 1$ bit sub-words. Particularly within the context of algorithms for symmetric cryptography, permutation of bits within a larger word is a common operation. In hardware, this operation has essentially no (computational) overhead in the sense that one simply "rewires" the inputs to form the outputs. In software, however, said overhead can

be significant. Standard implementations[8] of Data Encryption Standard (DES) commonly exhibit two approaches to addressing the challenge which results. First, they capitalise on special structure some permutations, e.g., IP and FP (the initial and final permutations); in modern alternatives to DES, such special structure might in fact exist by-design, so as to actively facilitate efficient implementation. Both Warren [War12, Chapter 7] and Knuth [Knu11, Section 7.1.3] survey a range of fairly generic techniques of a similar type. Second, they effectively pre-compute the action of some permutations, e.g., P and E (the P-box and expansion permutations), by "folding" them into existing S-box look-up tables.

Beyond such techniques, however, bit permutation is an obvious target for support ISEs which fall into one of two high-level classes. The first class are algorithm-specific ISEs in the sense they support one specific permutation. For example, Tehrani et al.[TGSD20, Section III.B] define ISEs intended to support the specific bit permutations used within the diffusion layer in PRESENT and GIFT. The second class are algorithm-agnostic ISEs in the sense they support any permutation: see, e.g., [SL00, YL00, ML01, LSY01, SL02, SYL03, LSY+04, Shi04, LYS05, HYL08, HL08, SYL08, Hil08, KKRM13, KAMS19]. Often, the instructions in such an ISE support a somewhat special-purpose permutation building-block which can be iterated in a structured manner (e.g., via a permutation network, such as omega-flip) to yield an arbitrary, general-purpose permutation; the building-block permutation is sometimes configurable, as in the case of Burke, McDonald, and Austin [BMA00, Section 5] which allow (partial) permutation of one register based on specification held in another.

Permutation for $w' > 1$ bit sub-words. The x86 `pshufb` [Int22b, Page 4-416–4-419] and RISC-V Zbkx extension [RIS24, Section 34.3.3.] (including `xperm8` and `xperm4`) adopt a similar design, which performs an in-place permutation of bytes (i.e., `pshufb` and `xperm8`) or nibbles (i.e., `xperm4`) in an *operand register* according to the other *control register*. This feature is beneficial for parallel implementations of (small) S-boxes, e.g., one can use one or more control registers to form a complete look-up table of an S-box. Cheng et al. [CGM+22, Section 3.3] present a parallel implementation of the 4-bit S-box of Spongent-π[160] [BKL+13], which is realised by `xperm4` on RV32. Eight simultaneous S-box look-ups take two `xperm4` and two `xor` instructions, plus three extra registers to hold the constants (i.e., S-box look-up tables and a mask), whose cost is much less than a pure software counterpart.

May, Penna, and Clark [MPC00, Section 3.1] are among the first[9] to specify a permutation-like operation termed `SWAPMOVE`. The most general form of `SWAPMOVE` could be viewed as an inter-word permutation, in the sense that some bits in an w-bit operand x are swapped with some bits in another w-bit operand y with further operands n and m controlling *which* bits. Although `SWAPMOVE` and variants thereof (e.g., as an intra-word permutation where $x = y$) can be applied in many use-cases, perhaps the most common is within fix-sliced [ANP20, AP20b] implementations: Adomnicai and Peyrin [AP20a] explore this fact in detail. Cheng et al. [CGM+22] present a special-purpose ISE design for `SWAPMOVE`, which employs a hardcoded set of n and m to obey the standard RISC-V instruction encoding format. Since only one destination register address can be used, their ISE includes 1) 1-operand instructions involving only x, i.e., a single instruction performs an *intra-word* `SWAPMOVE` permutation, and 2) 2-operand instructions, i.e., a pair of instructions together performs an *inter-word* `SWAPMOVE` permutation. In addition, they discuss a potential, more general-purpose ISE design for `SWAPMOVE` is possible, which should

[8] For example, the implementation of Outerbridge is widely available online and reproduced in print by Schneier [Sch96, Part V]; see Osvik [Osv03, Chapter 4]. for an accessible overview of the techniques involved.

[9] Their goal is efficient software implementation of permutations, such as those used by DES; they cite some prior art, e.g., noting "*[t]his technique is utilised in versions of DES available from the Internet (for example Eric Young's libdes)*".

use a *somewhat* general-purpose set of n and m.

Arithmetic in $\mathbb{Z}$, $\mathbb{Z}_N$, and $\mathbb{F}_p$. Most software libraries for Multi-Precision Integer (MPI) arithmetic represent the operands (which are integers of a length of n bits) via arrays of w-bit *digits* or *limbs*, where w equals the word-size of the underlying processor (called "full-radix" representation) or is a few bits below the word-size ("reduced-radix" representation). In general, a radix-2^w representation splits an n-bit integer A into $l = \lceil n/w \rceil$ digits/limbs so that $A = \sum_{i=0}^{l-1} a_i 2^{iw}$ where $0 \leq a_i < 2^w$ for $0 \leq i < l$. The most common word-sizes of general-purpose processors (resp. microcontrollers) are 8, 16, 32 and 64 bits. Algorithms for MPI arithmetic operate on the digits or limbs of such arrays by executing w-bit instructions supported by the processor, e.g., w-bit addition or $(w \times w)$-bit multiplication. The most basic techniques for MPI multiplication, namely the so-called *operand-scanning* method [MOV96] and *product-scanning* method [Com90], have complexity $\mathcal{O}(l^2)$ for l-digit operands, i.e., the number of $(w \times w)$-bit multiply instructions increases with the square of the operand length. Besides the length of the operands, the execution time of an MPI multiplication is also influenced by certain features of the Instruction Set Architecture (ISA), e.g., the actual semantics of the $(w \times w)$-bit multiplication and some other instructions, and the micro-architecture, e.g., the latency of the $(w \times w)$-bit multiply instruction.

The $(w \times w)$-bit multiply instruction is "special" in the sense that its result (i.e., the product of two w-bit digits or limbs) has a length of $2w$ bits and does, hence, not fit into a single register. This contrasts with most other arithmetic/logical instructions, such as instructions for Boolean operations like AND, OR, and XOR, which follow (at least in the case of RISC ISAs) the common 3-register instruction format, i.e., two source registers and one destination register. Over time, computer architects have come up with a broad and highly diverse range of approaches on how to deal with the "widening" aspect of the multiply instruction. For example, the multiply instructions of the 8-bit AVR and the MIPS32 architecture allows one to specify only the two source registers; the product is placed in either two fixed general-purpose registers (e.g., `r0` and `r1` in AVR) or special registers (e.g., `hi` and `lo` in MIPS32). In the latter case, the ISA provides instructions (e.g., `mvlo`, `mvhi`) that allow one to transfer the content of these two special registers to a general-purpose register. The SPARC V8 architecture implements a mix of these two approaches since its multiply instruction puts the lower half of the $2w$-bit product into a general-purpose register and upper half in a special register named `%y`. There are also some ISAs that offer separate multiply instructions for the lower and upper half of the product; well-known examples are the PowerPC architecture and RISC-V [RV19]. The ARM ISA provides the most flexible support for "widening" multiplication and includes also some other instructions that are useful for MPI arithmetic. For example, ARMv7-M contains the `umull` instruction for integer multiplication, which allows a programmer to specify two general-purpose registers for storing the 64-bit product, see [Arm21, Section A4.4.3]. However, this flexibility comes at a price, namely when single-cycle execution is desired (as is the case for Cortex-M4 microcontrollers), a total of four buses between the register file and the multiplier are needed and the register file has to provide a second write port (i.e., two read ports and two write ports altogether).

One of the first proposals of ISEs to speed up multi-precision arithmetic on a general-purpose RISC architecture was presented in [Gro02]. The target architecture was MIPS32, which, as explained above, uses two architecturally visible special-purpose registers (`hi` and `lo`) to store the 64-bit result of a multiplication. In order to speed up the inner loop of the operand-scanning method for multi-precision multiplication, the instruction `macil` is introduced, which executes an operation of the form $(u, v) \leftarrow a \times b + p + u$, i.e., two 32-bit words are multiplied and another two 32-bit words are added to the product. The result of this operation is at must 64 bits long and fits into two 32-bit registers. Normally,

an instruction for this operation would require four read-accesses and two write accesses to the register file, but by re-using `hi` and `lo` as local storage elements, only two read accesses and one write access are necessary. Consequently, the `macil` instruction adheres to the standard three-register instruction format of the MIPS32 architecture. Thanks to this instruction, the number of instructions executed in the inner loop of the operand scanning method can be reduced from 12 to only six.

Reference [Gro03] builds on [Gro02] and extends the ISE design to also support multi-precision squaring as well as Montgomery squaring. The squaring operation can be optimized so that is has to execute only about half of the word-level (i.e., single-precision) multiply instructions compared to a multiplication. In order to facilitate squaring on MIPS32, a special instruction called MACSQ (MAC for SQuaring) for computations of the form $(u, v) \leftarrow a \times b + p + u$ is introduced. Similar to `macil`, all operands have a length of 32 bits, but the result exceeds 64 bits since the product $a \times b$ is shifted left by one bit before the two additions are performed. Due to this left-shift, the result of this operation can have a length of up to 65 bits. In order to accommodate the extra bit, the result-accumulation register `hi` is extended to 33 bits and also the multiplier is modified. Thanks to these modifications, the inner loop of the squaring operation can be performed with only six instructions. A full 1024-bit Montgomery squaring can be executed in about 10,500 clock cycles on an extended MSP32 core, which is about 2,500 cycles faster than a 1024-bit Montgomery multiplication.

While the above two papers are based on the operand-scanning method for multiplication and squaring (resp., Montgomery multiplication and Montgomery squaring), the work described in [GK03a] uses the product-scanning method as foundation of an ISE design. The operation performed in the inner loop of product-scanning multiplication has the form $(t, u, v) \leftarrow (t, u, v)a \times b$, i.e., two words are multiplied and the 64-bit product is added to a cumulative sum (t, u, v). This is a classical MAC operation, well-known from, e.g., digital signal processing. When several such 64-bit products are added up, the sum exceeds 64 bits and can, therefore, not stored in two 32-bit registers anymore, which explains the purpose of t. In order to speed up this operation, [GK03a] proposes to (i) a modification of the native MIPS32 instruction `maddu`, (ii) and extension of the `hi` register to accommodate up to 40 bits, and (iii) and instruction `sha` to shift the cumulative sum in `hi` and `lo` 32 bits to the right, with the 32 least significant bit written to a general-purpose register. Furthermore, an instruction `m2addu` for squaring is proposed; it simply doubles the 64-bit product before it is added to the cumulative sum. The main advantage of these ISE for prudct-scanning multiplication is that the overall execution time is that a single-cycle multiplier is not needed to reach peak performance, i.e., the inner-loop operation can, to some extent, "hide" the latency of the multiplier.

Besides prime fields, elliptic curve cryptosystems can also be constructed over a binary extension field. The paper [GS04] presented a set of five custom instructions to accelerate arithmetic operations in both types of field. Multiplication and squaring are based on the product scanning technique and MIPS32 served again as based architecture. However, the paper also considered optimizing the modular reduction operation for a 192-bit generalized-Merseenne prime and an irreducible polynomial of degree 191, but standardized by the NIST. It was demonstrated that the proposed instructions can be easily integrated into MIPS32 and require only little extra hardware. The customn instructions enabled an extended MIPS32 core to perform an elliptic curve scalar multiplication over a 192-bit prime field in 36 msec, assuming a clock speed of 33 MHz. An elliptic curve scalar multiplication over the binary field GF(2^{191}) required only 21 msec, which is approximately six times faster than a software implementation on a standard MIPS32 processor.

Using SPARV V8 as base architecture, reference [GTS07] analyzed the performance of instruction set extensions for long integer arithmetic. The authors focussed more on the software side rather than instruction-set designor integration and discussed various

implementation options and optimization opportunities for both modular multiplication and exponentiation. In particular, they introduced a partial loop unrolling (PLU) technique for modular multiplication which allowed them to achieve large performance gains at the cost of a moderate increase in code size, while maintaining the full flexibility of a "rolled-loop" implementation. In addition, they studied window methods for modular exponentiation and analyzed their impact on performance and memory requirements. Experimental results, obtained with an FPGA prototype of the LEON-2 SPARC V8 core, showed that a full 1024-bit modular exponentiation can be performed in about $12.5 \cdot 10^6$ clock cycles.

Cheng et al. [CFG+24] presented two small sets of custom instructions for 64-bit RISC-V to accelerate multi-precision integer arithmetic, one for full-radix representation and the other for a reduced number-representation radix. Both adopt the R4 instruction encoding format for the custom integer multiply-add instructions, accepting the two operands to be multiplied and a third operand to be added to the product as inputs. The reduced-radix multiply-add instructions `madd57lu` and `madd57hu` operate on a fixed radix of 2^{57} (indicated by the instruction names), which was chosen according to the operand length of 511 bits of the target application. Both multiply two operands held in registers and add either the lower part (i.e., the 57 LSBs) or the upper part of the product to a third operand. The upper part is 64 bits long to avoid the so-called multiplier saturation problem when the operands are not fully reduced (i.e., exceed 57 bits).

SIMD-like operations. The SIMD extension is shown to be beneficial for accelerating various cryptographic constructions, e.g., [Cla97, BS12, MM12, GM13]. Rawat and Schaumont [RS16] propose a set of six custom SIMD instructions, targeting ARMv7 NEON unit, for accelerating Keccak-p permutation. The design is mainly integrating several bit-manipulations (e.g., logical, rotation, lane-wise permutation) into one instruction. Using a different element length in the NEON registers, their ISE can speed up the permutation that uses a state of 1600, 800, 400, or 200 bits. Li, Mentens, and Picek [LMP23] target also Keccak-p permutation (i.e., Keccak-f[1600], more specifically) and propose a SIMD ISE designed for the RISC-V vector unit. Their ISE relies on the large vector lengths, e.g., 1024 bits on the 64-bit RISC-V, and consists of the instructions to 1) efficiently permute and/or blend the elements across vectors, 2) accelerate various bitwise computations, and 3) speed up some specific steps in the permutation such as π step and ι step.

The designs of SIMD ISE have been extended to also PQC area, which usually introduce some mechanisms to speed up the associated (slow) data transfer (e.g., between the SIMD registers and the memory). Li, Mentens, and Picek [LMP22] present a RISC-V vector ISE to accelerate the polynomial arithmetic of Kyber, where they introduced three *register pools* to the vector unit for efficiently accessing the intermediate values (i.e., polynomial coefficients, coefficient indices, and twiddle factors). The custom vector instructions include mainly 1) polynomial load/store (data transfer between the RAM and a register pool) and read/write (data transfer between a register pool and the vector register file), 2) multiplication configuration, and 3) finite field computation (with a hardcoded Kyber modulus). Ye et al. [YSZ+24] extend a 32-bit RISC-V *scalar* core to make it support SIMD computing, and then based on this core present a SIMD ISE to aid the computation of Kyber and Dilithium. To enable the SIMD functionality, the modifications on the ISA side include: adding the SIMD register files and the associated PALU (Proposed ALU, for parallel operations), extending LSU to increase the datapath width, and using dual-issue design. On the ISE side, it consists of three categories of custom instructions (see [YSZ+24, Table 10]), namely SIMD arithmetic, SIMD Keccak-related, and SIMD load/store. In particular, the SIMD arithmetic instructions contain some special-purpose instructions to assist, e.g., modular operations (relying on a special register file named FIX to store the modulus q) and the centered binomial distribution sampling. Abdulrahman et al. [AOP+24] propose an ISE to assist the computation of Kyber and Dilithium for

OpenTitan Big Number (OTBN) ISA[10]. In OTBN ISA, there are 32 256-bit wide data registers (`w0` to `w31`) and a special modulus register `MOD` that holds the used modulus. Given the Keccak permutation can be accelerated by an existing KMAC core in OpenTitan, the ISE contains five SIMD instructions that are all related to polynomial arithmetic: three are respectively for coefficient addition, subtraction, and multiplication, one for data interleaving between two registers, and one for bit shifting.

4.4 Class $\mathcal{C}_4$: special-purpose state and access mechanisms

4.4.1 Concept

This general class of ISE covers a range of more specific cases, namely 1) support for special-purpose access to existing general-purpose state, and 2) addition of and support for access to special-purpose state. Considering the execution of some instruction defined as part of an ISE, the latter might be considered to serve various different purposes.

- Type-1 special-purpose state is introduced because it *facilitates* instruction execution. For example, the MIPS32 `mul` [MIPS01b, Page 169] instruction has the following semantics

  $$\mathsf{GPR}[\texttt{rd}] \leftarrow \mathrm{LSB}_{32}(\mathsf{GPR}[\texttt{rs}] \times \mathsf{GPR}[\texttt{rt}]).$$

  In contrast, the `mult` [MIPS01b, Page 169] instruction has the following semantics

  $$(\mathsf{GPR}[\texttt{lo}], \mathsf{GPR}[\texttt{hi}]) \leftarrow \mathsf{GPR}[\texttt{rs}] \times \mathsf{GPR}[\texttt{rt}].$$

  Put simply, the former explicitly uses *one* 32-bit general-purpose register to capture the least-significant 32 bits of the full product; the latter implicitly uses *two* 32-bit special-purpose registers `hi` and `lo` to capture the full product (`hi` captures the more- and `lo` the less-significant 32-bits respectivly). As such, the latter is facilitated by addition of these registers plus a suite of assoicated instructions, i.e. `mfhi` [MIPS01b, Page 146], `mflo` [MIPS01b, Page 147], `mthi` [MIPS01b, Page 166], and `mtlo` [MIPS01b, Page 167], which support access.
- Type-2 special-purpose state is introduced because it *configures* instruction execution. That is, one can distinguish between 1) dynamic state that is operated on by the instruction, or 2) static state that configures (or controls) the instruction, but is not operated on per se. The latter case could be further characterised as begin either semi-static, e.g., held within a register, memory, or some special-purpose, or fully-static, e.g., encoded as an immediate within the instruction. Within the context of cryptographic ISEs, a common use-case for such state is to provide a compromise between general-purpose and special-purpose behaviour: the idea is to use the state as an additional input which parameterises an otherwise special-purpose operation, thus rendering it more broadly applicable.

4.4.2 Examples

Type-1 state. Many ISEs which support multiplication use implicitly addressed type-1 state to cope with the challenge of increased output bandwidth (per Section 4.2); as alluded to above, instances relating to both integer multiplication (cf. Section 4.3) and carry-less multiplication (cf. Section 4.1) can be readily identified. For example, Fiskiran and Lee [FL04, Section 3.3] explore variants of the `bfmul` instruction which differ with respect to how the $(2 \cdot m)$-bit product of a carry-less multiplication is written-back to the general-purpose register file. What they refer to as case-1 introduces 2 special-purpose registers, `RH` and `RL`, which capture the most-significant (resp. least-significant) half of said product.

[10] https://opentitan.org/book/hw/ip/otbn/doc/isa.html

Type-2 state. There are a wide variety of ISE designs which involve some form of type-2 state, which could be further separated into examples where said state is tightly-integrated (or internal to the micro-architecture, e.g., using a register) or loosely-integrated (or external to the micro-architecture, e.g., using memory). Although various designs of this are also presented in other Section, a set of exemplars would include the following:

- Grabher, Großschädl, and Page [GGP08] propose an ISE that supports *configurable* 4-input, 2-output Boolean functions, vs. *fixed* 2-input, 1-output alternatives such as NOT, AND, OR, and XOR, e.g., for bit-sliced [Bih97] block cipher implementation; semi-static configuration for the function is held in a special-purpose register.
- Tehrani et al. propose an ISE that supports parallel application of a 4-bit S-box [TGS+19, Section III.A], i.e., a mapping $\{0,1\}^4 \rightarrow \{0,1\}^4$, to each nibble in a 32-bit word, e.g., for block cipher implementation; semi-static configuration for the S-box is held in "*three CSR registers*". Likewise, they propose an ISE that supports nibble-wise matrix-vector multiplication [TGS+19, Section III.D], the (compressed) matrix operand is held in "*eight 32-bit CSR registers*".
- In their ISE design which supports arithmetic in $\mathbb{F}_q$, Alkim et al. [AEL+20] explore variants where q and associated parameters are either 1) static, or 2) dynamic: the latter case is supported by state stored in "*internal registers*" and accessed by special-purpose instructions (namely `ffset` and `ffget`).
- In their ISE design which supports arithmetic in $\mathbb{F}_{2^m}$, Kuo, Garcia-Herrero, and Maestro [KGM21, Figure 1] use "*internal registers*" to store an irreducible polynomial required to define the finite field: this state is fixed via the `ffwidth` instruction, and used via the `ffred` instruction.

Specialist look-up addressing. Fiskiran and Lee [FL01] observe that availability of appropriate addressing modes, i.e., the ability to specify an effective address then used to access data, have an important role in the efficiency of cryptographic implementations. In [FL01, Section 3] they assess various addressing modes, including one ISE-like "*[load] instruction which combines index extraction, index scaling and [the] memory access*". Burke, McDonald, and Austin [BMA00, Figures 8+9] introduce a similar ISE via their `sbox` instruction.

The RISC-V compliant CV32E40P (formerly RI5CY) core supports[11] an optional, general-purpose ISE for auto-increment and register-based (versus immediate-based, per the ISA) index load and store instructions; this has various use-cases in cryptographic workloads.

Specialist look-up resourcing. Fiskiran and Lee [FL05b, FL05a] introduce an ISE to support efficient look-up table access, which is later improved by Lee and Chen [LC10]. The idea is that a so-called Parallel Table Lookup (PTLU) module houses 8 on-chip, 256-entry look-up tables with 32-bit entries; access to said tables (plus optional "in memory" computation, limited to logical operations, at the output port) can occur in parallel and with data-oblivious execution latency, with the effective addresses automatically extracted from sub-words of a register. Fiskiran and Lee [FL05b, Section 5.1] use the PTLU to store pre-computed S-boxes and thereby accelerate AES, with basically the same approach applied to the Whirlpool hash function by Hilewitz, Yin, and Lee [HYL08].

Altınay and Örs [AO21] introduce an ISE with similar motivation, intended to support computation of the Ascon S-box. Their instruction for doing so is CISC-like, in the sense it operates on data resident in memory: using an input register address rs_1, it loads five 32-bit inputs $x_i \leftarrow \mathsf{MEM}[\mathsf{GPR}[rs_1] + 4 \cdot i]^4$, applies the S-box to produce outputs r_i from the inputs x_i, then stores five 32-bit outputs $\mathsf{MEM}[\mathsf{GPR}[rs_1] + 4 \cdot i]^4 \leftarrow r_i$, where $0 \leq i < 5$

[11] See, e.g., https://docs.openhwgroup.org/projects/cv32e40p-user-manual

throughout. Put another way, the specialisation is more related to the load and store semantics, i.e., the input and output rather than content of the S-box.

To support their `sbox` instruction, Burke, McDonald, and Austin [BMA00, Section 5] propose an implementation which include (various configurations) of cache memory which are dedicated to S-box access. Writes through the cache are explicitly *not* visible to subsequent `sbox` instructions; the associated `sboxsync` instruction is therefore included to allow the enforcement of consistency.

Specialist support for instruction fetch. Within the context of cryptographic workloads, various authors have observed the utility of a mechanism for hardware-supported loops: although some kernels allow loops to be unrolled (e.g., a block cipher, based on a fixed number of rounds), others (e.g., an algorithm for multi-precision integer arithmetic, based on a variable number of limbs in the representation) often have short loop bodies so benefit from reduction of any loop overhead. Although not cryptography-specific, such mechanisms could be classified as *implicit*, e.g., a hardware-managed loop buffer or cache, or *explicit*, i.e., a managed manually in software via some form of ISE.

The RISC-V compliant CV32E40P (formerly RI5CY) core supports[12] an optional, general-purpose ISE for hardware loops which fits the latter class; this has various use-cases in cryptographic workloads. Grabher et al. [GGH+11, Section 3.2] investigate the concept of an Instruction Register File (IRF), as originally introduced by Hines et al. [HGTW05]: their approach allows a loop body to be "recorded" into the IRF and then "played back" multiple times without the overhead of memory access.

4.5 Class $\mathcal{C}_5$: security-critical properties

4.5.1 Concept

Per Section 1, a distinguishing characteristic of cryptographic workloads is their need to consider security as a quality metric. Consideration of this fact within the context of ISEs has positive *and* negative implications, both of which are explored in work arguably more diverse than that surveyed by other Sections.

From a positive perspective, for example, note that the implementation of an ISE is necessarily set within the context of some micro-architecture. Use of the ISE can therefore allow 1) access to or 2) control over resources that would be impossible using the ISA, and can be leveraged to provide functionality or enforce behaviour which is security-enhancing in some way. From a negative perspective, however, note that Saab, Rohatgi, and Hampel [SRH16] concluded that naive use of AES-NI allows a specific form of attack. One can argue reasonably that a similar attack applies to an implementation that use the ISA alone, so one cannot "blame" the ISE. However, there is a crucial difference, in the sense that various countermeasures are viable in an ISA-based implementation but non-viable in an ISE-based implementation: this fact stems from flexibility of software versus hardware (i.e., micro-architectural implementation of the ISE), leading to the challenge of designing ISEs which can be composed with countermeasures.

4.5.2 Examples

Security-informed design philosophy. Some work can be classified as taking a broader, and higher-level approach, in the sense it addresses challenges in (cyber-)security more generally (which then impact cryptography in specific instances) by fundamentally re-evaluating what an ISA should be or do within that context. Most examples are arguably better framed as altering versus extending the base ISA, although intersect with reasonable definitions of ISE even so.

[12] See, e.g., https://docs.openhwgroup.org/projects/cv32e40p-user-manual

- Ge, Yarom, and Heiser [GYH18] present the case for a "*new security-oriented hardware/software contract*" they dub the augmented ISA (aISA): see also [Hei18, LPAF+18]. Their argument, in essence, is that normal ISAs are failing to provide an appropriate abstraction because they abstract details that would otherwise allow provision of time protection (i.e., the prevention of unauthorised temporal interaction). Based on the high-level principle that it should be possible to 1) partition, and/or 2) reset resources in the micro-architecture, the aISA demands lower-level properties that render the ISA less opaque, *but*, as a trade-off, able to provide time protection and hence prevent associated micro-architectural attacks.
- Zagieboylo, Suh, and Myers [ZSM19] present a RISC-V based variant that supports enforcement of information-flow control and hence mitigation of side-channel attacks that leverage execution latency.
- Yu et al. [YHHF19] aim to "*present ISA design principles to block microarchitectural side channels*" while also offering "*support for efficient memory oblivious computation, and with safety features that allow modern hardware optimizations*", doing so in a concrete, RISC-V based variant they dub the Data Oblivious ISA extension (OISA).
- Escouteloup et al. [EFLL20] aim to "*prevent timing side-channels, strengthen control flow integrity and ensure micro-architectural state isolation*" using RISC-V as a base ISA; they propose RV32S, a "secure" ISA design based on RV32I, which embodies principles such as simplicity, principle of least privilege, transparency, defence in depth, etc. In concrete terms, they do so via a series of recommendations organised under 3 headings: these are 1) semantics of "SCA hardened" instructions (a "confidential" sub-set of the register file, automatic memory encryption, availability of an RNG), 2) improved security guarantees via stricter control-flow (fixed-length instruction encoding, removal of forward indirect jumps, availability of a dispatch instruction), and 3) support for Hardware Security Contexts (HSCs).
- Stolz et al. [SFSG23] aim to "*augment the RISC-V instruction set architecture with instructions to deter against the threats analyzed in this work*" using a variant of RISC-V. Per [SFSG23, Section 2], their scope spans embedded-focused threats including 1) software-based attacks via code injection and re-use, and 2) fault attacks via glitching. In concrete terms, they do so via a series of recommendations spanning 3 levels of granularity: these are the basic block level (e.g., hardware-assisted instruction hashing), the function level (e.g., extension to cater for inter- rather than intra-block control flow), and the global level (e.g., pointer protection).

Countermeasures: hiding. A hiding [MOP07, Chapter 7] style countermeasure can be characterised as attempting to prevent an associated side-channel attack by decreasing the Signal-to-Noise Ratio (SNR); this can be achieved by 1) decreasing signal (e.g., make observed samples constant), and/or 2) increasing noise (e.g., make observed samples random).

An example of the first class is support for data-oblivious (also "constant-time") versus data-dependant execution latency. In a generic sense, one *could* make the argument that ISEs often make this and similar approaches easier. ISEs for AES such AES-NI [Gue09, DGK19] can replace look-up style implementation techniques such as T-tables [DR02, Section 4.2], and thereby avoid data-dependant execution latency implied by memory access as a side-effect. Either way, however, more specific support is an emerging feature in ISAs and/or ISEs. First, via a processor mode: such an approach is taken by the the ARM Data Independent Timing (DIT) and the Intel Data Operand Independent Timing (DOIT) mechanisms, both of which enforce data-oblivious execution latency for a subset of instructions while the associated mode is enabled. Second, via stricter instruction semantics which constrain the micro-architectural implementation: such an approach is taken by the RISC-V Data Independent Execution Latency (DIEL) (meta-)extensions Zkt and

Zvkt [RIS24, Sections 34.6 and 35.2.15]. These extensions do not define functionality per se, but with them the processor "*attests that the machine has data-independent execution time for a safe subset of instructions.*" In other words, the Zkt and Zvkt extensions expand the hardware-software contract to include data-independent latency for some instructions.

An example of the second class is support for various forms of diversified (i.e., randomised) instruction execution, with a common sub-class being temporal reordering (or "shuffling"). In a generic sense, one *could* make the argument that ISEs often make this and similar approaches harder: an ISE-based implementation will typically comprise fewer instructions, for example, which implies less ILP to harness via shuffling; in turn, this acts to limit the security improvement possible. Either way, however, more specific support is considered by various ISEs. Bayrak et al. [BVIB12] focus on reordering instructions, in the sense their aim is to support execution instructions in a randomised order. They do so by adding an additional unit on the instruction fetch path between the core and memory (or instruction cache, more specifically). This unit is tasked with interpreting a pseudo-instruction which describes a subsequent block of instructions (see, e.g., [BVIB12, Figure 2]), then delivering those instructions to the fetch unit in a constrained random order. Zhou et al. [ZQL+23] focus on reordering data, in the sense their aim is to support permutation of indices then used, e.g., to randomised access to and hence operations on register- or memory-resident data. They do so by adding an additional unit to house and permute the content of dedicated shuffling registers (see, e.g., [ZQL+23, Table 2]), plus various instructions to, e.g., invoke the permutation and access said registers.

Countermeasures: masking. A masking [MOP07, Chapter 10] style countermeasure can be characterised as attempting to prevent an associated side-channel attack by employing a randomised representation, harnessing the underlying concept of secret sharing [Sha79]; doing so eliminates the relationship between data used during execution and that specified within the underlying (i.e., unmasked) algorithm.

At a high level, a given masking scheme specifies a masked representation, plus a means of performing masked computation. So, first, a d-th order masking scheme represents a variable x as $\hat{x} = \langle \hat{x}[0], \hat{x}[1], \ldots, \hat{x}[d] \rangle$, i.e., as $d+1$ statistically independent shares: a Boolean masking scheme demands that $x = \hat{x}[0] \oplus \hat{x}[1] \oplus \cdots \oplus \hat{x}[d]$, whereas an arithmetic masking scheme (typically) demands that $x = \hat{x}[0] + \hat{x}[1] + \cdots + \hat{x}[d] \pmod{2^k}$ for some k. Second, some functionality f which would be applied to x is translated into a compatible, masked alternative $\hat{f}$ applied to $\hat{x}$. The realisation of $\hat{f}$ can be categorised as either circuit-based or table-based: the former expresses the entire computation, i.e., both linear and non-linear components, as a (Boolean and/or an arithmetic) circuit of nodes termed gadgets, whereas the latter supports any non-linear components (e.g., the S-box) by using a look-up table.

ISEs for masking have thus far focused on circuit-based implementations, and therefore provision of instructions with gadget-like semantics. In the case of a Boolean masking scheme, for example, instructions might include masked variants of Boolean operators such as AND and OR which realise the SecAnd (or secure, masked AND) and SecOr (or secure, masked OR) gadgets described by Biryukov et al. [BDLU17, Table 1] for $d = 1$. This approach was proposed by Kiaei and Schaumont [KS20] and realised soon after by Gao et al. [GGM+21]. The evaluation presented in [GGM+21, Section 5] focuses on various symmetric primitives; Krausz et al. [KLS+23] used this as motivation, further expanding the ISE design and implementation to cater for various asymmetric, post-quantum primitives.

However, a limitation of such work is the focus on first-order masking where $d = 1$. Generalisation to higher-order masking where $d > 1$ is attractive, but challenging for various reasons including the number of input and output shares for a given gadget: this is essentially an instance of the increased input and/or output bandwidth challenge per Section 4.2. Marshall and Page [MP21] address the challenge by framing it as an instance

of vector processing, i.e., by treating $\hat{x}$ as a $(d+1)$-element vector then specifying an ISE which operates on a vector register file. Lozachmeur and Tisserand [LT23] address the challenge by framing it as an instance of scalar processing, instead "packing" blocks of shares into scalar registers. Consider application of a masked AND gadget to $\hat{x}$ and $\hat{y}$ in a case where $d = 3$. The ISE supports standard 3-address instructions for such gadgets, but assumes blocks of the $d + 1 = 4$ shares are packed into each input (resp. output) operand. So, rather than a single (complex) executed instruction with $2 \cdot (d + 1)$ input operands, the idea is to have multiple (simple) executed instructions with 2 input operands.

Countermeasures: suppression. Rather than *being* a countermeasure per se, some mechanisms are better classified as offering support *for* countermeasures. Mechanisms for leakage suppression fall into this class: their goal could be described as controlling or even eliminating certain leakage effects, which (ideally) means they cannot be exploited. At a high level, one could imagine at least two strategies: one could task hardware (e.g., the micro-architecture) with doing so *implicitly*, or software (stemming from, e.g., a developer or compiler) with doing so *explicitly* purely using the ISA or with support of an ISE. Numerous leakage effects (see, e.g., [CBCH23, Table 1]) relate to micro-architectural behaviour which is abstracted from, and thus inaccessible to software by design. In such cases, use of ISEs as a way to to selectively allow such access is therefore attractive. At a lower level, the mechanism employed by such an ISE might involve 1) spatial isolation, i.e., partitioning, and 2) temporal isolation, i.e., flushing (or resetting) to prevent abuse of shared micro-architectural state. For coarse-grained resources (which are, e.g., shared at a process granularity) both strategies are viable, but for fine-grained resources (which are, e.g., shared at an instruction granularity) the overheads typically associated with partitioning render it less so. flushing becomes the only viable option.

In existing literature, two broad classes of leakage suppression ISE can be identified. First, Cheng, Page, and Wang [CPW24] present an ISE that exemplifies leakage suppression via somewhat implicit resource flushing. The underlying idea is that general-purpose instructions are equipped with an additional "hint". Use of the hint results in the *same* functional properties, but *different* behavioural properties: specifically, the hint signals to the micro-architecture that overwriting should be prevented, e.g., by pre-flushing the destination resource. Second, various authors present ISEs that exemplify leakage suppression via somewhat explicit resource flushing. The underlying idea is that special-purpose instructions are added, which allow some form of control over resources in the micro-architecture. At least two approaches could be considered:

1. A lower-level approach focuses on the resources themselves. An example of such an approach would be allowing dedicated instructions to flush specific resources. Instances of this approach are adopted by flavours of x86 (e.g., instructions such as `clflush` [Int22b, Page 3-161] and `invd` [Int22b, Page 3-521]) and ARM (e.g., memory-mapped maintenance operations [Arm21, Section B2.2.7]).
2. A higher-level approach focuses on instruction execution. An example of such an approach would be support for fence (or barrier) instructions. When defined with respect to some class of instructions, a given fence will guarantee all instructions in said class *before* it (in program order) complete execution before any instructions *after* it; for the class of memory access instructions, for example, fences are often used to enforce a specific memory ordering model. Instances of this approach are adopted by flavours of x86 (e.g., via `mfence` [Int22b, Page 4-22], `sfence` [Int22b, Page 4-620], and `lfence` [Int22b, Page 3-585]), ARM (e.g., via `dmb` [Arm21, Section A7.7.33]), SPARC (e.g., via `membar` [SPA16, Section 8.4.3]), MIPS (e.g., via `sync` [MIPS01b, Pages 407–411]), and RISC-V (e.g., via `fence` [RV19, Section 2.7]). The same concept could be applied to enforce separation of instructions with respect to *any* micro-architectural resource, however, so, for example, a suitable fence instruction could be framed as a way to flush them.

For example, Gao et al. [GMPP20] focus on fine-grained resources (e.g., pipeline registers), and analogue forms of leakage (e.g., power or EM). Using RISC-V as a platform, they introduce the `fenl.fence` instruction: execution of said instruction flushes resource(s) specified at run-time by a special-purpose register (although an immediate variant is discussed in [GMPP20, Section 2.2.2]). Wistoff et al. [WSG+20] focus on coarse-grained resources (e.g., caches) and discrete forms of leakage (i.e., execution latency). Using RISC-V as a platform, they introduce the `fence.t` instruction: execution of said instruction flushes resource(s) specified at run-time by an immediate operand. Li, Hopkins, and Parameswaran [LHP20] focus on coarse-grained resources (e.g., caches) and discrete forms of leakage (i.e., execution latency). Using RISC-V as a platform, they introduce the `flushx` instruction: execution of said instruction flushes resource(s) specified at design-time which is termed the "*sphere of flushing*". Interestingly, architectural (e.g., the general-purpose register file) as well as micro-architectural resources can be included in that specification.

Countermeasures: hybrid. Although the term hybrid can be viewed as implying some combination of two (or more) techniques, it is overloaded in the sense that it depends on the techniques in question. In this Section we treat it as a catch-all for the combination of ISE-related countermeasure techniques, versus the non-hybrid or dedicated alternatives described elsewhere.

- Tillich, Herbst, and Mangard [THM07] taken an existing ISE for AES, namely [TG06], and consider how software-focused countermeasures against side-channel attack can be applied, i.e., how security can be enhanced through a focus on *use* versus *implementation* of the ISE. Per [THM07, Page 154] the outer-most rounds of AES are implemented using the ISA *with* countermeasures, whereas the outer-most rounds of AES are implemented using an ISE *without* countermeasures; doing so addresses the challenge of compsability of ISEs and countermeasures, allowing the outer-most, more vulnerable rounds to be more secure and the inner-most, less vulnerable rounds to be more efficient. This strategy mirrors wider study of non-uniform countermeasure use, e.g., by Verhamme, Cassiers, and Standaert [VCS22].
- Tillich and Großschädl [TG07a] taken an existing ISE for AES, namely [TG06], and consider how hardware-focused countermeasures against side-channel attack can be applied, i.e., how security can be enhanced through a focus on *implementation* versus *use* of the ISE. Their options #1 [TG07a, Section 4] and #2 [TG07a, Section 5] are more focused on the physical, i.e., more micro-architectural, implementation, in the sense they are based on use of a secure logic style and randomised pre-charging respectively; option #3 [TG07a, Section 6] is more focused on the logical, i.e., more architectural, implementation, in the sense it is based on use of an additional mechanism (a design concept later expanded upon in [TKS10]) which semi-automatically ensures inputs to and outputs from the AFU are masked. Note that [TG07a, Section 3] identifies AFU-related fowarding logic as a potential source of leakage, and removes this from the micro-architecture (based on the argument that for use of the ISE, forwarding is not required).
- Regazzoni et. al [RCS+09] investigate the hybridisation of option #1 presented by Tillich and Großschädl [TG07a, Section 4]. That is, they consider ISE implementations in which security-agnostic components are implemented using a standard Complementary Metal Oxide Semi-conductor (CMOS) logic style, whereas security-concious components are implemented using a secure MOS Current Mode Logic (MCML) [REP+09] logic style; doing so is framed within the context of security-aware Electronic Design Automation (EDA) tool and workflow.
- Kiaei et al. [KMD+20] present SKIVA, an extension of the SPARC-V compliant LEON3 core designed to support implementations based on the use of (aggregated) bit-slicing. More specifically, an ISE [KMD+20, Table 1] is used to efficiently support 1) hiding via

data-oblivious computation, 2) higher-order masking, 3) data-redundant computation, and 4) time-redundant computation: these act to harden an implementation, in the sense that 1) and 2) are framed as side-channel countermeasures while 3) and 3) are framed as fault induction countermeasures. SKIVA, and so the associated ISE, could therefore be described as hybrid in the sense it supports various forms of countermeasure.

Entropy sources, TRNGs, etc. Campbell-Kelly [Cam80] credits Alan Turing with the introduction of a random number instruction in Ferranti Mark I, built in 1949-51:

> *At the request of Turing an instruction to generate a random number from a noise source was provided. Unfortunately, the numbers turned out to be not particularly random [..]*

This quote also illustrates one unique property of random number generators and entropy sources as instructions; one has to specify their quality and security properties as well.

Mainstream Intel x86 ISA gained a cryptographic random number generator with the RDRAND instruction of Ivy Bridge processors (2012) [HKM12]. RDRAND is a user-mode instruction that is directly available to applications. Failure is signaled with the carry flag. Intel's implementation was intended to comply with NIST's recommendations for Random Bit Generation: SP 800-90A [BK15] and SP 800-90B [TBK+18].

Intel's RDRAND implementation uses CTR_DRBG [BK15] that is frequently reseeded from a Physical Entropy Source [TBK+18]. Early versions used AES-128 internally; hence, output strings from RDRAND were limited to 128 bits of entropy until the next reseed occurs after 511 blocks [Int18]. Generating keys above this 128-bit security level with RDRAND required gathering enough random words to force a reseed. With Broadwell (2014), Intel introduced a second instruction, RDSEED, whose output is intended to be "full entropy" in accordance with the relevant NIST standard SP 800-90C [BKM+24]; any n-bit output string from RDSEED should have close to n bits of entropy.

AMD started supporting the x86 RDRAND and RDSEED instructions in 2016 with their Cryptographic Coprocessor (CCP) 5.0 [AMD17]. AMD also makes raw noise samples available via a restricted MMIO interface, which is required in the NIST RBG validation process. Access to similar internal diagnostic resources of Intel DRNG is only possible in cooperation with Intel (the processor needs to "unlocked".)

The DARN (Deliver A Random Number) instruction first appeared in Power ISA 3.0 in 2015 [IBM15, Ope24]. Three modes are supported: 32-bit conditioned (SP 800-90C [BKM+24]), 64-bit conditioned, and 64-bit unconditioned (raw SP 800-90B [TBK+18] entropy samples for testing.) The return value `0xFFFFFFFF_FFFFFFFF` is not considered a random number but is used to signal both non-fatal and fatal errors, creating a slight (but cryptanalytically significant) bias if not used very carefully. It is likely that ISA designers were not aware that it is a compliance requirement that error conditions can be signalled. Still, the caller can't know if the noise source is depleted ("busy") or if the operating conditions are such that SP 800-90B health tests are failing.

There were early RISC-V "RNG ISA" proposals such as [LCW18] that largely ignored standards compliance requirements and hence would be limited to non-commercial use.

The development of the RISC-V Entropy Source (ES) interface is documented in [SNM22]. The ES interface was ratified in 2021 as the Zkr extension and later merged into the main RISC-V ISA specification [RIS24]. The Zkr extension defines an Entropy Source CSR (Control and Status Register) with specific entropy properties. On application-class processors, the physical RISC-V entropy source CSR is not intended to be available to user-level processes for multiple security reasons outlined [SNM22]; the expectation is that the obtained entropy is used to seed a (Kernel) RBG of arbitrary type, such as a fast DRBG that uses scalar or vector cryptography instructions. User applications then use operating system interfaces to obtain random bits of desired quality.

To avoid perceived problems with previous ISA approaches, the RISC-V Zkr interface can signal fatal and non-fatal errors via status bits reserved for that purpose. NIST SP 800-90A/B [BK15, TBK+18] and BSI AIS 20/31 [KS11] largely drow the technical randomness and self-monitoring features of Zkr. FIPS 140-3 and various Common Criteria protection profiles stipulate compliance with these standards. The [RIS24] specification also suggests an interface for testing and characterization to avoid the product certification problems caused by the unavailability or "lockdown" of such interfaces on some processors.

5 Conclusion

We have have surveyed, analysed, and evaluated cryptographic ISEs. We claim that the body of associated work captures an important point in what is a large, complex design space of implementation options for cryptography; this claim is evidenced by a rich history, *and* a vast corpus of both associated literature (e.g., per our 259-entry bibliography, and including notable theses such as [Shi04, Fis05, Hil08, Til08, Man11, Raw16, Nis21, Nis21, Fri22]) and concrete hardware and software artefacts. Within the paper, Section 3 focused on point 2), e.g., by systematising associated concepts and terminology. Section 4 focused on points 1) and 2), e.g., by surveying and systematising associated work.

Throughout the paper, but under point 2) in particular, there is significant volume of and diversity in the surveyed work; coupled with the design space, we attribute this fact to the topic spanning multiple disciplines. For example, the ISE design process involves a wide range of technical concepts, e.g., both hardware and software, *and* a wide range of stakeholders including 1) policy and decision makers, who fix constraints and select between options, e.g., based on pertinent use-cases, 2) cryptographers, who focus on design of the underlying constructions, 3) digital design engineers (cf. [BMT16]), who focus on hardware designs and implementations, 4) cryptographic engineers, who focus on software designs and implementations. One one hand, this fact could be viewed as a negative, or at least a challenge. On the other hand, however, it could also be viewed as a positive: cryptographic ISEs represent an accessible, active, and impactful topic, with the most effective work able encapsulating many different technical *and* non-technical perspectives and contributions.

Acknowledgements

This work has been supported in part by EPSRC via grant EP/R012288/1 under the RISE (http://www.ukrise.org) programme, and Innovate UK via project 10065634 (SCHEME: Safety Critical Harsh Environment Micro-processing Evolution), also, in part by the Lux4QCI project and by the Luxembourg National Research Fund (FNR) via the CORE project ImPAKT (C21/IS/16221219/ImPAKT).

References

[AAC+22] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and Y.-K. Liu. Status report on the third round of the NIST post-quantum cryptography standardization process. National Institute of Standards and Technology (NIST), Interagency Report 8413, 2022. https://doi.org/10.6028/NIST.IR.8413-upd1.

[Aca97] T. Acar. *High-Speed Algorithms & Architectures for Number Theoretic Cryptosystems*. PhD thesis, Department of Electrical & Computer Engineering, Oregon State University, 1997.

[AEL+20] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri. ISA extensions for finite field arithmetic: Accelerating Kyber and NewHope on RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(3):219–242, 2020. https://doi.org/10.13154/tches.v2020.i3.219-242.

[AMD17] AMD. AMD random number generator, 2017. https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/white-papers/amd-random-number-generator.pdf.

[ANP20] A. Adomnicai, Z. Najm, and T. Peyrin. Fixslicing: A new GIFT representation: Fast constant-time implementations of GIFT and GIFT-COFB on ARM Cortex-M. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(3):402–427, 2020. https://doi.org/10.13154/tches.v2020.i3.402-427.

[AO21] Ö. Altınay and B. Örs. Instruction extension of RV32I and GCC back end for Ascon lightweight cryptography algorithm. In *International Conference on Omni-Layer Intelligent Systems (COINS)*, pages 1–6, 2021. https://doi.org/10.1109/COINS51742.2021.9524190.

[AOP+24] A. Abdulrahman, F. Oberhansl, H.N.H. Pham, J. Philipoom, P. Schwabe, T. Stelzer, and A. Zankl. Towards ML-KEM & ML-DSA on OpenTitan. Cryptology ePrint Archive, Paper 2024/1192, 2024. https://eprint.iacr.org/2024/1192.

[AP20a] A. Adomnicai and T. Peyrin. Fixslicing - application to some NIST LWC round 2 candidates. In *4-th Lightweight Cryptography Workshop*, 2020. https://csrc.nist.gov/Events/2020/lightweight-cryptography-workshop-2020.

[AP20b] A. Adomnicai and T. Peyrin. Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(1):402–425, 2020. https://doi.org/10.46586/tches.v2021.i1.402-425.

[APRJ11] A. Arora, S. Parameswaran, R.G. Ragel, and D. Jayasinghe. A hardware/software countermeasure and a testing framework for cache based side channel attacks. In *Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1005–1014, 2011. https://doi.org/10.1109/TrustCom.2011.138.

[Arm20] Arm architecture reference manual: Armv8, for Armv8-A architecture profile. Technical report, 2020. https://static.docs.arm.com/ddi0487/fa/DDI0487F_a_armv8_arm.pdf.

[Arm21] ARMv7-M Architecture Reference Manual. Technical Report DDI-0403E.e, ARM Ltd., 2021. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0403e.e/index.html.

[Arm22] ARMv8-M Architecture Reference Manual. Technical Report DDI0553B.u, ARM Ltd., 2022. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0553b.u/index.html.

[BAK98] E. Biham, R. Anderson, and L. Knudsen. Serpent: A new block cipher proposal. In *Fast Software Encryption (FSE)*, LNCS 1372, pages 222–238. Springer-Verlag, 1998. https://doi.org/10.1007/3-540-69710-1_15.

[Bar86] P.D. Barrett. Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology (CRYPTO)*, LNCS 263, pages 311–323. Springer-Verlag, 1986. https://doi.org/10.1007/3-540-47721-7_24.

[BBFR06] G. Bertoni, L. Breveglieri, R. Farina, and F. Regazzoni. Speeding up AES by extending a 32-bit processor instruction set. In *Application-Specific Systems, Architectures and Processors (ASAP)*, pages 275–282, 2006. https://doi.org/10.1109/ASAP.2006.62.

[BBGM04] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli. A performance evaluation of ARM ISA extension for elliptic curve cryptography over binary finite fields. In *Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 238–245, 2004. https://doi.org/10.1109/SBAC-PAD.2004.5.

[BBGM08] S. Bartolini, I. Branovic, R. Giorgi, and E. Martinelli. Effects of instruction-set extensions on an embedded processor: A case study on elliptic curve cryptography over $GF(2^m)$. *IEEE Transactions on Computers*, 57(5):672–685, 2008. https://doi.org/10.1109/TC.2007.70832.

[BBGR09] R. Benadjila, O. Billet, S. Gueron, and M.J.B. Robshaw. The Intel AES instructions set and the SHA-3 candidates. In *Advances in Cryptology (ASIACRYPT)*, LNCS 5912, pages 162–178. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-10366-7_10.

[BDLU17] A. Biryukov, D. Dinu, Y. Le Corre, and A. Udovenko. Optimal first-order Boolean masking for embedded IoT devices. In *Smart Card Research and Advanced Applications (CARDIS)*, LNCS 10728, pages 22–41. Springer-Verlag, 2017. https://doi.org/10.1007/978-3-319-75208-2_2.

[BEM+15] N. Benhadjyoussef, W. Elhadjyoussef, M. Machhout, R. Tourki, and K. Torki. Enhancing a 32-bit processor core with efficient cryptographic instructions. *Journal of Circuits, Systems and Computers*, 24(10), 2015. https://doi.org/10.1142/S0218126615501583.

[BGHZ11] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology (CRYPTO)*, LNCS 6841, pages 71–90. Springer-Verlag, 2011. https://doi.org/10.1007/978-3-642-22792-9_5.

[BGM09] S. Bartolini, R. Giorgi, and E. Martinelli. Instruction set extensions for cryptographic applications. In Ç.K. Koç, editor, *Cryptographic Engineering*, chapter 9, pages 191–233. Springer, 2009. https://doi.org/10.1007/978-0-387-71817-0_9.

[BHO04] R. Buchty, N. Heintze, and D. Oliva. Cryptonite - a programmable crypto processor architecture for high-bandwidth applications, 2004. https://doi.org/10.1007/978-3-540-24714-2_15.

[Bih97] E. Biham. A fast new DES implementation in software. In *Fast Software Encryption (FSE)*, LNCS 1267, pages 260–272. Springer-Verlag, 1997. https://doi.org/10.1007/BFb0052352.

[BK15] E. Barker and J. Kelsey. Recommendation for random number generation using deterministic random bit generators. Special Publication SP 800-90A Revision 1, NIST, 2015. https://doi.org/10.6028/NIST.SP.800-90Ar1.

[BKL+07] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 4727, pages 450–466. Springer-Verlag, 2007. https://doi.org/10.1007/978-3-540-74735-2_31.

[BKL+13] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: The design space of lightweight cryptographic hashing. *IEEE Transactions on Computers*, 62(10):2041–2053, 2013. https://doi.org/10.1109/TC.2012.196.

[BKM+24] E. Barker, J. Kelsey, K. McKay, A. Roginsky, and M.S. Turan. Recommendation for random bit generator (RBG) constructions. Special Publication SP 800-90C Public Draft 4, NIST, 2024. https://doi.org/10.6028/NIST.SP.800-90C.4pd.

[BLCN20] M. Bie, W. Li, T. Chen, and L. Nan. Design and implementation of cryptographic instruction set. In *IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 1–3, 2020. https://doi.org/10.1109/ICSICT49897.2020.9278206.

[BMA00] J. Burke, J. McDonald, and T. Austin. Architectural support for fast symmetric-key cryptography. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 178–189, 2000. https://doi.org/10.1145/378993.379238.

[BMT16] W. Burleson, O. Mutlu, and M. Tiwari. Who is the major threat to tomorrow's security? you, the hardware designer. In *Design Automation Conference (DAC)*, pages 145:1–145:5, 2016. https://doi.org/10.1145/2897937.2905022.

[BOS11] J.W. Bos, O. Özen, and M. Stam. Efficient hashing using the AES instruction set. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 6917, pages 507–522. Springer-Verlag, 2011. https://doi.org/10.1007/978-3-642-23951-9_33.

[BS12] D.J. Bernstein and P. Schwabe. NEON crypto. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 7428, pages 320–339. Springer-Verlag, 2012. https://doi.org/10.1007/978-3-642-33027-8_19.

[BSP13] P. Bhagyasree, C. Silpa, and M.J.C. Prasad. A novel RISC processor with crypto specific instruction set. *International Journal of Soft Computing and Engineering (IJSCE)*, 3(5):124–128, 2013.

[BSS+13] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.

[BV14] A. Bosselaers and F. Vercauteren. YAES. Technical report, 2014. https://competitions.cr.yp.to/round1/yaesv2.pdf.

[BVIB12] A.G. Bayrak, N. Velickovic, P. Ienne, and W. Burleson. An architecture-independent instruction shuffler to protect against side-channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):20:1–20:19, 2012. https://doi.org/10.1145/2086696.2086699.

[Cam80] M. Campbell-Kelly. Programming the mark i: Early programming activity at the university of manchester. *Annals of the History of Computing*, 2(2):130–168, 1980. https://doi.org/10.1109/MAHC.1980.10018.

[CB23] S. Cui and J. Balasch. Efficient software masking of AES through instruction set extensions. In *Design, Automation, and Test in Europe (DATE)*, pages 1–6, 2023. 10.23919/DATE56975.2023.10137150.

[CBCH23] L. Casalino, N. Belleville, D. Couroussé, and K. Heydemann. A tale of resilience: On the practical security of masked software implementations. *IEEE Access*, 11:84651–84669, 2023. https://doi.org/10.1109/ACCESS.2023.3298436.

[CBG12a] J. Constantin, A. Burg, and F.K. Gürkaynak. Investigating the potential of custom instruction set extensions for SHA-3 candidates on a 16-bit microcontroller architecture. Cryptology ePrint Archive, Report 2012/050, 2012. https://eprint.iacr.org/2012/050.

[CBG12b] J.H.-F. Constantin, A.P. Burg, and F.K. Gürkaynak. Instruction set extensions for cryptographic hash functions on a microcontroller architecture. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 117–124, 2012. https://doi.org/10.1109/ASAP.2012.13.

[CCM97] A.D. Carlson, R.W. Castelino, and R.O. Mueller. Multimedia extensions for a 550-MHz RISC microprocessor. *Journal of Solid-State Circuits*, 32(11):1618–1624, 1997. https://doi.org/10.1109/4.641681.

[CFG+24] H. Cheng, G. Fotiadis, J. Großschädl, D. Page, T.H. Pham, and P.Y.A. Ryan. RISC-V instruction set extensions for multi-precision integer arithmetic. In *Design Automation Conference (DAC)*, 2024. https://doi.org/10.1145/3649329.3657347.

[CGM+22] H. Cheng, J. Großschädl, B. Marshall, D. Page, and T.H. Pham. RISC-V instruction set extensions for lightweight symmetric cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2023(1):193–237, 2022. https://doi.org/10.46586/tches.v2023.i1.193-237.

[CHW00] T.J. Callahan, J.R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *IEEE Computer*, 33(4):62–69, 2000. https://doi.org/10.1109/2.839323.

[Cla97] C.S.K. Clapp. Optimizing a fast stream cipher for VLIW, SIMD, and superscalar processors. In *Fast Software Encryption (FSE)*, LNCS 1267, pages 273–287. Springer-Verlag, 1997. https://doi.org/10.1007/BFb0052353.

[Com90] P.G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, 1990.

[CP20] L. Choquin and F. Piry. Arm custom instructions: Enabling innovation and greater flexibility on Arm. Technical report, Arm Ltd., 2020. https://www.arm.com/why-arm/technologies/custom-instructions.

[CPW24] H. Cheng, D. Page, and W. Wang. eLIMInate: a Leakage-focused ISE for Masked Implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2024(2):329–358, 2024. https://doi.org/10.46586/tches.v2024.i2.329-358.

[Cra] R.E. Crandall. Method and apparatus for public key exchange in a cryptographic system. U.S. Patent 5159632A. https://patents.google.com/patent/US5159632A.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. https://doi.org/10.2307/2003354.

[DB18] C. Dunham and J. Beard. This architecture tastes like microarchitecture. In *Workshop on Pioneering Processor Paradigms (WP3)*, 2018. http://www.jonathanbeard.io/pdf/db18a.pdf.

[DBB+23] M. De Kremer, M. Brohet, S. Banik, R. Avanzi, and F. Regazzoni. Resource-constrained encryption: Extending Ibex with a QARMA hardware accelerator. In *Application-specific Systems, Architectures and Processors (ASAP)*, pages 147–155, 2023. https://doi.org/10.1109/ASAP57973.2023.00034.

[DDHS00] K. Diefendorff, P. Dubey, R. Hochsprung, and H. Scales. AltiVec extension to PowerPC accelerates media processing. *IEEE Micro*, 20(2):85–95, 2000. https://doi.org/10.1109/40.848475.

[DEWW01] G. De Micheli, R. Ernst, W. Wolf, and M. Wolf, editors. *Readings in Hardware/Software Co-Design*. Systems on Silicon. Morgan Kaufmann, 2001.

[DGK19] N. Drucker, S. Gueron, and V. Krasnov. Making AES great again: The forthcoming vectorized AES instruction. In *Information Technology New Generations (ITNG)*, AISC 800, pages 37–41. Springer-Verlag, 2019. https://doi.org/10.1007/978-3-030-14070-0_6.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Din17] D. Dinu. *Efficient and Secure Implementations of Lightweight Symmetric Cryptographic Primitives*. PhD thesis, 2017. https://orbilu.uni.lu/bitstream/10993/33803/1/thesis.pdf.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002. https://doi.org/10.1007/978-3-662-04722-4.

[DR14] K.L. Divya and S.H. Rao. Implementation of Cryptographic Risc Processor (CRISC). *International Journal of Innovative Science, Engineering & Technology (IJISET)*, 1(10):358–363, 2014.

[EAD+22] W. El Hadj Youssef, A. Abdelli, F. Dridi, R. Brahim, and M. Machhout. An efficient lightweight cryptographic instructions set extension for IoT device security. *Security and Communication Networks*, 2022, 2022. https://doi.org/10.1155/2022/9709601.

[EES+16] A.S. Eissa, M.A. Elmohr, M.A. Saleh, K.E. Ahmed, and M.M. Farag. SHA-3 instruction set extension for a 32-bit RISC processor architecture. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 233–234, 2016. https://doi.org/10.1109/ASAP.2016.7760804.

[EFLL20] M. Escouteloup, J.J.A. Fournier, J.-L. Lanet, and R. Lashermes. Recommendations for a radically secure ISA. In *Computer Architecture Research with RISC-V (CARRV)*, 2020. https://carrv.github.io/2020.

[EKP+13] S. Engels, E.B. Kavun, C. Paar, T. Yalçin, and H. Mihajloska. A non-linear/linear instruction set extension for lightweight ciphers. In *IEEE Symposium on Computer Arithmetic (ARITH)*, pages 67–75, 2013. https://doi.org/10.1109/ARITH.2013.36.

[Elb07] A.J. Elbirt. Fast and efficient implementation of AES via instruction set extensions. In *Advanced Information Networking and Applications Workshops (AINAW)*, pages 396–403, 2007. https://doi.org/10.1109/AINAW.2007.182.

[Elb08] A.J. Elbirt. Accelerated AES implementations via generalized instruction set extensions. *Journal of Computer Security*, 16(3):265–288, 2008. .

[EMOC20] G.H. Eisenkraemer, F.G. Moraes, L.L. de Oliveira, and E. Carara. Lightweight cryptographic instruction set extension on Xtensa processor. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2020. https://doi.org/10.1109/ISCAS45731.2020.9180579.

[ESE+16] M.A. Elmohr, M.A. Saleh, A.S. Eissa, K.E. Ahmed, and M.M. Farag. Hardware implementation of a SHA-3 application-specific instruction set processor. In *International Conference on Microelectronics (ICM)*, pages 109–112, 2016. https://doi.org/10.1109/ICM.2016.7847921.

[Fis05] A.M. Fiskiran. *Instruction set architecture for accelerating cryptographic processing in wireless computing devices*. PhD thesis, Princeton University, 2005.

[FL01] A.M. Fiskiran and R.B. Lee. Performance impact of addressing modes on encryption algorithms. In *International Conference on Computer Design (ICCD)*, pages 542–545, 2001. https://doi.org/10.1109/ICCD.2001.955088.

[FL04] A.M. Fiskiran and R.B. Lee. Evaluating instruction set extensions for fast arithmetic on binary finite fields. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 125–136, 2004. https://doi.org/10.1109/ASAP.2004.1342464.

[FL05a] A.M. Fiskiran and R.B. Lee. Fast parallel table lookups to accelerate symmetric-key cryptography. In *Information Technology: Coding and Computing (ITCC)*, volume 1, pages 526–531, 2005. https://doi.org/10.1109/ITCC.2005.151.

[FL05b] A.M. Fiskiran and R.B. Lee. On-chip lookup tables for fast symmetric-key encryption. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 356–363, 2005. https://doi.org/10.1109/ASAP.2005.49.

[FLO18] A. Faz-Hernandez, J. López, and A.K.D.S. de Oliveira. SoK: A performance evaluation of cryptographic instruction sets on modern architectures. In *ASIA Public-Key Cryptography Workshop (APKC)*, pages 9–18, 2018. https://doi.org/10.1145/3197507.3197511.

[Fou07] J.J.A. Fournier. *Vector microprocessors for cryptography*. PhD thesis, University of Cambridge, 2007.

[FPP08] D. Fronte, A. Perez, and E. Payrat. Celator: A multi-algorithm cryptographic co-processor. In *Reconfigurable Computing and FPGAs (ReConFig)*, pages 438–443, 2008. https://doi.org/10.1109/ReConFig.2008.76.

[Fri22] T. Fritzmann. *Towards Secure Coprocessors and Instruction Set Extensions for Acceleration of Post-Quantum Cryptography.* PhD thesis, Technischen Universität München, 2022. https://mediatum.ub.tum.de/1650057.

[FSS20a] T. Fritzmann, G. Sigl, and J. Sepúlveda. Extending the RISC-V instruction set for hardware acceleration of the post-quantum scheme LAC. In *Design, Automation & Test in Europe (DATE)*, pages 1420–1425, 2020. https://doi.org/10.23919/DATE48585.2020.9116567.

[FSS20b] T. Fritzmann, G. Sigl, and J. Sepúlveda. RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(4):239–280, 2020. https://doi.org/10.13154/tches.v2020.i4.239-280.

[GBG+03] C. Grabbe, M. Bednara, J. von zur Gathen, J. Shokrollahi, and J. Teich. A high performance VLIW processor for finite field arithmetic. In *International Symposium on Parallel and Distributed Processing Workshops (IPDPSW)*, pages 396–403, 2003. https://doi.org/10.1109/IPDPS.2003.1213351.

[GBG+11] M. Grand, L. Bossuet, G. Gogniat, B. Le Gal, J.-P. Delahaye, and D. Dallet. A reconfigurable multi-core cryptoprocessor for multi-channel communication systems. In *International Symposium on Parallel and Distributed Processing Workshops (IPDPSW)*, pages 204–211, 2011. https://doi.org/10.1109/IPDPS.2011.143.

[GFG] S. Gueron, W.K. Feghali, and V. Gopal. Architecture and instruction set for implementing advanced encryption standard (AES). U.S. Patent 7949130B2. https://patents.google.com/patent/US7949130B2.

[GGH+11] P. Grabher, J. Großschädl, S. Hoerder, K. Järvinen, D. Page, S. Tillich, and M. Wójcik. An exploration of mechanisms for dynamic cryptographic instruction set extension. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 6917, pages 1–16. Springer-Verlag, 2011. https://doi.org/10.1007/978-3-642-23951-9_1.

[GGM+21] S. Gao, J. Großschädl, B. Marshall, D. Page, T.H. Pham, and F. Regazzoni. An instruction set extension to support software-based masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(4):283–325, 2021. https://doi.org/10.46586/tches.v2021.i4.283-325.

[GGP08] P. Grabher, J. Großschädl, and D. Page. Light-weight instruction set extensions for bit-sliced cryptography. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 5154, pages 331–345. Springer-Verlag, 2008. https://doi.org/10.1007/978-3-540-85053-3_21.

[GK03a] J. Großschädl and G.-A. Kamendje. Architectural enhancements for montgomery multiplication on embedded RISC processors. In *Applied Cryptography and Network Security (ACNS)*, LNCS 2846, pages 418–434. Springer-Verlag, 2003. https://doi.org/10.1007/978-3-540-45203-4_32.

[GK03b] J. Großschädl and G.-A. Kamendje. Instruction set extension for fast elliptic curve cryptography over binary finite fields GF(2^m). In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 455–468, 2003. https://doi.ieeecomputersociety.org/10.1109/ASAP.2003.1212868.

[GK03c] J. Großschädl and G.-A. Kamendje. Low-power design of a functional unit for arithmetic in finite fields GF(p) and GF(2^m). In *Workshop on Information Security Applications (WISA)*, LNCS 2908, pages 227–243. Springer-Verlag, 2003. https://doi.org/10.1007/978-3-540-24591-9_18.

[GKM+11] P. Gauravaram, L.R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläaffer, and S.S. Thomsen. Grøstl – a SHA-3 candidate. Technical report, 2011. http://www.groestl.info/Groestl.pdf.

[GKP04] J. Großschädl, S.S. Kumar, and C. Paar. Architectural support for arithmetic in optimal extension fields. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 111–124, 2004. https://doi.org/10.1109/ASAP.2004.10004.

[GLM24] C. Gewehr, L. Luza, and F.G. Moraes. Hardware acceleration of Crystals-Kyber in low-complexity embedded systems with RISC-V instruction set extensions. *IEEE Access*, 12:94477–94495, 2024. https://doi.org/10.1109/ACCESS.2024.3416812.

[GM13] J. Götzfried and T. Müller. Fast software encryption with SIMD (how to speed up symmetric block ciphers with the AVX/AVX2 instruction set). In *European Workshop on System Security (EUROSEC)*, 2013.

[GM23] C.G.D.A. Gewehr and F.G. Moraes. Improving the efficiency of cryptography algorithms on resource-constrained embedded systems via RISC-V instruction set extensions. In *SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2023. https://doi.org/10.1109/SBCCI60457.2023.10261964.

[GMPP20] S. Gao, B. Marshall, D. Page, and T.H. Pham. FENL: an ISE to mitigate analogue micro-architectural leakage. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2020(2):73–98, 2020. https://doi.org/10.13154/tches.v2020.i2.73-98.

[Gon00] R.E. Gonzalez. Xtensa: A configurable and extensible processor. *IEEE Micro*, 20(2):60–70, 2000. https://doi.org/10.1109/40.848473.

[GP12] H. Großand T. Plos. On using instruction-set extensions for minimizing the hardware-implementation costs of symmetric-key algorithms on a low-resource microcontroller. In *Radio Frequency Identification: Security and Privacy Issues (RFIDSec)*, LNCS 7739, pages 149–164. Springer-Verlag, 2012. https://doi.org/10.1007/978-3-642-36140-1_11.

[GPS08] S.D. Galbraith, K.G. Paterson, and N.P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156:3113–3121, 2008. https://doi.org/10.1016/j.dam.2007.12.010.

[Gro02] J. Großschädl. Instruction set extension for long integer modulo arithmetic on RISC-based smart cards. In *Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 13–19, 2002. https://doi.org/10.1109/CAHPC.2002.1180754.

[Gro03] J. Großschädl. Architectural support for long integer modulo arithmetic on RISC-based smart cards. *International Journal of High Performance Computing Applications (IJHPCA)*, 17(2):135–146, 2003. https://doi.org/10.1177/1094342003017002004.

[GS66] W.M. Gentleman and G. Sande. Fast fourier transforms: for fun and profit. In *American Federation of Information Processing Societies (AFIPS)*, volume 29, pages 563–578, 1966. https://doi.org/10.1145/1464291.1464352.

[GS04] J. Großschädl and E. Savaş. Instruction set extensions for fast arithmetic in finite fields $GF(p)$ and $GF(2^m)$. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 3156, pages 133–147. Springer-Verlag, 2004. https://doi.org/10.1007/978-3-540-28632-5_10.

[GTS07] J. Großschädl, S. Tillich, and A. Szekely. Performance evaluation of instruction set extensions for long integer modular arithmetic on a SPARC V8 processor. In *Euromicro Conference on Digital System Design (DSD)*, pages 680–689, 2007. https://doi.org/10.1109/DSD.2007.4341542.

[Gue09] S. Gueron. Intel's new AES instructions for enhanced performance and security. In *Fast Software Encryption (FSE)*, LNCS 5665, pages 51–66. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-03317-9_4.

[GYH18] Q. Ge, Y. Yarom, and G. Heiser. No security without time protection: we need a new hardware-software contract. In *Asia-Pacific Workshop on Systems (APSys)*, 2018. https://doi.org/10.1145/3265723.3265724.

[HCP+16] M.D. Hill, D. Christie, D. Patterson, J.J. Yi, D. Chiou, and R. Sendag. Proprietary versus open instruction sets. *IEEE Micro*, 36(4):58–68, 2016. https://doi.org/10.1109/MM.2016.61.

[Hei18] G. Heiser. For safety's sake: We need a new hardware-software contract! *IEEE Design & Test*, 35(2):27–30, 2018. https://doi.org/10.1109/MDAT.2017.2766559.

[HGTW05] S. Hines, J. Green, G. Tyson, and D. Whalley. Improving program efficiency by packing instructions into registers. In *International Symposium on Computer Architecture (ISCA)*, pages 260–271, 2005. https://doi.org/10.1109/ISCA.2005.32.

[Hil08] Y. Hilewitz. *Advanced Bit Manipulation Instructions: Architecture, Implementation And Applications*. PhD thesis, Princeton University, 2008.

[HKM12] M. Hamburg, P.C. Kocher, and M.E. Marson. Analysis of Intel's Ivy Bridge digital random number generator. Cryptography Research, Inc., 2012.

[HL08] Y. Hilewitz and R.B. Lee. Fast bit gather, bit scatter and bit permutation instructions for commodity microprocessors. *Journal of Signal Processing Systems*, 53(1-2):145–169, 2008. https://doi.org/10.1007/s11265-008-0212-8.

[HLL17] J. van der Hoeven, R. Larrieu, and G. Lecerf. Implementing fast carryless multiplication. In *Mathematical Aspects of Computer and Information Sciences (MACIS)*, LNCS 10693, pages 121–136. Springer-Verlag, 2017. https://doi.org/10.1007/978-3-319-72453-9_9.

[HV11] A. Hakkala and S. Virtanen. Accelerating cryptographic protocols: A review of theory and technologies. In *Communication Theory, Reliability, and Quality of Service (CTRQ)*, pages 103–109, 2011.

[HYL08] Y. Hilewitz, Y.L. Yin, and R.B. Lee. Accelerating the Whirlpool hash function using parallel table lookup and fast cyclical permutation. In *Fast Software Encryption (FSE)*, LNCS 5086, pages 173–188. Springer-Verlag, 2008. https://doi.org/10.1007/978-3-540-71039-4_11.

[IBM15] IBM. Power isa version 3.0, 2015.

[IL07] P. Ienne and R. Leupers, editors. *Customizable Embedded Processors*. Systems on Silicon. Morgan Kaufmann, 2007.

[Int18] Intel. Intel digital random number generator (DRNG), software implementation guide. Revision 2.1, 2018. https://www.intel.com/content/dam/develop/external/us/en/documents/drng-software-implementation-guide-2-1-185467.pdf.

[Int22a] Intel 64 and IA-32 architectures – software developer's manual (volume 1: Basic architecture). Technical Report 253665-077US, Intel Corp., 2022. http://software.intel.com/en-us/articles/intel-sdm.

[Int22b] Intel 64 and IA-32 architectures – software developer's manual (volume 2: Instruction set reference a-z). Technical Report 325383-077US, Intel Corp., 2022. http://software.intel.com/en-us/articles/intel-sdm.

[ITAO20] E.N. Işman, C. Topal, L. Akçay, and B. Örs. Instruction extension of an open source RV32IMC core for NTRU cryptosystem. In *European Conference on Circuit Theory and Design (ECCTD)*, pages 1–5, 2020. https://doi.org/10.1109/ECCTD49232.2020.9218372.

[JSG10] C. Jenkins, M. Schulte, and J. Glossner. Instruction set extensions for the advanced encryption standard on a multithreaded software defined radio platform. *International Journal of High Performance Systems Architecture (IJHPSA)*, 2(3/4):203–214, 2010. https://doi.org/10.1504/IJHPSA.2010.034541.

[KAMS19] B. Koppelmann, P. Adelt, W. Mueller, and C. Scheytt. RISC-V extensions for bit manipulation instructions. In *Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 41–48, 2019. https://doi.org/10.1109/PATMOS.2019.8862170.

[KCS23] P. Kiaei, T. Conroy, and P. Schaumont. Architecture support for bitslicing. *IEEE Transactions on Emerging Topics in Computing*, 11:497–510, 2023. https://doi.ieeecomputersociety.org/10.1109/TETC.2022.3215480.

[KGM21] Y.-M. Kuo, F. Garcia-Herrero, and J.A. Maestro. Versatile RISC-V ISA Galois field arithmetic extension for cryptography and error-correction codes. In *Computer Architecture Research with RISC-V (CARRV)*, 2021. https://carrv.github.io/2021.

[KGRM23] Y.-M. Kuo, F. García-Herrero, O. Ruano, and J.A. Maestro. RISC-V Galois field ISA extension for non-binary error-correction codes and classical and post-quantum cryptography. *IEEE Transactions on Computers*, 72:682–692, 2023. https://doi.org/10.1109/TC.2022.3174587.

[KKRM13] S. Khurana, S. Kolay, C. Rebeiro, and D. Mukhopadhyay. Lightweight cipher implementations on embedded processors. In *Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 82–87, 2013. https://doi.org/10.1109/DTIS.2013.6527783.

[KLA+19] K. Kiningham, P. Levis, M. Anderson, D. Boneh, M. Horowitz, and M. Shih. Falcon - a flexible architecture for accelerating cryptography. In *Mobile Ad Hoc and Sensor Systems (MASS)*, pages 136–144, 2019. https://doi.org/10.1109/MASS.2019.00025.

[KLS+23] M. Krausz, G. Land, F. Stolz, D. Naujoks, J. Richter-Brockmann, T. Güneysu, and L. Kogelheide. To extend or not to extend: Agile masking instructions for PQC. Cryptology ePrint Archive, Paper 2023/1287, 2023. https://eprint.iacr.org/2023/1287.

[KMD+20] P. Kiaei, D. Mercadier, P.-E. Dagand, K. Heydemann, and P. Schaumont. Custom instruction support for modular defense against side-channel and fault attacks. In *Constructive Side-Channel Analysis and Secure Design (COSADE)*, LNCS 12244, pages 221–253. Springer-Verlag, 2020. https://doi.org/10.1007/978-3-030-68773-1_11.

[KMPZ95] L. Kohn, G. Maturana, A. Prabhu, and G. Zyner. The visual instruction set (VIS) in UltraSPARC. In *Technologies for the Information Superhighway (COMPCON)*, pages 462–469, 1995. https://doi.org/10.1109/CMPCON.1995.512423.

[Knu11] D.E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*, volume 4A. Addison Wesley, 1 edition, 2011.

[KP04] S.S. Kumar and C. Paar. Reconfigurable instruction set extension for enabling ECC on an 8-bit processor. In *Field Programmable Logic and Application (FPL)*, LNCS 3203, pages 586–595. Springer-Verlag, 2004. https://doi.org/10.1007/978-3-540-30117-2_60.

[KS09] E. Käsper and P. Schwabe. Faster and timing-attack resistant AES-GCM. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 5747, pages 1–17. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-04138-9_1.

[KS11] W. Killmann and W. Schindler. A proposal for functionality classes for random number generators. AIS 20 / AIS 31, Version 2.0. Bundesamt für Sicherheit in der Informationstechnik (BSI), 2011. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf.

[KS20] P. Kiaei and P. Schaumont. Domain-oriented masked instruction set architecture for RISC-V. Cryptology ePrint Archive, Report 2020/465, 2020. https://eprint.iacr.org/2020/465.

[LC08] S.H. Lee and L. Choi. Accelerating symmetric and asymmetric ciphers with register file extension for multi-word and long-word operation. In *International Conference on Information Science and Security (ICISS)*, pages 102–107, 2008. https://doi.org/10.1109/ICISS.2008.32.

[LC10] R.B. Lee and Y.-Y. Chen. Processor accelerator for AES. In *Symposium on Application Specific Processors (SASP)*, pages 16–21, 2010. https://doi.org/10.1109/SASP.2010.5521153.

[LCW18] Y. Liu, R.C.C. Cheung, and H. Wong. Lightweight secure processor prototype on FPGA. In *Field Programmable Logic and Applications (FPL)*, pages 443–444, 2018. https://doi.org/10.1109/FPL.2018.00081.

[Lee95] R.B. Lee. Accelerating multimedia with enhanced micro-processors. *IEEE Micro*, 15(2):22–32, 1995. https://doi.org/10.1109/40.372347.

[Lee96] R.B. Lee. Subword parallelism with MAX-2. *IEEE Micro*, 16(4):51–59, 1996. https://doi.org/10.1109/40.526925.

[Lee03] R.B. Lee. Challenges in the design of security-aware processors. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 1–2, 2003. https://doi.org/10.1109/ASAP.2003.1212824.

[LF05] R.B. Lee and A.M. Fiskiran. PLX: An instruction set architecture and testbed for multimedia information processing. *VLSI Signal Processing Systems for Signal, Image and Video Technology*, 40(1):85–108, 2005. https://doi.org/10.1007/s11265-005-4940-8.

[LH96] R.B. Lee and J. Huck. 64-bit and multimedia extensions in the PA-RISC 2.0 architecture. In *Technologies for the Information Superhighway (COMPCON)*, pages 152–160, 1996. https://doi.org/10.1109/CMPCON.1996.501762.

[LHP20] T. Li, B. Hopkins, and S. Parameswaran. SIMF: Single-instruction multiple-flush mechanism for processor temporal isolation. *CoRR*, abs/2011.10249, 2020. https://arxiv.org/abs/2011.10249.

[LM90] X. Lai and J.L. Massey. A proposal for a new block encryption standard. In *Advances in Cryptology (EUROCRYPT)*, LNCS 473, pages 389–404. Springer-Verlag, 1990. https://doi.org/10.1007/3-540-46877-3_35.

[LMP22] H. Li, N. Mentens, and S. Picek. A scalable SIMD RISC-V based processor with customized vector extensions for CRYSTALS-kyber. In *Design Automation Conference (DAC)*, pages 733–738, 2022. https://doi.org/10.1145/3489517.3530552.

[LMP23] H. Li, N. Mentens, and S. Picek. Maximizing the potential of custom RISC-V vector extensions for speeding up SHA-3 hash functions. In *Design, Automation, and Test in Europe (DATE)*, pages 1–6. IEEE, 2023. https://doi.org/10.23919/DATE56975.2023.10137009.

[LN16] P. Longa and M. Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In S. Foresti and G. Persiano, editors, *Cryptology and Network Security (CANS)*, LNCS 10052, pages 124–139. Springer-Verlag, 2016. https://doi.org/10.1007/978-3-319-48965-0_8.

[LPAF+18] J. Lowe-Power, V. Akella, M.K. Farrens, S.T. King, and C.J. Nitta. A case for exposing extra-architectural state in the ISA. In *Hardware and Architectural Support for Security and Privacy (HASP)*, pages 8:1–8:6, 2018. https://doi.org/10.1145/3214292.3214300.

[LQYW24] L. Li, G. Qin, Y. Yu, and W. Wang. Compact instruction set extensions for Kyber. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(3):756–760, 2024. https://doi.org/10.1109/TCAD.2023.3327104.

[LSY01] R.B. Lee, Z. Shi, and X. Yang. Efficient permutation instructions for fast software cryptography. *IEEE Micro*, 21(6):56–69, 2001. https://doi.org/10.1109/40.977759.

[LSY+04] R.B. Lee, Z. Shi, Y.L. Yin, R.L. Rivest, and M.J.B. Robshaw. On permutation operations in cipher design. In *Information Technology: Coding and Computing (ITCC)*, volume 2, pages 569–577, 2004. https://doi.org/10.1109/ITCC.2004.1286714.

[LT23] F. Lozachmeur and A. Tisserand. A RISC-V instruction set extension for flexible hardware/software protection of cryptosystems masked at high orders. In *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 360–364, 2023. https://doi.org/10.1109/MWSCAS57524.2023.10405991.

[LTQ+24] L. Li, Q. Tian, G. Qin, S. Chen, and W. Wang. Compact instruction set extensions for Dilithium. *ACM Transactions on Embedded Computing Systems*, 23(2):23:1–23:21, 2024. https://doi.org/10.1145/3643826.

[LWDZ15] W. Liao, M. Wan, K. Dai, and X. Zou. Scalable instruction set extension for dual-field public-key cryptosystem. In *International MultiConference of Engineers and Computer Scientists (IMECS)*, pages 129–133, 2015.

[LYS04] R.B. Lee, X. Yang, and Z. Shi. Validating word-oriented processors for bit and multi-word operations. In *Advances in Computer Systems Architecture (ACSAC)*, LNCS 3189, pages 473–488. Springer-Verlag, 2004. https://doi.org/10.1007/978-3-540-30102-8_40.

[LYS05] R.B. Lee, X. Yang, and Z. Shi. Single-cycle bit permutations with MOMR execution. *Journal of Computer Science Technology*, 20(5):577–585, 2005. https://doi.org/10.1007/s11390-005-0577-0.

[MA07] D. Montgomery and A. Akoglu. Methodology and toolset for ASIP design and development targeting cryptography-based applications. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 365–370, 2007. https://doi.org/10.1109/ASAP.2007.4459291.

[Man11] R. Manley. *Program generation for Intel AES new instructions*. PhD thesis, Trinity College Dublin, 2011.

[May09] D. May. The XMOS XS1 architecture. Technical report, XMOS Ltd., 2009. http://www.xmos.com/published/xmos-xs1-architecture.

[MBB+23] K. Miteloudi, J. Bos, O. Bronchain, B. Fay, and J. Renes. PQ.V.ALU.E: Post-quantum RISC-V custom ALU extensions on dilithium and kyber. Cryptology ePrint Archive, Paper 2023/1505, 2023. https://eprint.iacr.org/2023/1505.

[MD07] S. Majzoub and H. Diab. Instruction-set extension for cryptographic applications on reconfigurable platform. *Journal of Circuits, Systems and Computers*, 16(6):911–927, 2007. https://doi.org/10.1142/S0218126607004076.

[MIPS01b] MIPS32 architecture for programmers, volume II: The MIPS32 instruction set. Technical report, MIPS Technologies, Inc., 2001.

[ML01] J.P. McGregor and R.B. Lee. Architectural enhancements for fast subword permutations with repetitions in cryptographic applications. In *International Conference on Computer Design (ICCD)*, pages 453–461, 2001. https://doi.org/10.1109/ICCD.2001.955065.

[MM12] S. Matsuda and S. Moriai. Lightweight cryptography for the cloud: Exploit the power of bitslice implementation. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 7428, pages 408–425. Springer-Verlag, 2012. https://doi.org/10.1007/978-3-642-33027-8_24.

[MNP+21] B. Marshall, G.R. Newell, D. Page, M.-J.O. Saarinen, and C. Wolf. The design of scalar AES instruction set extensions for RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021(1):109–136, 2021. https://doi.org/10.46586/tches.v2021.i1.109-136.

[MOP07] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007. https://doi.org/10.1007/978-0-387-38162-6.

[MOV96] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MP21] B. Marshall and D. Page. SME: Scalable Masking Extensions. Cryptology ePrint Archive, Paper 2021/1416, 2021. https://eprint.iacr.org/2021/1416.

[MPC00] L. May, L. Penna, and A. Clark. An implementation of bitsliced DES on the Pentium MMX^{TM} processor. In *Australasian Conference on Information Security and Privacy (ACISP)*, LNCS 1841, pages 112–122. Springer-Verlag, 2000. https://doi.org/10.1007/10718964_10.

[MPP20] B. Marshall, D. Page, and T.H. Pham. Implementing the draft RISC-V scalar cryptography extensions. In *Hardware and Architectural Support for Security and Privacy (HASP)*, 2020. https://doi.org/10.1145/3458903.3458904.

[MPP21] B. Marshall, D. Page, and T. Pham. A lightweight ISE for ChaCha on RISC-V. In *Application-specific Systems, Architectures and Processors (ASAP)*, pages 25–32. IEEE, 2021. https://doi.org/10.1109/ASAP52443.2021.00011.

[NDZ+21] P. Nannipieri, S. Di Matteo, L. Zulberti, F. Albicocchi, S. Saponara, and L. Fanucci. A RISC-V post quantum cryptography instruction set extension for number theoretic transform to speed-up CRYSTALS algorithms. *IEEE Access*, 9:150798–150808, 2021. https://doi.org/10.1109/ACCESS.2021.3126208.

[NIK04] K. Nadehara, M. Ikekawa, and I. Kuroda. Extended instructions for the AES cryptography and their efficient implementation. In *Signal Processing Systems (SIPS)*, pages 152–157, 2004. https://doi.org/10.1109/SIPS.2004.1363041.

[Nis21] G. Nishandji. Performance evaluation of cryptographic algorithms in software and with hardware implementation of specialized instructions for the RISC-V instruction set architecture. MSc thesis, Northern Arizona University, 2021. https://openknowledge.nau.edu/id/eprint/5648.

[NIST01] Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197, 2001. http://csrc.nist.gov.

[NIST07] Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards and Technology (NIST) Special Publication 800-38D, 2007. http://csrc.nist.gov.

[NIST15] Secure Hash Standard (SHS). National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 180-4, 2015. http://csrc.nist.gov.

[NIST99] Data Encryption Standard (DES). National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 46-3, 1999. http://csrc.nist.gov.

[NOOS95] E. Nahum, S. O'Malley, H. Orman, and R. Schroeppel. Towards high performance cryptographic software. In *High Performance Communication Subsystems (HPCS)*, pages 69–72, 1995. https://doi.org/10.1109/HPCS.1995.662009.

[NRE+12] K.D. Nima, E.A. Reza, A. Erfan, T. Ahmad, R. Mahsa, and M. Mina. CIARP: Crypto instruction-aware RISC processor. In *IEEE Symposium on Computers & Informatics (ISCI)*, pages 208–213, 2012. https://doi.org/10.1109/ISCI.2012.6222696.

[NST10] A. Nohl, F. Schirrmeister, and D. Taussig. Application specific processor design: Architectures, design methods and tools. In *International Conference on Computer-Aided Design (ICCAD)*, pages 349–352, 2010. https://doi.org/10.1109/ICCAD.2010.5653632.

[OE08] S. O'Melia and A.J. Elbirt. Instruction set extensions for enhancing the performance of symmetric-key cryptography. In *Annual Computer Security Applications Conference (ACSAC)*, pages 465–474, 2008. https://doi.org/10.1109/ACSAC.2008.10.

[OE10] S. O'Melia and A.J. Elbirt. Enhancing the performance of symmetric-key cryptography via instruction set extensions. *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, 18(11):1505–1518, 2010. https://doi.org/10.1109/TVLSI.2009.2025171.

[OFP+24] F. Oberhansl, T. Fritzmann, T. Pöppelmann, D.B. Roy, and G. Sigl. Uniform instruction set extensions for multiplications in contemporary and post-quantum cryptography. *Journal of Cryptographic Engineering (JCEN)*, 14(1):1–18, 2024. https://doi.org/10.1007/s13389-023-00332-2.

[OFW99] S. Oberman, G. Favor, and F. Weber. AMD 3DNow! technology: architecture and implementations. *IEEE Micro*, 19(2):37–48, 1999. https://doi.org/10.1109/40.755466.

[Ope24] OpenPOWER. Power isa version 3.1 c, 2024. https://files.openpower.foundation/s/9izgC5Rogi5Ywmm.

[Osv03] D.A. Osvik. *Efficient Implementation of the Data Encryption Standard*. Candidatus scientiarum, Department of Informatics, Universitas Bergensis, 2003.

[Paa02] C. Paar. The future of the art of cryptographic implementations. In *STORK Cryptography Workshop*, 2002. http://www.stork.eu.org/program.html.

[Pow18] Power ISA. Technical Report 2.07 B, IBM, 2018. https://ibm.ent.box.com/s/jd5w15gz301s5b5dt375mshpq9c3lh4u.

[PS81] D.A. Patterson and C.H. Séquin. RISC I: A reduced instruction set VLSI computer. In *International Symposium on Computer Architecture (ISCA)*, pages 216–230, 1981. https://doi.org/10.1145/285930.285981.

[PS82] D.A. Patterson and C.H. Séquin. A VLSI RISC. *IEEE Computer*, 15(9):8–21, 1982. https://doi.org/10.1109/MC.1982.1654133.

[PSP08] C. Puttmann, J. Shokrollahi, and M. Porrmann. Resource efficiency of instruction set extensions for elliptic curve cryptography. In *Information Technology New Generations (ITNG)*, pages 131–136, 2008. https://doi.org/10.1109/ITNG.2008.130.

[PW96] A. Peleg and U. Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4):42–50, 1996. https://doi.org/10.1109/40.526924.

[Raw16] H.K. Rawat. Vector instruction set extensions for efficient and reliable computation of Keccak. MSc thesis, Virginia Polytechnic Institute and State University, 2016. https://vtechworks.lib.vt.edu/bitstream/handle/10919/72857/Rawat_HK_T_2016.pdf.

[RCS+09] F. Regazzoni, A. Cevrero, F.-X. Standaert, S. Badel, T. Kluter, P. Brisk, Y. Leblebici, and P. Ienne. A design flow and evaluation framework for DPA-resistant instruction set extensions. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 5747, pages 205–219. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-04138-9_15.

[REP+09] F. Regazzoni, T. Eisenbarth, A. Poschmann, J. Großschädl, F. Gurkaynak, M. Macchetti, Z. Toprak, L. Pozzi, C. Paar, Y. Leblebici, and P. Ienne. Evaluating resistance of MCML technology to power analysis attacks using a simulation-based methodology. In *Transactions on Computational Science IV*, LNCS 5430, pages 230–243. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-01004-0_13.

[Res18] E. Rescorla. The Transport Layer Security (TLS) protocol version 1.3. Internet Engineering Task Force (IETF) Request for Comments (RFC) 8446, 2018. http://tools.ietf.org/html/rfc8446.

[RI16] F. Regazzoni and P. Ienne. Instruction set extensions for secure applications. In *Design, Automation, and Test in Europe (DATE)*, pages 1529–1534, 2016.

[RIS24] RISC-V. The RISC-V instruction set manual volume I: Unprivileged architecture. Ratified ISA Release 20240411, RISC-V International, 2024. https://github.com/riscv/riscv-isa-manual/releases/download/20240411/unpriv-isa-asciidoc.pdf.

[Riv94] R.L. Rivest. The RC5 encryption algorithm. In *Fast Software Encryption (FSE)*, LNCS 1008, pages 86–96. Springer-Verlag, 1994. https://doi.org/10.1007/3-540-60590-8_7.

[RKL+04] S. Ravi, P.C. Kocher, R.B. Lee, G. McGraw, and A. Raghunathan. Security as a new dimension in embedded system design. In *Design Automation Conference (DAC)*, pages 753–760, 2004. https://doi.org/10.1145/996566.996771.

[RRKH04] S. Ravi, A. Raghunathan, P.C. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computer Systems*, 3(3):461–491, 2004. https://doi.org/10.1145/1015047.1015049.

[RRSY98] R.L. Rivest, M.J.B. Robshaw, R. Sidney, and Y.L. Yin. The RC6 block cipher, 1998. http://people.csail.mit.edu/rivest/pubs/RRSY98.pdf.

[RS16] H.K. Rawat and P. Schaumont. SIMD instruction set extensions for Keccak with applications to SHA-3, Keyak and Ketje. In *Hardware and Architectural Support for Security and Privacy (HASP)*, pages 1–8, 2016. https://doi.org/10.1145/2948618.2948622.

[RV19] The RISC-V instruction set manual. Technical Report Volume I: User-Level ISA (version 20190608-Base-Ratified), 2019. http://riscv.org/specifications.

[RVB21] RISC-V bit-manipulation ISA-extensions (version 1.0.0). Technical report, 2021. https://github.com/riscv/riscv-bitmanip.

[RVK21] RISC-V cryptographic extension proposals. Technical Report Volume II: Vector Instructions (version 0.7.0), 2021. https://github.com/riscv/riscv-crypto.

[RVK22] RISC-V cryptographic extension proposals. Technical Report Volume I: Scalar & Entropy Source Instructions (version 1.0.1), 2022. https://github.com/riscv/riscv-crypto.

[SA10] M.I. Soliman and G. Y. Abozaid. FastCrypto: parallel AES pipeline extension for general-purpose processors. *Neural, Parallel, and Scientific Computations*, 18(1):47–58, 2010.

[Saa19] M.-J. O. Saarinen. SNEIK on microcontrollers: AVR, ARMv7-M, and RISC-V with custom instructions. Cryptology ePrint Archive, Report 2019/936, 2019. http://eprint.iacr.org/2019/936.

[Saa20] M.-J.O. Saarinen. A lightweight ISA extension for AES and SM4. 2020. https://ascslab.org/conferences/secriscv/program.html.

[SC14] G. Sayilar and D. Chiou. Cryptoraptor: high throughput reconfigurable cryptographic processor. In *International Conference on Computer-Aided Design (ICCAD)*, pages 155–161, 2014. https://doi.org/10.5555/2691365.2691398.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[SCS+09] N. Suarez, G.M. Callico, R. Sarmiento, O. Santana, and A.A. Abbo. Processor customization for software implementation of the AES algorithm for wireless sensor networks. In *Power and Timing Modeling, Optimization and Simulation (PATMOS)*, LNCS 5953, pages 326–335. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-11802-9_37.

[SFSG23] F. Stolz, M. Fyrbiak, P. Sasdrich, and T. Güneysu. Recommendation for a holistic secure embedded ISA extension, 2023. https://doi.org/10.1007/978-3-031-33491-7_3.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613, 1979. https://doi.org/10.1145/359168.359176.

[Shi04] Z. Shi. *Bit Permutation Instructions: Architecture, Implementation, And Cryptographic Properties*. PhD thesis, Princeton University, 2004.

[SL00] Z. Shi and R.B. Lee. Bit permutation instructions for accelerating software cryptography. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 138–148, 2000. https://doi.org/10.1109/ASAP.2000.862385.

[SL02] Z. Shi and R.B. Lee. Subword sorting with versatile permutation instructions. In *International Conference on Computer Design (ICCD)*, pages 234–241, 2002. https://doi.org/10.1109/ICCD.2002.1106776.

[SNM22] Markku-Juhani O. Saarinen, G. Richard Newell, and Ben Marshall. Development of the RISC-V entropy source interface. *J. Cryptogr. Eng.*, 12(4):371–386, 2022. https://doi.org/10.1007/s13389-021-00275-6.

[SP21] S. Steinegger and R. Primas. A fast and compact RISC-V accelerator for Ascon and friends. In *Smart Card Research and Advanced Applications (CARDIS)*, LNCS 12609, pages 53–67. Springer-Verlag, 2021. https://doi.org/10.1007/978-3-030-68487-7_4.

[SPA16] Oracle SPARC architecture 2011. Technical Report D1.0.0, Oracle Corp., 2016. https://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/140521-ua2011-d096-p-ext-2306580.pdf.

[SRH16] S. Saab, P. Rohatgi, and C. Hampel. Side-channel protections for cryptographic instruction set extensions. Cryptology ePrint Archive, Report 2016/700, 2016. https://eprint.iacr.org/2016/700.

[SS05] N.T. Slingerland and A.J. Smith. Multimedia extensions for general purpose microprocessors: a survey. *Microprocessors and Microsystems*, 29(5):225–246, 2005. https://doi.org/10.1016/j.micpro.2004.10.002.

[STK00] E. Savaš, A.F. Tenca, and Ç.K. Koç. A scalable and unified multiplier architecture for finite fields gf(p) and gf(2^m). In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 1965, pages 277–292. Springer-Verlag, 2000. https://doi.org/10.1007/3-540-44499-8_22.

[SYL03] Z. Shi, X. Yang, and R.B. Lee. Arbitrary bit permutations in one or two cycles. In *Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 237–247, 2003. https://doi.org/10.1109/ASAP.2003.1212847.

[SYL08] Z. Shi, X. Yang, and R.B. Lee. Alternative application-specific processor architectures for fast arbitrary bit permutations. *International Journal of Engineering and Science (IJES)*, 3(4):219–228, 2008. https://doi.org/10.1504/IJES.2008.022393.

[TBK+18] M.S. Turan, E. Barker, J. Kelsey, K.A. McKay, M.L. Baish, and M. Boyle. Recommendation for the entropy sources used for random bit generation. Special Publication SP 800-90B, NIST, 2018. https://doi.org/10.6028/NIST.SP.800-90B.

[TG04] S. Tillich and J. Großschädl. A simple architectural enhancement for fast and flexible elliptic curve cryptography over binary finite fields GF(2^m). In *Advances in Computer Systems Architecture (ACSAC)*, LNCS 3189, pages 282–295. Springer-Verlag, 2004. https://doi.org/10.1007/978-3-540-30102-8_24.

[TG05] S. Tillich and J. Großschädl. Accelerating AES using instruction set extensions for elliptic curve cryptography. In *International Conference Computational Science and Its Applications (ICCSA)*, volume 2 of *LNCS 3481*, pages 665–675. Springer-Verlag, 2005. https://doi.org/10.1007/11424826_70.

[TG06] S. Tillich and J. Großschädl. Instruction set extensions for efficient AES implementation on 32-bit processors. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 4249, pages 270–284. Springer-Verlag, 2006. https://doi.org/10.1007/11894063_22.

[TG07a] S. Tillich and J. Großschädl. Power analysis resistant AES implementation with instruction set extensions. In *Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 4727, pages 303–319. Springer-Verlag, 2007. https://doi.org/10.1007/978-3-540-74735-2_21.

[TG07b] S. Tillich and J. Großschädl. VLSI implementation of a functional unit to accelerate ECC and AES on 32-bit processors. In *Workshop on the Arithmetic of Finite Fields (WAIFI)*, LNCS 4547, pages 40–54. Springer-Verlag, 2007. https://doi.org/10.1007/978-3-540-73074-3_5.

[TGS05] S. Tillich, J. Großschädl, and A. Szekely. An instruction set extension for fast and memory-efficient AES implementation. In *Communications and Multimedia Security (CMS)*, LNCS 3677, pages 11–21. Springer-Verlag, 2005. https://doi.org/10.1007/11552055_2.

[TGS+19] E. Tehrani, T. Graba, A. Si-Merabet, S. Guilley, and J.-L. Danger. Classification of lightweight block ciphers for specific processor accelerated implementations. In *International Conference on Electronics, Circuits and Systems (ICECS)*, pages 747–750, 2019. https://doi.org/10.1109/ICECS46596.2019.8965156.

[TGSD20] E. Tehrani, T. Graba, A. Si-Merabet, and J.-L. Danger. RISC-V extension for lightweight cryptography. In *Euromicro Conference on Digital System Design (DSD)*, pages 222–228, 2020. https://doi.org/10.1109/DSD51259.2020.00045.

[TH99] S. Thakkar and T. Huff. Internet streaming SIMD extensions. *IEEE Computer*, 32(12):26–34, 1999. https://doi.org/10.1109/2.809248.

[THM07] S. Tillich, C. Herbst, and S. Mangard. Protecting AES software implementations on 32-bit processors against power analysis. In *Applied Cryptography and Network Security (ACNS)*, LNCS 4521, pages 141–157. Springer-Verlag, 2007. https://doi.org/10.1007/978-3-540-72738-5_10.

[Til08] S. Tillich. *Instruction Set Extensions for Support of Cryptography on Embedded Systems*. PhD thesis, 2008.

[TKS10] S. Tillich, M. Kirschbaum, and A. Szekely. SCA-resistant embedded processors: The next generation. In *Annual Computer Security Applications Conference (ACSAC)*, pages 211–220, 2010. https://doi.org/10.1145/1920261.1920293.

[TSP09] D. Theodoropoulos, A. Siskos, and D. Pnevmatikatos. CCproc: A custom VLIW cryptography co-processor for symmetric-key ciphers. In *Applied Reconfigurable Computing: Architectures, Tools and Applications (ARC)*, pages 318–323. Springer-Verlag, 2009. https://doi.org/10.1007/978-3-642-00641-8_35.

[UK24] H. Uzuner and E.B. Kavun. NLU-V: A family of instruction set extensions for efficient symmetric cryptography on RISC-V. *Cryptography*, 8(1), 2024. https://doi.org/10.3390/cryptography8010009.

[VCS22] C. Verhamme, G. Cassiers, and F.-X. Standaert. Analyzing the leakage resistance of the NIST's lightweight crypto competition's finalists. In *Smart Card Research and Advanced Applications (CARDIS)*, LNCS 13820, pages 290–308. Springer-Verlag, 2022. https://doi.org/10.1007/978-3-031-25319-5_15.

[VPG07] T. Vejda, D. Page, and J. Großschädl. Instruction set extensions for pairing-based cryptography. In *Pairing-Based Cryptography (Pairing)*, LNCS 4575, pages 208–224. Springer-Verlag, 2007. https://doi.org/10.1007/978-3-540-73489-5_11.

[VSI+19] A. Varici, G. Saglam, S. Ipek, A. Yildiz, S. Gören, A. Aysu, D. Iskender, T.B. Aktemur, and H.F. Ugurdag. Fast and efficient implementation of lightweight crypto algorithm PRESENT on FPGA through processor instruction set extension. In *East-West Design & Test Symposium (EWDTS)*, pages 1–5, 2019. https://doi.org/10.1109/EWDTS.2019.8884397.

[War12] H.S. Warren Jr. *Hackers Delight*. Addison Wesley, 2 edition, 2012.

[Wat16] A. Waterman. *Design of the RISC-V Instruction Set Architecture*. PhD thesis, University of California at Berkeley, 2016. https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf.

[WSG+20] N. Wistoff, M. Schneider, F.K. Gürkaynak, L. Benini, and G. Heiser. Prevention of microarchitectural covert channels on an open-source 64-bit RISC-V core. In *Computer Architecture Research with RISC-V (CARRV)*, 2020. https://carrv.github.io/2020.

[WSWH15] Y. Wang, Y. Shi, C. Wang, and Y. Ha. FPGA-based SHA-3 acceleration on a 32-bit processor via instruction set extension. In *Electron Devices and Solid-State Circuits (EDSSC)*, pages 305–308, 2015. https://doi.org/10.1109/EDSSC.2015.7285111.

[WWA01] L. Wu, C. Weaver, and T. Austin. CryptoManiac: A fast flexible architecture for secure communication. In *International Symposium on Computer Architecture (ISCA)*, pages 110–119, 2001. https://doi.org/10.1109/ISCA.2001.937439.

[YHHF19] J. Yu, L. Hsiung, M. El Hajj, and C.W. Fletcher. Data oblivious ISA extensions for side channel-resistant and high performance computing. In *Network and Distributed System Security Symposium (NDSS)*, 2019. https://www.ndss-symposium.org/ndss-paper/data-oblivious-isa-extensions-for-side-channel-resistant-and-high-performance-computing.

[YHMH19] X. Yang, Y. Hou, J. Ma, and H. He. CDSP: A solution for privacy and security of multimedia information processing in industrial big data and Internet of Things. *Sensors*, 19(3):556:1–556:16, 2019. https://doi.org/10.3390/s19030556.

[YL00] X. Yang and R.B. Lee. Fast subword permutation instructions using omega and flip network stages. In *International Conference on Computer Design (ICCD)*, pages 15–21, 2000. https://doi.org/10.1109/ICCD.2000.878264.

[YSZ+24] Z. Ye, R. Song, H. Zhang, D. Chen, R.C.C. Cheung, and K. Huang. A highly-efficient lattice-based post-quantum cryptography processor for iot applications. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2024(2):130–153, 2024. https://doi.org/10.46586/tches.v2024.i2.130-153.

[YYDZ08] X.-H. Yang, X.-R. Yu, Z.B. Dai, and Y.F. Zhang. Accelerated flexible processor architecture for crypto information. In *International Conference on Circuits and Systems for Communications (ICCSC)*, pages 628–632, 2008. https://doi.org/10.1109/ICCSC.2008.139.

[ZQL+23] J. Zhou, G. Qin, L. Li, C. Guo, and W. Wang. ISA extensions of shuffling against side-channel attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. https://doi.org/10.1109/TCAD.2023.3323165.

[ZSM19] D. Zagieboylo, G. Suh, and A. Myers. Using information flow to design an ISA that controls timing channels. In *Computer Security Foundations Symposium (CSF)*, 2019. https://doi.ieeecomputersociety.org/10.1109/CSF.2019.00026.