Scalar Crypto Arch Review Results

Ben and Richard, following is the aggregated Arch Review feedback from the members of the AR committee. Many of the items reflect required changes, while some raise concerns or questions for which there may be other overriding considerations that you all better appreciate.

Ch. 1

This chapter is fine for public review, but for the actual arch spec content to be incorporated into the Unpriv/Priv arch specs, sections 1.1 and 1.2 should be dropped and section 1.3 would become non-normative commentary about the guiding principles behind these extensions.

Ch. 2

To properly talk about misa.K, keep in mind that there is not a “cryptography extension” and instead there is this group of crypto extensions and there will be more crypto extensions in the future (as 1.3 notes). (Btw, using “ISA extensions” is more consistent with general RV terminology than “instruction set extensions”.)

The spec should say that misa.K covers all present and future RISC-V Crypto-related ISA extensions. And that misa.K is set if any crypto-related extension is implemented. Presumably this also includes the Zbk[bcx] extensions - which should be explicitly clarified. (Might it be worth also clarifying that clearing misa.K does not disable/hide all implemented crypto extensions as if they are not implemented?)

Detection of “fine-grained functionality” is misleading. The granularity of detection (aka discovery) is whole extensions.

Regarding the non-normative notes in 2.1, 2.2, and 2.3, keep in mind that there will not be two versions of copies of the Zbk[bcx] extensions. There should be no “inevitable divergences”. The Zk[bcx] specs created by Ben as additions to the doc that contains Zb[abcs] is the one golden arch reference and what is in this spec document, only for the sake of the ratification package and public review, is presumably an accurate temporary copy of Zb[abcs] and Zbk[bcx] instructions in the “new” BitManip spec.

Ch.3

Is it a deliberate choice that e.g. aes32esi (RV32) and aes64es (RV64) share the same opcode? This may be the first time something like this is done in the 32-bit encoding space. But this has

been done extensively in the 16-bit encoding space, so software tools already need to cope with this concept. So this is probably OK; just checking.

In the description of aes64ks1i, replace the entire last sentence with "The values 0xB..0xF are reserved." (The deleted text about illegal-instruction exceptions is implied by the definition of "reserved".)

We want to confirm that aes64ks2 is used frequently enough (relative to other instructions in Zkn) to justify its inclusion, given that it can be emulated with a 5-instruction sequence (srli + xor + srli + xor + pack).

The SAIL code for the aes32 instructions is parameterized on XLEN and uses 6-bit shift amounts, despite the fact that these are RV32-only instructions. By contrast, the SAIL code for the aes64 instructions is hardcoded to assume XLEN=64. This stylistic inconsistency needs to be resolved, perhaps by changing the SAIL code for the aes32 instructions to assume XLEN=32 (and, hence, to use a 5-bit shift amount).

The SAIL code for aes64ks1i includes an EXTZ, even though its argument is already 64 bits. Many other instructions omit the EXTZ, which seems defensible. Regardless of which is chosen, there should be consistency.

Reminder: The Zks/Zkn instructions that produce 32-bit results on RV64 should sign-extend their results.

For the instructions that are not included in both RV32 and RV64, this fact should be reflected in the "Included in" section at the end of each instruction page. For example, packw might say “Included in Zbkb (RV64 only)” or something like that.

Some instructions that are listed as being in Zbb are not, in fact, in Zbb--namely, pack, packh, and packw.

The funct3 field is misformatted ("0" instead of "000"), as are other fields that hold the value 0.

Ch. 4

In addition to defining CSRs sentropy and mnoise, Chapter 4 appears to assert multiple requirements on what software must do to satisfy cryptography standards NIST SP 800-90B and AIS-31 PTG.2, perhaps others. While we don’t dispute the veracity of the stated requirements for satisfying those standards, Zkr is defined as an extension of the RISC-V ISA, which specifies the boundary between hardware and software. Except in the interactions between hardware and software, it isn’t the role of the RISC-V ISA to say what software may or may not do. Hence, attempts to dictate what software must do to test and condition the values read from sentropy are mostly out of scope for Zkr.

We believe the guidance given for how to conform with NIST SP 800-90B and AIS-31 PTG.2 is valuable, but a clearer distinction should be made between this advice and exactly what Zkr requires of the hardware, the latter being within the purview of the RISC-V ISA. Large parts of Sections 4.4 (“Entropy Source Requirements”), 4.6.1 (“Hypervisor Trap and Emulate”), and 4.6.2 (“Direct S-Mode access”) probably should be moved to an appendix, maybe Appendix B or a separate appendix focusing solely on what driver software must additionally do to conform to the desired standards. Or, with some shuffling, certain subsections of Chapter 4 could be clearly delineated as advice beyond the usual scope of the ISA.

RISC-V International should be able some day to create a conformance test for Zkr that does not involve testing the behavior of any vendor-supplied “driver software”. Put another way, there should be a way to determine whether an implementation of the sentropy interface conforms to Zkr, independent of any specific driver software tied to the hardware. Ideally, the normative parts of Chapter 4 would leave readers understanding, in outline, what would be required of such a conformance test.

We do not see sufficient value in the pseudoinstructions pollentropy and getnoise. There’s no reason for the pollentropy pseudoinstruction based on the justification given in section 4.2, which says:

> Software should only access sentropy using the pseudo-instruction. This allows implementers to more easily create faster access paths to sentropy, without any of the more heavyweight logic associated with other CSR accesses.

Keep in mind that pollentropy would be a pseudoinstruction, i.e. just a mnemonic alias for a CSRRS instruction that reads the sentropy CSR. Hardware always only sees a CSR instruction. (This doesn’t prevent an implementation from recognizing CSR instructions that use this specific CSR number and optimizing the implementation.)

Regarding CSR mnoise and pseudoinstruction getnoise, there appears to be no way for these to be used by standard-conforming RISC-V software. Section 4.5 says:

> The optional GetNoise interface ... is intended for manufacturer tests and validation of physical entropy source modules. It must not be used as a source of randomness or for other production use. The contents and behavior of the register must be interpreted in the context of mvendorid, marchid, and mimpid CSR identifiers.

If that's true, there is no reason to standardize mnoise and getnoise. Vendors already have the freedom to define custom test interfaces via custom CSRs.

If the TG wants to keep it, section 4.5 and everything associated with NOISE_TEST = 1 should be moved into an appendix as non-normative / non-standard advice, with mnoise suggested as a custom CSR. So no standard CSR number is assigned for mnoise.

If an entropy source is shared among multiple harts, what is supposed to happen if the NOISE_TEST field of CSR mnoise is set to 1 at some harts and to 0 at other harts? Can one hart set NOISE_TEST to 1 and prevent all other harts from accessing entropy bits? For a shared entropy source, we believe implementers would definitely prefer to have a single memory-mapped register for the noise test feature, rather than mnoise at each hart. Would the risks of that really be any worse?

There‘s no need to strongly emphasize that reads of sentropy are non-blocking. Currently all CSR instructions are considered to be non-blocking. (It would be a very notable and possibly contentious first in the ISA to have a blocking CSR.) The current “non-blocking” note should just talk about the typical usage scenario for reading sentropy as being based on polling. Also, the note about “fast” paths should be eliminated or heavily reworded. It appears to improperly mix the idea of blocking with the idea of execution latency of CSR instructions.

The seed output from sentropy needs to be more explicitly defined to require that a given 16-bit seed is only returned once to software and that the next OPST=ES16 read of sentropy returns a new random 16-bit seed (unless that is not an intended requirement!?) Section 4.1 fails to say if multiple reads of sentropy with OPST=ES16 may keep returning the same seed value, and if not, why not. Appendix B, Section B.3.4, "Information Flows", says,

> To guarantee that no sensitive data is read twice and that
> different callers don’t get correlated output, it is suggested that
> hardware implements wipe-on-read on the randomness pathway during
> each read (successful poll).

We think this suggestion is too weak. Presumably the intended normal operational behavior is that sentropy reads either return OPST=ES16 with a “new” seed value, or return OPST=WAIT.

Assuming reads of the sentropy CSR have side-effects (by removing 16 entropy bits from the head of the FIFO), then the assigned CSR number for sentropy is tentatively 0xD60. [Whether standard RISC-V CSRs should be allowed to have side-effects on reads is currently being debated. Hopefully with a resolution soon. If the conclusion is not, then a R/W address will be allocated and the definition of sentropy adjusted accordingly.]

It appears to us that under certain conditions software might want to treat OPST=BIST the same as OPST=WAIT. That would be a little easier to do if the OPST codes for ES16 and WAIT were swapped, so:

00 = BIST, 01 = WAIT, 10 = ES16, 11 = DEAD

An RV64 program could then load the constant 0x80000000 into a register and test for sentropy >= 0x80000000 using BGE. An RV32 program could just test whether the sign bit of sentropy is set, using BGEZ. Although only a minor improvement, there doesn't appear to be any significance to the current ordering of bits 31 and 30 in sentropy, in which case swapping them should be harmless.

Use of WFI in the entropy polling loop is incorrect. WFI is permitted to stall the processor forever if no interrupt becomes pending; there's no guarantee that entropy becoming available will ever break the processor out of WFI. The PAUSE instruction is the more appropriate recommendation.

The Zkr spec should not prescribe the behavior of other bits of mseccfg. You can just delete the sentence beginning with "If Zkr is the only implemented feature".

More generally, since mseccfg is going to be shared by a growing set of multiple extensions over time, the plan is to move the definition of the overall mseccfg CSR to the Machine-Level chapter of the Priv 1.12 spec (as part of 1.12, not as an extension). Details to be decided, but that definition will define the layout of bit fields associated with various extensions and will cross-reference (ideally via hyper-links) the chapters for those extensions for the detailed individual bit definitions within each bit field.

So the Zkr spec should focus on defining its mseccfg bits and their bit positions. Assume for now that the current bit positions remain unchanged. This hopefully will also get resolved soon.

In Section 4.1, the phrase "Reserved for custom" should be replaced with "Designated for custom".

The reference to "S / HS" should be to "S/HS mode".

The footnote should refer to ePMP as "Enhanced PMP extension (Smepmp)". The current reference is bound to become stale when that extension gets incorporated into the Priv spec. Probably at most the chapter title might include something like Enhanced PMP, but not the current long phrase.

In Section 4.6, Table 1, the abbreviations "PE" and "GN" are not explained. Not every reader is going to readily realize that these are meant to be "pollentropy" and "getnoise". And, given earlier comments, these should now refer to sentropy and to mnoise.

The same section says the CSR address of mseccfg is 0x390, but it is currently 0x747. (And eventually, after public review, this number should only be specified in Priv 1.12 and not also in this extension.)

When section 4.6.1, "Hypervisor Trap and Emulate", says

> "The hypervisor (or M-mode elements) can trap and feed a less privileged guest virtual entropy source words."

it should say what causes the trap, presumably an attempted access to sentropy.
(By the way, the 'a' before 'less' was missing.)

That section also uses the unusual and non-standard term "S-mode instance", which presumably is supposed to mean a guest OS or a virtual machine. This needs to be suitably corrected.

Ch. 5

We recommend not making division half-constant time for now. If the use case becomes less hypothetical, a future extension can make division half-constant or full-constant time based on what is more clearly appropriate at that point.

In 5.4.3, delete C.SRLI64, C.SRAI64, and C.SLLI64. These are RV128 instructions, not RV64 instructions. C.SRLI, C.SRAI, and C.SLLI are available on both RV32 and RV64.

We are confused by the inclusion of the Zkr instructions in Zkt. The entropy source might not be ready, and so even if the instructions themselves are constant-time, the loop might run for a variable number of iterations, making the overall time spent in the polling loop non-constant. With that in mind, what is the harm in the instructions themselves not being constant time?

Other

The crypto extensions should avoid unnecessary statements about what other extensions require, such as these from appendix section A.4.2:

> Cores which also implement the Bit-manipulation extension must implement the complete grevi instruction as specified there, not the minimal subset described here.

> Cores which also implement the Bit-manipulation extension must implement the [un]shfli instruction as specified there, not the minimal subset described here

If the other extensions are edited, these sorts of statements can easily become incorrect and misleading. Further, there is not a BitManip extension. There is only a set of ratified (or soon to be ratified) bitmanip extensions. All other parts of the old BitManip spec are just proposals - which ratified extensions should not be referring to.

Section B.2.5 speaks of unidentified "instances". Instances of what?