RISC-V and Cryptography

Developing XCrypto: An Embedded Class Crypto ISE

Ben Marshall, Daniel Page, Thinh Pham

University of Bristol

Computer Science Department

Outline

- Context and terms
- Implementing Cryptography on RISC-V
- XCrypto Overview
- Overlaps between XCrypto, P and B Extensions
- Side channels
- Cryptography on RISC-V: Where next?

Context

- We are funded by the UK government (EPSRC Grant EP/R012288/1).
- All of our work is open source / free to use / MIT licensed / on Github.
- An overarching project goal is to develop a side-channel resiliant RISC-V CPU core.
- Along the way, we are doing lots of research into what decisions you can make at a (micro-)architectural level that help/hinder side channel resiliance.
- We have focused on “embedded” class / IoT cores.
- Concretely, we are concerned with RV32 as a base ISA.

Terms

- **Side channel**: measurements of a device / algorithms runtime, power consumption or electromagnetic energy emissions used to undermine a cryptographic construction.
- **Embedded CPU core**: Think ARM Cortex-M class, or anything with an IoT sticker. Probably doesn't have a cache. *Probably doesn't do macro-op fusion*.

Cryptography on RISC-V

The Bad News:

- The base ISA has a way to go to just be *on par* with peer ISAs.

The Good News:

- We know exactly what functionality is needed to put it back on par.
- We know what is needed to make it *better than the competition*.
  - Lots of this has been independently developed in other extensions.
  - XCrypto has some helpful functionality in it not present in other extensions.
- There is **a lot** of prior art in this area.
  - Academic conferences like CHES feature existing research which has been under-used or re-invented.

Implementing Crypto on RISC-V

- Considering the "general" or otherwise common RISC-V architecture strings:
  - Application Class: `RV32G`, `RV64G` (possibly with `C` too.)
  - Microcontrollers: `RV32IMC`
- These base architectures are deficient compared to ARM, MIPS in terms of performance, static and dynamic code density for most varieties of cryptographic workload.
- RISC-V puts the **R** in RISC.
  1. Lack of a rotate instruction.
  2. Limited memory addressing modes.
  3. Long arithmetic / integer carry support.
  4. Rigid adherence to 2-read-1-write.

1. Lack of a rotate instruction

Rotate is extremely popular in block ciphers and hashing algorithms.
ARX (Add, *Rotate*, XOR) constructions are common in light weight block ciphers designed for IoT devices.
By default, rotate is a 3-instruction sequence on RISC-V.
It's hard to overstate the impact of this:

	Cycle Count		
Algorithm	Without Rotate	With Bitmanip Rotate	Gain
SHA256	4205	3077	1.37x
ChaCha20	4149	2381	1.75x

As measured using Spike on RV64G, with(out) the draft B extension.
This will affect all RISC-V UNIX class systems going forward.

2. Limited addressing modes

- RISC-V does not support indexed addressing.
- This leads to *lots* of pointer arithmetic when stepping through arrays.
- Block ciphers/hash functions need to step non-linearly through state: AES/SHA3 are good examples.
  - In performance optimised crypto workloads for larger systems, you can unroll the loops and the problem goes away.
  - In embedded class cores, where loop unrolling is not an option, you see inner loops dominated by pointer arithmetic.
- Long-arithmetic also becomes dominated by pointer arithmetic.
- An argument against indexed load/store: the store variant requires three register read ports.
  - Counter argument: even the ARM M0+ has these sorts of instructions.

3. Long arithmetic support

- No carry detection in RISC-V, requires extra `sltu` instruction.
  - Lack of carry flags is nice sometimes, but this is the cost.
- ARM has `umlal` instructions: unsigned long multiply with accumulate. Present on ARM Cortex-M0 and up.
  - No equivilent on RISC-V
- Long arithmetic typically steps through three arrays at once: two sources and a destination.
  - Lack of indexed load means two extra instructions per inner loop iteration to increment pointers.

4. Rigid adherence to 2-read-1-write

- Having very simple encoding schemes is great for lots of reasons.
  - Lots of instructions only need to read two operands and write one result.
  - Fewer ports on a register file is nice, *but not as nice as it was 20+ years ago*.
- Lots of cryptographic functions are fundamentally reduction operations:
  - A plaintext *and* a key, reduced to a ciphertext.
  - N bytes hashed down to $M < N$ bytes.
- The more data you can feed to instructions, the faster reductions will run.
- At the ISA level: add more operands to your instructions.
- R.B. Lee, X. Yang, and Z. Shi. Validating Word-Oriented Processors for Bit and Multi-word Operations. In: Annual Computer Security Applications Conference (ACSAC). 2004, pp. 473488

Implementing Crypto on RISC-V: Summary

- Cryptographic workloads are an excellent way to identify the shortcommings of RISC-V when compared to similar architectures.
- These shortcomings are not always cryptography specific, though cryptography is disproportionatly affected.

XCrypto: our custom crypto extension to RISC-V

XCrypto is designed to:

- Address the shortcomings in RISC-V already described.
- See *what else* we could add to RISC-V to make it as efficient at executing cryptographic workloads as possible.
- Be used as a vehicle to research what *(micro-)architectural* decisions you can make to help/hinder side channel resiliance.
- Enable software based side channel countermeasures in a tighter performance budget.

XCrypto is not:

- A “drop in” embedded class crypto extension for RISC-V. (But we think it might guide the design of one...)

XCrypto: Key design choices and assumptions

- Doesn't use the V extension as a base.
- Tries to add RISC instructions with general utility over very efficient but specialised CISC instructions.
- Adds an extra 16-element, 32-bit wide register file.
  - Assumes 3 read ports and 2 write ports.
  - Additional instructions do compute on this register file.
  - Memory address arithmetic still done in the normal GPRs.
- Tries to maximise implementation options:
  - Multi-cycle v.s. pipelined.
  - Integrated data path v.s Separate *secure* datapath.
  - In turn, maximises opportunities to implement side-channel countermeasures while minimising their cost.

XCrypto: New Instructions

XCrypto includes instructions to address all of the problems we identified earlier:

- Rotate!
- Indexed and immediate offset Load/Store Unsigned Byte/Halfword/Word
  - All XCrypto load/store instructions have *update semantics*. This lets us load a register with bytes/halfwords directly without shift-and-or sequences.
- Multi-operand instructions for long-integer arithmetic.
  - Add/subtract, Multiply, Accumulate, Compare

XCrypto: New Instructions

General Purpose Instructions:

- **Randomness:** Seeding, sampling and quality checking of entropy source.
- **Memory:** Sub-word scatter/gather operations.
- **Bit:** permutations, bitfield insert/extract, lookup-tables
- **Packed:** SIMD-within-a-register (SWAR) arithmetic operations on 32/16/8/4/2 bit packed values.

Special Purpose Instructions:

- **AES:** lightweight AES sub-bytes and mix-columns acceleration
- **SHA3:** code-dense, energy efficient state index generation

XCrypto: New Instructions - Randomness

- A good source of randomness is essential for cryptography.
- Many side-channel countermeasures assume fast access to randomness.
- We don't specify how the randomness is generated, only that it is *cryptographically appropriate*.
- We only define an instruction based *interface* to a randomness source. That might be implemented as an alias for a CSR read, or memory mapped peripheral access.
- Instructions can seed a generator, sample the generator, and check if the generator thinks it currently has enough entropy to be useful.

XCrypto: New Instructions - Scatter/Gather

- These instructions split a register into bytes or halfwords, and use each byte/halfword as an offset into memory from a base address.
- Useful for code-dense S-Box implementation.
- Amenable to hiding based side-channel countermeasures if defined appropriately.
- Still allows for masked S-Boxes.

`xc.gather.b crd, rs1, crs2`:

$$
\begin{array}{l}
\textbf{begin} \\
\quad addr_0 \leftarrow \mathsf{GPR}[\texttt{rs1}] + \mathsf{XCR}[\texttt{crs2}]^8_0 \\
\quad addr_1 \leftarrow \mathsf{GPR}[\texttt{rs1}] + \mathsf{XCR}[\texttt{crs2}]^8_1 \\
\quad addr_2 \leftarrow \mathsf{GPR}[\texttt{rs1}] + \mathsf{XCR}[\texttt{crs2}]^8_2 \\
\quad addr_3 \leftarrow \mathsf{GPR}[\texttt{rs1}] + \mathsf{XCR}[\texttt{crs2}]^8_3 \\
\quad \mathsf{XCR}[\texttt{crd}]^8_0 \leftarrow \mathsf{MEM}[addr_0] \\
\quad \mathsf{XCR}[\texttt{crd}]^8_1 \leftarrow \mathsf{MEM}[addr_1] \\
\quad \mathsf{XCR}[\texttt{crd}]^8_2 \leftarrow \mathsf{MEM}[addr_2] \\
\quad \mathsf{XCR}[\texttt{crd}]^8_3 \leftarrow \mathsf{MEM}[addr_3] \\
\textbf{end}
\end{array}
$$

XCrypto: New Instructions - Bitwise

- Adds instructions for bit permutation, field insert/extract. All of which has been independently developed/added in the bitmanip extension.
- We also add support for very efficient 4-bit S-Box/LUTs and generalised ternary bitwise expressions.
- `xc.bop` (bit op) can implement any 3-variable bitwise expression. The example below is taken from SHA2.
- `xc.bop c0, c1, c2, 0xE8 // c0 = (c1&c2) | ((c1|c2) & c0)`

XCrypto: New Instructions - Packed Arithmetic

- We add the standard set of arithmetic operations:
  - Add/sub/multiply/shift/rotate
  - Plus carryless multiply.
- All of these work as a SIMD instruction, on data elements 32, 16, 8, 4, or 2 bits wide.
- This is similar / identical to what the (DS)P extension proposes.
- We support narrower bit-widths to accelerate bit-sliced or secret shared implementations, designed for side-channel resiliance.

XCrypto: New Instructions - AES

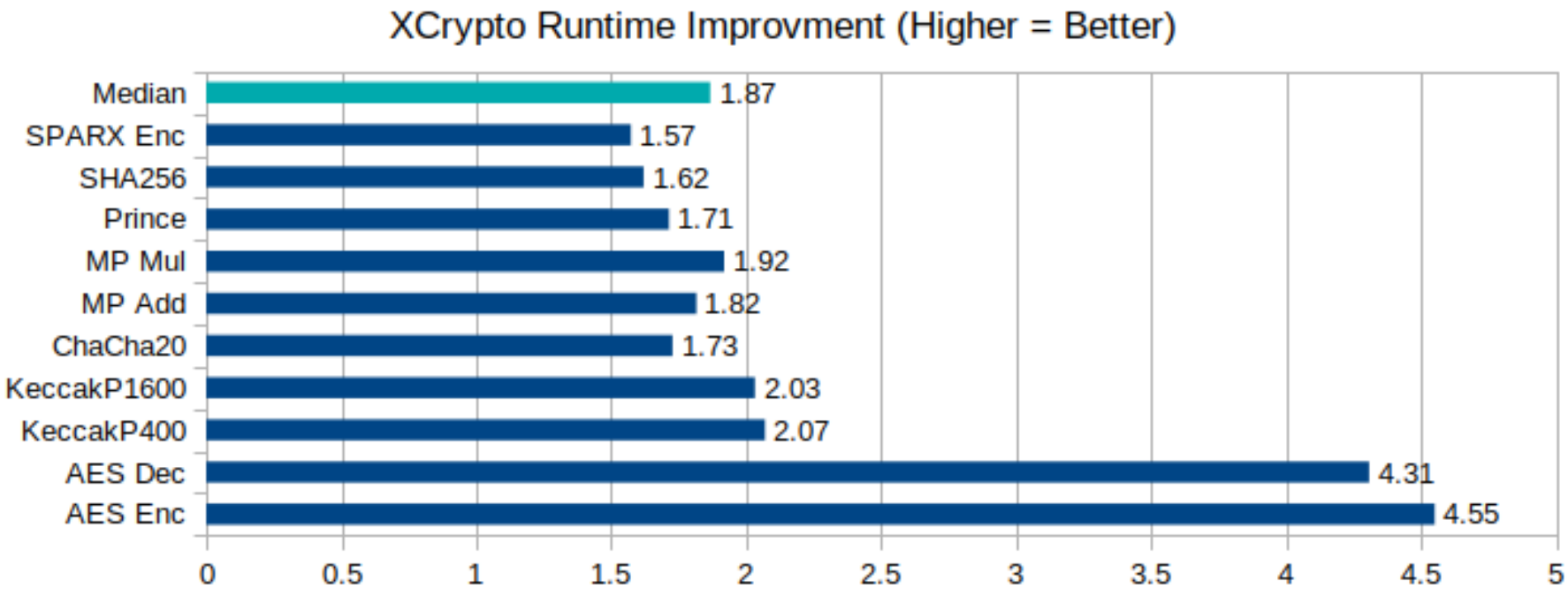
- Light weight acceleration of the sub-bytes and mix-columns steps of AES.
- Based on existing academic literature:
  - Tillich, Stefan, and Johann Groschdl. “Instruction set extensions for efficient AES implementation on 32-bit processors.” International workshop on cryptographic hardware and embedded systems. Springer, Berlin, Heidelberg, 2006.

XCrypto: New Instructions - SHA3

XCrypto does a good job with SHA2. What about SHA3?

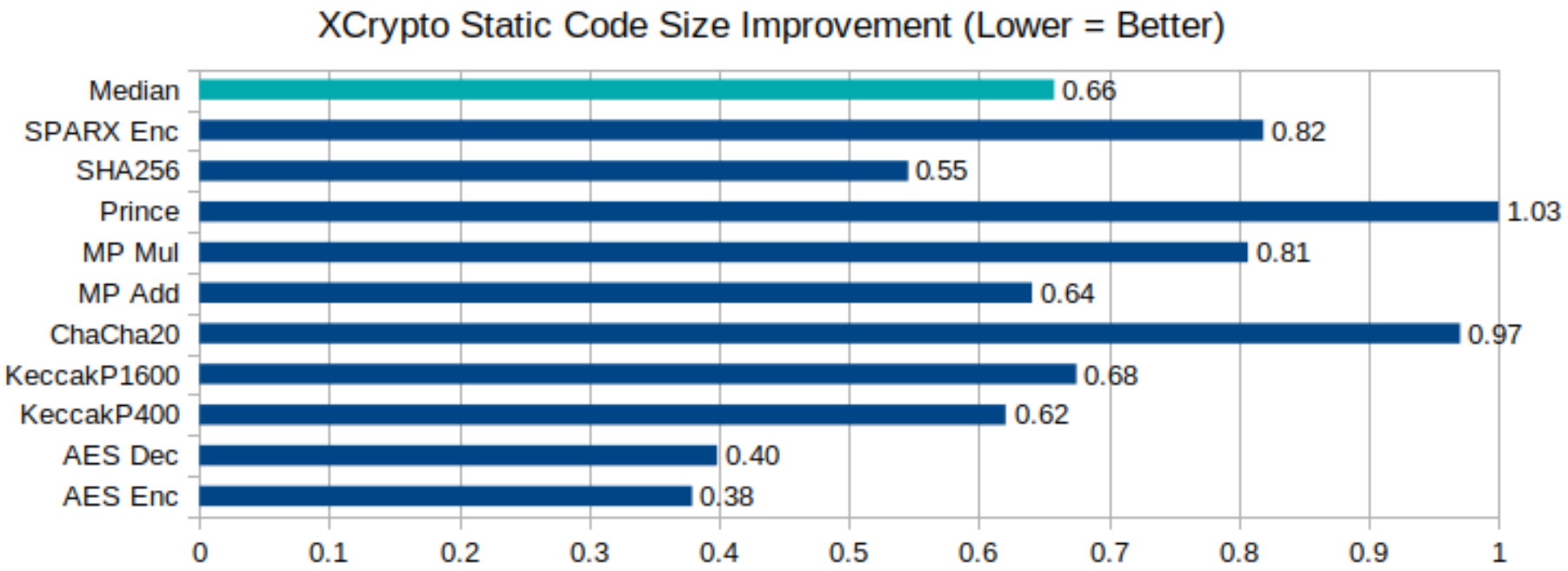
- The compute operations in SHA3 are simple: rotations and XORs.
- An unrolled implementation is 4K of code on `RV32IMC`.
- Unrolling is not realistic in embedded environments.
- Rolled-up implementations dominated by generating state matrix indexes:
  - `index(x, y) = (x%5)+5*(y%5)`
- Add novel instructions to generate the indexes from loop counter variables.
- Results in fast, code dense implementation with no extra lookup tables.
- Best performance increase per logic gate added we could find.
- 2x faster and 40% smaller across parameter sets.

XCrypto: Runtime Improvement



As measured using the SPIKE ISA simulator and our area optimised reference implementation. We use hand-optimised RISC-V assembly as a baseline, unless the compiler is better than the human.

XCrypto: Static Code Size

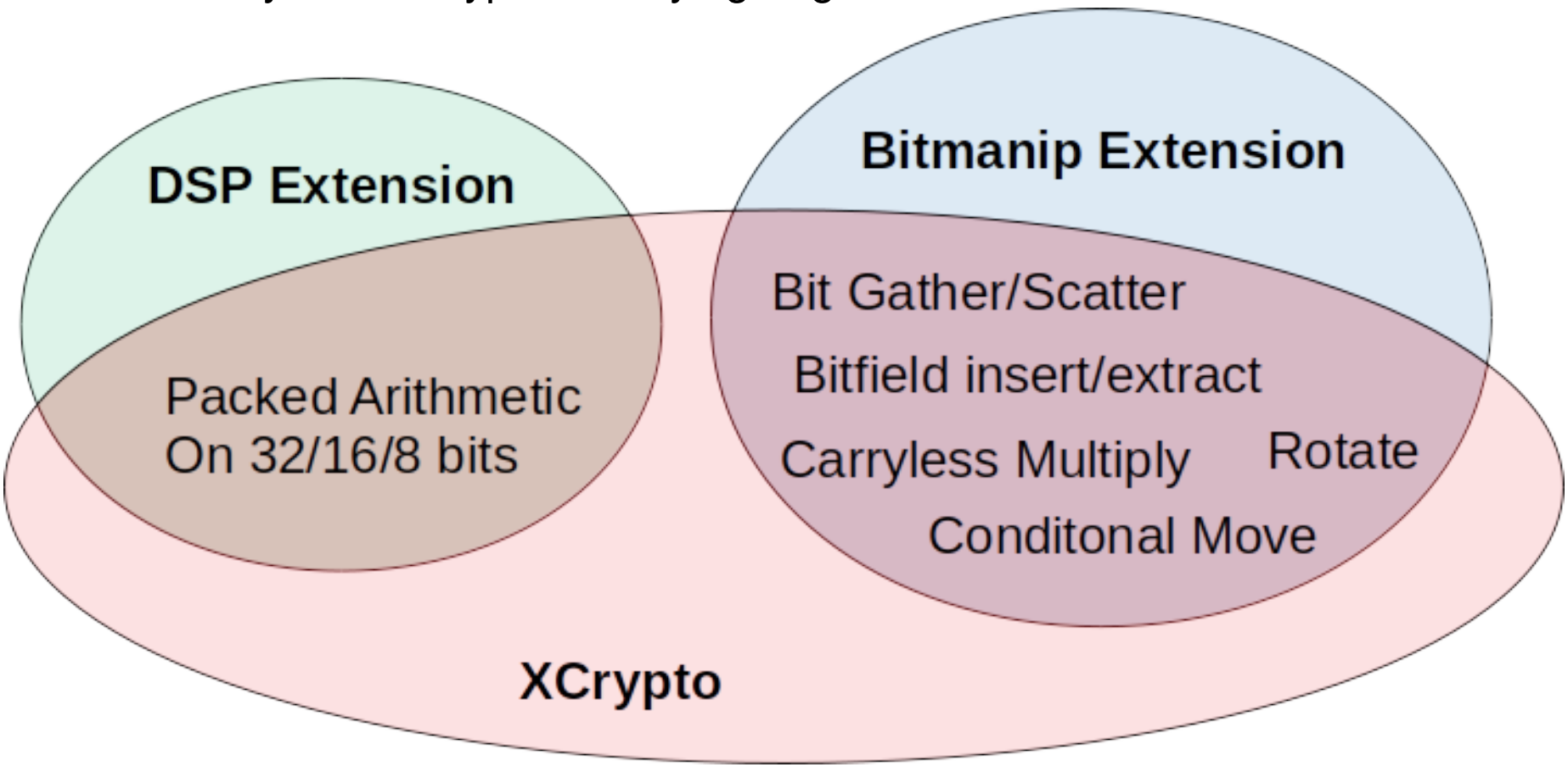


XCrypto: Implementation Overhead

- We have a reference area-optimised Verilog implementation of XCrypto, which uses PicoRV32 as a base. When implemented on an FPGA:
  - XCrypto itself is 1.9K LUTs / 215 FFs Plus some DMEM slices.
  - The entire PicoRV32+XCrypto subsystem is 6.4K LUTs and 1.1K DFFs.
  - This is with no datapath sharing, and a DSP slices for the multiplier.
  - Most of the datapaths could be shared.
  - In an abstract CMOS gate flow, the area is 50% single-cycle multiplier.
- Takeaway: you can make this really small.

Overlapping Functionality:

We've found that the DSP extension, and the Bitmanip extension overlap in terms of functionality with XCrypto to varying degrees.



Side Channels

Side channel attacks are increasingly important:

- IoT implies lots of small devices needing to communicate.
- Communication implies a need for cryptography.
- IoT-class devices are very suceptible to side-channel attacks.

Question: Which decisions I make at an architectural level affect side channel resiliance and how?

- Overwrite Semantics of Instructions
- Instruction Specification
- Implementation Flexibility
- Software Countermeasure Acceleration

The RISC-V Crypto Extension: Our View

- It makes perfect sense to have whole/all-round SHA2/AES instructions *iff*:
  - Your implementation has the vector extension already.
  - Your implementation has enough vector lanes.
- There are a lot of cores which sit below this performance point *which need to do crypto efficiently*.
  - This class of core is the kind most likely to be subject to power and EM side-channel attacks.
  - They also most need to consider energy efficiency.
  - Currently, the public proposals regarding the crypto extension do not solve this problem.
- A RISC-V crypto extension should cater to all classes of core.

Cryptography on RISC-V - What now?

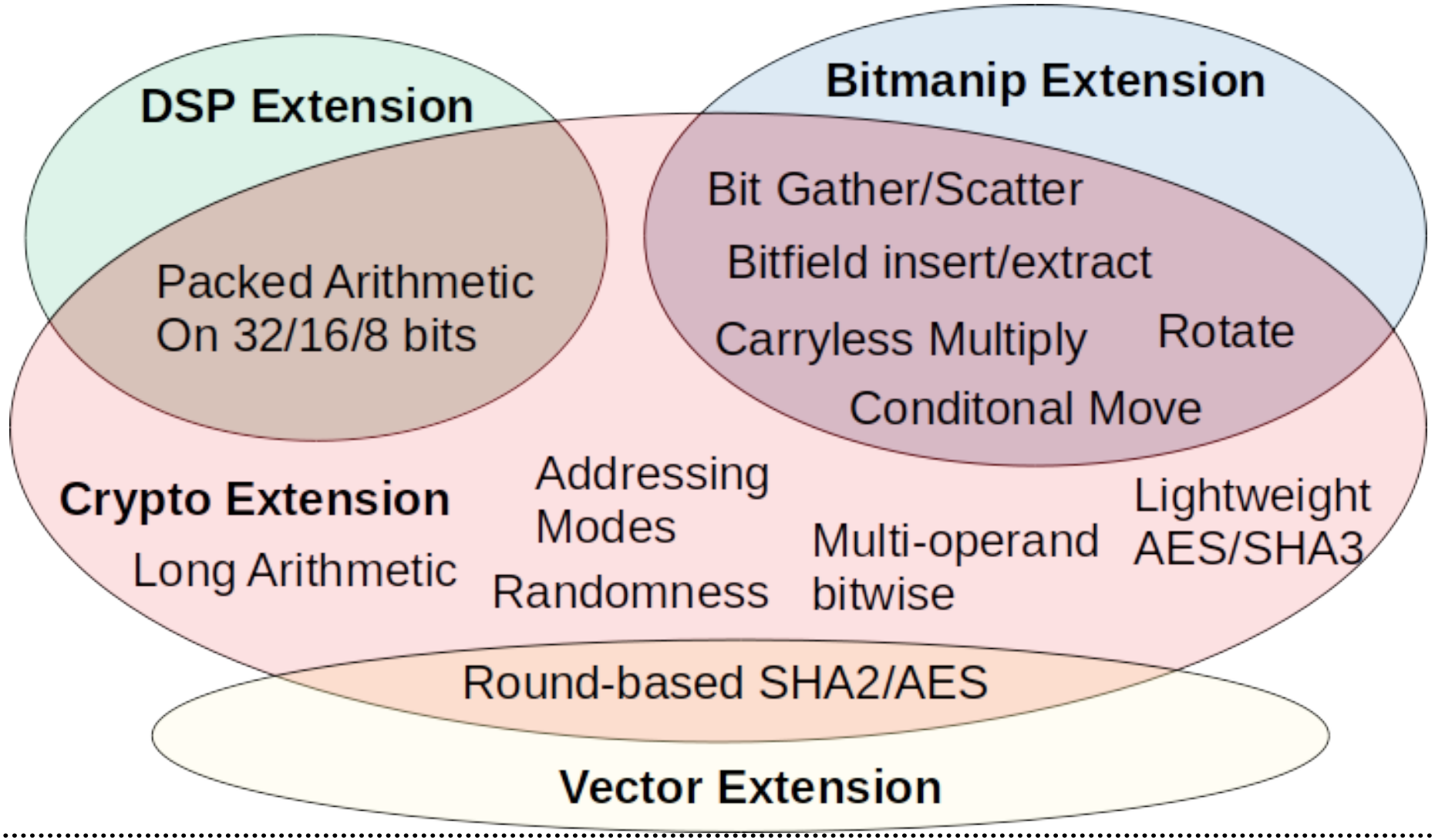
Current Trajectory:

- Large Cores: Base ISA + Bitmanip + Vector Crypto
- Small Cores:
  - Base ISA + Bitmanip
  - Base ISA + *Custom Extensions*

Possible Alternative:

- Large Cores: Base ISA + Bitmanip + Vector Crypto
- Small Cores: Base ISA + *Embedded Profile Crypto Extension*
- All Cores: Base ISA + *Modular Crypto Extension*

Modular Crypto ISE



Embedded Profile Crypto ISE

XCrypto was designed to be an embedded profile Crypto ISE.

Going forward, there are some possible options:

- Leave it as a custom extension:
  - It's a drop in, but with no compiler support etc.
- Do some work to turn it into a standard extension:
  - Encoding Space, Extra State Choices etc.
  - Handling overlaping functionality.
  - Needs to fix a remit.
  - This can be done if there is an appetite for it?

Cryptography on RISC-V

The Bad News:

- The base ISA has a way to go to just be *on par* with peer ISAs.

The Good News:

- We know exactly what functionality is needed to put it back on par.
- We know what is needed to make it *better than the competition*.
  - Lots of this has been independently developed in other extensions.
  - XCrypto has some helpful functionality in it not present in other extensions.
- There is **a lot** of prior art in this area.
  - Academic conferences like CHES feature existing research which has been under-used or re-invented.
- **We are keen to help take all of this forward.**