RISC-V Summit Europe Barcelona

Cryptography Extensions

June 5, 2023

Speaker

Markku-Juhani O. Saarinen

- Acting Post-Quantum TG Chair
- PQShield LTD, Oxford UK

On behalf of:

G. Richard Newell (Microchip Technology)

Ken Dockser (Tenstorrent)

- CETG Chair & Vice Chair

Crypto Extensions

- Ratified in Late 2021: Scalar Crypto Extensions
  - Scalar, resource optimized: AES, SHA2, SM3, SM4, Entropy Source
  - Supporting Bit Manipulations (helps SHA3, Ascon, also CLMUL/GHASH)
  - Data Independent Timing (for scalar)
- Present: Vector Crypto Extensions (almost frozen)
  - Vector, performance optimized: AES, SHA2, SM3, SM4
  - Assorted arithmetic manipulations (+ helps SHA3, Ascon)
  - CLMUL and GHASH, Data Independent Timing for Vector
- Future:
  - Full-Rounds AES – for key management / side-channel
  - Post Quantum (Kyber and Dilithium)
  - Still classical RSA/ECC crypto? Other cipher suites?

Present:
Vector Crypto

"Frozen Soon"

Topics to be covered

1. Scope of the Current Instructions
2. Vector Element Groups
3. Vector-Scalar Instructions
4. Vector Crypto Extension Groups
5. Data Independent Execution Latency

Scope of The Current Extension

- **AES-GCM** in TLS provides bulk data confidentiality (AES-CTR) and integrity (GHASH) for >50% of internet data.
- AES also for Storage (disk) encryption (XTS), DRBGs, etc.
- **SHA2-256/512** for Certificate Processing, also integrity of bulk data, content addressed storage, etc.
- **ShangMi** in China: SM4 (block cipher), SM3 (hash).

<u>Advantages:</u>

Lower network (or storage) latency, better energy efficiency, security by addressing (timing) attacks.

Vector Extensions refresher

- 32 Vector registers
  - register width (bits): **VLEN** (is ≥ 128 for “vk”)
- Vector register groups
  - 1, 2, 4 or 8 registers used as a single operand
- Instructions
  - Load/Store
  - Set configuration (e.g., vl, vtype)
  - operations on multiple elements

Vector Crypto *can* be built on any Vector Extension base

- “vk” with VLEN ≥ 256 is prefered
- ELEN<64 or XLEN<64 block some extensions

Element Groups (1)

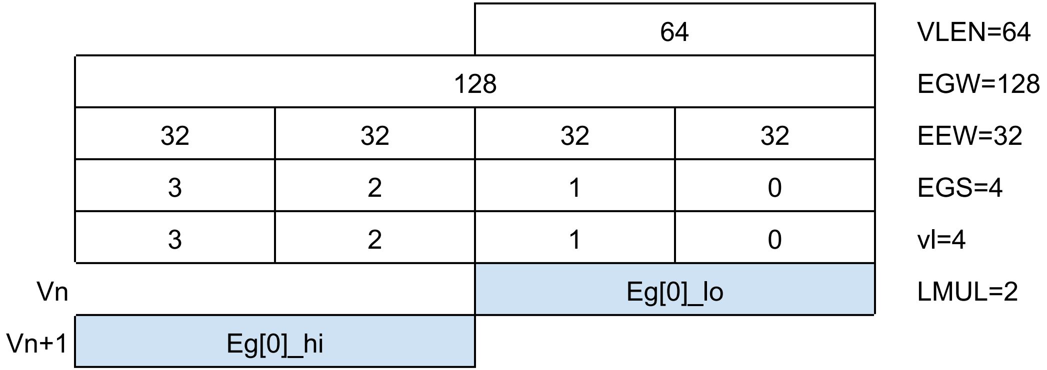
<table>
<tr><td></td><td colspan="8">256</td><td>VLEN=256</td></tr>
<tr><td></td><td colspan="4">128</td><td colspan="4">128</td><td>EGW=128</td></tr>
<tr><td></td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>32</td><td>EEW=32</td></tr>
<tr><td></td><td>3</td><td>2</td><td>1</td><td>0</td><td>3</td><td>2</td><td>1</td><td>0</td><td>EGS=4</td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>vl=8</td></tr>
<tr><td>Vn</td><td colspan="4">Eg[1]</td><td colspan="4">Eg[0]</td><td></td></tr>
</table>

- Provide support for data wider than 64-bits
- Vector Crypto has first extensions to adopt this concept

https://github.com/riscv/riscv-v-spec/blob/master/element_groups.adoc

Element Group Width (**EGW**):	Total number of bits in an element group.
Effective Element Width (**EEW**):	Number of bits in each element.
Element Group Size (**EGS**):	Number of elements in an element group.

Element Groups (2)

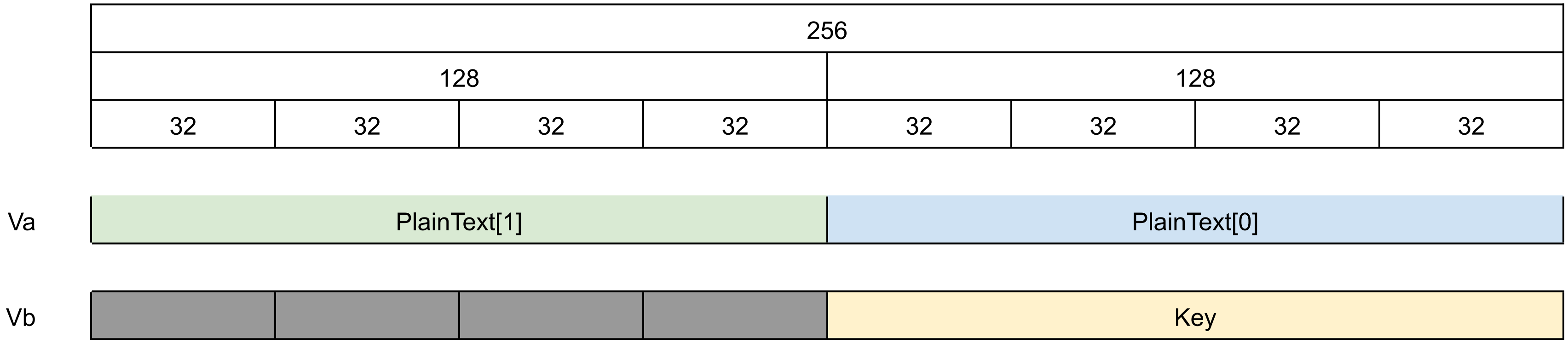


- Element *groups* can cross register boundaries by using *register* groups
  - Enables narrower implementations to support larger EGW instructions
- Elements are still not allowed to cross register boundaries

Element Groups in Vector Crypto

Instructions	Extension	EGW	EEW	EGS
AES	Zvkned	128	32	4
SHA256	Zvknh[ab]	128	32	4
SHA512	Zvknhb	256	64	4
GHASH	Zvkg	128	32	4
SM4 cipher	Zvksed	128	32	4
SM3 hash	Zvksh	256	32	8

Vector-Scalar instructions



- Allows an *element group* to be used as a scalar
- Re-uses the `.vs` suffix
- Applies scalar to each element group
  - for example, one AES key could apply to all “lanes”

“NIST” Selective Suites

Zvkned: NIST Suite: Vector AES Block Cipher

Zvknh[ab]: NIST Suite: Vector SHA-2 Secure Hash

Zvknc: NIST Algorithm Suite with carryless multiply

Zvkng: NIST Algorithm Suite with GCM

Zvkn: NIST Algorithms (Zvkned, Zvknhb, Zvbb, Zvkt)

“ShangMi” Selective Suites

Zvksed: ShangMi Suite: SM4 Block Cipher

Zvksh: ShangMi Suite: SM3 Secure Hash

Zvksc: ShangMi Algorithm Suite with carryless mult.

Zvksg: ShangMi Algorithm Suite with GCM

Zvks: ShangMi Algorithms (Zvksed, Zvksh, Zvbb, Zvkt)

Common Suites

Zvbb - Vector Bit-manipulation used in Cryptography

vandn.[vv,vx], vbrev.v, vbrev8.v, vrev8.v, vclz.v, vctz.v, vcpop.v, vrol.[vv,vx], vror.[vv,vx,vi], vwsll.[vv,vx,vi]

Zvbc - Vector Carryless Multiplication

vclmul.[vv,vx], vclmulh.[vv,vx]

Zvkg - Vector GCM/GMAC (128-bit fixed modulus)

vghsh.vv, vgmul.vv

Zvkt - Vector Data-Independent Execution Latency

Zvkt DIEL

“Constant Time”

25+ Years of Timing Attacks

<u>Examples over the years:</u>

- P.C. Kocher: *"Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems."* (CRYPTO 1996. Target: RSAREF 2.0 running on MS-DOS.)
- D. Brumley and D. Boneh: *"Remote timing attacks are practical."* (USENIX Security 2003. OpenSSL RSA remote key recovery, CVE-2003-0147.)
- B. Brumley and N. Toveri: *"Remote Timing Attacks Are Still Practical."* (ESORICS 2011. OpenSSL ECDSA remote key recovery, CVE-2011-1945.)
- *.. mature crypto implementations (e.g. OpenSSL) are nowadays mostly okay.*

<u>But new stuff keeps happening:</u>

- Q. Guo, T. Johansson. A. Nilsson, *"A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM."* (Crypto 2020.)

Addressed via constant-time implementation techniques.

Sources of Timing leaks

1. Secret-controlled branches and loops:

```
if <secret> then { delay1(); } else { delay2(); }
```

2. Memory accesses (cache timing attacks). Can be a load or store.

```
ct = SBox[pt ^ key];   // observe latency with different inputs.
```

3. Arithmetic operations whose processing time just depends on inputs

```
x = y % q;   // division and remainder ops are rarely constant-time.
```

Need <u>Data Independent Execution Latency</u> to process data.

Zvkt: Data Independent Latency

- One can use static analysis or dynamic variable tainting (in emulator) to verify that compiled code is using only the right (DIEL) instructions to handle secret data.

- But: ``*Constant-timeness*” of **Intel** and **ARM** instructions: derived mostly from experiments. A lot of platforms..

- RISC-V CETG codified timing as the **Zkt** extension for scalar, and the *(brand new)* **Zvkt** DIEL list for vector.

Zvkt DIEL Listings in Sect 2.14

All dedicated crypto instructions (Zk*) are DIEL. If a CPU asserts Zvkt then these instructions are DIEL in relation to data operands:

- Vector Bitmanip in Crypto Spec: Zvbb, Zvbc
- General arithmetic: Add/sub, compare and set, copy, extend, Boolean, multiply, multiply-add, integer merge, shift
- With limitations (only “data” registers): permute, slide

<u>Excluded</u>:

- All Load/store, floating point operations
- Clip, compress, divide, remainder, average, mask op, min/max, multiply-saturate, reduce, shift round, vset, ..

Zvk instructions

Skimming..

AES instructions - Zvkns

EGW	Mnemonic	Description
128	`vaesef.v[vs] vd, vs2`	Vector AES encrypt final round
128	`vaesem.v[vs] vd, vs2`	Vector AES encrypt middle round
128	`vaesdf.v[vs] vd, vs2`	Vector AES decrypt final round
128	`vaesdm.v[vs] vd, vs2`	Vector AES decrypt middle round
128	`vaeskf1.vi vd, vs2, uimm`	Vector AES-128 Forward KeySchedule
128	`vaeskf2.vi vd, vs2, uimm`	Vector AES-256 Forward KeySchedule
128	`vaesz.vs vd, vs2`	Vector AES round zero (encrypt/decrypt)

- All Vector AES instructions have 2 source operands
- Vd is used as a source to save instruction encoding space

SHA-2 instructions - Zvknh[ab]

Extension	SEW	EGW	Mnemonic	Description
Zvknha/b	32	128	vsha2ms.vv vd, vs2, vs1	Vector SHA-256 Message Schedule
Zvknha/b	32	128	vsha2ch.vv vd, vs2, vs1	Vector SHA-256 Compression high
Zvknha/b	32	128	vsha2cl.vv vd, vs2, vs1	Vector SHA-256 Compression low

Extension	SEW	EGW	Mnemonic	Description
Zvknhb	64	256	vsha2ms.vv vd, vs2, vs1	Vector SHA-512 Message Schedule
Zvknhb	64	256	vsha2ch.vv vd, vs2, vs1	Vector SHA-512 Compression high
Zvknhb	64	256	vsha2cl.vv vd, vs2, vs1	Vector SHA-512 Compression low

Bitmanip - Zvbb (not Zvkb anymore)

Mnemonic	Description
`vclmul.v[vx] vd, vs2, vs1, vm`	Vector Carryless Multiply
`vclmulh.v[vx] vd, vs2, vs1, vm`	Vector Carryless Multiply Return High Half
`vrol.v[vx] vd, vs2, [vr]s1, vm`	Vector Rotate Left
`vror.v[vx] vd, vs2, [vr]s1, vm` `vror.vi vd, vs2, uimm, vm`	Vector Rotate Right
`vbrev8.v vd, vs2, vm`	Vector Reverse Bits in Bytes
`vrev8.v vd, vs2, vm`	Vector Reverse Bytes
`vandn.v[vx] vs2, [vr]s1, vm` `vandn.vi vs2, imm, vm`	Vector And-Not

More Bitmanip - Zvbb (via AR)

Mnemonic	Description
`vclz.v vd, vs2, vm`	Vector Count Leading Zeros
`vctz.v vd, vs2, vm`	Vector Count Trailing Zeros
`vcpop.v vd, vs2, vm`	Vector Population Count

`vwsll.vv vd, vs2, vs1, vm`
`vwsll.vx vd, vs2, rs1, vm`
`vwsll.vi vd, vs2, uimm, vm`
	Vector Widening Shift Left Logical

GHASH instruction for GCM/GMAC - Zvkg

EGW	Mnemonic	Definition
128	`vgmul.vv vd, vs2`	GHASH Multiply
128	`vghsh.vv vd, vs2, vs1`	Vector GHASH Add-Multiply

vgmul computes vd*vs2 where * is a 128x128 carryless multiplication reduced to 128 bits it by GHASH's irreducible poly: $x^{128} + x^7 + x^2 + x + 1$.

The `vghmac` instruction performs a single iteration of the $GHASH_H$ algorithm. It computes (vd ^ vs1) * vs2 with reduction as in vgmul.
(note: vghsh was previously `vghmac`*, had a different order of multiply/add.)*

SM3 Secure Hash - Zvksh

EGW	EEW	Mnemonic	Definition
256	32	`vsm3me.vv vd, vs2, vs1`	Vector SM3 Message Expansion (8 rounds)
256	32	`vsm3c.vi vd, vs2,` *`uimm`*	Vector SM3 Compression (2 rounds)

- vsm3me has 3 source operands
- vsm3c has 2 source operands

- There is 1 message expansion instruction for every 4 compressions
  - `vslideDown` can be used to provide the current word pair
- This approach was chosen as it is expected to be more performant than having to execute 1 compression instruction per word pair.

SM4 Block Cipher - Zvksed

EGW	Mnemonic	Definition
128	vsm4k.vi vd, vs2, uimm	Vector SM4 four Rounds Key Expansion
128	vsm4rv.[vs] vd, vs2	SM4 four Rounds Encryption/Decryption

- vsm4k has 2 source operands (one is an immediate)
- vsm4r has 2 source operands

Vector Crypto Status

https://github.com/riscv/riscv-crypto/releases

- Spec has done several rounds with the Architecture Review; feedback incorporated
- Has binutils, Spike, OpenSSL support..
- Work will probably continue separately with **AES "All Rounds"** instructions.
- *"Crystallizing into Frozen.."*

Future: Post-Quantum

Introduction

Post-Quantum (1 minute intro)

- RSA and Elliptic Curve decimated by quantum (Shor's).
- Post-Quantum Cryptography (PQC) = Cryptography that is **not** vulnerable to a quantum algorithms.
- Symmetric cryptography is not affected much (Grover's) – most current RISC-V Zk extensions are actually fine.
- ~2015 U.S. Government decided to transition to PQC. NIST Standardization of PQC 2016-2024.

NIST Post-Quantum Standards

Selected July 2022, Standards 2023-2024.

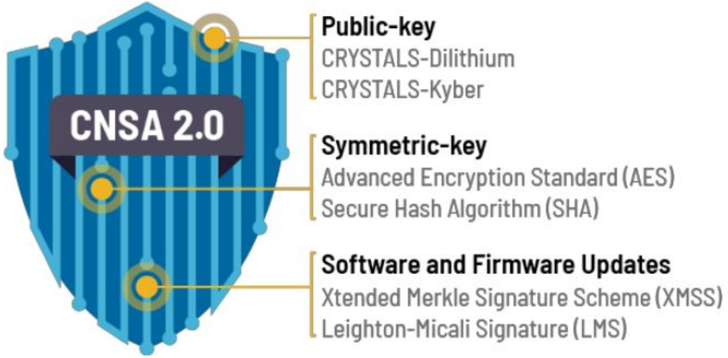
Kyber (+ Round 4 KEMs)
Replaces EC(DH), RSA key establishment.

Dilithium, Falcon, SPHINCS+
Replaces EC(DSA), RSA signatures.

Especially for U.S. Government Entities:

- *Active transition effort expected (presidential directives NSM-08, NSM-10).*
- *Regulations mandate FIPS 140-3 cryptography -> also for PQC modules.*

National Security Systems (NSS)



Transition 2025-2030-2035:
"Note that this will effectively deprecate [in NSS] the use of RSA, Diffie-Hellman (DH), and elliptic curve cryptography (ECDH and ECDSA) when mandated."

Table III: CNSA 2.0 quantum-resistant public-key algorithms

Algorithm	Function	Specification	Parameters
CRYSTALS-Kyber	Asymmetric algorithm for key establishment	TBD	Use Level V parameters for all classification levels.
CRYSTALS-Dilithium	Asymmetric algorithm for digital signatures	TBD	Use Level V parameters for all classification levels.

Kyber & Dilithium: Arithmetic

- **NTT** (Number Theoretic Transform) uses butterfly operations and a mul/add/sub mod fixed special q:
  q=0xD01 (Kyber)   q=0x7FE001 (Dilithium)

- **Better SHAKE/SHA3**: On an ARM microcontroller ~50% cycles is spent on the Keccak Permutation.

- Rejection sampling / bit gather (esp. for A matrix).

- Bitmanip (16- and 32-bit) shifts for CBD, bit packing.

Vector NTT: (mod q) arithmetic

- Montgomery technique used for (mod q) multiplication requires widening, extra reduction multiplication, shift, add.
- Vector Single-Width Integer Multiply-Add Instructions for mod q arithmetic (`vmaccq`, `vnmsacq`, `vmaddq`, `vnmsubq`).

Example Proposal:

```
vmaccq.vv vd, vs1, vs2, vm
# vd[i] = +(vs1[i] * vs2[i])+vd[i] (mod q)
```

If SEW=16: q=0xD01, else if SEW=32: q=0x7FE001.

(or set modulus q in special register – *Need experiments.*)

Better SHA3 / Keccak? Samplers?

<u>Scalar and vector crypto already have SHA3 support.</u>

- In Zvk: `vandn` (Chi), `vrol` (Theta) are there for Keccak.
- With `vrgather`, `vrgatherei` (Pi) this reasonably good.
- Do we need even more speedup for permutation?
- SHA3 also for SPHINCS+, XMSS, LMS/HSS.

<u>Look at gather / compress sequences for samplers:</u>

- Dilithium: Extract a 24-bit segment, clear high bit (bit 23), compare and select if $x<q$, expand to 32 bits for use.
- Kyber: 12-bit segment x, select x if $x<q$, expand to 16 bits.

PQC Workplan

- Elect officials (I'm the Acting Chair, Richard acting VC)
- Initial focus on Kyber and Dilithium which are a transition priority. NIST Should be releasing Draft PQC Standards in mid-2023 (yes, *very soon*) for Kyber and Dilithium.
- Do quantitative analysis with real-life benchmarks from PQ TLS ciphersuites, certificate processing, etc use cases
- Don't propose instructions unless they show advantage
- Proceed into freeze by the time NIST PQC Standards review is complete (not many changes expected)
- Try to time RVI ratification shortly after NIST's ratification

Thanks!

Questions?


RISC-V
SUMMIT