

# High-Performance FPGA Accelerator for the Post-quantum Signature Scheme CROSS

**Patrick Karl<sup>1</sup>, Francesco Antognazza<sup>2</sup>,  
Alessandro Barenghi<sup>3</sup>, Gerardo Pelosi<sup>3</sup>,  
Georg Sigl<sup>1,4</sup>**

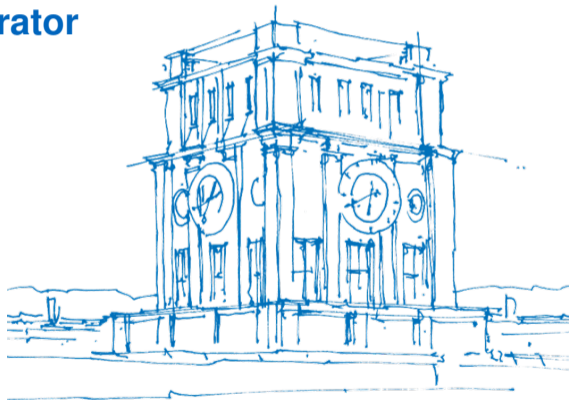
<sup>1</sup> Technical University of Munich

<sup>2</sup> then Politecnico di Milano, now UC Irvine

<sup>3</sup> Politecnico di Milano

<sup>4</sup> Faunhofer AISEC

Sixth PQC Standardization Conference  
NIST, Gaithersburg, September 25, 2025



*TUM Uhrenturm*

- 1** Introduction
- 2 Design Rationale, Architecture, Scheduling
- 3 Evaluation Methodology and Results
- 4 Conclusion

# Motivation

## Goal:

1. Spec compliant HW implementation of CROSS [Bal+25] (18 params, compile time)
2. Focus on efficiency w.r.t. Area-Time (AT) product

# Motivation

## Goal:

1. Spec compliant HW implementation of CROSS [Bal+25] (18 params, compile time)
2. Focus on efficiency w.r.t. Area-Time (AT) product

## Challenges:

1. Simple operations within CROSS (actually a good thing)
  - prioritizing low execution time over area not always rewarding in terms of AT product
2. Control-flow (scheduling, data movement etc.) plays an increasing role

# CROSS in a Nutshell

## ■ Fiat-Shamir transform applied to Zero-Knowledge protocol

- 5-pass protocol
- $t$  parallel repetitions

## ■ Hard problem: variant of Syndrome Decoding Problem

→ Two finite fields  $\mathbb{F}_p$  and  $\mathbb{F}_z$

→  $(z, p) = (7, 127)$  for R-RSDP

→  $(z, p) = (127, 509)$  for R-RSDP( $G$ )

### Restricted-SDP (R-SDP)

$$s = eH^T, \quad s \in \mathbb{F}_p^{(n-k)}, \quad H \in \mathbb{F}_p^{(n-k) \times n}$$

$$e \in \mathbb{E}^n, \quad \mathbb{E} = \{g^i \mid i \in \{1, \dots, z\}\}$$

$$g \in \mathbb{F}_p^* \text{ of prime order } z$$

### R-SDP w/ subgroup $G$ (R-SDP( $G$ ))

$$s = eH^T, \quad s \in \mathbb{F}_p^{(n-k)}, \quad H \in \mathbb{F}_p^{(n-k) \times n}$$

$$G = \left\{ \star_{i=1}^m \mathbf{a}_i^{\bar{u}_i} \mid \bar{u}_i \in \mathbb{F}_z \right\} \text{ for } \mathbf{a}_i \in \mathbb{E}^n$$

$$e \in G, \quad g \in \mathbb{F}_p^* \text{ of prime order } z$$

# CROSS in a Nutshell

- Fiat-Shamir transform applied to Zero-Knowledge protocol
  - 5-pass protocol
  - $t$  parallel repetitions
  
- Hard problem: variant of Syndrome Decoding Problem
  - Two finite fields  $\mathbb{F}_p$  and  $\mathbb{F}_z$ 
    - $(z, p) = (7, 127)$  for R-RSDP
    - $(z, p) = (127, 509)$  for R-RSDP( $G$ )

## CROSS parameters

Version and Security Level	Optim. Corner	$p$	$z$	$n$	$k$	$m$	$t$	$w$
R-SDP 1	fast	127	7	127	76	-	157	82
	balanced	127	7	127	76	-	256	215
	small	127	7	127	76	-	520	488
R-SDP 3	fast	127	7	187	111	-	239	125
	balanced	127	7	187	111	-	384	321
	small	127	7	187	111	-	580	527
R-SDP 5	fast	127	7	251	150	-	321	167
	balanced	127	7	251	150	-	512	427
	small	127	7	251	150	-	832	762
R-SDP( $G$ ) 1	fast	509	127	55	36	25	147	76
	balanced	509	127	55	36	25	256	220
	small	509	127	55	36	25	512	484
R-SDP( $G$ ) 3	fast	509	127	79	48	40	224	119
	balanced	509	127	79	48	40	268	196
	small	509	127	79	48	40	512	463
R-SDP( $G$ ) 5	fast	509	127	106	69	48	300	153
	balanced	509	127	106	69	48	356	258
	small	509	127	106	69	48	642	575

## Signature Generation: costly operations

### Commitment computation

- Sampling 2 vectors from 1 seed  $\bar{e}'[i], \mathbf{u}'[i] \leftarrow \text{CSPRNG}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_z^n \times \mathbb{F}_p^n)$
  - Matrix-vector multiplication(s)  $s'[i] \leftarrow \mathbf{u}[i] \mathbf{H}^\top$
  - 2 Hash computations  $\text{cmt}_0[i] \leftarrow \text{HASH}(s'[i] \parallel \bar{v}[i] \parallel \text{salt}),$   
 $\text{cmt}_1[i] \leftarrow \text{HASH}(\text{seed}[i] \parallel \text{salt})$
- *in each of  $t$  rounds*

# Signature Generation: costly operations

## Commitment computation

- Sampling 2 vectors from 1 seed
  - Matrix-vector multiplication(s)
  - 2 Hash computations
- *in each of  $t$  rounds*

$$\bar{e}'[i], \mathbf{u}'[i] \leftarrow \text{CSPRNG}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_z^m \times \mathbb{F}_p^m)$$

$$s'[i] \leftarrow \mathbf{u}'[i] \mathbf{H}^\top$$

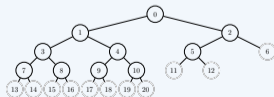
$$\text{cmt}_0[i] \leftarrow \text{HASH}(s'[i] \parallel \bar{v}[i] \parallel \text{salt}),$$

$$\text{cmt}_1[i] \leftarrow \text{HASH}(\text{seed}[i] \parallel \text{salt})$$

## Other things

- First response ( $t$  times)
  - Seed- and Merkle tree
- $t - 1$  SHAKE calls each

$$\mathbf{y}[i] \leftarrow \mathbf{u}'[i] + \text{chall}_1[i] e'[i]$$



# Signature Generation and Verification

## Algorithm 2 CROSS.SIGN

**Require:**  $sk = \text{seed}_k \in \{0, 1\}^{2\lambda}$ ,  $\text{msg} \in \{0, 1\}^*$   
**Ensure:**  $\text{sig} = (\text{salt} \in \{0, 1\}^{2\lambda}, \text{dig}_{\text{cnt}} \in \{0, 1\}^{2\lambda}, \text{dig}_{\text{chall}_2} \in \{0, 1\}^{2\lambda}, \text{path}, \text{proof}, \text{resp})$   
*▷ Expanding the secret key (according to lines 2-7 in CROSS.KEYGEN)*

- 1:  $\bar{e}, \bar{g}_G, H^T, \bar{M} \leftarrow \text{EXPANDSK}(\text{seed}_k)$   $\bar{e}, H^T \leftarrow \text{EXPANDSK}(\text{seed}_k)$   
*▷ Computing the commitments*
- 2:  $\text{seed} \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda$ ;  $\text{salt} \stackrel{\$}{\leftarrow} \{0, 1\}^{2\lambda}$
- 3:  $(\text{seed}[0], \dots, \text{seed}[t-1]) \leftarrow \text{SEEDLEAVES}(\text{seed}, \text{salt})$
- 4: **for**  $i \leftarrow 0$  **to**  $t-1$  **do** *▷ Compute the transformation  $v[i]$  such that  $v[i] \odot e^i[i] = e$*   
 $\bar{e}'_G[i], u^i[i] \leftarrow \text{CSRPRNG}_{2t-1+i}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_p^n \times \mathbb{F}_p^n)$   
 $\bar{e}'^i[i], u^i[i] \leftarrow \text{CSRPRNG}_{2t-1+i}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_p^n \times \mathbb{F}_p^n)$
- 5:  $\bar{e}'[i] \leftarrow \bar{e}'_G[i] \bar{M}$   $\bar{e}'^i[i], u^i[i] \leftarrow \text{CSRPRNG}_{2t-1+i}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_p^n \times \mathbb{F}_p^n)$
- 6:  $\bar{v}_G[i] \leftarrow \bar{e}_G - \bar{e}'_G[i]$
- 7:  $\bar{v}[i] \leftarrow \bar{e} - \bar{e}'[i]$
- 8:  $v[i] \leftarrow g^{\bar{v}[i]}$
- 9:  $u^i[i] \leftarrow v[i] \odot u^i[i]$
- 10:  $s^i[i] \leftarrow u^i[i] H^T$
- 11:  $\text{cnt}_0[i] \leftarrow \text{HASH}_{2t-1+i}(s^i[i] \parallel \bar{v}_G[i] \parallel \text{salt})$   $\text{cnt}_0[i] \leftarrow \text{HASH}_{2t-1+i}(s^i[i] \parallel \bar{v}[i] \parallel \text{salt})$
- 12:  $\text{dig}_{\text{cnt}_0}, \text{dig}_{\text{cnt}_1} \leftarrow \text{TREEROOT}(\text{cnt}_0[0] \parallel \dots \parallel \text{cnt}_0[t-1])$
- 13:  $\text{dig}_{\text{cnt}} \leftarrow \text{HASH}_0(\text{dig}_{\text{cnt}_0} \parallel \text{dig}_{\text{cnt}_1})$  *▷ Computing the first challenge*
- 14:  $\text{dig}_{\text{msg}} \leftarrow \text{HASH}_0(\text{msg})$
- 15:  $\text{dig}_{\text{chall}_1} \leftarrow \text{HASH}_0(\text{dig}_{\text{msg}} \parallel \text{dig}_{\text{cnt}} \parallel \text{salt})$
- 16:  $\text{chall}_1 \leftarrow \text{CSRPRNG}_{3t-1}(\text{dig}_{\text{chall}_1}, (\mathbb{F}_p^n)^t)$  *▷ Computing the first response*
- 17: **for**  $i \leftarrow 0$  **to**  $t-1$  **do**
- 18:  $e^i[i] \leftarrow g^{\bar{e}'^i[i]}$
- 19:  $y[i] \leftarrow u^i[i] + \text{chall}_1[i] e^i[i]$  *▷ Computing the second challenge*
- 20:  $\text{dig}_{\text{chall}_2} \leftarrow \text{HASH}_0(y[0] \parallel \dots \parallel y[t-1] \parallel \text{dig}_{\text{chall}_1})$
- 21:  $\text{chall}_2 \leftarrow \text{CSRPRNG}_{3t}(\text{dig}_{\text{chall}_2}, \mathcal{B}_{t,w})$  *▷ Computing the second response*
- 22:  $\text{proof} \leftarrow \text{TREEPROOF}(\text{cnt}_0[0] \parallel \dots \parallel \text{cnt}_0[t-1], \text{chall}_2)$
- 23:  $\text{path} \leftarrow \text{SEEDPATH}(\text{seed}, \text{salt}, \text{chall}_2)$
- 24: **for**  $i \leftarrow 0$  **to**  $t-1$  **do**
- 25: **if**  $\text{chall}_2[i] = 0$  **then**
- 26:  $\text{resp}[i]_0 \leftarrow (y[i], \bar{v}_G[i])$   $\text{resp}[i]_0 \leftarrow (y[i], \bar{v}[i])$
- 27:  $\text{resp}[i]_1 \leftarrow \text{cnt}_1[i]$
- 28: **return**  $\text{sig} = (\text{salt}, \text{dig}_{\text{cnt}}, \text{dig}_{\text{chall}_2}, \text{path}, \text{proof}, \text{resp})$

## Algorithm 3 CROSS.VERIFY

**Require:**  $\text{pk} = (\text{seed}_k \in \{0, 1\}^{2\lambda}, s \in \mathbb{F}_p^{n-k}), \text{msg} \in \{0, 1\}^*$ ,  
 $\text{sig} = (\text{salt} \in \{0, 1\}^{2\lambda}, \text{dig}_{\text{cnt}} \in \{0, 1\}^{2\lambda}, \text{dig}_{\text{chall}_2} \in \{0, 1\}^{2\lambda}, \text{path}, \text{proof}, \text{resp})$   
**Ensure:**  $\text{valid} \in \{\text{true}, \text{false}\}$   
*▷ Recover the public key*

- 1:  $\bar{W}, V^T \leftarrow \text{CSRPRNG}_{3t+2}(\text{seed}_k, \mathbb{F}_p^{n \times (n-m)} \times \mathbb{F}_p^{k \times (n-k)})$   $V^T \leftarrow \text{CSRPRNG}_{3t+2}(\text{seed}_k, \mathbb{F}_p^{k \times (n-k)})$
- 2:  $\bar{M} \leftarrow [\bar{W}, I_m]$   $H^T \leftarrow [V^T \mid I_{n-k}]$
- 3:  $\text{dig}_{\text{msg}} \leftarrow \text{HASH}_0(\text{msg})$  *▷ Compute the challenges*
- 4:  $\text{dig}_{\text{chall}_1} \leftarrow \text{HASH}_0(\text{dig}_{\text{msg}} \parallel \text{dig}_{\text{cnt}} \parallel \text{salt})$
- 5:  $\text{chall}_1 \leftarrow \text{CSRPRNG}_{3t-1}(\text{dig}_{\text{chall}_1}, (\mathbb{F}_p^n)^t)$
- 6:  $\text{chall}_2 \leftarrow \text{CSRPRNG}_{3t}(\text{dig}_{\text{chall}_2}, \mathcal{B}_{t,w})$  *▷ Compute the commitments*
- 7:  $(\text{seed}[i])_{i, \text{chall}_2[i]=1} \leftarrow \text{REBUILDLEAVES}(\text{path}, \text{chall}_2, \text{salt})$
- 8: **for**  $i \leftarrow 0$  **to**  $t-1$  **do**
- 9: **if**  $\text{chall}_2[i] = 1$  **then**
- 10:  $\text{cnt}_1[i] \leftarrow \text{HASH}_{2t-1+i}(\text{seed}[i] \parallel \text{salt})$
- 11:  $\bar{e}'_G[i], u^i[i] \leftarrow \text{CSRPRNG}_{2t-1+i}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_p^n \times \mathbb{F}_p^n)$   $\bar{e}'^i[i], u^i[i] \leftarrow \text{CSRPRNG}_{2t-1+i}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_p^n \times \mathbb{F}_p^n)$
- 12:  $\bar{e}'[i] \leftarrow \bar{e}'_G[i] \bar{M}$
- 13:  $e^i[i] \leftarrow g^{\bar{e}'^i[i]}$
- 14:  $y^i[i] \leftarrow u^i[i] + \text{chall}_1[i] e^i[i]$
- 15: **if**  $\text{chall}_2[i] = 0$  **then**
- 16:  $\text{cnt}_1[i] \leftarrow \text{resp}[i]_1$   $(y[i], \bar{v}[i]) \leftarrow \text{resp}[i]_0$
- 17:  $(y[i], \bar{v}_G[i]) \leftarrow \text{resp}[i]_0$   $(y[i], \bar{v}[i]) \leftarrow \text{resp}[i]_0$
- 18: **Check if**  $\bar{v}_G[i] \in \mathbb{F}_p^m$  **Check if**  $\bar{v}[i] \in \mathbb{F}_p^n$
- 19:  $\bar{v}[i] \leftarrow \bar{v}_G[i] \bar{M}$
- 20:  $v[i] \leftarrow g^{\bar{v}[i]}$
- 21:  $y^i[i] \leftarrow v[i] \odot y^i[i]$
- 22:  $s^i[i] \leftarrow y^i[i] H^T - \text{chall}_1[i] s$
- 23:  $\text{cnt}_0[i] \leftarrow \text{HASH}_{2t-1+i}(s^i[i] \parallel \bar{v}_G[i] \parallel \text{salt})$   $\text{cnt}_0[i] \leftarrow \text{HASH}_{2t-1+i}(s^i[i] \parallel \bar{v}[i] \parallel \text{salt})$
- 24: *▷ Check the digests*
- 25:  $\text{dig}_{\text{cnt}_0}, \text{dig}_{\text{cnt}_1} \leftarrow \text{RECOMPUTEROOT}(\text{cnt}_0, \text{proof}, \text{chall}_2), \text{HASH}_0(\text{cnt}_1[0] \parallel \dots \parallel \text{cnt}_1[t-1])$
- 26:  $\text{dig}'_{\text{cnt}}, \text{dig}'_{\text{chall}_2} \leftarrow \text{HASH}_0(\text{dig}_{\text{cnt}_0} \parallel \text{dig}_{\text{cnt}_1}), \text{HASH}_0(y[0] \parallel \dots \parallel y[t-1] \parallel \text{dig}_{\text{chall}_1})$
- 27: **if**  $\text{dig}_{\text{cnt}} \neq \text{dig}'_{\text{cnt}} \vee \text{dig}_{\text{chall}_2} \neq \text{dig}'_{\text{chall}_2}$  **then**
- 28: **return**  $\text{valid} = \text{false}$
- 29: **return**  $\text{valid} = \text{true}$

# Signature Generation and Verification

Algorithm 2 CROSS.S

Require:  $sk = \text{seed}_k \in \{0, 1\}^{2\lambda}$   
 Ensure:  $\text{sig} = (\text{salt} \in \{0, 1\}^{2\lambda})$

```

1:  $\bar{e}, \bar{e}_G, H^\top, \bar{M} \leftarrow \text{EXPANDSK}(\text{seed}_k)$ 
2:  $\text{seed} \stackrel{\$}{\leftarrow} \{0, 1\}^\lambda; \text{salt} \stackrel{\$}{\leftarrow} \{0, 1\}^{2\lambda}$ 
3:  $(\text{seed}[0], \dots, \text{seed}[t-1]) \leftarrow \text{SEEDPATH}(\text{seed}, \text{salt})$ 
4: for  $i \leftarrow 0$  to  $t-1$  do
5:    $\bar{e}'_G[i], \mathbf{u}'[i] \leftarrow \text{CSPRNG}_{2t-1+i}(\text{seed}[i])$ 
6:    $\bar{e}'[i] \leftarrow \bar{e}_G[i] \bar{M}$ 
7:    $\bar{\mathbf{v}}_G[i] \leftarrow \bar{e}_G - \bar{e}'_G[i]$ 
8:    $\mathbf{v}[i] \leftarrow g^{\bar{\mathbf{v}}_G[i]}$ 
9:    $\mathbf{u}[i] \leftarrow \mathbf{v}[i] \odot \mathbf{u}'[i]$ 
10:   $\text{cmt}_0[i] \leftarrow \text{HASH}_{2t-1+i}(\mathbf{u}'[i])$ 
11:   $\text{cmt}_1[i] \leftarrow \text{HASH}_{2t-1+i}(\text{seed}[i] \parallel \bar{\mathbf{v}}_G[i])$ 
12:   $\text{dig}_{\text{cmt}_0}, \text{dig}_{\text{cmt}_1} \leftarrow \text{TREEROOT}(\text{cmt}_0, \text{cmt}_1)$ 
13:   $\text{dig}_{\text{cmt}} \leftarrow \text{HASH}_0(\text{dig}_{\text{cmt}_0} \parallel \text{dig}_{\text{cmt}_1})$ 
14:   $\text{dig}_{\text{msg}} \leftarrow \text{HASH}_0(\text{msg})$ 
15:   $\text{dig}_{\text{chall}_1} \leftarrow \text{HASH}_0(\text{dig}_{\text{msg}} \parallel \text{dig}_{\text{cmt}})$ 
16:   $\text{chall}_1 \leftarrow \text{CSPRNG}_{3t-1}(\text{dig}_{\text{chall}_1})$ 
17: for  $i \leftarrow 0$  to  $t-1$  do
18:    $\mathbf{e}'[i] \leftarrow g^{\bar{\mathbf{e}}'[i]}$ 
19:    $\mathbf{y}[i] \leftarrow \mathbf{u}'[i] \odot \text{chall}_1[i] \mathbf{e}'[i]$ 
20:   $\text{dig}_{\text{chall}_2} \leftarrow \text{HASH}_0(\mathbf{y}[0] \parallel \dots \parallel \mathbf{y}[t-1])$ 
21:   $\text{chall}_2 \leftarrow \text{CSPRNG}_{3t}(\text{dig}_{\text{chall}_2})$ 
22:  $\text{proof} \leftarrow \text{TREPROOF}(\text{cmt}_0[0], \dots, \text{cmt}_0[t-1])$ 
23:  $\text{path} \leftarrow \text{SEEDPATH}(\text{seed}, \text{salt})$ 
24: for  $i \leftarrow 0$  to  $t-1$  do
25:   if  $\text{chall}_2[i] = 0$  then
26:     $\text{resp}[i]_0 \leftarrow (\mathbf{y}[i], \bar{\mathbf{v}}_G[i])$ 
27:     $\text{resp}[i]_1 \leftarrow \text{cmt}_1[i]$ 
28: return  $\text{sig} = (\text{salt}, \text{dig}_{\text{cmt}}, \text{dig}_{\text{chall}_1}, \text{chall}_1, \text{chall}_2, \text{proof}, \text{path}, \text{resp})$ 

```

Operation	Keygen	Sign	Verify
CSPRNG( $\cdot$ )	✓	✓	✓
Hash( $\cdot$ )	✗	✓	✓
mul mat-vec $\mathbb{F}_z$ (RSDP(G))	✓ (5)	✓ (5)	✓ (11, 16)
mul mat-vec $\mathbb{F}_p$	✓ (7)	✓ (9)	✓ (19)
mul vec-vec $\mathbb{F}_p$	✗	✓ (8, 19)	✓ (18)
exponentiation	✓ (6)	✓ (7, 18)	✓ (12, 17)
sub vec-vec $\mathbb{F}_z$	✗	✓ (5, 6)	✗
sub vec-vec $\mathbb{F}_p$	✗	✗	✓ (19)
add vec-vec $\mathbb{F}_p$	✗	✓ (19)	✓ (13)
Tree operations	✗	✓	✓
Compression	✓	✓	✓
Decompression	✗	✗	✓

→ Unified design for keygen, sign, verify

resp)

▷ Recover the public key

$\text{seed}_k \in \mathbb{F}_p^{k \times (n-k)}$

▷ Compute the challenges

Compute the commitments

$\text{seed}[i] \parallel \bar{\mathbf{v}}_G[i] \in \mathbb{F}_p^n$

$(\mathbf{u}'[i] \parallel \bar{\mathbf{v}}_G[i] \parallel \text{salt})$

▷ Check the digests

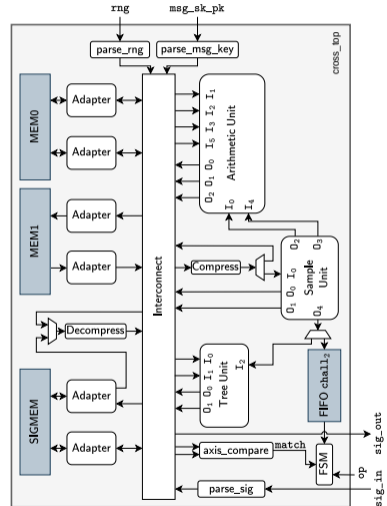
$[t-1]$

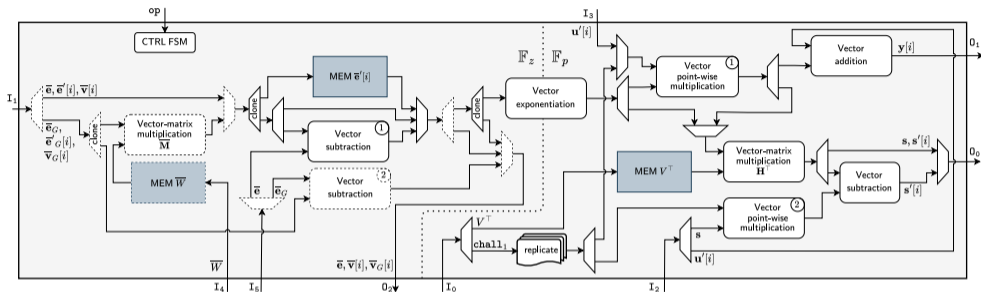
# Outline

- 1 Introduction
- 2 Design Rationale, Architecture, Scheduling**
- 3 Evaluation Methodology and Results
- 4 Conclusion

# Design Rationale and Architecture

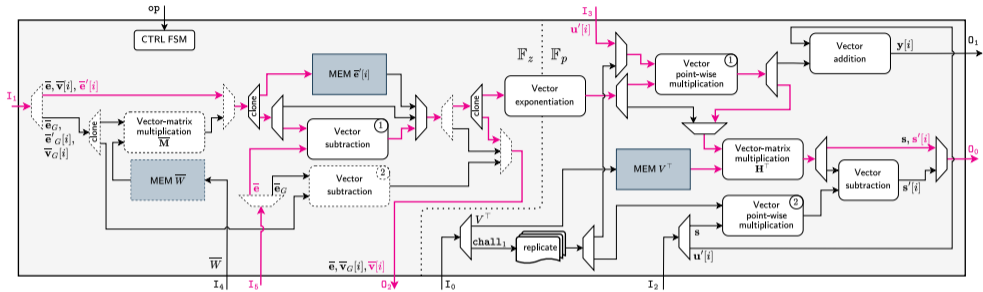
- Point-to-point connections, pipelining
  - overlap generation and processing of data
  - prevent unnecessary memory accesses
  
- Parallelization, unrolling
  - unroll keccak
  - vectorization of arithmetics
  - parallel sampling and hashing (or seed expansions)
  
- Parallel scheduling
  - hide data- and resource independent operations
  - hide memory access latencies





- Two domains for  $\mathbb{F}_z$  and  $\mathbb{F}_p$  separated by  $e = g^{\bar{e}}$ ,  $\bar{e} \in \mathbb{F}_z^n$  (R-SDP),  $e \in \mathbb{F}_p^n$
- Local memories for public variables / intermediates required later
- Operations vectorized to word width (exclusion mat-vec mult)
- Shift-and-add reduction for Mersenne primes  $z = 7, z = 127$  and  $p = 127$ , Crandall reduction for  $p = 509$

# Arithmetic Unit



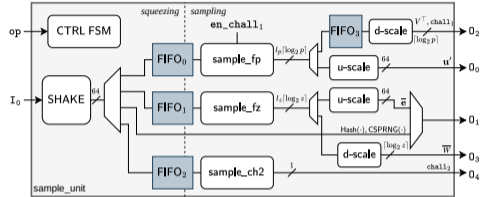
- Two domains for  $\mathbb{F}_z$  and  $\mathbb{F}_p$  separated by  $e = g^{\bar{e}}$ ,
- Local memories for public variables / intermediate
- Operations vectorized to word width (exclusion ma
- Shift-and-add reduction for Mersenne primes  $z =$  reduction for  $p = 509$

computation of  $s'[i]$  in committment loop

$$\begin{aligned}
 \bar{e}'[i], u'[i] &\leftarrow \text{CSPRNG}(\text{seed}[i] \parallel \text{salt}) \\
 \bar{v}[i] &\leftarrow \bar{e} - \bar{e}'[i] \\
 v[i] &\leftarrow g^{\bar{v}[i]} \\
 u[i] &\leftarrow v[i] \odot u'[i] \\
 s'[i] &\leftarrow u[i] H^T
 \end{aligned}$$

# Sample Unit

- Contains SHAKE module, rejection samplers ( $\mathbb{F}_z$ ,  $\mathbb{F}_p$ ), Fisher-Yates shuffling module
- Separation of *squeezing* SHAKE and sampling
  - prevents blocking SHAKE while sampling
- Leverages parallelism, e.g.
  - sample from both  $\mathbb{F}_z$  and  $\mathbb{F}_p$
  - sample public matrix and expand seed tree
  - sample  $\text{chall}_1$  while processing single entries



$$\bar{e}'[i], \mathbf{u}'[i] \leftarrow \text{CSPRNG}(\text{seed}[i] \parallel \text{salt}, \mathbb{F}_z^{Tz} \times \mathbb{F}_p^{Tz})$$

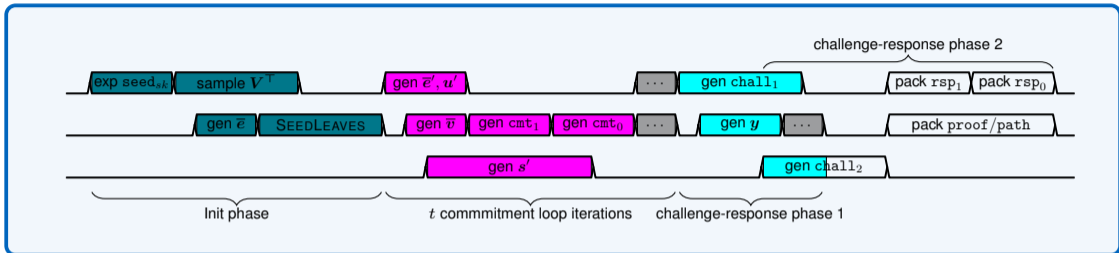
squeeze fill FIFO<sub>1</sub> fill FIFO<sub>0</sub>

sample  $\bar{e}'[i]$  sample\_fz active ...

sample  $\mathbf{u}'[i]$  sample\_fp active ...

→ in R-SDP  $\mathbf{u}'[i]$  is free (performance-wise)!

# Signing schedule (R-SDP)

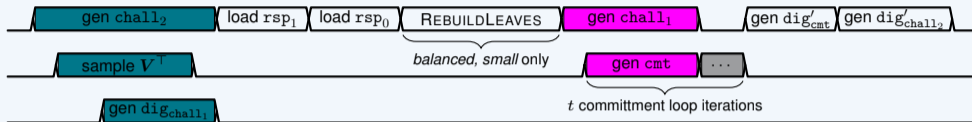


```
(seede, seedpk) ← (seedsk)
VT ← CSPRNG(seedpk)
 $\bar{e}$  ← CSPRNG(seede)
(seed[0], ...) ← SEEDLEAVES(...)
```

```
 $\bar{e}'[i], u'[i] \leftarrow \text{CSPRNG}(\text{seed}[i] \parallel \text{salt})$ 
 $\bar{v}[i] \leftarrow \bar{e} - \bar{e}'[i]$ 
 $v[i] \leftarrow g^{\bar{v}[i]}$ 
 $u[i] \leftarrow v[i] \odot u'[i]$ 
 $s'[i] \leftarrow u[i] H^T$ 
cmt0[i] ← HASH( $s'[i] \parallel \bar{v}[i] \parallel \text{salt}$ )
cmt1[i] ← HASH(seed[i]  $\parallel$  salt)
```

```
chall1 ← CSPRNG(digchall1)
For i in 0 to t - 1 :
     $e'[i] \leftarrow g^{\bar{e}'[i]}$ 
     $y[i] \leftarrow u'[i] + \text{chall}_{1}[i] e'[i]$ 
digchall2 ← HASH(y[0]  $\parallel$  ...  $\parallel$  digchall1)
```

# Verification schedule (R-SDP)



```

chall2 ← CSPRNG(digchall2)
VT ← CSPRNG(seedpk)
digmsg ← HASH(msg)
digchall1 ← HASH(digmsg || digcmt || salt)

```

```

chall1 ← CSPRNG(digchall1)
If chall2[i] = 0 :
    v[i] ← gv̄[i]
    y'[i] ← v[i] ⊙ y[i]
    s'[i] ← y'[i]HT - chall1[i]s
    cmt0[i] ← HASH(s'[i] || v̄[i] || salt)
If chall2[i] = 1 :
    cmt1[i] ← HASH(seed[i] || salt)
    ē'[i], u'[i] ← CSPRNG(seed[i] || salt)
    e'[i] ← gē'[i]
    y[i] ← u'[i] + chall1[i]e'[i]

```

→ less potential for parallelizing, but faster in general

# Outline

- 1 Introduction
- 2 Design Rationale, Architecture, Scheduling
- 3 Evaluation Methodology and Results**
- 4 Conclusion

# Evaluation Methodology

- Target: AMD Artix-7 (xc7a200tffbg484-3)
  - Vivado 2023.1 for synthesis and PnR
  
- Verilator and cocotb for simulation
  - integrated C reference implementation for testing
  
- Area measured in *equivalent Slices (eSlice)*
  - combines area of LUTs, FFs, DSPs, BRAMs
  - evaluates LUTs and FFs from out-of-context synthesis of DSP/BRAM
  - **DSP**: 538 LUTs and 232 FFs
  - **SDP BRAM**: 848 LUTs and 548 FFs
  - $\max(\lceil (\#LUTs)/4 \rceil, \lceil (\#FFs)/8 \rceil)$

# Evaluation Methodology

- Target: AMD Artix-7 (xc7a200tffbg484-3)
  - Vivado 2023.1 for synthesis and PnR
- Verilator and cocotb for simulation
  - integrated C reference implementation for testing
- Area measured in *equivalent Slices (eSlice)*
  - combines area of LUTs, FFs, DSPs, BRAMs
  - evaluates LUTs and FFs from out-of-context synthesis of DSP/BRAM
  - **DSP**: 538 LUTs and 232 FFs
  - **SDP BRAM**: 848 LUTs and 548 FFs
  - $\max(\lceil (\#LUTs)/4 \rceil, \lceil (\#FFs)/8 \rceil)$

## Disclaimer

Results shown differ from paper due to work-in-progress improvements. [Eprint](#) will be updated!

# Performance and Area Results

Security level	Param.	Resources					Freq. MHz	KEYGEN		SIGN		VERIFY	
		LUT	FF	BRAM	DSP	KeSlice		ms	AT	ms	AT	ms	AT
1	RSDP-1-f	25805	11525	44.5	0	15.8	125	0.033	0.529	0.502	7.97	0.438	6.95
	RSDP-1-b	26524	11654	57.0	0	18.7	117	0.036	0.666	0.902	16.8	0.731	13.6
	RSDP-1-s	26947	11744	111.0	0	30.2	105	0.040	1.20	2.01	61.0	1.57	47.7
	RSDPG-1-f	27456	11706	28.5	0	12.9	117	0.009	0.114	0.352	4.54	0.316	4.08
	RSDPG-1-b	28026	11879	37.0	0	14.8	126	0.008	0.122	0.643	9.55	0.495	7.34
	RSDPG-1-s	28620	12210	63.0	0	20.5	105	0.010	0.202	1.54	31.6	1.14	23.4
3	RSDP-3-f	28196	13569	96.5	0	27.5	111	0.079	2.18	1.15	31.8	1.06	29.3
	RSDP-3-b	29019	13780	121.0	0	32.9	119	0.074	2.44	1.74	57.4	1.46	48.1
	RSDP-3-s	29067	13872	147.5	0	38.5	119	0.074	2.85	2.61	100	2.11	81.5
	RSDPG-3-f	31167	13265	43.5	0	17.0	125	0.016	0.266	0.628	10.6	0.596	10.1
	RSDPG-3-b	32031	13482	68.0	0	22.4	120	0.016	0.365	0.880	19.7	0.753	16.8
	RSDPG-3-s	31512	13769	122.5	0	33.8	116	0.017	0.570	1.71	57.9	1.36	46.2
5	RSDP-5-f	31323	15270	150.5	0	39.7	108	0.145	5.76	1.95	77.5	1.89	75.1
	RSDP-5-b	32286	15431	201.5	0	50.7	111	0.141	7.16	2.98	151	2.62	133
	RSDP-5-s	33922	15568	279.5	0	67.7	99	0.158	10.7	5.35	362	4.54	308
	RSDPG-5-f	35183	15338	91.5	0	28.1	119	0.027	0.761	1.13	32.1	1.08	30.4
	RSDPG-5-b	35223	15494	122.5	0	34.7	115	0.028	0.972	1.54	53.6	1.29	44.9
	RSDPG-5-s	35392	15821	182.5	0	47.5	114	0.028	1.34	2.74	130	2.13	101

■ **Keccak**: unrolled 2 rounds per cc; **Mat-vec-mul**: 1 vector entry multiplied by 1 matrix row per cc

# Comparison with AVX2-optimized SW – Security Level 1

## Clock cycles ( $\times 10^3$ )

		Sign	Verify	Freq.
This	RSDP-1-f	62.7	54.7	125MHz
	RSDP-1-b	105	85.5	117MHz
	RSDP-1-s	212	166	105MHz
[Bal+25]	RSDP-1-f	1366	781	3.6GHz
	RSDP-1-b	2361	1539	3.6GHz
	RSDP-1-s	4783	3290	3.6GHz
This	RSDPG-1-f	41.2	37.0	125MHz
	RSDPG-1-b	81.1	62.3	117MHz
	RSDPG-1-s	162	120	105MHz
[Bal+25]	RSDPG-1-f	744	482	3.6GHz
	RSDPG-1-b	1452	989	3.6GHz
	RSDPG-1-s	2849	1995	3.6GHz

## Latency in ms

		Sign	Verify	Freq.
This	RSDP-1-f	0.502	0.438	125MHz
	RSDP-1-b	0.902	0.731	117MHz
	RSDP-1-s	2.01	1.57	105MHz
[Bal+25]	RSDP-1-f	0.379	0.217	3.6GHz
	RSDP-1-b	0.656	0.428	3.6GHz
	RSDP-1-s	1.33	0.914	3.6GHz
This	RSDPG-1-f	0.352	0.316	117MHz
	RSDPG-1-b	0.643	0.495	126MHz
	RSDPG-1-s	1.54	1.14	105MHz
[Bal+25]	RSDPG-1-f	0.207	0.134	3.6GHz
	RSDPG-1-b	0.403	0.275	3.6GHz
	RSDPG-1-s	0.791	0.554	3.6GHz

# Comparison with AVX2-optimized SW – Security Level 1

## Clock cycles ( $\times 10^3$ )

		Sign	Verify	Freq.
This	RSDP-1-f	62.7	54.7	125MHz
	RSDP-1-b	105	85.5	117MHz
	RSDP-1-s	212	166	105MHz

compared to AVX2 [Bal+25]:

$\times 18 - 22 \downarrow$  cc for signing

$\times 13 - 20 \downarrow$  cc for verification

	RSDPG-1-s	162	120	105MHz
[Bal+25]	RSDPG-1-f	744	482	3.6GHz
	RSDPG-1-b	1452	989	3.6GHz
	RSDPG-1-s	2849	1995	3.6GHz

## Latency in ms

		Sign	Verify	Freq.
This	RSDP-1-f	0.502	0.438	125MHz
	RSDP-1-b	0.902	0.731	117MHz
	RSDP-1-s	2.01	1.57	105MHz

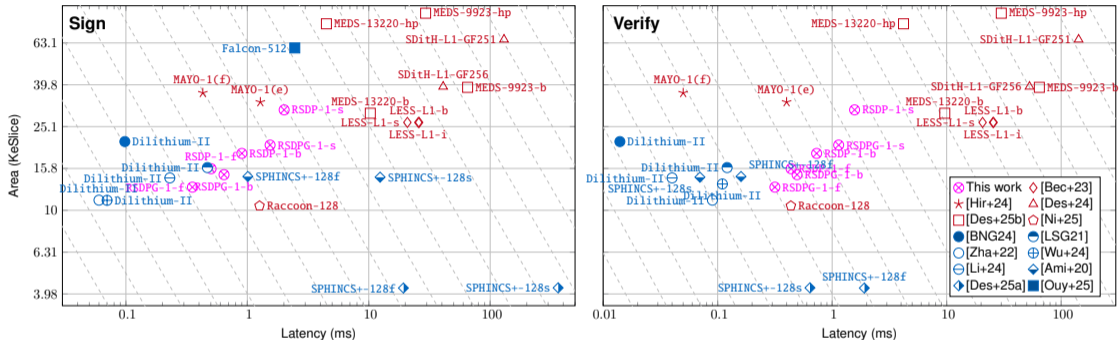
compared to AVX2 [Bal+25]

$\times 1.3 - 1.9 \uparrow$  ms for signing

$\times 1.7 - 2.36 \uparrow$  ms for verification

	RSDPG-1-s	1.54	1.14	105MHz
[Bal+25]	RSDPG-1-f	0.207	0.134	3.6GHz
	RSDPG-1-b	0.403	0.275	3.6GHz
	RSDPG-1-s	0.791	0.554	3.6GHz

# Comparison with SoA – Security Level 1 or 2



→ Competitive w.r.t. implementations of other on-ramp schemes

→ AT figures (dotted lines) between Dilithium and SPHINCS+, better than Falcon

# Outline

- 1 Introduction
- 2 Design Rationale, Architecture, Scheduling
- 3 Evaluation Methodology and Results
- 4 Conclusion**

## Conclusion and Outlook

- Competitive implementation of CROSS
  - parallel sampling, pipelining and parallel scheduling, vectorization, unrolling
- All parameter sets fit on Artix-7 despite large memory requirements
- Further numbers for other security levels, design-space explorations etc. in the paper
- New results upcoming:
  - parallelize sampling of multiple  $\mathbb{F}_z$  or  $\mathbb{F}_p$  elements
    - $\approx 10\%$  cc reduction for signing (R-SDP)
    - up to  $\approx 23\%$  cc reduction for verification (R-SDP)



eprint

# References I

- [Ami+20] D. Amiet et al. “FPGA-based SPHINCS<sup>+</sup> Implementations: Mind the Glitch”. In: *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*. IEEE, 2020, pp. 229–237.
- [Bal+25] M. Baldi et al. *CROSS Documentation*. 2025.
- [Bec+23] L. Beckwith et al. “A High-Performance Hardware Implementation of the LESS Digital Signature Scheme”. In: *Post-Quantum Cryptography - 14th International Workshop, PQCrypto 2023*. Ed. by T. Johansson and D. Smith-Tone. Springer, Cham, Aug. 2023, pp. 57–90.
- [BNG24] L. Beckwith, D. T. Nguyen, and K. Gaj. “Hardware Accelerators for Digital Signature Algorithms Dilithium and FALCON”. In: *IEEE Des. Test* 41.5 (2024), pp. 27–35.
- [Des+24] S. Deshpande et al. “SDitH in Hardware”. In: *IACR TCHES 2024.2* (2024), pp. 215–251.
- [Des+25a] S. Deshpande et al. “SPHINCSLET: An Area-Efficient Accelerator for the Full SPHINCS+ Digital Signature Algorithm”. In: *ACM Trans. Embed. Comput. Syst.* (Apr. 2025). Just Accepted.
- [Des+25b] S. Deshpande et al. *Unified MEDS Accelerator*. Cryptology ePrint Archive, Paper 2025/796. 2025.
- [Hir+24] F. Hirner et al. “Whipping the Multivariate-based MAYO Signature Scheme using Hardware Platforms”. In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*. Ed. by B. Luo et al. ACM, 2024, pp. 3421–3435.

# References II

- [Li+24] X. Li et al. “A High Speed Post-Quantum Crypto-Processor for Crystals-Dilithium”. In: *IEEE Trans. Circuits Syst. II Express Briefs* 71.1 (2024), pp. 435–439.
- [LSG21] G. Land, P. Sasdrich, and T. Güneysu. “A Hard Crystal - Implementing Dilithium on Reconfigurable Hardware”. In: *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*. Ed. by V. Grosso and T. Pöppelmann. Vol. 13173. Lecture Notes in Computer Science. Springer, 2021, pp. 210–230.
- [Ni+25] Z. Ni et al. “HRaccoon: A High-performance Configurable SCA Resilient Raccoon Hardware Accelerator”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.3 (2025), 413–436.
- [Ouy+25] Y. Ouyang et al. “FalconSign: An Efficient and High-Throughput Hardware Architecture for Falcon Signature Generation”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025.1 (2025), pp. 203–226.
- [Wu+24] Z. Wu et al. “An Efficient Hardware Implementation of Crystal-Dilithium on FPGA”. In: *Information Security and Privacy - 29th Australasian Conference, ACISP 2024, Sydney, NSW, Australia, July 15-17, 2024, Proceedings, Part II*. Ed. by T. Zhu and Y. Li. Vol. 14896. Lecture Notes in Computer Science. Springer, 2024, pp. 64–83.
- [Zha+22] C. Zhao et al. “A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.1 (2022), pp. 270–295.