IMES

FPGA-BASED ACCELERATOR FOR POST-QUANTUM SIGNATURE SCHEME SPHINCS-256

Dorian Amiet[1], Andreas Curiger[2] and Paul Zbinden[1]

[1] HSR Hochschule für Technik, Rapperswil, Switzerland

[2] Securosys SA, Zürich, Switzerland

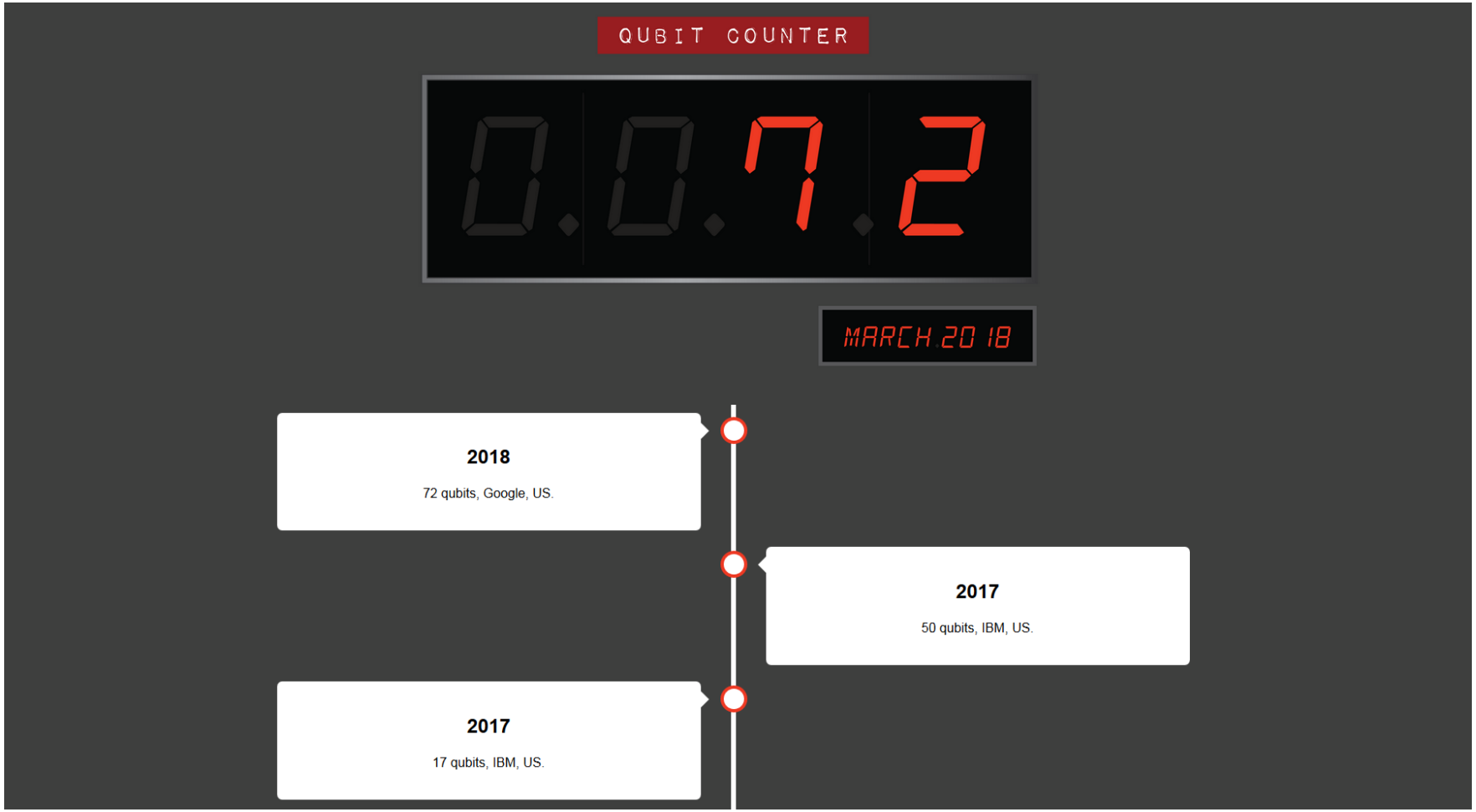
CHES 2018

12.09.2018

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Quantum Computer Progress



www.qubitcounter.com

Impact on Current Algorithms

Function	Algorithm	Key length/ Hash length (bits)	Security level (bits)		Quantum Algorithm
			Classical	Quantum	
PKI: Signing, Key Exchange....	RSA-3072	3072	128	0	Shor
	ECC-256	256	128	0	Shor
Symmetric Encryption	AES-128	128	128	64	Grover
	AES-256	256	256	128	Grover
Hash	SHA-256	256	256	128	Grover
	SHA3-512	512	512	256	Grover

Agenda

- **Hash-based signatures**
  - OTS (one-time signature)
  - Merkle trees
  - SPHINCS-256
- **SPHINCS-256 FPGA implementation**
- **Adjustments to SPHINCS+**
- **SPHINCS+ FPGA implementation**
- **Performance results**

} **New, unpublished results!**

Post-Quantum Signature Algorithms…

- **…enable secure signing while an adversary has a quantum computer**
- **Several approaches:**
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others

Post-Quantum Signature Algorithms…

- **…enable secure signing while an adversary has a quantum computer**
- **Several approaches:**
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others

} **All signing protocols need a hash function (message digest)**

Post-Quantum Signature Algorithms…

- **…enable secure signing while an adversary has a quantum computer**
- **Several approaches:**
  - Lattice-based
  - Code-based
  - Supersingular isogeny
  - Others

  } **All signing protocols need a hash function (message digest)**
- **Hash based signature schemes**
  - Security relies on hardness of (second-) pre-image attack
  - Cryptanalysis: Hash functions are very well analyzed and understood
  - If hash functions are broken, all signing protocols are broken

  => Simply the most conservative choice in terms of security

Lamport One-Time Signature (OTS)

Example: OTS with 256 bit security

1. **Generate 2x256 random numbers, each 256 bits long**
   - $X_{0,0}, X_{0,1}, X_{2,0} \ldots X_{255,1}$
   - $X_{i,j}$ = private key

rand 0		rand 1	
$X_{0,0}$		$X_{0,1}$	
$X_{1,0}$		$X_{1,1}$	
$X_{2,0}$		$X_{2,1}$	
$X_{\ldots,0}$		$X_{\ldots,1}$	
$X_{255,0}$		$X_{255,1}$	

Lamport One-Time Signature (OTS)

Example: OTS with 256 bit security

1. **Generate 2x256 random numbers, each 256 bits long**
   - $X_{0,0}, X_{0,1}, X_{2,0} \ldots X_{255,1}$
   - $X_{i,j}$ = private key
2. **Calculate all digests from random numbers**
   - $Y_{0,0} = h(X_{0,0}), Y_{0,1} = h(X_{0,1}), \ldots, Y_{255,1} = h(X_{255,1})$
   - $Y_{i,j}$ = public key

rand 0	*h*(rand 0)	rand 1	*h*(rand1)
$X_{0,0}$	$Y_{0,0}$	$X_{0,1}$	$Y_{0,1}$
$X_{1,0}$	$Y_{1,0}$	$X_{1,1}$	$Y_{1,1}$
$X_{2,0}$	$Y_{2,0}$	$X_{2,1}$	$Y_{2,1}$
$X_{\ldots,0}$	$Y_{\ldots,0}$	$X_{\ldots,1}$	$Y_{\ldots,1}$
$X_{255,0}$	$Y_{255,0}$	$X_{255,1}$	$Y_{255,1}$

Lamport One-Time Signature (OTS)

Example: OTS with 256 bit security

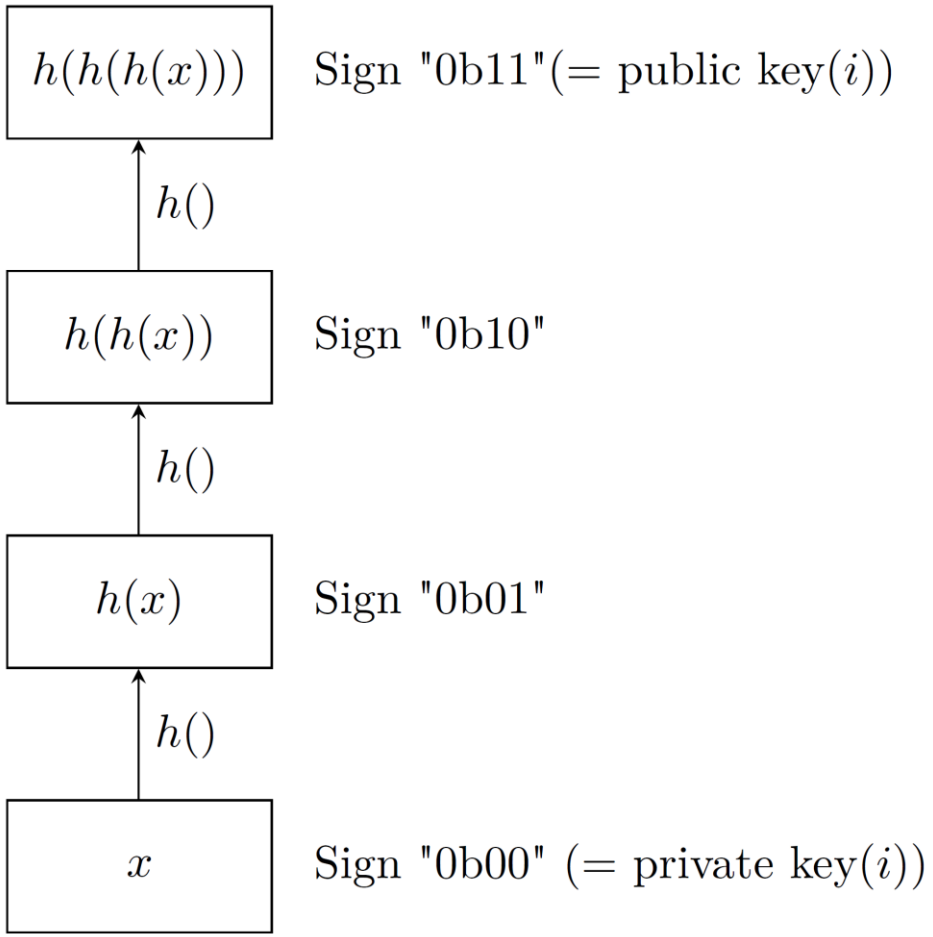
1. **Generate 2x256 random numbers, each 256 bits long**
   - $X_{0,0}, X_{0,1}, X_{2,0} \ldots X_{255,1}$
   - $X_{i,j}$ = private key
2. **Calculate all digests from random numbers**
   - $Y_{0,0} = h(X_{0,0}), Y_{0,1} = h(X_{0,1}), \ldots, Y_{255,1} = h(X_{255,1})$
   - $Y_{i,j}$ = public key
3. **Sign:**
   1. Calculate digest from message $d = h(m)$
   2. For $i = 0$ to 255
      1. If $d_i = 0$, then $\upsilon_i <= X_{i,0}$
      2. Else $\upsilon_i <= X_{i,1}$

rand 0	*h*(rand 0)	rand 1	*h*(rand1)
$X_{0,0}$	$Y_{0,0}$	$X_{0,1}$	$Y_{0,1}$
$X_{1,0}$	$Y_{1,0}$	$X_{1,1}$	$Y_{1,1}$
$X_{2,0}$	$Y_{2,0}$	$X_{2,1}$	$Y_{2,1}$
$X_{\ldots,0}$	$Y_{\ldots,0}$	$X_{\ldots,1}$	$Y_{\ldots,1}$
$X_{255,0}$	$Y_{255,0}$	$X_{255,1}$	$Y_{255,1}$

***h(m)* = 0b010…1**

=> Signature(m) = ($X_{0,0}, X_{1,1}, X_{2,0}, \ldots, Y_{255,1}$)

W-OTS+ Shorter Signatures for Hash-Based Signature Schemes

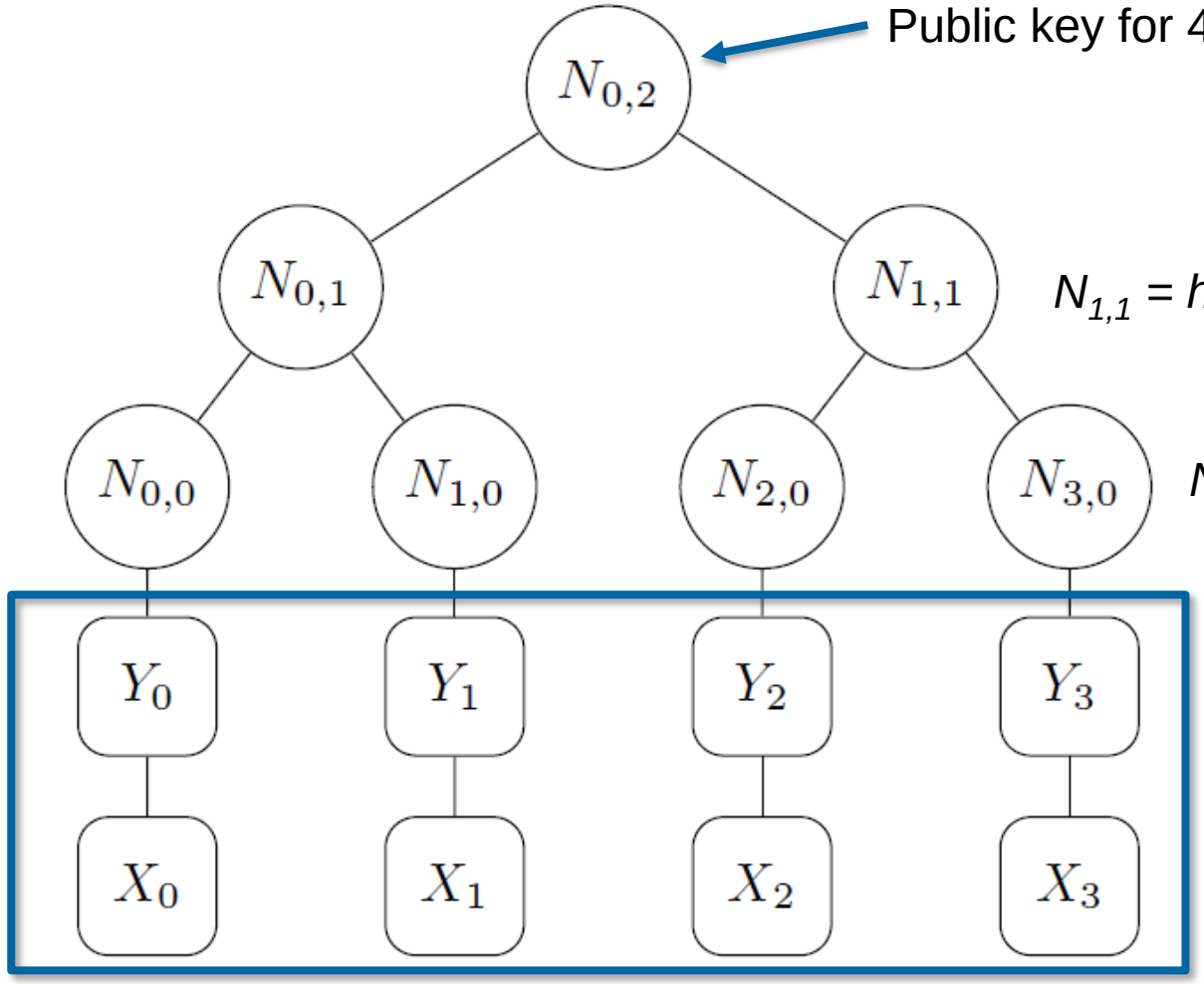


- **Sign a few bits per random number**
- **Increases processing time**
- **Decreases key and signature sizes**

W-OTS+

+ **Signature system which security is based only on security of hash function**
+ **Quantum secure**
+ **Very fast**
– **One signature per key pair**

Merkle Tree



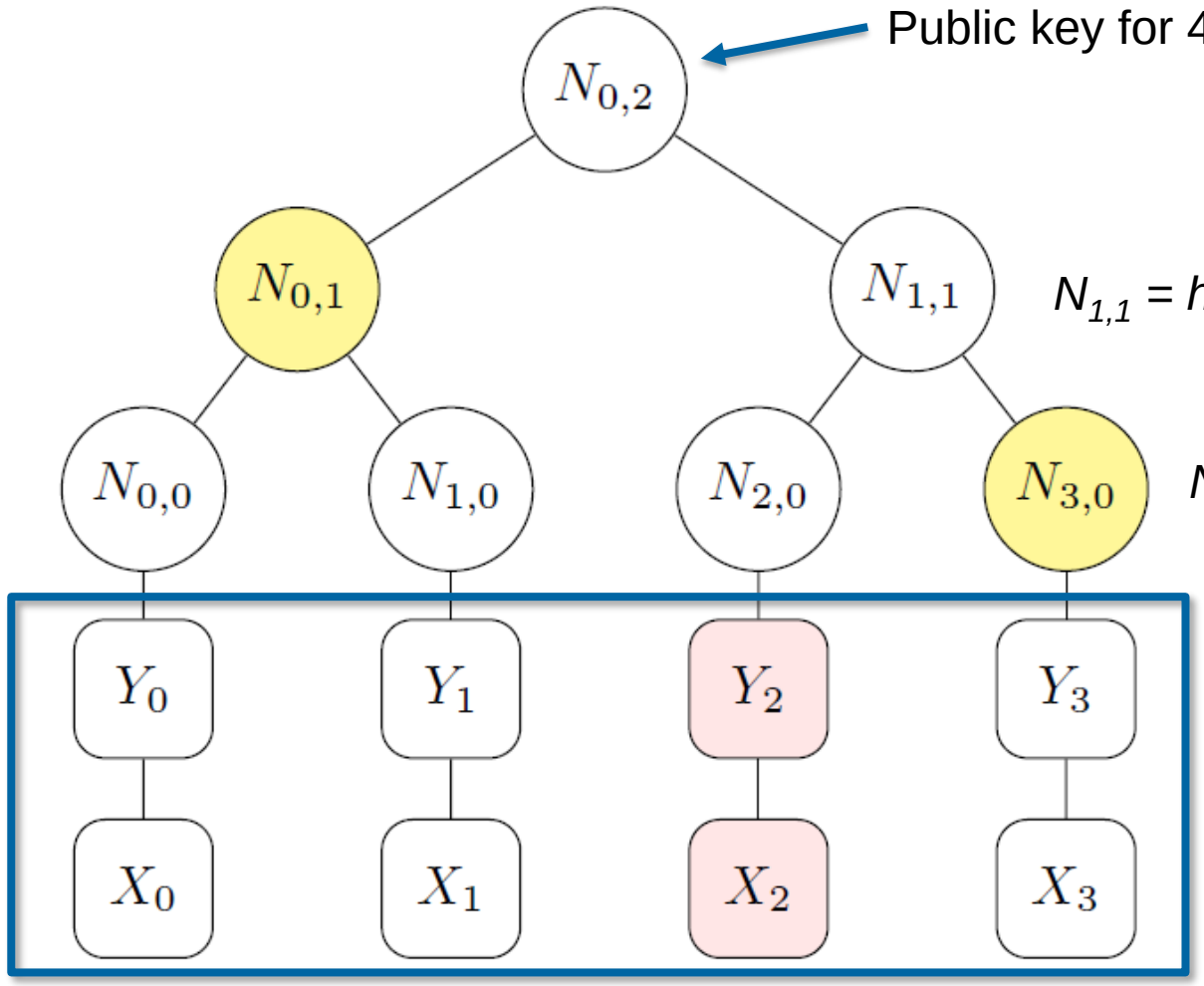
Public key for 4 signatures

$N_{1,1} = h(N_{2,0} \,||\, N_{3,0})$

$N_{3,0} = h(Y_3)$

4 W-OTS+ key pairs

Merkle Tree



Public key for 4 signatures

$N_{1,1} = h(N_{2,0} \,||\, N_{3,0})$

$N_{3,0} = h(Y_3)$

4 W-OTS+ key pairs

Merkle Tree

+ **Signature system which security is based only on security of hash function**
+ **Quantum secure**
+ **Fast operations**
– **State-based**

  => Check-list required: Which W-OTS+ key pairs (leaves of the tree) are already used?

SPHINCS

- **Make a hyper-tree (tree of trees)**
  - Increases number of leaves dramatically
- **Use a FTS (few-time signature) at bottom layer instead of OTS**
- **Choose starting point at random**

SPHINCS

- **Make a hyper-tree (tree of trees)**
  - Increases number of leaves dramatically
- **Use a FTS (few-time signature) at bottom layer instead of OTS**
- **Choose starting point at random**

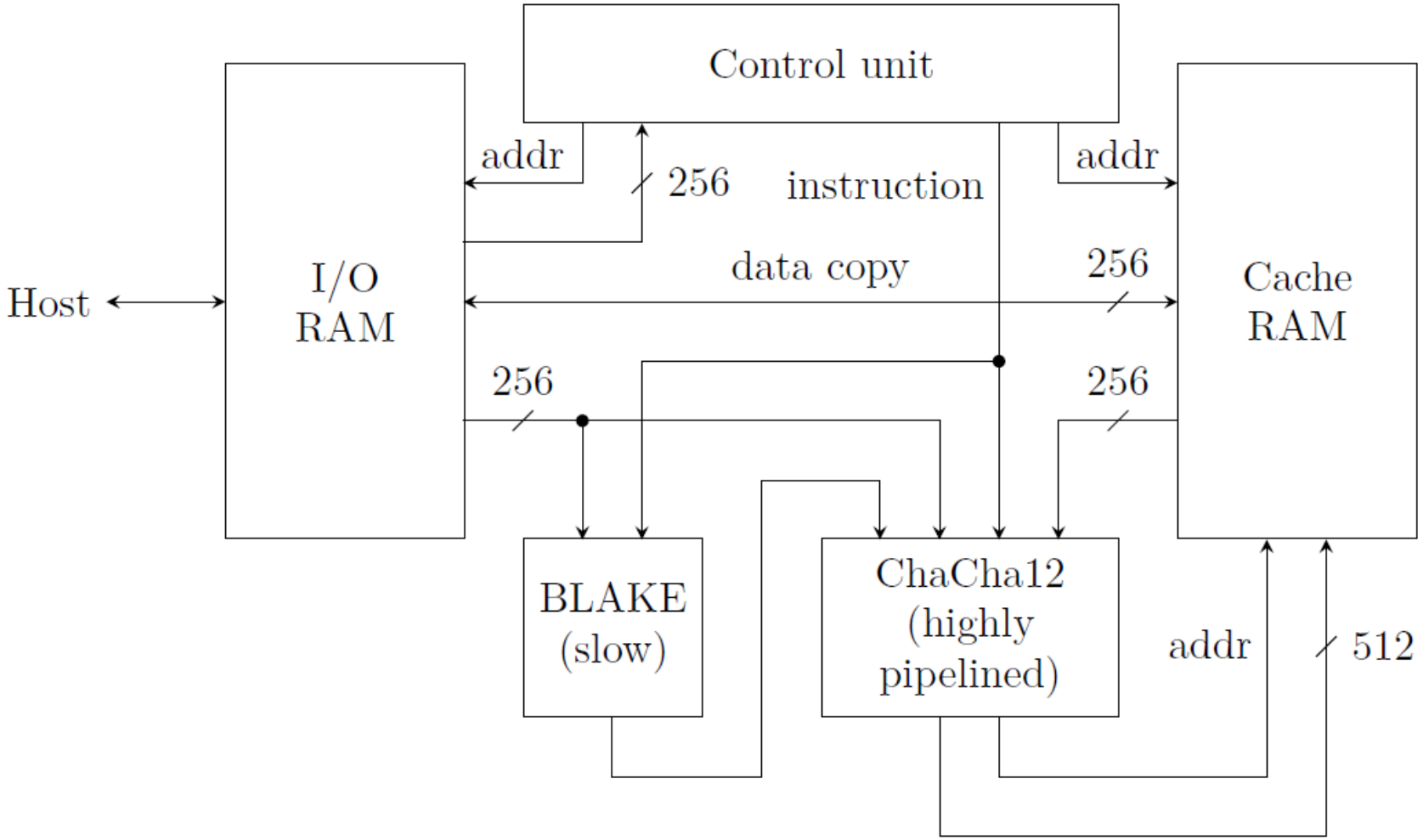
Source: https://sphincs.cr.yp.to/

=> Stateless, practical, hash-based, incredibly nice cryptographic signatures (SPHINCS)

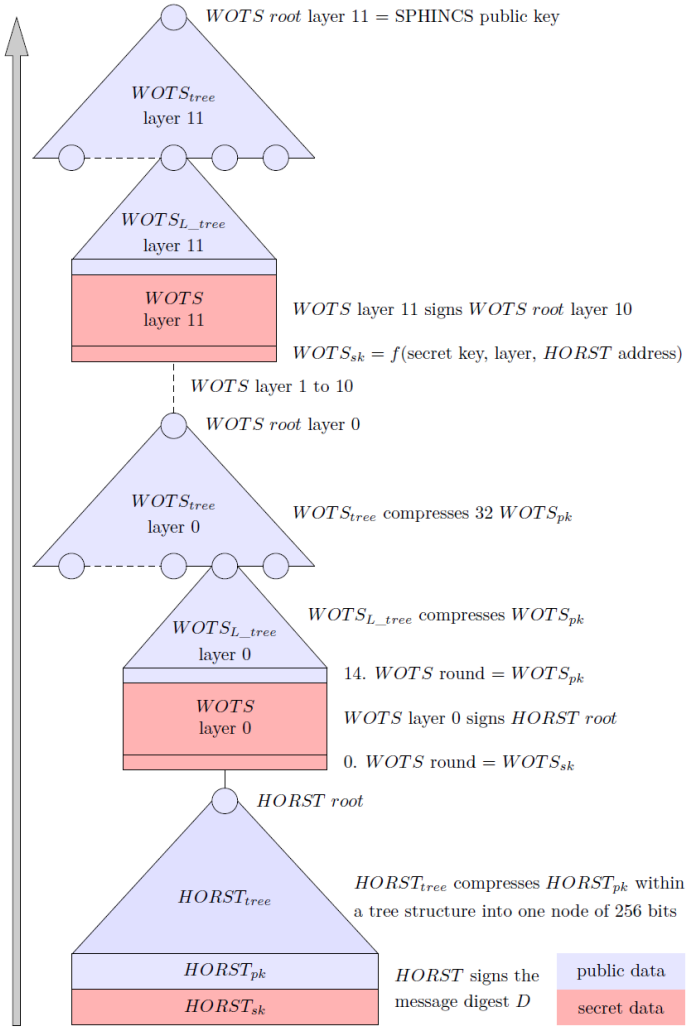
SPHINCS-256 Operation Count

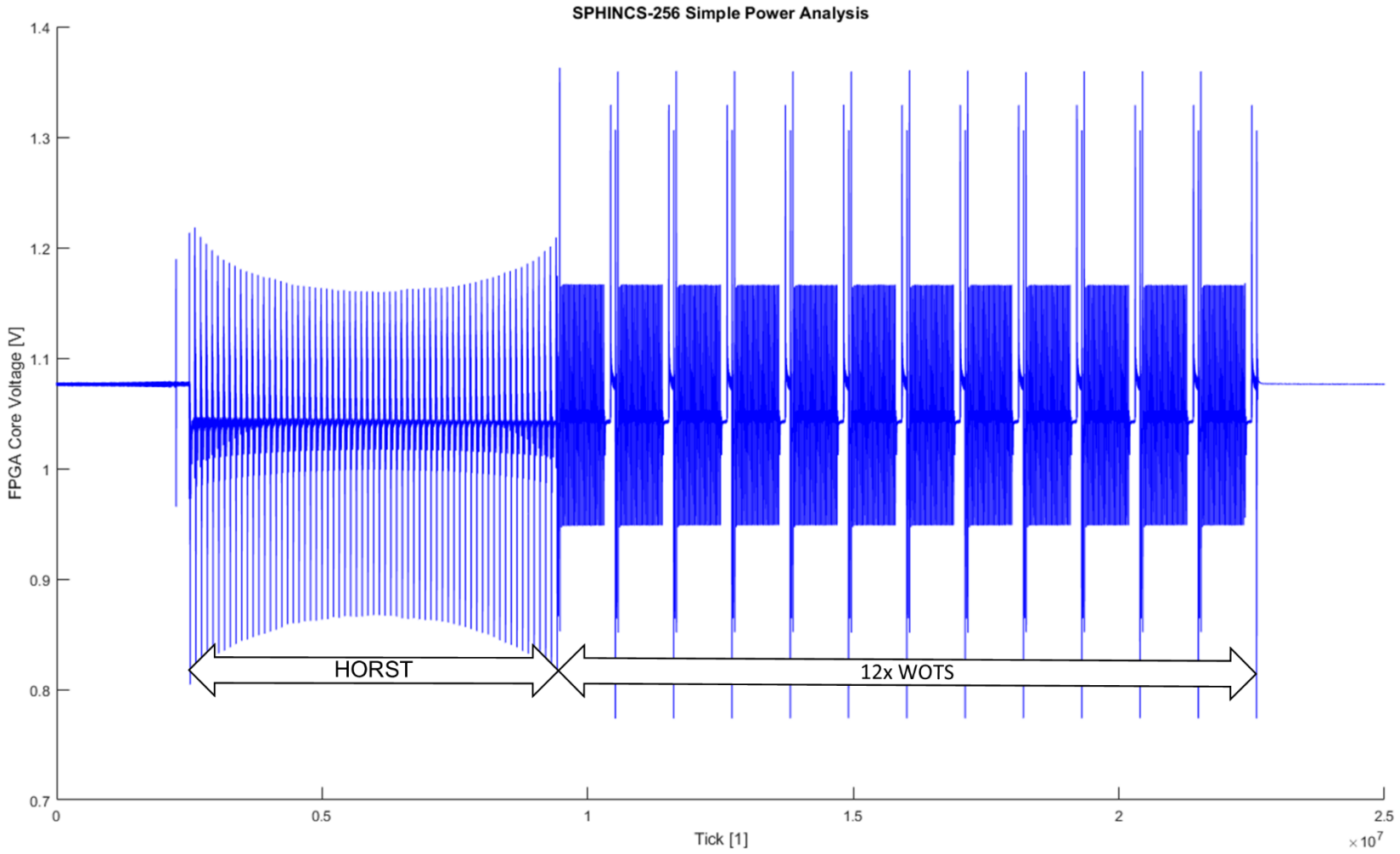
Function	Signing					Verification
Part	Start	HORST	WOTS	Overhead	Total	Total
BLAKE-256	0	1	384	12	**397**	0
ChaCha12	0	32,768	13,056	408	**46,232**	0
π_{ChaCha}	0	193,410	437,352	≈9000	**640,000**	≈9000
BLAKE-512	2	0	0	0	**2**	1

SPHINCS-256 Core Top



Simple Power Analysis

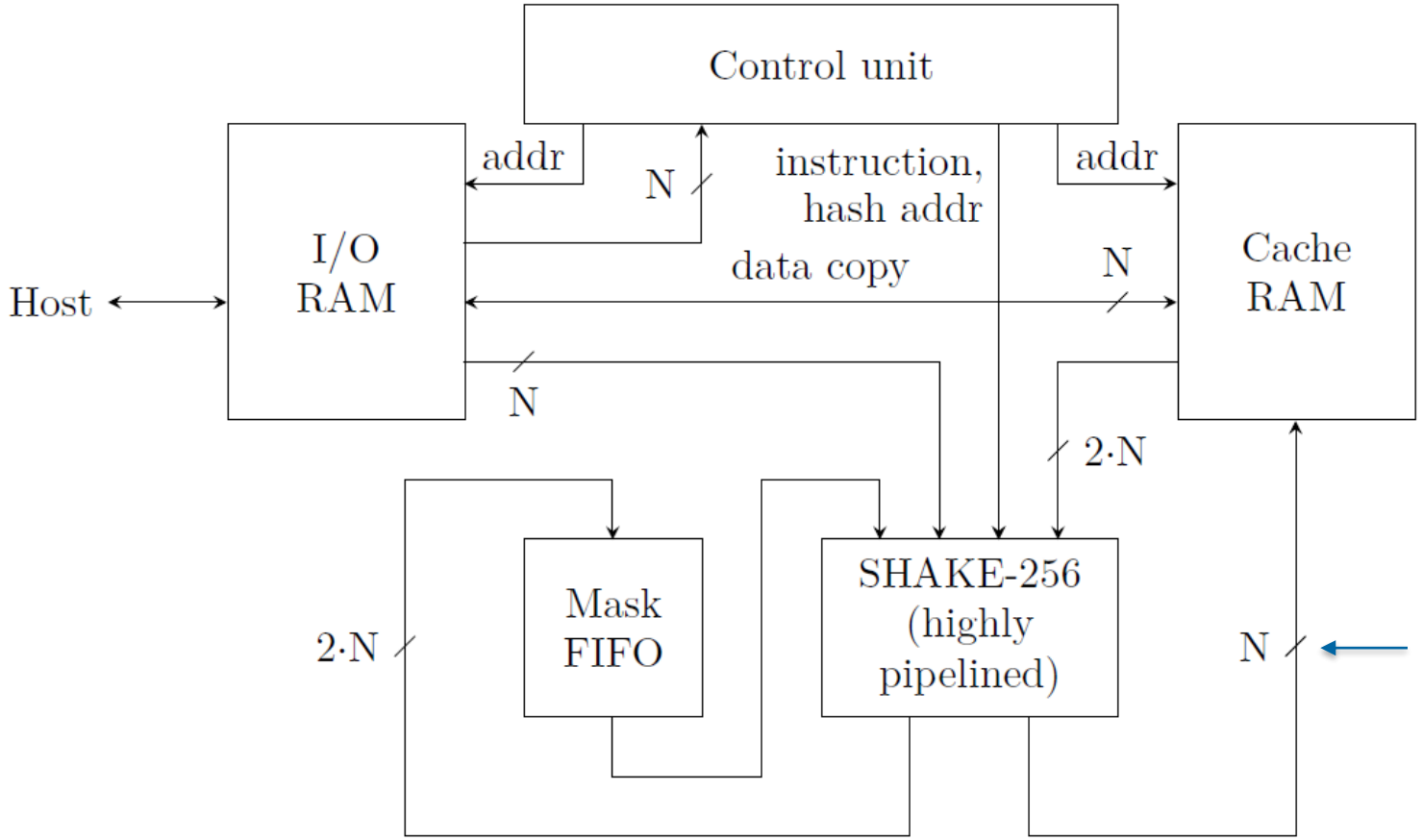




SPHINCS+

- **Submitted to the NIST post-quantum project**
- **Some adjustments to SPHINCS-256**
  - Few-Time signature is now more efficient (security, processing time, signature size)
  - Change underlying hash function
  - Masks are generated (PRNG) => reduces key sizes
- **Several instances**
  - Security level 1, 3, and 5 (≙ 128, 192, and 256 bit)
  - Different hash functions
    - SHAKE-256 (SHA-3)
    - SHA-256
    - Haraka
  - Always a fast (larger signature) and a small (slower processing) version

SPHINCS+ Core Top



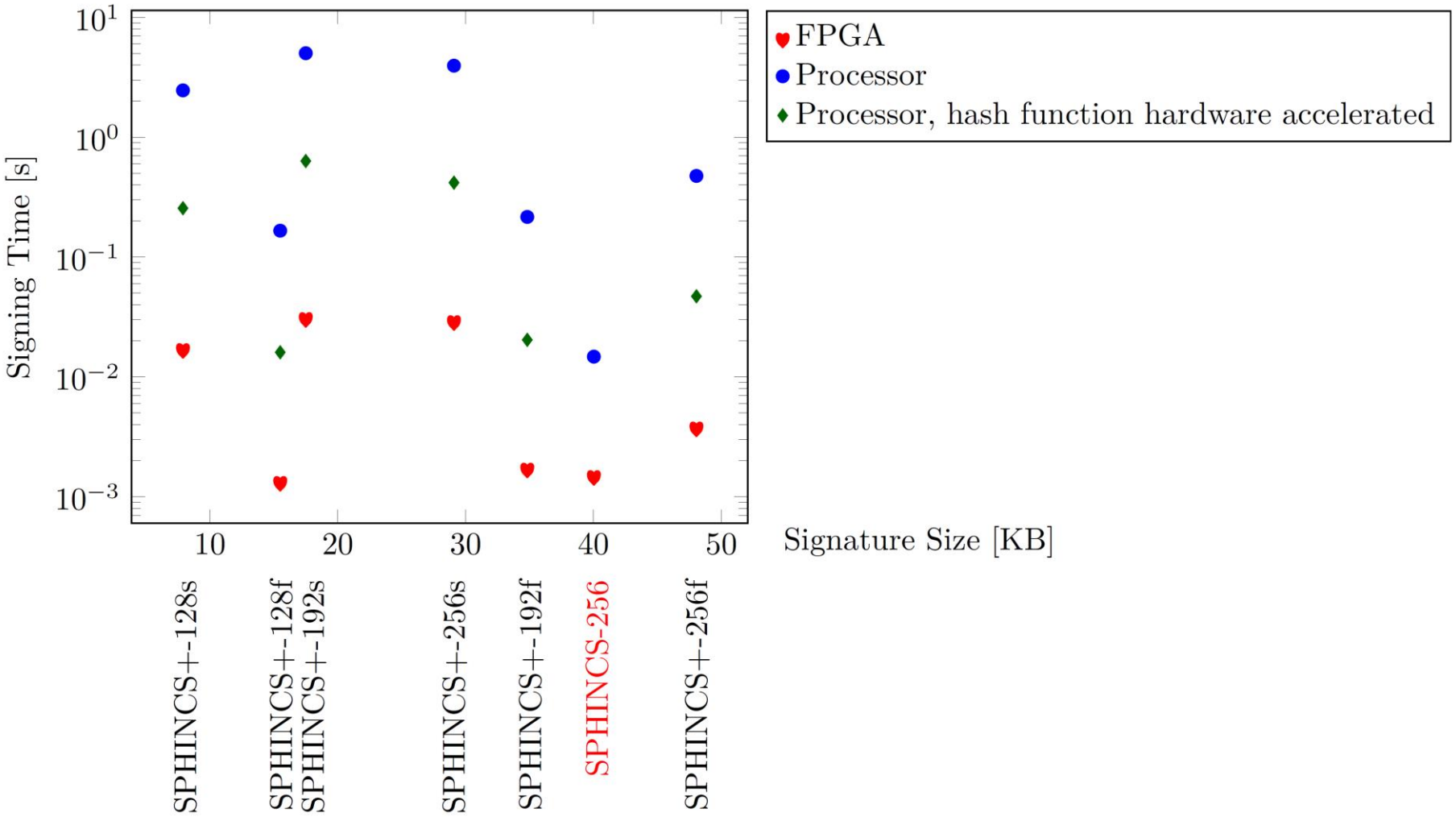
Performance Results

Instance	Sign	FPGA Resources			Sign	Clock	T sign	T verify
	[KByte]	LUT	FF	BRAM	[clks]	[MHz]	[ms]	[ms]
SPHINCS-256	40	19k	38k	36	805k	525	1.53	0.07
SPHINCS+-SHAKE256-128s	7.9	49k	73k	15.5	5,275k	300*	17.58	0.09
SPHINCS+-SHAKE256-128f	16.6	47k	73k	15.5	410k	300*	1.37	0.19
SPHINCS+-SHAKE256-192s	16.7	50k	74k	22.5	9,569k	300*	31.90	0.12
SPHINCS+-SHAKE256-192f	34.8	50k	74k	22.5	530k	300*	1.77	0.25
SPHINCS+-SHAKE256-256s	29.1	50k	76k	30	9,025k	300*	30.08	0.17
SPHINCS+-SHAKE256-256f	48	52k	76k	30	1,169k	300*	3.90	0.28

*Clock frequency of SHAKE-256 pipeline runs at 600 MHz

All results are related to Xilinx Kintex-7 device (XC7K325T-2)

Performance Comparison



Summary

- **FPGA Implementation SPHINCS-256**
  - >600 sign/s, >15000 verifications/s for SPHINCS-256
- **FPGA Implementation SPHINCS+-SHAKE256-128f**
  - >700 sign/s, >5000 verifications/s for
- **SPA: Protected**
- **DPA: Robust**
  - We tried hard, but could not extract any key bits.

Thank you

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IMES INSTITUTE FOR MICROELECTRONICS AND EMBEDDED SYSTEMS

securosys

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Innosuisse – Swiss Innovation Agency

This work was supported by Innosuisse

Why is SPHINCS+ slower than SPHINCS-256?

- **Factor two is lost due to the mask computation**
- **The hash function SHAKE-256 needs more computational effort than ChaCha12**
- **L-tree computation is faster than the calculation of SHAKE-256 with a long input.**
  - The latter holds only for our highly pipelined FPGA implementation and is caused by pipeline stalls.

Implementation Results

Ref	Scheme	Security Classic	Security PQ	FPGA	Area LUT/FF/DSP/BRAM	f MHz	t ms	t·area s·LUT
this	SPHINCS-256	256	128	K7	19,067/38,132/3/36	525	1.53	29.4
[PDG14]	BLISS-IV	192	?	S6	6,438/6,198/5/7	135	0.35	2.25
[ACZ16]	ECDSA-256	128	0	V7	6,816/4,442/20/0	225	1.49	10.2
[ACZ16]	ECDSA-521	256	0	V7	8,273/7,689/64/0	161	5.02	41.5
[SA14]	RSA-2048	112	0	V7	3,558 slices/54/0	399	5.68	≈60
[BHH+15]	SPHINCS-256	256	128	Haswell CPU E3-1275 (1 core)		3500	14.7	-