On Forging SPHINCS^{+}-Haraka Signatures

on a Fault-tolerant Quantum Computer

Robin Berger & Marcel Tiepelt | 2021

Origins

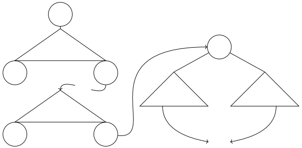
- LatinCrypt 2021 (eprint)

SPHINCS$^+$-128 Explicit (Universal) Forgery

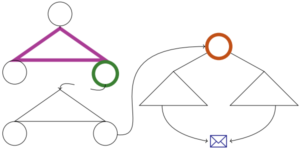
Q#-implementation of SHAKE-256 and Haraka

Quantum Resource Estimate inspired by [Amy et al. 2017]

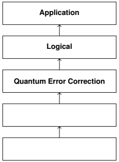
Outline



SPHINCS$^+$



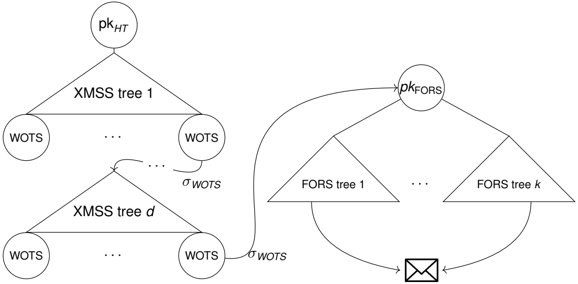
Points of Attack



Results: Resource Estimate

SPHINCS⁺-Components

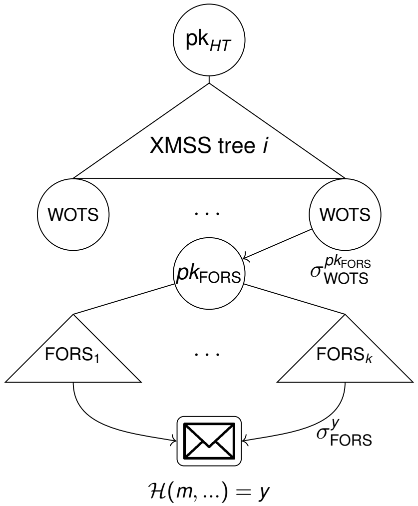
- Hash function $\mathcal{H}$
- Forest Of Random Subsets (FORS)
- Winternitz One-Time Signatures (WOTS)
- eXtended Merkle Signature Scheme (XMSS)



Keys: $pk_{HT} := pk_{\text{SPHINCS}^+}$
sk_{WOTS}, sk_{FORS}

Signature: $\sigma^{m}_{\text{SPHINCS}^+} := \left(\ldots, Path_{\text{XMSS}}, \sigma^{pk_{\text{FORS}}}_{\text{WOTS}}, \sigma^{\mathcal{H}(m,\ldots)}_{\text{FORS}}\right)$

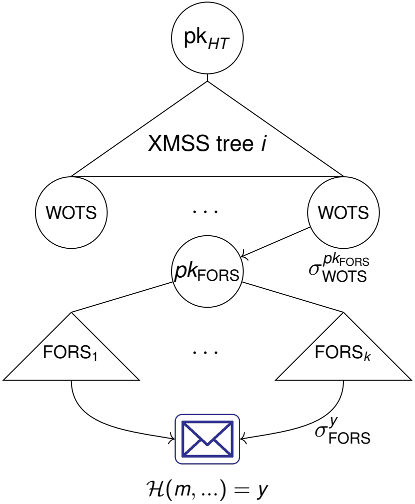
SPHINCS+



pk_{HT}
XMSS tree i
WOTS
...
WOTS
pk_{FORS}
$\sigma^{pk_{FORS}}_{WOTS}$
$FORS_1$
...
$FORS_k$
σ^{y}_{FORS}
$\mathcal{H}(m, ...) = y$

SPHINCS+

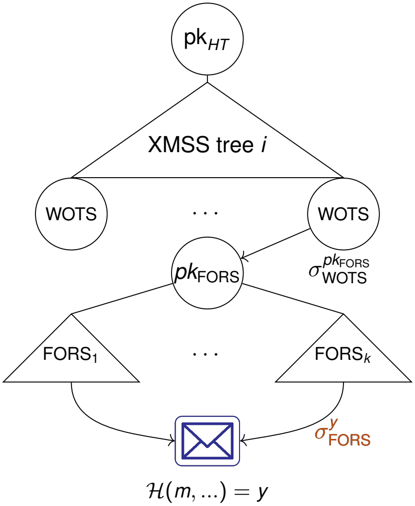
1. Compute message digest $\mathcal{H}(m, ...) := y$



SPHINCS+

1. Compute message digest $\mathcal{H}(m, ...) := y$

2. Generate FORS instance (pk_{FORS}) and sign message digest σ^{y}_{FORS}

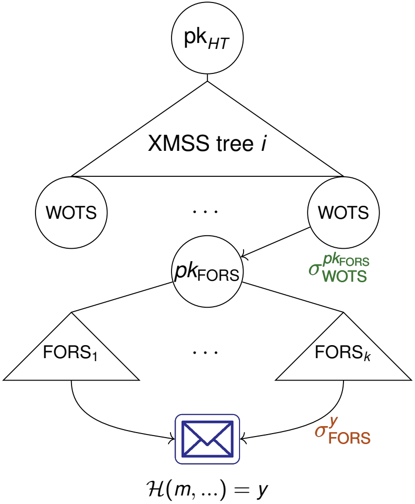


SPHINCS⁺

1. Compute message digest $\mathcal{H}(m, ...) := y$

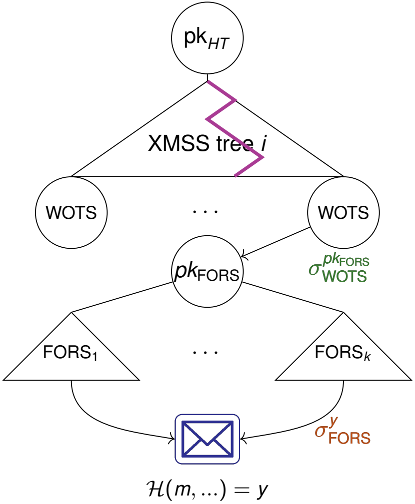
2. Generate FORS instance (pk_{FORS}) and sign message digest σ^{y}_{FORS}

3. Sign pk_{FORS} with WOTS $\sigma^{pk_{\text{FORS}}}_{\text{WOTS}}$

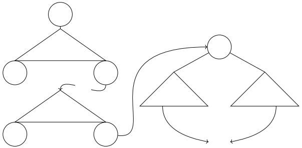


SPHINCS+

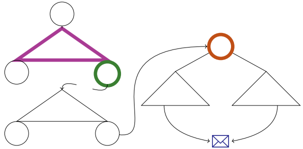
1. Compute message digest $\mathcal{H}(m, ...) := y$
2. Generate FORS instance (pk_{FORS}) and sign message digest σ^y_{FORS}
3. Sign pk_{FORS} with WOTS $\sigma^{pk_{\text{FORS}}}_{\text{WOTS}}$
4. Compute XMSS path to pk_{SPHINCS^+} $\text{Path}_{\text{XMSS}}$



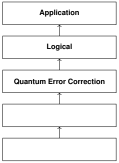
Outline



SPHINCS$^+$

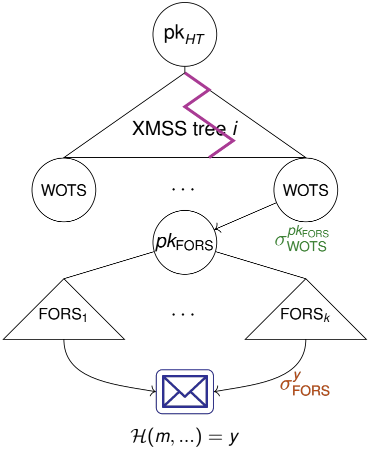


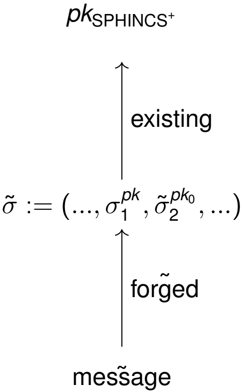
Points of Attack

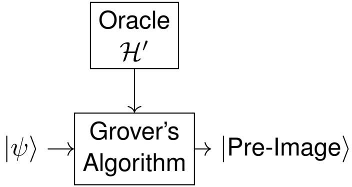


Results: Resource Estimate

General Attack Scheme







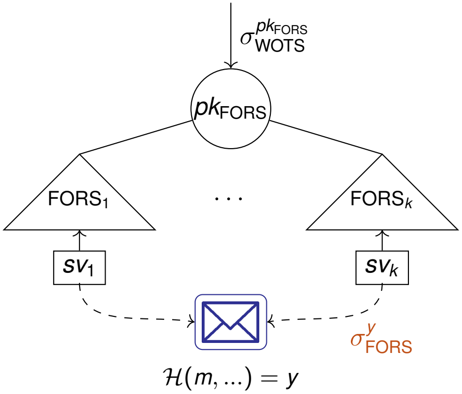
Forgeries

Karlsruhe Institute of Technology

Target	Component	Existential Forgery	Universal Forgery
$\mathcal{H}(m, ...) := y$	$\mathcal{H}$	✓ (oracle depth $=$ 1)	✗
σ^{y}_{FORS}	FORS	✓	✓ (oracle depth $=$ 2) (or multiple pre-images)
$\sigma^{pk_{FORS}}_{\text{HT}}$	WOTS	✓	✓ (oracle depth $=$ 5)
$\text{Path}_{\text{XMSS}}$	XMSS Path	✓	✓ (oracle depth $=$ 1)

Forest Of Random Subsets: Sign and Verify

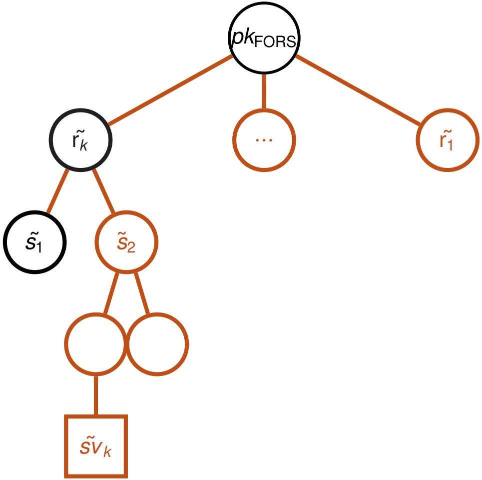
- k trees generated from sk_{FORS}
- pk is hash of all roots
- $\mathcal{H}(m, \ldots)$ determines which (private-key) leaves used
- σ_{FORS} contains $\text{Path}_{\text{FORS}}$ for each tree



Forest Of Random Subsets: Forgery

- First *mutable* sibling $\tilde{s}_1$
- Find $pk_{\text{FORS}} := \mathcal{H}(\tilde{r}_1, ..., \tilde{r}_k := \mathcal{H}(\tilde{s}_1, \tilde{s}_2))$

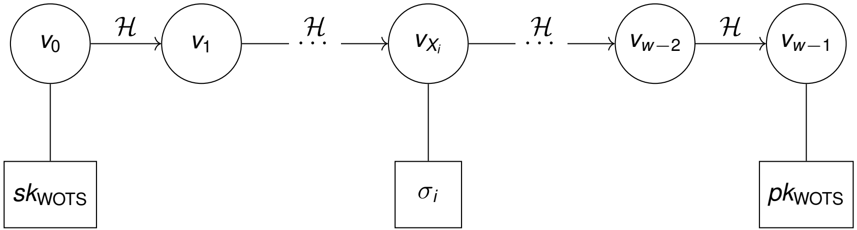
Pre-image search with oracle depth 2



Winternitz One-Time Signatures (WOTS)

- pk_{WOTS} generated from σ_{WOTS}
- σ_i generated from chain of hashes
- Find $pk_{\text{WOTS}} := \mathcal{H}(\mathcal{H}(\mathcal{H}(...)))$

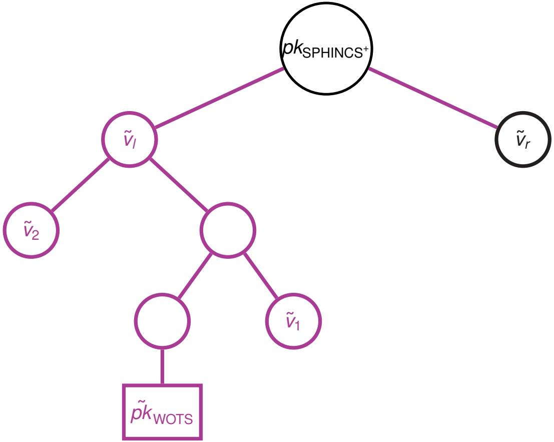
Pre-image search with oracle depth 5



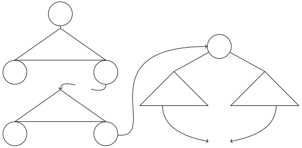
eXtended Merkle Signature Scheme (XMSS)

- First *mutable* sibling $\tilde{v}_r$
- Find $pk_{\text{sphincs}} := \mathcal{H}(\tilde{v}_l, \tilde{v}_r)$

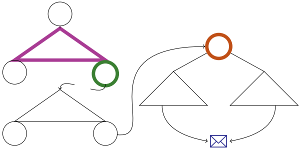
Pre-image search with oracle depth 1



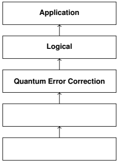
Outline



SPHINCS⁺



Points of Attack



Results: Resource Estimate

Target: Hash Functions

Level	Security
I	128-bit
III	192-bit
V	256-bit

Haraka ← I

I, III, V → SHAKE-256

- Collision on Haraka-Sponge

  $\rightsquigarrow$ second-pre-image

- $\approx 2^{129.5}$ classical hash function invocations [Bertoni et al. 2011]

- Fault-tolerant cost following [N. C. Jones et al. 2012]
  - Assumptions on *current* state-of-the-art
  - Optimizations for magic-state distillation

Metrics

- (Physical, Logical) #Qubits
- #surface code cycles
- #T-gates
- Logical-Qubit-Cycle ($\approx$ classical hash function invoc.) [Amy et al. 2017]
- ...

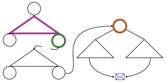
Assumptions

1. Cost Fault-tolerant QC $\approx$ surface codes
2. #physical qubits to embed log. qubit into surface code [Gidney and Ekerå 2021]
3. Error rates qubits p_{in}, gates p_{gate}, time for SCC [Fowler, Devitt, and C. Jones 2013]
4. Quantum gates distributed uniformly across layers

Results

SPHINCS+-			SHAKE-256	Haraka
Collision Attack Chailloux, Naya-Plasencia, and Schrottenloher 2017	#Grover Iterations		—	$1.32 \cdot 2^{102}$
	Time-Space Product		—	$1.51 \cdot 2^{153}$
	#Classical hash function invocations		—	$2^{129.5}$
Path$_{\text{XMSS}}$ on SPHINCS+-128	#Distilleries	ϕ	83×3	3×3
	#Log. Qubits	Q^{log}	23876	2120
	#Total Phys. Qubits	Q^{phy}	$8.65 \cdot 10^6$	$2.03 \cdot 10^6$
	#Total ECC cycles	COST_{SCC}	$1.6 \cdot 2^{84}$	$1.5 \cdot 2^{90}$
	logical-qubit-cycles	COST_{lqc}	$2.65 \cdot 2^{99}$	$1.55 \cdot 2^{101}$

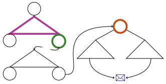
Conclusion



Points of Attack

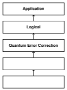
Target	Component	Universal Forgery
σ^{y}_{FORS}	FORS	(oracle depth = 2) (or multiple pre-images)
$\sigma^{pk_{FORS}}_{HT}$	WOTS	(oracle depth = 5)
$Path_{XMSS}$	XMSS Path	(oracle depth = 1)

Conclusion



Points of Attack

Target	Component	Universal Forgery
σ^{y}_{FORS}	FORS	(oracle depth $=2$) (or multiple pre-images)
$\sigma^{pk_{FORS}}_{HT}$	WOTS	(oracle depth $=5$)
$\text{Path}_{\text{XMSS}}$	XMSS Path	(oracle depth $=1$)

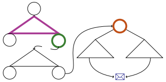


Resource Estimate

SPHINCS^{+}-128		Haraka
$Path_{\text{XMSS}}$	$\#_{\text{Log. Qubits}}$	2120
	$\#_{\text{Total Phys. Qubits}}$	$2.03 \cdot 10^{6}$
	$\#_{\text{Total ECC cycles}}$	$1.5 \cdot 2^{90}$
	logical-qubit-cycles	$1.55 \cdot 2^{101}$

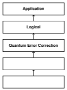
Conclusion

Karlsruhe Institute of Technology



Points of Attack

Target	Component	Universal Forgery
σ^{y}_{FORS}	FORS	(oracle depth $= 2$) (or multiple pre-images)
$\sigma^{pk_{FORS}}_{HT}$	WOTS	(oracle depth $= 5$)
$\text{Path}_{\text{XMSS}}$	XMSS Path	(oracle depth $= 1$)

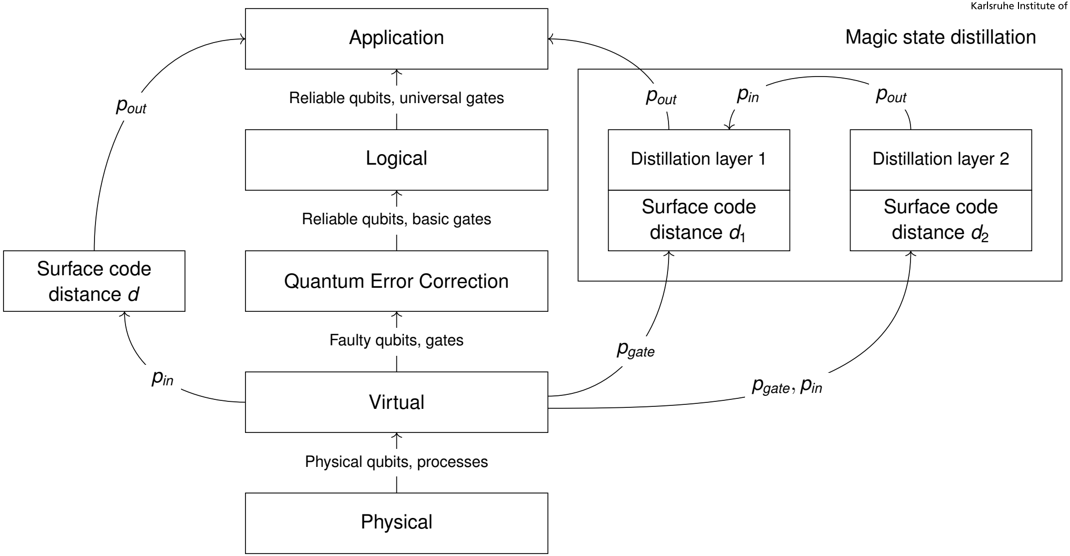


Resource Estimate

SPHINCS+-128		Haraka
$Path_{\text{XMSS}}$	$\#_{\text{Log. Qubits}}$	2120
	$\#_{\text{Total Phys. Qubits}}$	$2.03 \cdot 10^{6}$
	$\#_{\text{Total ECC cycles}}$	$1.5 \cdot 2^{90}$
	logical-qubit-cycles	$1.55 \cdot 2^{101}$

Questions?

Architecture



Optimization: Magic-State Distillation