Tp4

May 31, 2023

1 Sphincs+

1.1 TP4 Grupo 15

Nuno Mata PG44420

Pedro Araujo PG50684

O esquema Sphincs+ é o último de uma cadeia de propostas de esquemas HBS ("hash based signatures"), isto é, esquemas de assinatura digital que apenas usam funções de "hash". Essencialmente, a segurança do esquema de assinaturas depende apenas da segurança das funções de "hash" usadas.

Este esquema foi desenvolvido com o objetivo de ser seguro contra adversários "quânticos" e clássicos/tradicionais.

O Sphincs+ dependes dos seguintes principais conceitos:

```
- WOTS+ (Winternitz One-Time Signature+)
- Árvores de Merkle (Merkle Trees)
- HORST (Hierarchical Offset Ring Signature Tree)
- XMSS (Extended Merkle Signature Scheme)
```

[1]:

```python
import hashlib
```

[2]:

```python
# parametros inciais do SPHINCS+
HASH_ALGO = hashlib.sha256
N_LEVELS = 20
TREE_HEIGHT = 10
WOTS_W = 16 # Representa o numero de bits em cada elemento da assinatura WOTS+
WOTS_LOGW = 4 # Depende do valor WOTS_W, neste caso | log(16) base 2 = 4
```

[3]:

```python
# Funcao que faz a computacao dos Hashes de cada node
def hash_node(data):
    return HASH_ALGO(data).digest()

# Funcao que gera a Merkle Tree
def generate_merkle_tree(seed):
    tree = []
    for i in range(N_LEVELS):
        level = []
        for j in range(2 ** TREE_HEIGHT):
            level.append(hash_node(seed))
        tree.append(level)
        seed = b''.join(level)
    return tree

# Funcao para gerar a private key WOTS+
def generate_wots_sk(seed):
    sk = []
    for i in range(WOTS_W):
        sk.append(seed)
        seed = hash_node(seed)
    return sk

# Funcao que faz a computacao da public key WOTS+
def compute_wots_pk(sk):
    pk = b''
    for i in range(len(sk)):
        for j in range(2 ** WOTS_W):
            pk += hash_node(sk[i] + bytes([j % 256]))
    return pk

# Funcao para fazer a computacao da assinatura WOTS+
def sign_wots(message, sk):
    signature = []
    for i in range(len(message)):
        for j in range(WOTS_W):
            idx = message[i] % (2 ** WOTS_LOGW)
            signature.append(hash_node(sk[j] + bytes([idx])))
            message[i] //= (2 ** WOTS_LOGW)
    return signature

# Funcao para verificar a assinatura WOTS+
def verify_wots(message, signature, pk):
    for i in range(len(message)):
        for j in range(WOTS_W):
            idx = message[i] % (2 ** WOTS_LOGW)
            pk_chunk = pk[:HASH_ALGO().digest_size]
            pk = pk[HASH_ALGO().digest_size:]
            if hash_node(signature[j] + bytes([idx])) != pk_chunk:
                return False
            message[i] //= (2 ** WOTS_LOGW)
    return True

# HORST scheme que vai assinar a mensagem
def sign_horst(message, sk, merkle_tree):
    wots_sk = generate_wots_sk(sk)
    wots_pk = compute_wots_pk(wots_sk)
    wots_signature = sign_wots(message, wots_sk)
    horst_signature = b''.join(wots_signature)
    horst_signature += wots_pk
    return horst_signature

# Funcao que verifica a assinatura usando o HORST scheme
def verify_horst(message, signature, pk, merkle_tree):
    wots_signature = []
    for i in range(len(signature) // HASH_ALGO().digest_size):
        wots_signature.append(signature[:HASH_ALGO().digest_size])
        signature = signature[HASH_ALGO().digest_size:]
    wots_pk = signature
    wots_pk_chunks = [wots_pk[i:i+HASH_ALGO().digest_size] for i in range(0,␣
 ↪len(wots_pk), HASH_ALGO().digest_size)]

    if not verify_wots(message, wots_signature, wots_pk_chunks):
        return False

    auth_path = []
    index = int.from_bytes(wots_pk, 'big')
    for level in merkle_tree:
        auth_path.append(level[index])
        index >>= 1

    leaf = wots_pk_chunks[0]
    for i in range(len(message)):
        leaf = hash_node(message[i].to_bytes(1, 'big') + leaf) + auth_path[i]

    return leaf == merkle_tree[0][int.from_bytes(pk, 'big')]
```

```
[4]: def main():
    message = [1, 2, 3]  # Mensagem que vai ser assinada

    # Chaves pub e privada
    sk = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
    pk = b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

    seed = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a'  # Valor da seed para␣
 ↪gerar a Merkle tree

    # Generate the Merkle tree
    merkle_tree = generate_merkle_tree(seed)

    # Assinar a mensagem com o HORST scheme
    signature = sign_horst(message, sk, merkle_tree)

    # Verificar a assinatura usando outra vez o HORST scheme
    is_valid = verify_horst(message, signature, pk, merkle_tree)

    if is_valid:
        print("Signature is valid.")
    else:
        print("Signature is invalid.")
```

[ ]:

```
if __name__ == '__main__':
    main()
```

[ ]:

[ ]: