Sphincs+

June 23, 2020

1 Trabalho Prático 4

1.1 Sphincs+

```
In [1]: from sphincs_aux import *
        from xmss import Xmss
        from fors import Fors


        class Sphincs:

            def __init__(self):
                #self._randomize = True

                #Parametros

                self._n = 16 #Parametro de segurança
                self._w = 16 #Parametro de Winternitz (4, 16 ou 256)
                self._h = 64 #Altura da Hypertree
                self._d = 8 #Camadas da Hypertree
                self._k = 10 #Numero de arvores no FORS (Forest of Random Subsets)
                self._a = 15 #Numero de folhas de cada arvore no FORS

                self._len_1 = ceil(8 * self._n / log(self._w, 2))
                self._len_2 = floor(log(self._len_1 * (self._w - 1), 2) / log(self._w, 2)) + 1
                self._len_0 = self._len_1 + self._len_2 # n-bit values in WOTS+ sk, pk, and si

                self._h_prime = self._h // self._d
                self._t = 2 ** self._a

                # XMSS e FORS
                self.xmss = Xmss()
                self.fors = Fors()


                self.size_md = floor((self._k * self._a + 7) / 8)
                self.size_idx_tree = floor((self._h - self._h // self._d + 7) / 8)
                self.size_idx_leaf = floor((self._h // self._d + 7) / 8)


            # Implementação SPHINCS+
            # Return: sk, pk
            def sphincs_keygen(self):

                # Geração dos seeds
                secret_seed = os.urandom(self._n) # Para gerar sk do WOTS
                secret_prf = os.urandom(self._n)
                public_seed = os.urandom(self._n)

                public_root = self.xmss.hypertree_pk_gen(secret_seed, public_seed)

                return [secret_seed, secret_prf, public_seed, public_root], [public_seed, publi


            def sphincs_sign(self, m, secret_key):
                #  Assinatura FORS do hash da mensagem, assinatura WOTS+ da pk do FORS corresp
                # uma série de caminhos de autenticação, além das assinaturas WOTS+ para auten

                adrs = ADRS()

                #Obter seeds
                secret_seed = secret_key[0]
                secret_prf = secret_key[1]
                public_seed = secret_key[2]
                public_root = secret_key[3]

                #opt = bytes(self._n)
                opt = os.urandom(self._n)
                r = prf_msg(secret_prf, opt, m, self._n)
                sig = [r]


                #hash da mensagem
                #comprime a mensagem a ser assinada
                digest = hash_msg(r, public_seed, public_root, m, self.size_md + self.size_idx
                tmp_md = digest[:self.size_md]
                tmp_idx_tree = digest[self.size_md:(self.size_md + self.size_idx_tree)]
                tmp_idx_leaf = digest[(self.size_md + self.size_idx_tree):len(digest)]

                md_int = int.from_bytes(tmp_md, 'big') >> (len(tmp_md) * 8 - self._k * self._a
                md = md_int.to_bytes(ceil(self._k * self._a / 8), 'big')

                idx_tree = int.from_bytes(tmp_idx_tree, 'big') >> (len(tmp_idx_tree) * 8 - (se
                idx_leaf = int.from_bytes(tmp_idx_leaf, 'big') >> (len(tmp_idx_leaf) * 8 - (se
                # Armazena os endereços
                adrs.set_layer_address(0)
                adrs.set_tree_address(idx_tree)
                adrs.set_type(ADRS.FORS_TREE)
                adrs.set_key_pair_address(idx_leaf)

                # Assinatura do FORS
                sig_fors = self.fors.fors_sign(md, secret_seed, public_seed, adrs.copy())
                sig += [sig_fors]

                pk_fors = self.fors.fors_pk_from_sig(sig_fors, md, public_seed, adrs.copy())

                # Assinatura das PK do FORS utilizando a Hypertree
                adrs.set_type(ADRS.TREE)
                sig_hypertree = self.xmss.hypertree_sign(pk_fors, secret_seed, public_seed, idx
                sig += [sig_hypertree]

                return sig


            def sphincs_verify(self, m, sig, public_key):
                adrs = ADRS()
                r = sig[0]
                sig_fors = sig[1]
                sig_hypertree = sig[2]

                public_seed = public_key[0]
                public_root = public_key[1]

                #hash da mensagem
                #comprime a mensagem a ser assinada
                digest = hash_msg(r, public_seed, public_root, m, self.size_md + self.size_idx_
                tmp_md = digest[:self.size_md]
                tmp_idx_tree = digest[self.size_md:(self.size_md + self.size_idx_tree)]
                tmp_idx_leaf = digest[(self.size_md + self.size_idx_tree):len(digest)]

                md_int = int.from_bytes(tmp_md, 'big') >> (len(tmp_md) * 8 - self._k * self._a)
                md = md_int.to_bytes(ceil(self._k * self._a / 8), 'big')

                idx_tree = int.from_bytes(tmp_idx_tree, 'big') >> (len(tmp_idx_tree) * 8 - (sel
                idx_leaf = int.from_bytes(tmp_idx_leaf, 'big') >> (len(tmp_idx_leaf) * 8 - (sel

                #Armazena os endereços
                adrs.set_layer_address(0)
                adrs.set_tree_address(idx_tree)
                adrs.set_type(ADRS.FORS_TREE)
                adrs.set_key_pair_address(idx_leaf)
                # Obter PK do FORS através da assinatura
                pk_fors = self.fors.fors_pk_from_sig(sig_fors, md, public_seed, adrs)

                adrs.set_type(ADRS.TREE)
                # Se Hypertree verifica a PK do FORS, retorna True
                return self.xmss.hypertree_verify(pk_fors, sig_hypertree, public_seed, idx_tre

            # IMPLEMENTAÇÃO DO SPHINCS

            def keygen(self):
                """
                Gerar um par de chaves para o Sphincs+ signatures
                :return: secret key e public key
                """
                sk, pk = self.sphincs_keygen()
                sk_0, pk_0 = bytes(), bytes()

                for i in sk:
                    sk_0 += i
                for i in pk:
                    pk_0 += i

                return sk_0, pk_0

            def sign(self, m, sk):
                """
                Assinar uma mensagem com o algoritmo Sphincs
                :param m: Mensagem a ser assinada
                :param sk: Secret Key
                :return: Assinatura da mensagem com a Secret key
                """

                sk_tab = [sk[(i * self._n):((i + 1) * self._n)] for i in range(4)]

                sig_tab = self.sphincs_sign(m, sk_tab)

                sig = sig_tab[0]  # R
                for i in sig_tab[1]:  # SIG FORS
                    sig += i
                for i in sig_tab[2]:  # SIG Hypertree
                    sig += i

                return sig

            def verify(self, m, sig, pk):
                """
                Verificar a assinatura
                :param m: Mensagem
                :param sig: Assinatura
                :param pk: Public Key
                :return: Boolean True se assinatura correta
                """

                pk_tab = [pk[(i * self._n):((i + 1) * self._n)] for i in range(2)]

                sig_tab = []

                sig_tab += [sig[:self._n]]  # R

                sig_tab += [[]]  # SIG FORS
                for i in range(self._n,
                               self._n + self._k * (self._a + 1) * self._n,
                               self._n):
                    sig_tab[1].append(sig[i:(i + self._n)])

                sig_tab += [[]]  # SIG Hypertree
                for i in range(self._n + self._k * (self._a + 1) * self._n,
                               self._n + self._k * (self._a + 1) * self._n + (self._h + self._
                               self._n):
                    sig_tab[2].append(sig[i:(i + self._n)])

                return self.sphincs_verify(m, sig_tab, pk_tab)
```

1.2 Teste

```
In [2]: sphincs = Sphincs()
        sk, pk = sphincs.keygen()
        print("sk: ", sk)
        print("\npk: ", pk)

        m = os.urandom(32)
        print("\nMensagem: ", m)

        signature = sphincs.sign(m, sk)

        print("\nAssinatura correta? ", sphincs.verify(m, signature, pk))
```

```
sk:  b'\xafS2\xd7q\xb5a\xf9\x17>\xf9.3L\x10\xcb\xfa!4_\x14G\xa9:\xa4\xeb/\xb9\x05\x04\xc45}D\xc

pk:  b'}D\xcaE]r{\x01g\x1d\xeb?\xf1\xb5\xea?\xa7\xf7\x9f\x92\xb1\x1e\x96NN\x03\xbc\x84q\xa2\x0

Mensagem:  b"\x17_2W\r\x05\x01\xb8\x84\x98\x8c'\xc1-u\x8c\x06\xb8\x89Z\xf6\x07\x13]\x8fIn\x17\

Assinatura correta?  True
```