Hash-based signatures II

Lamport and Winternitz one-time signatures

Tanja Lange
(with some slides by Daniel J. Bernstein)

Eindhoven University of Technology

SAC – Post-quantum cryptography

Lamport's 1-time signature system

Sign arbitrary-length message by signing its 256-bit hash:

```
def keypair():
  keys = [signbit.keypair() for n in range(256)]
  public,secret = zip(*keys)
  return  public,secret

def sign(message,secret):
  msg = message.to_bytes(200, byteorder="little")
  h = sha3_256(msg)
  hbits = [1 & (h[i//8])>>(i%8) for i in range(256)]
  sigs = [signbit.sign(hbits[i],secret[i]) for i in range(256)]
  return sigs, message

def open(sm,public):
  message = sm[1]
  msg = message.to_bytes(200, byteorder="little")
  h = sha3_256(msg)
  hbits = [1 & (h[i//8])>>(i%8) for i in range(256)]
  for i in range(256):
    if hbits[i] != signbit.open(sm[0][i],public[i]):
      raise Exception('bit %d of hash does not match' % i)
  return message
```

Want to sign 4 bits with just 32 bytes

- Lamport's signatures have 2×256 hash outputs (each 32 bytes) as public key and the signature has 256 times 32 bytes.
- Define
$$H^i(x) = H(H^{i-1}(x)) = \underbrace{H(H(\dots(H(x))))}_{i \text{ times}}.$$
- Pick random sk, compute pk$= H^{16}(\mathrm{sk})$.
- For message m reveal $s = H^m(sk)$ as signature.
- To verify check that pk$= H^{16-m}(s)$.

Weak Winternitz

```
def keypair():
  secret = sha3_256(os.urandom(32))
  public = sha3_256(secret)
  for i in range(16): public = sha3_256(public)
  return public,secret

def sign(m,secret):
  if type(m) != int: raise Exception('message must be int')
  if m < 0 or m > 15: raise Exception('message must be between 0
  sign = secret
  for i in range(m): sign = sha3_256(sign)
  return sign, m

def open(sm,public):
  if type(sm[1]) != int: raise Exception('message must be int')
  if sm[1] < 0 or sm[1] > 15: raise Exception('message must be b
  check = sm[0]
  for i in range(16-sm[1]): check = sha3_256(check)
  if sha3_256(check) != public: raise Exception('bad signature')
  return sm[1]
```

Want to sign 4 bits with just 32 bytes

- Lamport's signatures have 2×256 hash outputs (each 32 bytes) as public key and the signature has 256 times 32 bytes.
- Define
$$H^i(x) = H(H^{i-1}(x)) = \underbrace{H(H(\ldots(H(x))))}_{i \text{ times}}.$$
- Pick random sk, compute pk$= H^{16}(\text{sk})$.
- For message m reveal $s = H^m(sk)$ as signature.
- To verify check that pk$= H^{16-m}(s)$.
- This works – but is insecure!

Want to sign 4 bits with just 32 bytes

- Lamport's signatures have 2×256 hash outputs (each 32 bytes) as public key and the signature has 256 times 32 bytes.
- Define
$$H^i(x) = H(H^{i-1}(x)) = \underbrace{H(H(\ldots(H(x))))}_{i \text{ times}}.$$
- Pick random sk, compute pk$= H^{16}(\text{sk})$.
- For message m reveal $s = H^m(sk)$ as signature.
- To verify check that pk$= H^{16-m}(s)$.
- This works – but is insecure!
Eve can take $H(s)$ as signature on $m+1$ (for $m < 15$).

Want to sign 4 bits with just 32 bytes

- Lamport's signatures have 2×256 hash outputs (each 32 bytes) as public key and the signature has 256 times 32 bytes.
- Define
$$H^i(x) = H(H^{i-1}(x)) = \underbrace{H(H(\dots(H(x))))}_{i \text{ times}}.$$
- Pick random sk, compute pk$= H^{16}(\text{sk})$.
- For message m reveal $s = H^m(sk)$ as signature.
- To verify check that pk$= H^{16-m}(s)$.
- This works – but is insecure!
  Eve can take $H(s)$ as signature on $m+1$ (for $m < 15$).
- Fix by doubling the key-sizes again, running one chain forward, one in reverse.

Slow Winternitz 1-time signature system for 4 bits

Could stop at 15 iterations, but convenient to reuse code here:

```
import weak_winternitz
def keypair():
  keys = [weak_winternitz.keypair() for n in range(2)]
  public,secret = zip(*keys)
  return public,secret

def sign(m,secret):
  sign0 = weak_winternitz.sign(m,secret[0])
  sign1 = weak_winternitz.sign(16-m,secret[1])
  return sign0, sign1, m

def open(sm,public):
  m0 = weak_winternitz.open(sm[0],public[0])
  m1 = weak_winternitz.open(sm[1],public[1])
  if m0 != sm[2] or  m1 != (16-sm[2]): raise Exception('Invalid
  return sm[2]
```

Winternitz 1-time signature system

- Define parameter w. Each chain will run for 2^w steps.
- For signing a 256-bit hash this needs $t_1 = \lceil 256/w \rceil$ chains.
  Write m in base 2^w (integers of w bits):

  $$m = (m_{t_1-1}, \ldots, m_1, m_0)$$

  (zero–padding if necessary).
- Put

  $$c = \sum_{i=0}^{t_1-1} (2^w - m_i)$$

  Note that $c \leq t_1 2^w$.
- The checksum c gets larger if m_i is smaller.
- Write c in base 2^w. This takes $t_2 = 1 + \lceil \lfloor (\log_2 t_1 \rfloor + 1)/w \rceil$ w-bit integers

  $$c = (c_{t_2-1}, \ldots, c_1, c_0).$$

- Publish $t_1 + t_2$ public keys, sign with chains of lengths

  $$m_{t_1-1}, \ldots, m_1, m_0, c_{t_2-1}, \ldots, c_1, c_0.$$

Winternitz 1-time signature system for $w = 8$

- Define parameter $w = 8$. Each chain will run for $2^8 = 256$ steps.
- For signing a 256-bit hash this needs $t_1 = \lceil 256/8 \rceil = 32$ chains. Write m in base 2^8 (integers of 8 bits):

  $$m = (m_{31}, \ldots, m_1, m_0)$$

  (zero–padding if necessary).
- Put

  $$c = \sum_{i=0}^{31} (2^8 - m_i)$$

  Note that $c \leq 32 \cdot 2^8 = 2^{13}$.
- The checksum c gets larger if m_i is smaller.
- Write c in base 2^8. This takes $t_2 = 1 + \lceil (5+1)/8 \rceil = 2$ 8-bit integers

  $$c = (c_1, c_0).$$
- Publish $t_1 + t_2 = 34$ public keys, sign with chains of lengths

  $$m_{31}, \ldots, m_1, m_0, c_1, c_0.$$