Lecture outline

- Unit testing
- What I did in industry

What can go wrong with a crypto implementation?

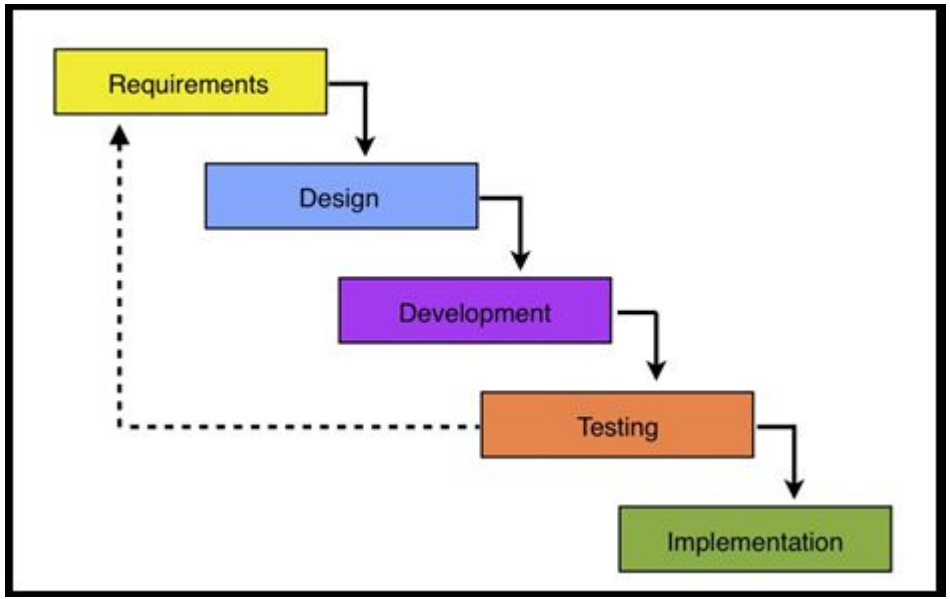
What can go wrong with a crypto implementation?

Pretty much everything!!

How can we ensure things *probably* won't go wrong?

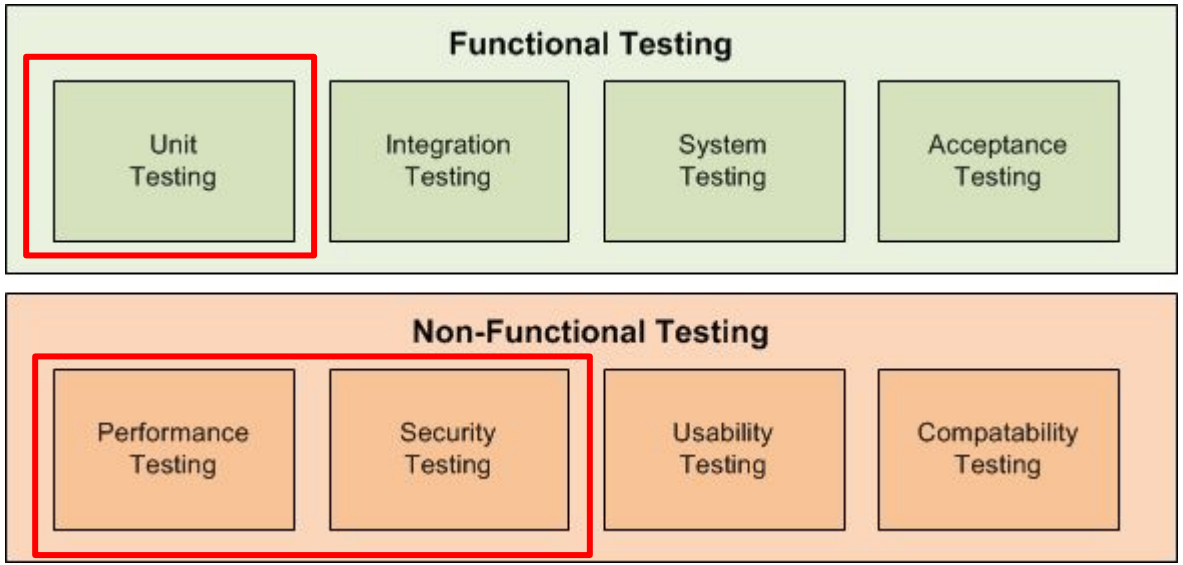
Testing!

- Unit tests
- Integration tests
- Whole-system tests
- Performance tests
- Regression tests



Portions of this lesson are taken from
http://katyhuff.github.io/python-testing/

For crypto libraries...



What is a unit test?

Verify behavior of smallest possible piece

Implementation and interfaces

Document behavior on correct and incorrect inputs

Three basic parts:

1. setup
2. assertion(s)
3. teardown

Interface

```
#Encrypts data
def encrypt(self, data):
    …

#Decrypts data
def decrypt(self, ciphertext):
    ...
```

```
def setup(self):
    key = base64.urlsafe_b64encode('1234567890abcdef')
    self._lpc6 = LengthPreservingCipher(key, 6)

def test_basic(self):
    self.setup()
    txt = b'abcdef'
    ctx = self._lpc6.encrypt(txt)
    ttxt = self._lpc6.decrypt(ctx)
    assert ttxt == txt

def test_random_msg(self):
    self.setup()
    for i in xrange(101):
        txt = os.urandom(6)
        ctx = self._lpc6.encrypt(txt)
        ttxt = self._lpc6.decrypt(ctx)
        assert txt == ttxt,
        "Input and out messages do not match.”
```

```
def test_wrong_length_encrypt(self):
        self.setup()
        with pytest.raises(AssertionError) as e:
            self._lpc6.encrypt('a')
        with pytest.raises(AssertionError) as e:
            self._lpc6.encrypt('a'*10)

    def test_wrong_length_decrypt(self):
        self.setup()
        msg = 'a'
        with pytest.raises(AssertionError) as e:
            self._lpc6.decrypt('a')
        with pytest.raises(AssertionError) as e:
            self._lpc6.decrypt('a'*10)

    def test_length_of_cipher(self):
        self.setup()
        txt = b'abcdef'
        ctx = self._lpc6.encrypt(txt)
        assert len(ctx) == len(txt)
```

How to write unit tests?

Decompose code into atomic units of functionality

Verify input/output behavior for each piece

Check edge cases!

Recombine pieces, check whole parts

100% code coverage is necessary but not usually sufficient

Code comments

Document assumptions about inputs

What happens if assumptions are violated?

Think about your future self

```
#Encrypts data
def encrypt(self, data):
    …

#Decrypts data
def decrypt(self, ciphertext):
    ...
```

vs.

```
#takes a six-byte plaintext,
#returns a six-byte ciphertext
def encrypt(self, data):
    …

#takes a six-byte ciphertext,
#returns a six-byte plaintext
def decrypt(self, ciphertext):
    ...
```

Practice problem

```
def xor(a,b):
    """
    xors two raw byte streams.
    """
    assert len(a) == len(b), "Lengths of two strings are not same. a = "\
        "{}, b = {}\n{}\n{}".format(len(a), len(b), a , b)
    return ''.join(chr(ord(ai)^ord(bi)) for ai,bi in zip(a,b))

class Insecure:
    def __init__(self, backend=None):
        if backend is None:
            self._backend = default_backend()
        else:
            self._backend = backend
        self.block_size_bytes = hashes.SHA256().block_size

        self.opad = chr(0x5c)*self.block_size_bytes
        self.ipad = chr(0x36)*self.block_size_bytes

    def make_key(self, k):
        if len(k) <= self.block_size_bytes:
            return k + chr(0x00)*(self.block_size_bytes - len(k))
        else:
            digest = hashes.Hash(hashes.SHA256(),
backend=default_backend())
            digest.update(k)
            retval = digest.finalize()
            return retval + chr(0x00)*(self.block_size_bytes - len(retval))
```

```
def func(self, k, m):
        digest = hashes.Hash(hashes.SHA256(), backend=self._backend)
        padded_key = self.make_key(k)
        i_key_pad = xor(padded_key,self.ipad)
        o_key_pad = xor(padded_key,self.opad)
        digest.update(i_key_pad + m)
        intermediate = digest.finalize()
        second_digest = hashes.Hash(hashes.SHA256(), backend=self._backend)
        second_digest.update(o_key_pad + intermediate)
        output = second_digest.finalize()
        return output

    def func2(self, k, m, t):
        digest = hashes.Hash(hashes.SHA256(), backend=self._backend)
        padded_key = self.make_key(k)
        i_key_pad = xor(padded_key,self.ipad)
        o_key_pad = xor(padded_key,self.opad)
        digest.update(i_key_pad + m)
        intermediate = digest.finalize()
        second_digest = hashes.Hash(hashes.SHA256(), backend=self._backend)
        second_digest.update(o_key_pad + intermediate)
        output = second_digest.finalize()
        if not len(t) == len(output):
            return False
        for b in xrange(len(output)):
            if not output[b] == t[b]:
                return False
        return True
```

Interoperability testing

Implementations of crypto have to be interoperable, especially if you're implementing a cross-platform specification (like in hw4!)

How do we do this?

-Cross-impl correctness

-Known-answer tests

Certifications

NIST Cryptographic algorithm verification program (CAVP)

-Standardized known-answer tests for USG standards

Skyhigh Networks 1601 S. De Anza Blvd. Ste. 248 Cupertino, CA 95014 USA -Kaushik Narayan TEL: 408 564-0278	**Skyhigh Secure Gateway** Version 1	Intel Core i7 w/ Mac OS X-10.9 with JVM 1.7.0.45	10/31/2014	**CBC** (e/d; 128 , 256); **CTR** (ext only; 128 , 256) "Cloud Encryption Gateway" *11/06/14: Updated implementation information;*

Certifications

Federal information processing standard (FIPS) 140-2

-The worst thing in the world, incredibly outdated and harmful for security

-critical vulnerabilities *cannot be fixed* without invalidating FIPS certification

-To call it a garbage fire is actually an insult to honest garbage fires, which get rid of trash and generate heat

Preview of other testing methods

- Randomness “tests”
  - Can we develop an algorithm that, given some input string, will determine whether or not it is random?
- Fuzzers
- Formal verification
- Project Wycheproof
  - https://github.com/google/wycheproof
- Check for non-constant-time behavior
- (A subsequent lecture may discuss these in more detail)

Questions? Comments?