HW4: Password-Based Authenticated Encryption

(due Friday, April 14th)

In this assignment, you will build an implementation of the PWFernet specification. The specification document can be found at this url. The PWFernet spec was created in response to issue #1333 on the cryptography.io library's Github page.

Grading

(50%): Fully implement the PWFernet spec. Your code should be clean, well-commented, and easy to read. It should not contain debugging print statements, dead code, or other cruft. We will split this 50% up as follows:

(10%): Quality and cleanliness of code and comments.
(15%): Self-correctness (whether it can decrypt something it encrypted).
(25%): Correctly implements PWFernet spec. For example, can you decrypt a ciphertext encrypted with our implementation?

(25%): Write a unit testing suite for your implementation. Your tests should cover basic correctness as well as proper handling of error conditions and edge cases. Ideally, your tests will cover 100% of the lines of your implementation. Code coverage metrics can be checked using a tool like pytest-cov.

(25%): Interoperability testing. After you're confident you've implemented the spec correctly, generate five test vectors and upload them along with your solution. The test vectors should be Python dicts of the following format:

```
dict(password = b'password',
        salt = b'salt',
        size='small',
        timestamp=1490650404,
        message=b'a great secret message'
        result = '6d1bb878eee9ce4a7b77d7a44103574d4cbfe3c15ae3940f0ffe75cd5e1e0afa')
```

where the 'size' is one of {'small', 'medium', 'large', 'xlarge'} and the timestamp is an integer representing a Unix timestamp. **This is not a correct test vector**; it merely illustrates the format.

Once everyone has submitted, we will cross-test every implementation with the test vectors of the other implementations. You'll receive full credit for this part if your implementation works correctly for other students' test vectors. Obviously, the test vectors will first be checked for correctness with respect to our reference implementation so you aren't penalized if another student's implementation is wrong.

Deliverables

Submit a single Python file (pwfernet.py) containing your implementation and a text document (i.e. with a .txt extension) containing the five test vectors.

Other instructions

Because the cryptography library's current stable version does not support the OpenSSL Scrypt bindings on all platforms, you will be using a pure Python implementation for this assignment. We will provide a file containing the implementation as well as a class implementing the Scrypt interface for cryptography.io. The only thing different in your code will be instantiating the backend: rather than default_backend() instantiated as

backend = default_backend()

we will use a MultiBackend containing both the provided Scrypt backend and the default backend instantiated as

backend = MultiBackend([NewScryptBackend(),default_backend()]).

Note that the 'NewScryptBackend()' *must* come first in the list.

We're using the Python Scrypt implementation to make it easier to write the PWFernet implementation, but it is important to note that because the Python version is much slower, a production implementation of the spec should use the native OpenSSL Scrypt. This is not just a performance issue - slow implementations force systems to use weaker parameters to derive passwords, which makes brute-force attacks more efficient.