**Danger**

This is a "Hazardous Materials" module. You should **ONLY** use it if you're 100% absolutely sure that you know what you're doing because this module is full of land mines, dragons, and dinosaurs with laser guns.

# Cipher-based message authentication code (CMAC)

Cipher-based message authentication codes (or CMACs) are a tool for calculating message authentication codes using a block cipher coupled with a secret key. You can use an CMAC to verify both the integrity and authenticity of a message.

A subset of CMAC with the AES-128 algorithm is described in **RFC 4493**.

*class* **cryptography.hazmat.primitives.cmac.CMAC**(*algorithm*, *backend=None*)

*New in version 0.4.*

CMAC objects take a `BlockCipherAlgorithm` instance.

```
>>> from cryptography.hazmat.primitives import cmac
>>> from cryptography.hazmat.primitives.ciphers import algorithms
>>> c = cmac.CMAC(algorithms.AES(key))
>>> c.update(b"message to authenticate")
>>> c.finalize()
b'CT\x1d\xc8\x0e\x15\xbe4e\xdb\xb6\x84\xca\xd9Xk'
```

If the backend doesn't support the requested `algorithm` an `UnsupportedAlgorithm` exception will be raised.

If `algorithm` isn't a `BlockCipherAlgorithm` instance then `TypeError` will be raised.

To check that a given signature is correct use the `verify()` method. You will receive an exception if the signature is wrong:

```
>>> c = cmac.CMAC(algorithms.AES(key))
>>> c.update(b"message to authenticate")
>>> c.verify(b"an incorrect signature")
Traceback (most recent call last):
...
cryptography.exceptions.InvalidSignature: Signature did not match digest.
```

**Parameters:**
- **algorithm** – An instance of `BlockCipherAlgorithm`.
- **backend** – An optional instance of `CMACBackend`.

**Raises:**
- TypeError – This is raised if the provided `algorithm` is not an instance of `BlockCipherAlgorithm`
- cryptography.exceptions.UnsupportedAlgorithm – This is raised if the provided `backend` does not implement `CMACBackend`

### `update`(*data*)

**Parameters:** **data** (*bytes*) – The bytes to hash and authenticate.

**Raises:**
- cryptography.exceptions.AlreadyFinalized – See `finalize()`
- TypeError – This exception is raised if `data` is not `bytes`.

### copy()

Copy this `CMAC` instance, usually so that we may call `finalize()` to get an intermediate value while we continue to call `update()` on the original instance.

**Returns:** A new instance of `CMAC` that can be updated and finalized independently of the original instance.

**Raises:** cryptography.exceptions.AlreadyFinalized – See `finalize()`

### `verify`(*signature*)

Finalize the current context and securely compare the MAC to `signature`.

**Parameters:** **signature** (*bytes*) – The bytes to compare the current CMAC against.

**Raises:**

- cryptography.exceptions.AlreadyFinalized – See `finalize()`
- cryptography.exceptions.InvalidSignature – If signature does not match digest
- TypeError – This exception is raised if `signature` is not `bytes`.

### `finalize()`

Finalize the current context and return the message authentication code as bytes.

After `finalize` has been called this object can no longer be used and `update()`, `copy()`, `verify()` and `finalize()` will raise an `AlreadyFinalized` exception.

**Return bytes:** The message authentication code as bytes.

**Raises:** cryptography.exceptions.AlreadyFinalized –