

### ⚠ 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.

## Key derivation functions

Key derivation functions derive bytes suitable for cryptographic operations from passwords or other data sources using a pseudo-random function (PRF). Different KDFs are suitable for different tasks such as:

- Cryptographic key derivation

Deriving a key suitable for use as input to an encryption algorithm. Typically this means taking a password and running it through an algorithm such as `PBKDF2HMAC` or `HKDF`. This process is typically known as [key stretching](#).

- Password storage

When storing passwords you want to use an algorithm that is computationally intensive. Legitimate users will only need to compute it once (for example, taking the user’s password, running it through the KDF, then comparing it to the stored value), while attackers will need to do it billions of times. Ideal password storage KDFs will be demanding on both computational and memory resources.

## Variable cost algorithms

### Argon2 Family

The Argon2 family of key derivation functions are designed for password storage and is described in [RFC 9106](#). It consists of three variants that differ only how they access an internal memory buffer, which leads to different trade-offs in resistance to hardware attacks.

Each of the classes constructors and parameters are the same; only details of Argon2id are defined before, for brevity.

---

```
class cryptography.hazmat.primitives.kdf.argon2.Argon2d(*, salt, length, it
memory_cost, ad=None, secret=None) \[source\]
```

 [latest](#) ▼

Added in version 47.0.0.

This variant of the Argon2 family maximizes resistance to time-memory-trade-off attacks, but introduces possible side-channels

---

```
class cryptography.hazmat.primitives.kdf.argon2.Argon2i(*, salt, length, iterations, lanes,
memory_cost, ad=None, secret=None) \[source\]
```

Added in version 47.0.0.

This variant of the Argon2 family resists side-channel attacks, but is vulnerable to time-memory-trade-off attacks

---

```
class cryptography.hazmat.primitives.kdf.argon2.Argon2id(*, salt, length, iterations, lanes,
memory_cost, ad=None, secret=None) \[source\]
```

Added in version 44.0.0.

Argon2id is a blend of the previous two variants. Argon2id should be used by most users, as recommended in [RFC 9106](#).

This class conforms to the `KeyDerivationFunction` interface.

```
>>> import os
>>> from cryptography.hazmat.primitives.kdf.argon2 import Argon2id
>>> salt = os.urandom(16)
>>> # derive
>>> kdf = Argon2id(
...     salt=salt,
...     length=32,
...     iterations=1,
...     lanes=4,
...     memory_cost=64 * 1024,
...     ad=None,
...     secret=None,
... )
>>> key = kdf.derive(b"my great password")
>>> # verify
>>> kdf = Argon2id(
...     salt=salt,
...     length=32,
...     iterations=1,
...     lanes=4,
...     memory_cost=64 * 1024,
...     ad=None,
...     secret=None,
... )
>>> kdf.verify(b"my great password", key)
```

All arguments to the constructor are keyword-only.

- Parameters:**
- **salt** (*bytes*) – A salt should be unique (and randomly generated) per password and is recommended to be 16 bytes or longer
  - **length** (*int*) – The desired length of the derived key in bytes.
  - **iterations** (*int*) – Also known as passes, this is used to tune the running time independently of the memory size.
  - **lanes** (*int*) – The number of lanes (parallel threads) to use. Also known as parallelism.
  - **memory\_cost** (*int*) – The amount of memory to use in kibibytes. 1 kibibyte (KiB) is 1024 bytes. This must be at minimum `8 * lanes`.
  - **ad** (*bytes*) – Optional associated data.
  - **secret** (*bytes*) – Optional secret data; used for keyed hashing.

[RFC 9106](#) has recommendations for [parameter choice](#).

**Raises:** [cryptography.exceptions.UnsupportedAlgorithm](#) – If Argon2id is not supported by the OpenSSL version `cryptography` is using.

**derive(key\_material)**

- Parameters:** **key\_material** (*bytes-like*) – The input key material.
- Return bytes:** the derived key.
- Raises:**
- [TypeError](#) – This exception is raised if `key_material` is not `bytes`.
  - [cryptography.exceptions.AlreadyFinalized](#) – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates and returns a new key from the supplied password.

**derive\_into(key\_material, buffer)**

*Added in version 47.0.0.*

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material.
  - **buffer** – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** The number of bytes written to the buffer.
- Raises:**
- [TypeError](#) – This exception is raised if `key_material` is not `bytes`.
  - [ValueError](#) – This exception is raised if the buffer is too small for the derived key.
  - [cryptography.exceptions.AlreadyFinalized](#) – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates a new key from the supplied password and writes it directly into the provided buffer.

 [latest](#) ▼

**verify(key\_material, expected\_key)**

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.
- Raises:**
- **`cryptography.exceptions.InvalidKey`** – This is raised when the derived key does not match the expected key.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match. This can be used for checking whether the password a user provides matches the stored derived key.

### `derive_phc_encoded(key_material)`

*Added in version 45.0.0.*

- Parameters:** **key\_material** (*bytes-like*) – The input key material.
- Return str:** A PHC-formatted string containing the parameters, salt, and derived key.
- Raises:** **`cryptography.exceptions.AlreadyFinalized`** – This is raised when any method is called more than once.

This method generates and returns a new key from the supplied password, formatting the result as a string according to the Password Hashing Competition (PHC) format. The returned string includes the algorithm, all parameters, the salt, and the derived key in a standardized format:

```
$argon2id$v=19$m=<memory_cost>,t=<iterations>,p=<lanes>${<salt>}${<key>}
```

This format is suitable for password storage and is compatible with other Argon2id implementations that support the PHC format.

### `classmethod verify_phc_encoded(key_material, phc_encoded, secret=None)`

*Added in version 45.0.0.*

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive_phc_encoded()`.
  - **phc\_encoded** (*str*) – A PHC-formatted string as returned by `derive_phc_encoded()`.
  - **secret** (*bytes*) – Optional secret data; used for keyed hashing.
- Raises:** **`cryptography.exceptions.InvalidKey`** – This is raised when the derived key does not match the key in the encoded string or when the format of the encoded string is invalid.

This class method verifies whether the supplied `key_material` matches the key

contained in the PHC-formatted string. It extracts the parameters from the string, recomputes the key with those parameters, and compares the result to the key in the string.

This is useful for validating a password against a stored PHC-formatted hash string.

```
>>> import os
>>> from cryptography.hazmat.primitives.kdf.argon2 import Argon2id
>>> salt = os.urandom(16)
>>> # Create an Argon2id instance and derive a PHC-formatted string
>>> kdf = Argon2id(
...     salt=salt,
...     length=32,
...     iterations=1,
...     lanes=4,
...     memory_cost=64 * 1024,
... )
>>> encoded = kdf.derive_phc_encoded(b"my great password")
>>> # Later, verify the password
>>> Argon2id.verify_phc_encoded(b"my great password", encoded)
```

## PBKDF2

---

```
class cryptography.hazmat.primitives.kdf.pbkdf2.PBKDF2HMAC(algorithm, length, salt,
iterations) \[source\]
```

Added in version 0.2.

**PBKDF2** (Password Based Key Derivation Function 2) is typically used for deriving a cryptographic key from a password. It may also be used for key storage, but an alternate key storage KDF such as `Scrypt` is generally considered a better solution.

This class conforms to the `KeyDerivationFunction` interface.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
>>> # Salts should be randomly generated
>>> salt = os.urandom(16)
>>> # derive
>>> kdf = PBKDF2HMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     iterations=1_200_000,
... )
>>> key = kdf.derive(b"my great password")
>>> # verify
>>> kdf = PBKDF2HMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     iterations=1_200_000,
... )
>>> kdf.verify(b"my great password", key)
```

 [latest](#) ▼

- Parameters:**
- **algorithm** – An instance of `HashAlgorithm`.
  - **length** (*int*) – The desired length of the derived key in bytes. Maximum is  $(2^{32} - 1) * \text{algorithm.digest\_size}$ .
  - **salt** (*bytes*) – A salt. Secure values <sup>1</sup> are 128-bits (16 bytes) or longer and randomly generated.
  - **iterations** (*int*) – The number of iterations to perform of the hash function. This can be used to control the length of time the operation takes. Higher numbers help mitigate brute force attacks against derived keys. A [more detailed description](#) can be consulted for additional information.
- Raises:**
- **TypeError** – This exception is raised if `salt` is not `bytes`.

### `derive(key_material)`

- Parameters:** **key\_material** (*bytes-like*) – The input key material. For PBKDF2 this should be a password.
- Return bytes:** the derived key.
- Raises:**
- **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.
  - **TypeError** – This exception is raised if `key_material` is not `bytes`.

This generates and returns a new key from the supplied password.

### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material. For PBKDF2 this should be a password.
  - **buffer** – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** The number of bytes written to the buffer.
- Raises:**
- **TypeError** – This exception is raised if `key_material` is not `bytes`.
  - **ValueError** – This exception is raised if the buffer is too small for the derived key.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates a new key from the supplied password and writes it directly into the provided buffer.

### `verify(key_material, expected_key)`

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.
- Raises:**
- **`cryptography.exceptions.InvalidKey`** – This is raised when the derived key does not match the expected key.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match. This can be used for checking whether the password a user provides matches the stored derived key.

## Script

---

`class cryptography.hazmat.primitives.kdf.script.Script(salt, length, n, r, p)` [\[source\]](#)

*Added in version 1.6.*

Script is a KDF designed for password storage by Colin Percival to be resistant against hardware-assisted attackers by having a tunable memory cost. It is described in [RFC 7914](#).

This class conforms to the `KeyDerivationFunction` interface.

```
>>> import os
>>> from cryptography.hazmat.primitives.kdf.script import Script
>>> salt = os.urandom(16)
>>> # derive
>>> kdf = Script(
...     salt=salt,
...     length=32,
...     n=2**14,
...     r=8,
...     p=1,
... )
>>> key = kdf.derive(b"my great password")
>>> # verify
>>> kdf = Script(
...     salt=salt,
...     length=32,
...     n=2**14,
...     r=8,
...     p=1,
... )
>>> kdf.verify(b"my great password", key)
```

- Parameters:**
- **salt** (*bytes*) – A salt.
  - **length** (*int*) – The desired length of the derived key in bytes.
  - **n** (*int*) – CPU/Memory cost parameter. It must be larger than 1 and be a power of 2.
  - **r** (*int*) – Block size parameter.
  - **p** (*int*) – Parallelization parameter.

The computational and memory cost of Scrypt can be adjusted by manipulating the 3 parameters: `n`, `r`, and `p`. In general, the memory cost of Scrypt is affected by the values of both `n` and `r`, while `n` also determines the number of iterations performed. `p` increases the computational cost without affecting memory usage. A more in-depth explanation of the 3 parameters can be found [here](#).

**RFC 7914** recommends values of `r=8` and `p=1` while scaling `n` to a number appropriate for your system. [The scrypt paper](#) suggests a minimum value of `n=2**14` for interactive logins ( $t < 100\text{ms}$ ), or `n=2**20` for more sensitive files ( $t < 5\text{s}$ ).

- Raises:**
- **`cryptography.exceptions.UnsupportedAlgorithm`** – If Scrypt is not supported by the OpenSSL version `cryptography` is using.
  - **`TypeError`** – This exception is raised if `salt` is not `bytes`.
  - **`ValueError`** – This exception is raised if `n` is less than 2, if `n` is not a power of 2, if `r` is less than 1 or if `p` is less than 1.

#### `derive(key_material)`

- Parameters:** `key_material` (*bytes-like*) – The input key material.
- Return bytes:** the derived key.
- Raises:**
- **`TypeError`** – This exception is raised if `key_material` is not `bytes`.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates and returns a new key from the supplied password.

#### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

- Parameters:**
- `key_material` (*bytes-like*) – The input key material.
  - `buffer` (*bytes-like*) – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** the number of bytes written to the buffer.

**Raises:**

- **ValueError** – This exception is raised if the buffer length does not match the specified `length`.
- **TypeError** – This exception is raised if `key_material` or `buffer` is not `bytes`.
- **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates a new key from the supplied password and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

```
verify(key_material, expected_key)
```

**Parameters:**

- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
- **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- **cryptography.exceptions.InvalidKey** – This is raised when the derived key does not match the expected key.
- **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match. This can be used for checking whether the password a user provides matches the stored derived key.

## Fixed cost algorithms

### ConcatKDF

---

```
class cryptography.hazmat.primitives.kdf.concatkdf.ConcatKDFHash(algorithm, length,  
otherinfo) \[source\]
```

*Added in version 1.0.*

ConcatKDFHash (Concatenation Key Derivation Function) is defined by the NIST Special Publication [NIST SP 800-56Ar3](#) document, to be used to derive keys for use after a Key Exchange negotiation operation.

#### ⚠ Warning

ConcatKDFHash should not be used for password storage.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.concatkdf import ConcatKDFHash
>>> otherinfo = b"concatkdf-example"
>>> ckdf = ConcatKDFHash(
...     algorithm=hashes.SHA256(),
...     length=32,
...     otherinfo=otherinfo,
... )
>>> key = ckdf.derive(b"input key")
>>> ckdf = ConcatKDFHash(
...     algorithm=hashes.SHA256(),
...     length=32,
...     otherinfo=otherinfo,
... )
>>> ckdf.verify(b"input key", key)
```

- Parameters:**
- **algorithm** – An instance of `HashAlgorithm`.
  - **length** (*int*) – The desired length of the derived key in bytes. Maximum is `hashlen * (2^32 - 1)`.
  - **otherinfo** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.
- Raises:**
- **`TypeError`** – This exception is raised if `otherinfo` is not `bytes`.

### `derive(key_material)`

- Parameters:** **key\_material** (*bytes-like*) – The input key material.
- Return bytes:** The derived key.
- Raises:**
- **`TypeError`** – This exception is raised if `key_material` is not `bytes`.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material.

### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material.
  - **buffer** (*bytes-like*) – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** the number of bytes written to the buffer.

**Raises:**

- **ValueError** – This exception is raised if the buffer length does not match the specified `length`.
- **TypeError** – This exception is raised if `key_material` or `buffer` is not `bytes`.
- **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

`verify(key_material, expected_key)`

**Parameters:**

- **key\_material** (`bytes`) – The input key material. This is the same as `key_material` in `derive()`.
- **expected\_key** (`bytes`) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- **cryptography.exceptions.InvalidKey** – This is raised when the derived key does not match the expected key.
- **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

---

`class cryptography.hazmat.primitives.kdf.concatkdf.ConcatKDFHMAC(algorithm, length, salt, otherinfo)` [\[source\]](#)

*Added in version 1.0.*

Similar to `ConcatKDFHash` but uses an HMAC function instead.

**⚠ Warning**

`ConcatKDFHMAC` should not be used for password storage.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.concatkdf import ConcatKDFMAC
>>> salt = os.urandom(16)
>>> otherinfo = b"concatkdf-example"
>>> ckdf = ConcatKDFMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     otherinfo=otherinfo,
... )
>>> key = ckdf.derive(b"input key")
>>> ckdf = ConcatKDFMAC(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     otherinfo=otherinfo,
... )
>>> ckdf.verify(b"input key", key)
```

**Parameters:**

- **algorithm** – An instance of `HashAlgorithm`.
- **length** (*int*) – The desired length of the derived key in bytes. Maximum is `hashlen * (232 - 1)`.
- **salt** (*bytes*) – A salt. Randomizes the KDF's output. Optional, but highly recommended. Ideally as many bits of entropy as the security level of the hash: often that means cryptographically random and as long as the hash output. Does not have to be secret, but may cause stronger security guarantees if secret; If `None` is explicitly passed a default salt of `algorithm.block_size` null bytes will be used.
- **otherinfo** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.

**Raises:** `TypeError` – This exception is raised if `salt` or `otherinfo` is not `bytes`.

**derive(key\_material)**

**Parameters:** `key_material` (*bytes*) – The input key material.

**Return bytes:** The derived key.

- Raises:**
- `TypeError` – This exception is raised if `key_material` is not `bytes`.
  - `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material.

**derive\_into(key\_material, buffer)**

Added in version 47.0.0.

 [latest](#) ▼

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material.
  - **buffer** (*bytes-like*) – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** the number of bytes written to the buffer.
- Raises:**
- **ValueError** – This exception is raised if the buffer length does not match the specified `length`.
  - **TypeError** – This exception is raised if `key_material` or `buffer` is not `bytes`.
  - **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

`verify(key_material, expected_key)`

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.
- Raises:**
- **cryptography.exceptions.InvalidKey** – This is raised when the derived key does not match the expected key.
  - **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

## HKDF

---

`class cryptography.hazmat.primitives.kdf.hkdf.HKDF(algorithm, length, salt, info)` [\[source\]](#)

*Added in version 0.2.*

**HKDF** (HMAC-based Extract-and-Expand Key Derivation Function) is suitable for deriving keys of a fixed size used for other cryptographic operations.

### ⚠ Warning

HKDF should not be used for password storage.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.hkdf import HKDF
>>> salt = os.urandom(16)
>>> info = b"hkdf-example"
>>> hkdf = HKDF(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     info=info,
... )
>>> key = hkdf.derive(b"input key")
>>> hkdf = HKDF(
...     algorithm=hashes.SHA256(),
...     length=32,
...     salt=salt,
...     info=info,
... )
>>> hkdf.verify(b"input key", key)
```

**Parameters:**

- **algorithm** – An instance of `HashAlgorithm`.
- **length** (*int*) – The desired length of the derived key in bytes. Maximum is `255 * (algorithm.digest_size // 8)`.
- **salt** (*bytes*) – A salt. Randomizes the KDF's output. Optional, but highly recommended. Ideally as many bits of entropy as the security level of the hash: often that means cryptographically random and as long as the hash output. Worse (shorter, less entropy) salt values can still meaningfully contribute to security. May be reused. Does not have to be secret, but may cause stronger security guarantees if secret; see [RFC 5869](#) and the [HKDF paper](#) for more details. If `None` is explicitly passed a default salt of `algorithm.digest_size // 8` null bytes will be used. See [understanding HKDF](#) for additional detail about the salt and info parameters.
- **info** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.

**Raises:**

**TypeError** – This exception is raised if `salt` or `info` is not `bytes`.

**static** `extract(algorithm, salt, key_material)`

*Added in version 47.0.0.*

**Note**

Extract is a component of the complete HKDF algorithm. Unless needed for implementing an existing protocol, users should ignore this method and use call `derive()`.

- Parameters:**
- **algorithm** – An instance of `HashAlgorithm`.
  - **salt** (*bytes*) – A salt. Randomizes the KDF's output. Optional, but highly recommended. Ideally as many bits of entropy as the security level of the hash: often that means cryptographically random and as long as the hash output. Worse (shorter, less entropy) salt values can still meaningfully contribute to security. May be reused. Does not have to be secret, but may cause stronger security guarantees if secret; see [RFC 5869](#) and the [HKDF paper](#) for more details. If `None` is explicitly passed a default salt of `algorithm.digest_size // 8` null bytes will be used. See [understanding HKDF](#) for additional detail about the salt and info parameters.
  - **key\_material** (*bytes-like*) – The input key material.
- Return bytes:** The extracted value.
- Raises:** `TypeError` – This exception is raised if `key_material`, `salt`, or `algorithm` are the wrong type.

### `derive(key_material)`

- Parameters:** **key\_material** (*bytes-like*) – The input key material.
- Return bytes:** The derived key.
- Raises:**
- `TypeError` – This exception is raised if `key_material` is not `bytes`.
  - `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material by performing both the extract and expand operations.

### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material.
  - **buffer** – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** The number of bytes written to the buffer.
- Raises:**
- `TypeError` – This exception is raised if `key_material` is not `bytes`.
  - `ValueError` – This exception is raised if the buffer is too small for the derived key.
  - `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material by performing both the extract and expand operations, writing the result into the provided buffer.

```
verify(key_material, expected_key)
```

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.
- Raises:**
- **`cryptography.exceptions.InvalidKey`** – This is raised when the derived key does not match the expected key.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

---

**class** `cryptography.hazmat.primitives.kdf.hkdf.HKDFExpand`(*algorithm, length, info*) [\[source\]](#)

*Added in version 0.5.*

HKDF consists of two stages, extract and expand. This class exposes an expand only version of HKDF that is suitable when the key material is already cryptographically strong.

### ⚠ Warning

HKDFExpand should only be used if the key material is cryptographically strong. You should use `HKDF` if you are unsure.

```
>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.hkdf import HKDFExpand
>>> info = b"hkdf-example"
>>> key_material = os.urandom(16)
>>> hkdf = HKDFExpand(
...     algorithm=hashes.SHA256(),
...     length=32,
...     info=info,
... )
>>> key = hkdf.derive(key_material)
>>> hkdf = HKDFExpand(
...     algorithm=hashes.SHA256(),
...     length=32,
...     info=info,
... )
>>> hkdf.verify(key_material, key)
```

- Parameters:**
- **algorithm** – An instance of `HashAlgorithm`.
  - **length** (*int*) – The desired length of the derived key in bytes. Maximum is `255 * (algorithm.digest_size // 8)`.
  - **info** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.
- Raises:** `TypeError` – This exception is raised if `info` is not `bytes`.

#### `derive(key_material)`

- Parameters:** **key\_material** (*bytes*) – The input key material.
- Return bytes:** The derived key.
- Raises:**
- `TypeError` – This exception is raised if `key_material` is not `bytes`.
  - `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material by only performing the expand operation.

#### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

- Parameters:**
- **key\_material** (*bytes*) – The input key material.
  - **buffer** – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** The number of bytes written to the buffer.
- Raises:**
- `TypeError` – This exception is raised if `key_material` is not `bytes`.
  - `ValueError` – This exception is raised if the buffer is too small for the derived key.
  - `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material by only performing the expand operation, writing the result into the provided buffer.

#### `verify(key_material, expected_key)`

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- [cryptography.exceptions.InvalidKey](#) – This is raised when the derived key does not match the expected key.
- [cryptography.exceptions.AlreadyFinalized](#) – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.
- [TypeError](#) – This is raised if the provided `key_material` is a `unicode` object

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

## KBKDF

---

```
class cryptography.hazmat.primitives.kdf.kbkdf.KBKDFHMAC(algorithm, mode, length, rlen, llen, location, label, context, fixed) \[source\]
```

*Added in version 1.4.*

KBKDF (Key Based Key Derivation Function) is defined by the [NIST SP 800-108](#) document, to be used to derive additional keys from a key that has been established through an automated key-establishment scheme.

### ⚠ Warning

KBKDFHMAC should not be used for password storage.

```

>>> import os
>>> from cryptography.hazmat.primitives import hashes
>>> from cryptography.hazmat.primitives.kdf.kbkdf import (
...     CounterLocation, KBKDFHMAC, Mode
... )
>>> label = b"KBKDF HMAC Label"
>>> context = b"KBKDF HMAC Context"
>>> kdf = KBKDFHMAC(
...     algorithm=hashes.SHA256(),
...     mode=Mode.CounterMode,
...     length=32,
...     rlen=4,
...     llen=4,
...     location=CounterLocation.BeforeFixed,
...     label=label,
...     context=context,
...     fixed=None,
... )
>>> key = kdf.derive(b"input key")
>>> kdf = KBKDFHMAC(
...     algorithm=hashes.SHA256(),
...     mode=Mode.CounterMode,
...     length=32,
...     rlen=4,
...     llen=4,
...     location=CounterLocation.BeforeFixed,
...     label=label,
...     context=context,
...     fixed=None,
... )
>>> kdf.verify(b"input key", key)

```

**Parameters:**

- **algorithm** – An instance of `HashAlgorithm`.
- **mode** – The desired mode of the PRF. A value from the `Mode` enum.
- **length** (*int*) – The desired length of the derived key in bytes.
- **rlen** (*int*) – An integer that indicates the length of the binary representation of the counter in bytes.
- **llen** (*int*) – An integer that indicates the binary representation of the `length` in bytes.
- **location** – The desired location of the counter. A value from the `CounterLocation` enum.
- **label** (*bytes*) – Application specific label information. If `None` is explicitly passed an empty byte string will be used.
- **context** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.
- **fixed** (*bytes*) – Instead of specifying `label` and `context` you may supply your own fixed data. If `fixed` is specified, `label` and `context` is ignored.
- **break\_location** (*int*) – A keyword-only argument. An integer that indicates the bytes offset where counter bytes are truncated. Required when `location` is `MiddleFixed`.

**Raises:**

- **`TypeError`** – This exception is raised if `label` or `context` is not `bytes`. Also raised if `rlen`, `llen`, or `break_location` is not `int`.
- **`ValueError`** – This exception is raised if `rlen` or `llen` is greater than 4 or less than 1. This exception is also raised if you specify a `label` or `context` and `fixed`. This exception is also raised if you specify `break_location` and `location` is not `MiddleFixed`.

**`derive(key_material)`**

**Parameters:** `key_material` (*bytes-like*) – The input key material.

**Return bytes:** The derived key.

- Raises:**
- **`TypeError`** – This exception is raised if `key_material` is not `bytes`.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material.

**`derive_into(key_material, buffer)`**

*Added in version 47.0.0.*

**Parameters:**

- `key_material` (*bytes-like*) – The input key material.
- `buffer` (*bytes-like*) – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.

**Return int:** the number of bytes written to the buffer.

- Raises:**
- **`ValueError`** – This exception is raised if the buffer length does not match the specified `length`.
  - **`TypeError`** – This exception is raised if `key_material` or `buffer` is not `bytes`.
  - **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

**`verify(key_material, expected_key)`**

- Parameters:**
- `key_material` (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - `expected_key` (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- [cryptography.exceptions.InvalidKey](#) – This is raised when the derived key does not match the expected key.
- [cryptography.exceptions.AlreadyFinalized](#) – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

---

`class cryptography.hazmat.primitives.kdf.kbkdf.KBKDFCMAC(algorithm, mode, length, rlen, llen, location, label, context, fixed)` [\[source\]](#)

*Added in version 35.0.0.*

KBKDF (Key Based Key Derivation Function) is defined by the [NIST SP 800-108](#) document, to be used to derive additional keys from a key that has been established through an automated key-establishment scheme.

**⚠ Warning**

KBKDFCMAC should not be used for password storage.

```
>>> from cryptography.hazmat.primitives.ciphers import algorithms
>>> from cryptography.hazmat.primitives.kdf.kbkdf import (
...     CounterLocation, KBKDFCMAC, Mode
... )
>>> label = b"KBKDF CMAC Label"
>>> context = b"KBKDF CMAC Context"
>>> kdf = KBKDFCMAC(
...     algorithm=algorithms.AES,
...     mode=Mode.CounterMode,
...     length=32,
...     rlen=4,
...     llen=4,
...     location=CounterLocation.BeforeFixed,
...     label=label,
...     context=context,
...     fixed=None,
... )
>>> key = kdf.derive(b"32 bytes long input key material")
>>> kdf = KBKDFCMAC(
...     algorithm=algorithms.AES,
...     mode=Mode.CounterMode,
...     length=32,
...     rlen=4,
...     llen=4,
...     location=CounterLocation.BeforeFixed,
...     label=label,
...     context=context,
...     fixed=None,
... )
>>> kdf.verify(b"32 bytes long input key material", key)
```

**Parameters:**

- **algorithm** – A class implementing a block cipher algorithm being a subclass of `CipherAlgorithm` and `BlockCipherAlgorithm`.
- **mode** – The desired mode of the PRF. A value from the `Mode` enum.
- **length** (*int*) – The desired length of the derived key in bytes.
- **rlen** (*int*) – An integer that indicates the length of the binary representation of the counter in bytes.
- **llen** (*int*) – An integer that indicates the binary representation of the `length` in bytes.
- **location** – The desired location of the counter. A value from the `CounterLocation` enum.
- **label** (*bytes*) – Application specific label information. If `None` is explicitly passed an empty byte string will be used.
- **context** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.
- **fixed** (*bytes*) – Instead of specifying `label` and `context` you may supply your own fixed data. If `fixed` is specified, `label` and `context` is ignored.
- **break\_location** (*int*) – A keyword-only argument. An integer that indicates the bytes offset where counter bytes are to be located. Required when `location` is `MiddleFixed`.

**Raises:**

- **`cryptography.exceptions.UnsupportedAlgorithm`** – This is raised if `algorithm` is not a subclass of `CipherAlgorithm` and `BlockCipherAlgorithm`.
- **`TypeError`** – This exception is raised if `label` or `context` is not `bytes`, `rlen`, `llen`, or `break_location` is not `int`, `mode` is not `Mode` or `location` is not `CounterLocation`.
- **`ValueError`** – This exception is raised if `rlen` or `llen` is greater than 4 or less than 1. This exception is also raised if you specify a `label` or `context` and `fixed`. This exception is also raised if you specify `break_location` and `location` is not `MiddleFixed`.

**`derive(key_material)`**

**Parameters:** `key_material` (*bytes-like*) – The input key material.

**Return bytes:** The derived key.

**Raises:**

- **`TypeError`** – This exception is raised if `key_material` is not `bytes`.
- **`ValueError`** – This exception is raised if `key_material` is not a valid key for `algorithm` passed to `KBKDFCMAC` constructor.
- **`cryptography.exceptions.AlreadyFinalized`** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material.

```
derive_into(key_material, buffer)
```

Added in version 47.0.0.

- Parameters:**
- **key\_material** (*bytes-like*) – The input key material.
  - **buffer** – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.
- Return int:** the number of bytes written to the buffer.
- Raises:**
- **ValueError** – This exception is raised if the buffer length does not match the specified `length`, or if `key_material` is not a valid key for `algorithm` passed to `KBKDFCMAC` constructor.
  - **TypeError** – This exception is raised if `key_material` or `buffer` is not `bytes`.
  - **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

```
verify(key_material, expected_key)
```

- Parameters:**
- **key\_material** (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.
  - **expected\_key** (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.
- Raises:**
- **cryptography.exceptions.InvalidKey** – This is raised when the derived key does not match the expected key.
  - **cryptography.exceptions.AlreadyFinalized** – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match.

---

```
class cryptography.hazmat.primitives.kdf.kbkdf.Mode \[source\]
```

An enumeration for the key based key derivative modes.

```
CounterMode
```

The output of the PRF is computed with a counter as the iteration variable.

---

```
class cryptography.hazmat.primitives.kdf.kbkdf.CounterLocation \[source\]
```

An enumeration for the key based key derivative counter location.

```
BeforeFixed
```

The counter iteration variable will be concatenated before the fixed input data.

 [latest](#) ▼

### AfterFixed

The counter iteration variable will be concatenated after the fixed input data.

### MiddleFixed

Added in version 38.0.0.

The counter iteration variable will be concatenated in the middle of the fixed input data.

## X963KDF

```
class cryptography.hazmat.primitives.kdf.x963kdf.X963KDF(algorithm, length, otherinfo)  
[source]
```

Added in version 1.1.

X963KDF (ANSI X9.63 Key Derivation Function) is defined by ANSI in the [ANSI X9.63:2001](#) document, to be used to derive keys for use after a Key Exchange negotiation operation.


SECG in [SEC 1 v2.0](#) recommends that `ConcatKDFHash` be used for new projects. This KDF should only be used for backwards compatibility with pre-existing protocols.

### ⚠ Warning

X963KDF should not be used for password storage.

```
>>> import os  
>>> from cryptography.hazmat.primitives import hashes  
>>> from cryptography.hazmat.primitives.kdf.x963kdf import X963KDF  
>>> sharedinfo = b"ANSI X9.63 Example"  
>>> xkdf = X963KDF(  
...     algorithm=hashes.SHA256(),  
...     length=32,  
...     sharedinfo=sharedinfo,  
... )  
>>> key = xkdf.derive(b"input key")  
>>> xkdf = X963KDF(  
...     algorithm=hashes.SHA256(),  
...     length=32,  
...     sharedinfo=sharedinfo,  
... )  
>>> xkdf.verify(b"input key", key)
```

### Parameters:

- **algorithm** – An instance of `HashAlgorithm`.
- **length** (*int*) – The desired length of the derived key in bytes. Maximum is `hashlen * (232 - 1)`.  [latest](#) ▼
- **sharedinfo** (*bytes*) – Application specific context information. If `None` is explicitly passed an empty byte string will be used.

**Raises:** `TypeError` – This exception is raised if `sharedinfo` is not `bytes`.

### `derive(key_material)`

**Parameters:** `key_material` (`bytes-like`) – The input key material.

**Return bytes:** The derived key.

**Raises:**

- `TypeError` – This exception is raised if `key_material` is not `bytes`.
- `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material.

### `derive_into(key_material, buffer)`

*Added in version 47.0.0.*

**Parameters:**

- `key_material` (`bytes-like`) – The input key material.
- `buffer` (`bytes-like`) – A writable buffer to write the derived key into. The buffer must be equal to the length supplied in the constructor.

**Return int:** the number of bytes written to the buffer.

**Raises:**

- `ValueError` – This exception is raised if the buffer length does not match the specified `length`.
- `TypeError` – This exception is raised if `key_material` or `buffer` is not `bytes`.
- `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

Derives a new key from the input key material and writes it into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

### `verify(key_material, expected_key)`

**Parameters:**

- `key_material` (`bytes`) – The input key material. This is the same as `key_material` in `derive()`.
- `expected_key` (`bytes`) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- `cryptography.exceptions.InvalidKey` – This is raised when the derived key does not match the expected key.
- `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` same key as the `expected_key`, and raises an exception if they do not n

# Interface

---

`class cryptography.hazmat.primitives.kdf.KeyDerivationFunction` [\[source\]](#)

Added in version 0.2.

`derive(key_material)` [\[source\]](#)

**Parameters:** `key_material` (*bytes*) – The input key material. Depending on what key derivation function you are using this could be either random bytes, or a user supplied password.

**Returns:** The new key.

**Raises:** `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates and returns a new key from the supplied key material.

`derive_into(key_material, buffer)` [\[source\]](#)

Added in version 47.0.0.

**Parameters:**

- `key_material` (*bytes-like*) – The input key material. Depending on what key derivation function you are using this could be either random bytes, or a user supplied password.
- `buffer` – A writable buffer to write the derived key into.

**Return int:** the number of bytes written to the buffer.

**Raises:**

- `ValueError` – This exception is raised if the buffer length does not match the expected key length.
- `TypeError` – This exception is raised if `key_material` or `buffer` is not `bytes`.
- `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This generates a new key from the supplied key material and writes it directly into the provided buffer. This is useful when you want to avoid allocating new memory for the derived key.

`verify(key_material, expected_key)` [\[source\]](#)

**Parameters:**

- `key_material` (*bytes*) – The input key material. This is the same as `key_material` in `derive()`.

- `expected_key` (*bytes*) – The expected result of deriving a new key, this is the same as the return value of `derive()`.

**Raises:**

- `cryptography.exceptions.InvalidKey` – This is raised when the derived key does not match the expected key.

- `cryptography.exceptions.AlreadyFinalized` – This is raised when `derive()`, `derive_into()`, or `verify()` is called more than once.

This checks whether deriving a new key from the supplied `key_material` generates the same key as the `expected_key`, and raises an exception if they do not match. This can be used for something like checking whether a user's password attempt matches the stored derived key.

- [1] See [NIST SP 800-132](#).