IOTA TUTORIAL 9.1

Key, Digests & Address

Key = Private Key

Address = Public Key

INTRO

- In IOTA tutorial 9, I quickly explained how an IOTA address is calculated.
- In this video I will explain in detail how the key, digests and address are calculated.

CALCULATE KEY

- The generation of a key, digests and address, all starts with a seed.

- How the key is calculated, see:
  https://github.com/iotaledger/iota.crypto.js/blob/v0.4.2/lib/crypto/signing/signing.js

```
var key = function(seed, index, length)
```

- Every index number refers to an address, also known as public key, and each address has a corresponding unique key, also known as the private key.

- seed = the randomly generated 81 trytes (A-Z9), converted to trits (81x3=243 trits)
  index = an integer (0,1,2 …. 9007199254740991) and every address has a corresponding index number.
  length = the security level (1,2 or 3)

CALCULATE KEY

```
var key = function(seed, index, length) {

    while ((seed.length % 243) !== 0) {
      seed.push(0);
    }

    var indexTrits = Converter.fromValue( index );
    var subseed = add( seed.slice( ), indexTrits );

    var kerl = new Kerl( );

    kerl.initialize( );
    kerl.absorb(subseed, 0, subseed.length);
    kerl.squeeze(subseed, 0, subseed.length);

    kerl.reset( );
    kerl.absorb(subseed, 0, subseed.length);

    var key = [], offset = 0, buffer = [];

    while (length-- > 0) {

        for (var i = 0; i < 27; i++) {

            kerl.squeeze(buffer, 0, subseed.length);
            for (var j = 0; j < 243; j++) {

                key[offset++] = buffer[j];
            }
        }
    }
    return key;
}
```

A

B

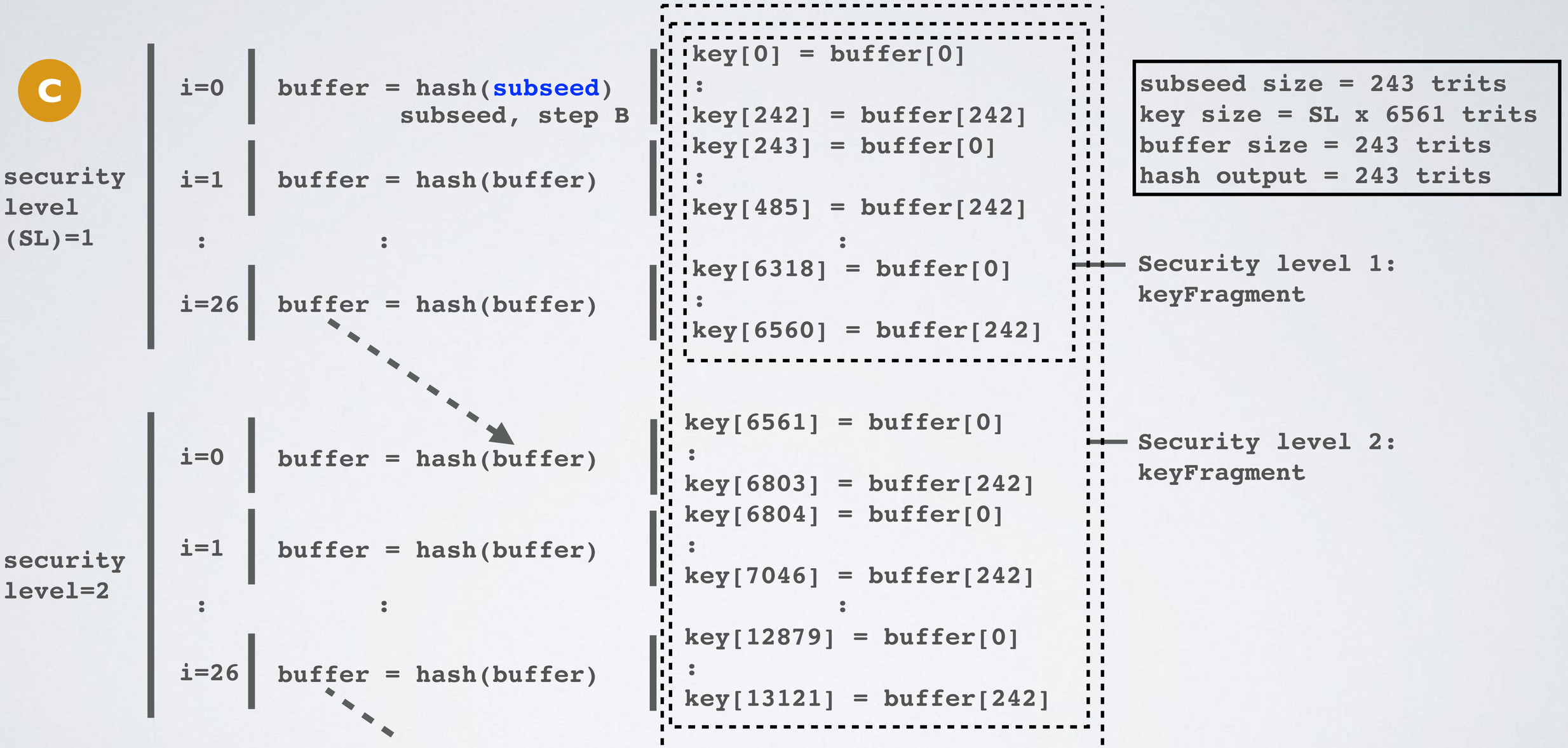
C

iota.crypto.js (v0.4.2)

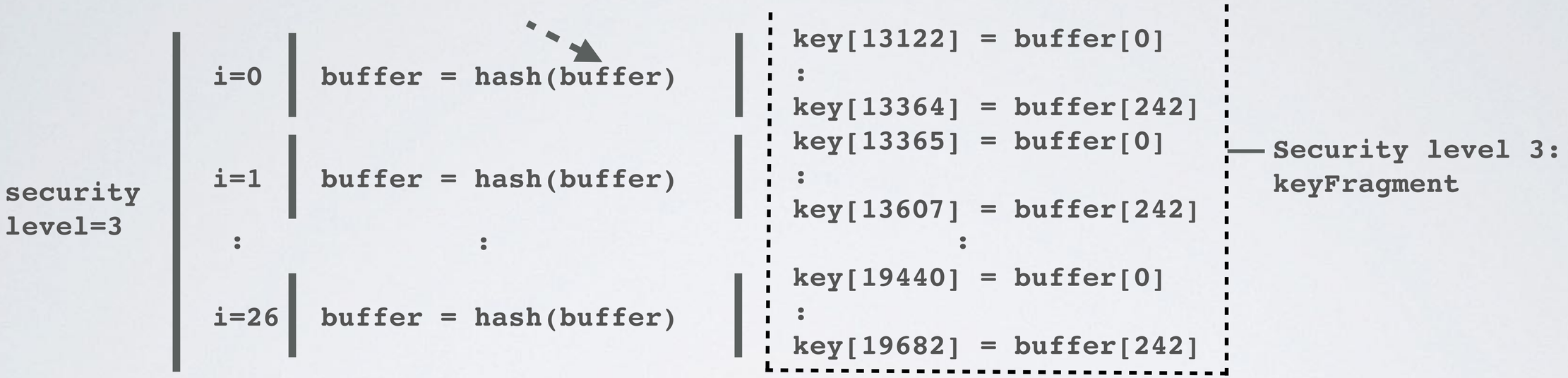
- Step A: subseed = seed + index
  subseed size = 243 trits
- Step B: subseed = hash(subseed)
  subseed size = 243 trits
- Step C: Explained in the next 2 slides

Note:
Kerl uses the keccak-384 hash algorithm

CALCULATE KEY



CALCULATE KEY

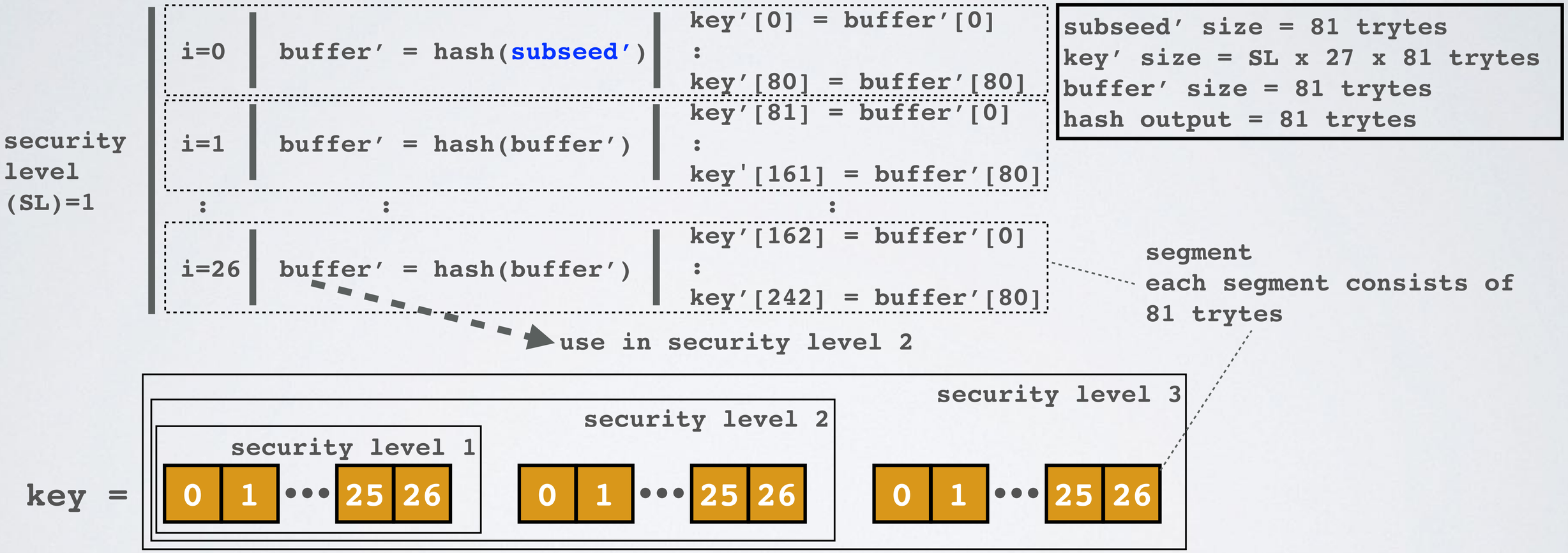



```
Security level 1, key size = 1x27x243 = 6561 trits
Security level 2, key size = 2x27x243 = 13122 trits
Security level 3, key size = 3x27x243 = 19683 trits

Security level 1, key size = 6561/3 = 2187 trytes
Security level 2, key size = 13122/3 = 4374 trytes
Security level 3, key size = 19683/3 = 6561 trytes
```

CALCULATE KEY

- Visual representation how the key is calculated when using trytes instead of trits.



CALCULATE KEY

- Security level 1:
  key has 1x27 = 27 segments, each segment consists of 81 trytes.
  key has in total 27 x 81 = 2187 trytes
  Security level 2:
  key has 2x27 = 54 segments, each segment consists of 81 trytes.
  key has in total 2 x 27 x 81 = 2 x 2187 = 4374 trytes
  Security level 3:
  key has 3x27 = 81 segments, each segment consists of 81 trytes.
  key has in total 3 x 27 x 81 = 3 x 2187 = 6561 trytes

- Security level 1, key size = (2187 x 3 x ln(3) / ln(2)) / 8 = ~1300 bytes
  Security level 2, key size = (4374 x 3 x ln(3) / ln(2)) / 8 = ~2600 bytes
  Security level 3, key size = (6561 x 3 x ln(3) / ln(2)) / 8 = ~3900 bytes

CALCULATE DIGESTS

- How the digests is calculated, see:
  https://github.com/iotaledger/iota.crypto.js/blob/v0.4.2/lib/crypto/signing/signing.js

```
var digests = function(key)
```

CALCULATE DIGESTS

```
var digests = function(key) {
    var digests = [], buffer = [];
    for (var i = 0; i < Math.floor(key.length / 6561); i++) {
        var keyFragment = key.slice(i * 6561, (i + 1) * 6561);
        for (var j = 0; j < 27; j++) {
            buffer = keyFragment.slice(j * 243, (j + 1) * 243);

            for (var k = 0; k < 26; k++) {
                var kKerl = new Kerl();
                kKerl.initialize();
                kKerl.absorb(buffer, 0, buffer.length);
                kKerl.squeeze(buffer, 0, Curl.HASH_LENGTH);
            }
            for (var k = 0; k < 243; k++) {

                keyFragment[j * 243 + k] = buffer[k];
            }
        }

        var kerl = new Kerl()

        kerl.initialize();
        kerl.absorb(keyFragment, 0, keyFragment.length);
        kerl.squeeze(buffer, 0, Curl.HASH_LENGTH);

        for (var j = 0; j < 243; j++) {

            digests[i * 243 + j] = buffer[j];
        }
    }
    return digests;
}
```

26x

D

E

F

G

H

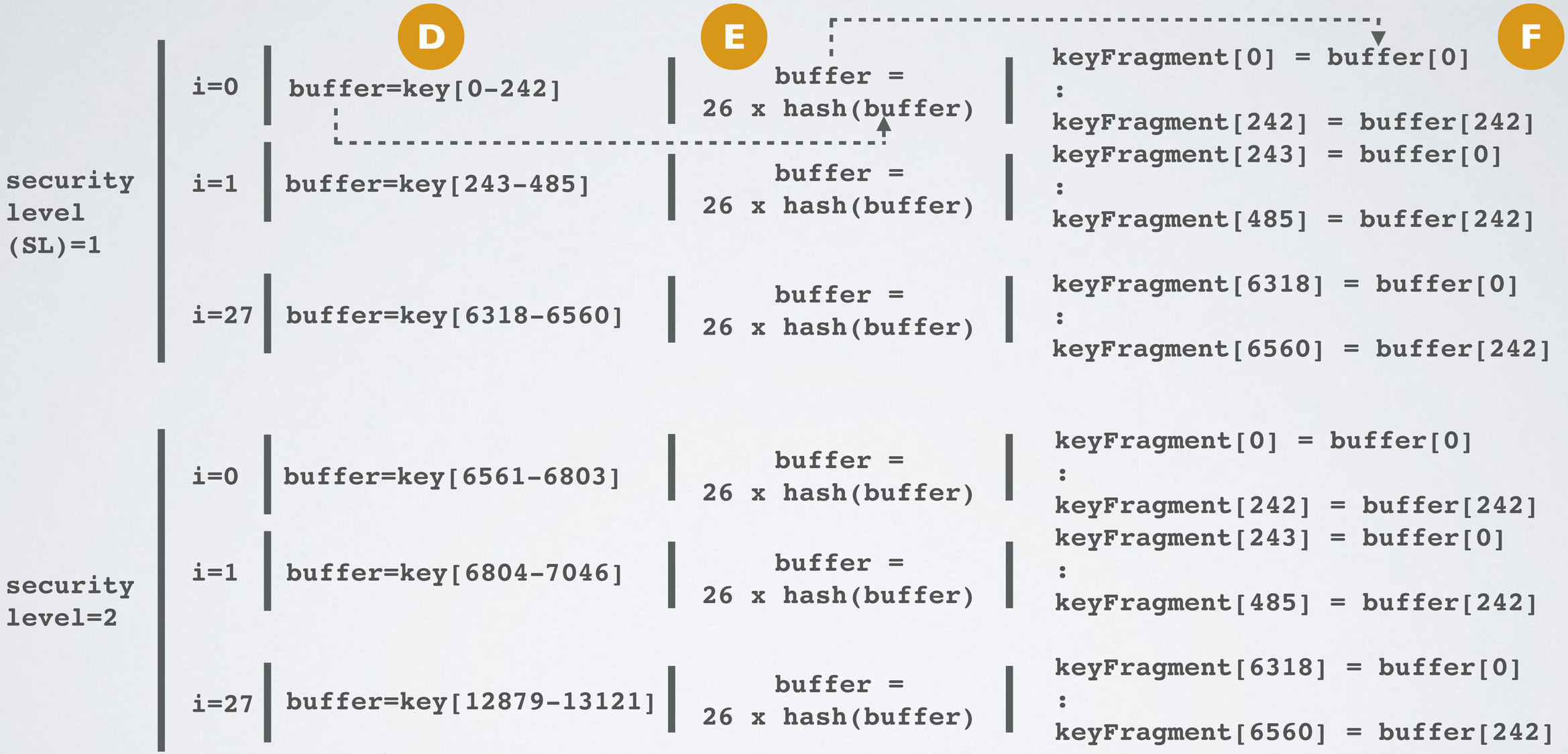
iota.crypto.js (v0.4.2)

Note:
Curl.HASH_LENGTH = 243

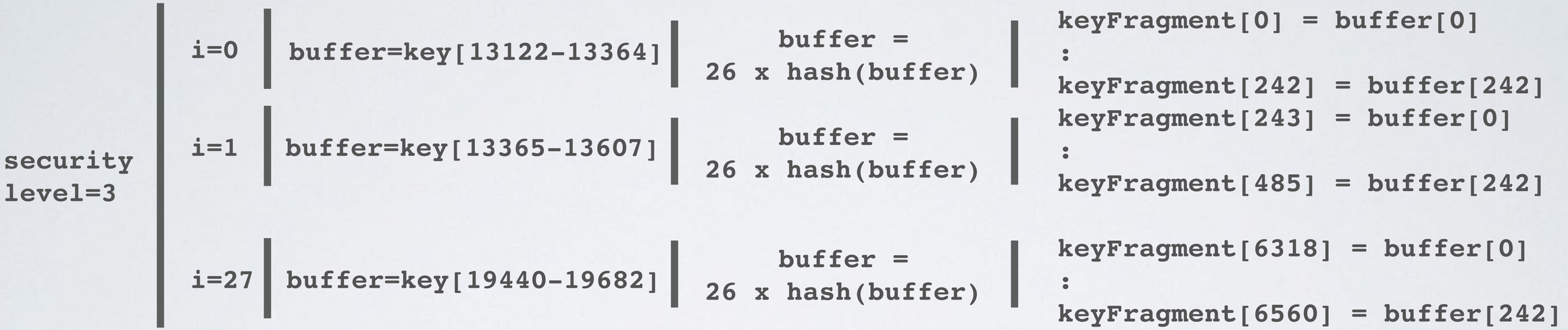
The buffer is hashed 26x (see E). This is NOT an arbitrary chosen number.

This number plays a key role in creating a signature and validating a signature.

CALCULATE DIGESTS

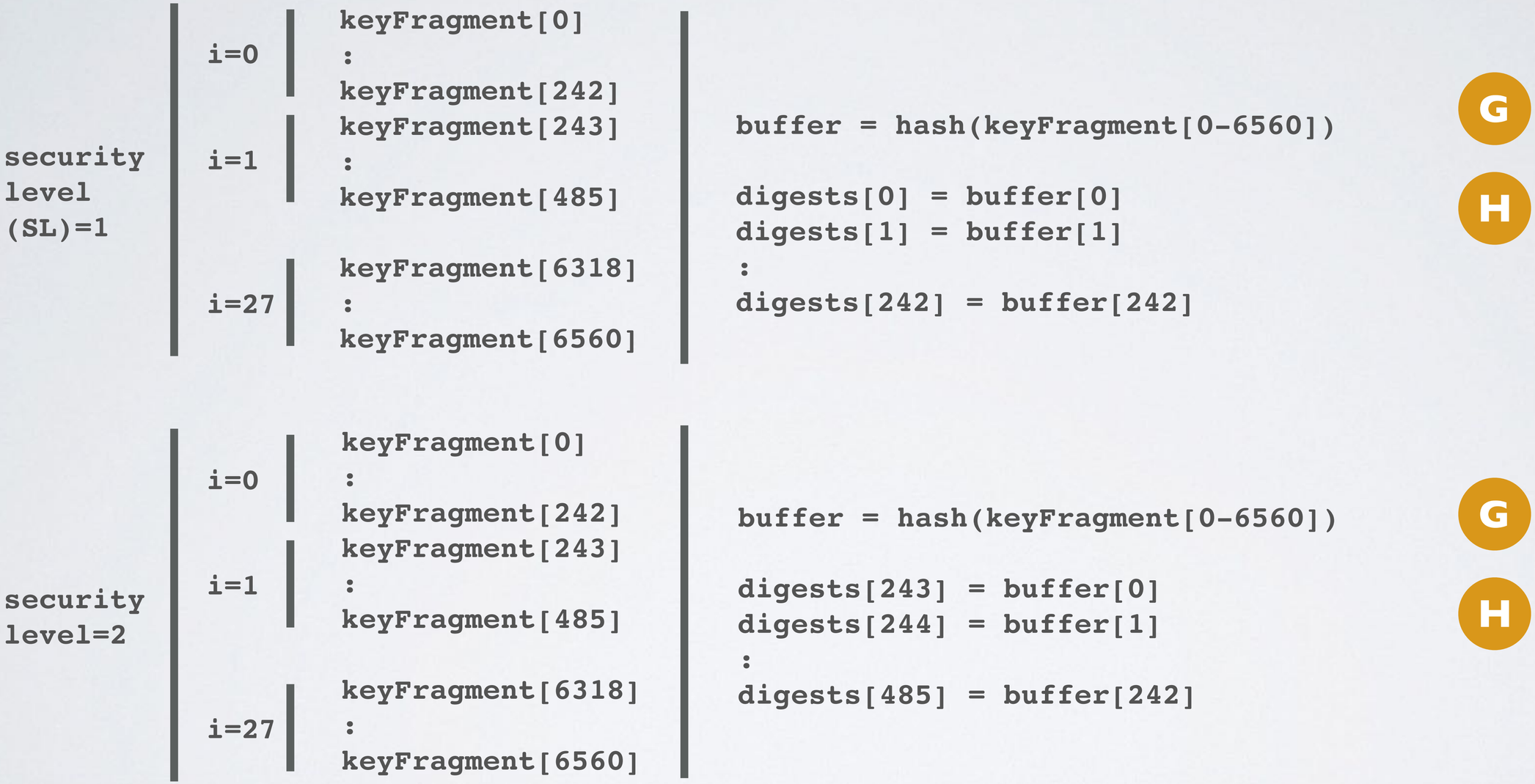


CALCULATE DIGESTS

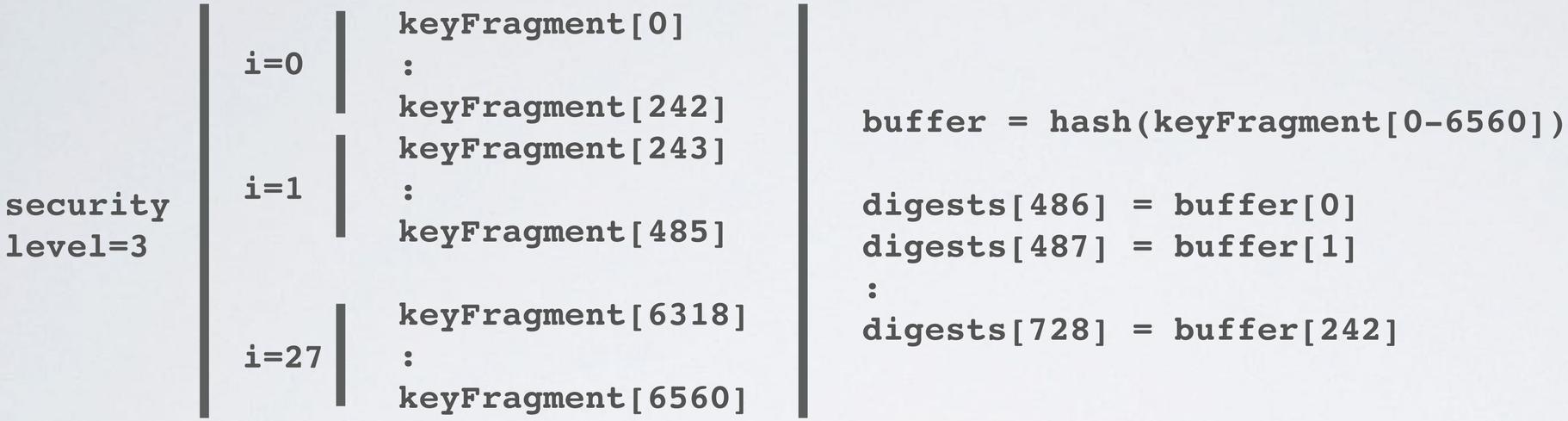



```
key size = SL x 6561 trits
buffer size = 243 trits
keyFragment size = 6561 trits
hash output = 243 trits
```

CALCULATE DIGESTS



CALCULATE DIGESTS

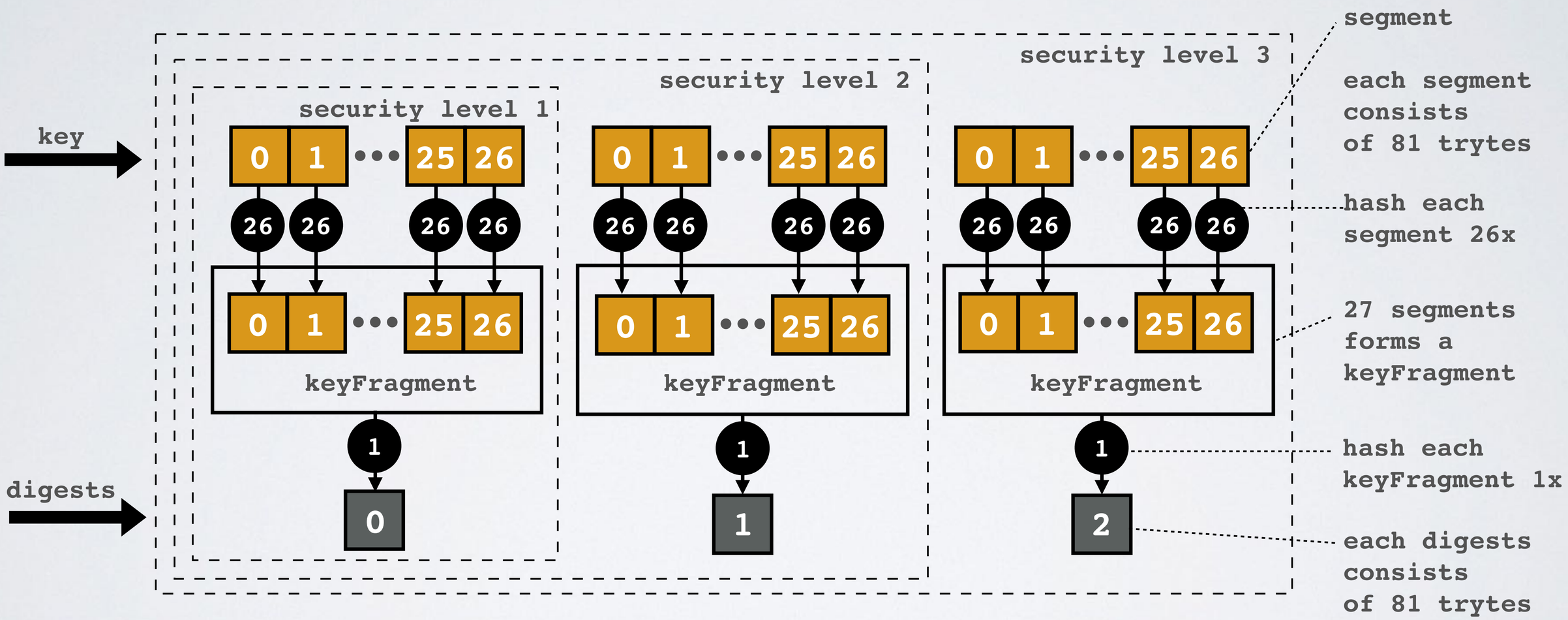



```
digests size = SL x 243 trits
buffer size = 243 trits
keyFragment size = 6561 trits
hash output = 243 trits
```

```
Security level 1, digests size = 1x243 = 243 trits = 243 / 3 = 1 x 81 trytes
Security level 2, digests size = 2x243 = 486 trits = 486 / 3 = 2 x 81 trytes
Security level 3, digests size = 3x243 = 729 trits = 729 / 3 = 3 x 81 trytes
```

CALCULATE DIGESTS

- Visual representation how the digests is calculated when using trytes instead of trits.



CALCULATE ADDRESS

- How the address is calculated, see:
  https://github.com/iotaledger/iota.crypto.js/blob/v0.4.2/lib/crypto/signing/signing.js

```
var address = function(digests)
```

CALCULATE ADDRESS

```
var address = function(digests) {

    var addressTrits = [];

    var kerl = new Kerl();

    kerl.initialize();
    kerl.absorb(digests, 0, digests.length);
    kerl.squeeze(addressTrits, 0, Curl.HASH_LENGTH);

    return addressTrits;
}
```

iota.crypto.js (v0.4.2)

```
digests size = SL x 243 trits
address size = 243 trits
hash output = 243 trits
```

- Security level 1, address = hash(digests[0-242])
  Security level 2, address = hash(digests[0-485])
  Security level 3, address = hash(digests[0-728])

- Address size is always 243 trits = 243 / 3 = 81 trytes

CALCULATE DIGESTS