

IOTA TUTORIAL 9.I

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 ($81 \times 3 = 243$ 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

C

security
level
(SL)=1

```
i=0 | buffer = hash(subseed)
      | subseed, step B
i=1 | buffer = hash(buffer)
:
i=26 | buffer = hash(buffer)
```

security
level=2

```
i=0 | buffer = hash(buffer)
i=1 | buffer = hash(buffer)
:
i=26 | buffer = hash(buffer)
```

```
key[0] = buffer[0]
:
key[242] = buffer[242]
key[243] = buffer[0]
:
key[485] = buffer[242]
:
key[6318] = buffer[0]
:
key[6560] = buffer[242]
```



```
key[6561] = buffer[0]
:
key[6803] = buffer[242]
key[6804] = buffer[0]
:
key[7046] = buffer[242]
:
key[12879] = buffer[0]
:
key[13121] = buffer[242]
```

subseed size = 243 trits
key size = SL x 6561 trits
buffer size = 243 trits
hash output = 243 trits

Security level 1:
keyFragment

Security level 2:
keyFragment

CALCULATE KEY

security
level=3

```
i=0 | buffer = hash(buffer)
    |
i=1 | buffer = hash(buffer)
    :
    :
i=26 | buffer = hash(buffer)
```

```
key[13122] = buffer[0]
:
key[13364] = buffer[242]
key[13365] = buffer[0]
:
key[13607] = buffer[242]
    :
key[19440] = buffer[0]
:
key[19682] = buffer[242]
```

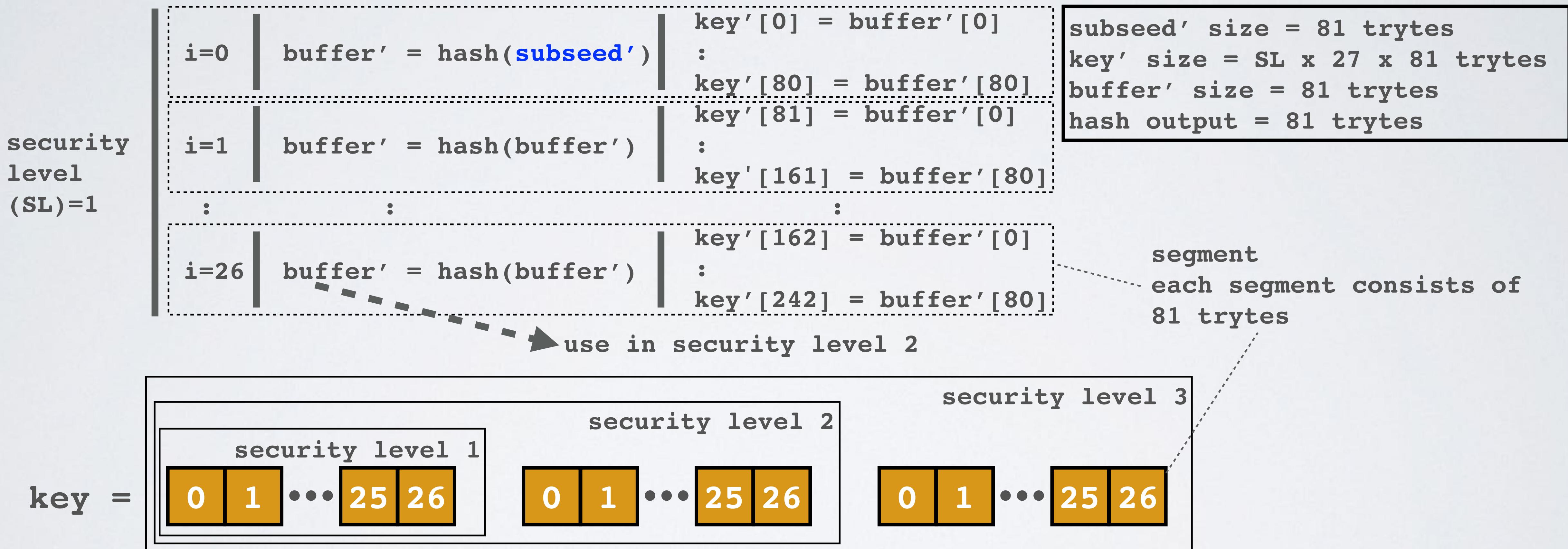
Security level 3:
keyFragment

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 $1 \times 27 = 27$ segments, each segment consists of 81 trytes.

key has in total $27 \times 81 = 2187$ trytes

Security level 2:

key has $2 \times 27 = 54$ segments, each segment consists of 81 trytes.

key has in total $2 \times 27 \times 81 = 2 \times 2187 = 4374$ trytes

Security level 3:

key has $3 \times 27 = 81$ segments, each segment consists of 81 trytes.

key has in total $3 \times 27 \times 81 = 3 \times 2187 = 6561$ trytes

- Security level 1, key size = $(2187 \times 3 \times \ln(3) / \ln(2)) / 8 = \sim 1300$ bytes

Security level 2, key size = $(4374 \times 3 \times \ln(3) / \ln(2)) / 8 = \sim 2600$ bytes

Security level 3, key size = $(6561 \times 3 \times \ln(3) / \ln(2)) / 8 = \sim 3900$ 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);

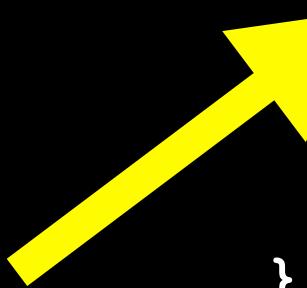
            for (var k = 0; k < 26; k++) {
                var kerl = new Kerl();
                kerl.initialize();
                kerl.absorb(buffer, 0, buffer.length);
                kerl.squeeze(buffer, 0, Curl.HASH_LENGTH);
            }
            for (var k = 0; k < 243; k++) {
                keyFragment[j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

security
level
(SL)=1

i=0 | buffer=key[0-242]

:

i=1 | buffer=key[243-485]

i=27 | buffer=key[6318-6560]

D

E

F

buffer =
26 x hash(buffer)

buffer =
26 x hash(buffer)

buffer =
26 x hash(buffer)

keyFragment[0] = buffer[0]
:
keyFragment[242] = buffer[242]
keyFragment[243] = buffer[0]
:
keyFragment[485] = buffer[242]

keyFragment[6318] = buffer[0]
:
keyFragment[6560] = buffer[242]

security
level=2

i=0 | buffer=key[6561-6803]

i=1 | buffer=key[6804-7046]

i=27 | buffer=key[12879-13121]

buffer =
26 x hash(buffer)

buffer =
26 x hash(buffer)

buffer =
26 x hash(buffer)

keyFragment[0] = buffer[0]
:
keyFragment[242] = buffer[242]
keyFragment[243] = buffer[0]
:
keyFragment[485] = buffer[242]

keyFragment[6318] = buffer[0]
:
keyFragment[6560] = buffer[242]

CALCULATE DIGESTS

security
level=3

i=0	buffer=key[13122-13364]	buffer = 26 x hash(buffer)	keyFragment[0] = buffer[0] : keyFragment[242] = buffer[242]
i=1	buffer=key[13365-13607]	buffer = 26 x hash(buffer)	keyFragment[243] = buffer[0] : keyFragment[485] = buffer[242]
i=27	buffer=key[19440-19682]	buffer = 26 x hash(buffer)	keyFragment[6318] = buffer[0] : keyFragment[6560] = buffer[242]

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

CALCULATE DIGESTS

**security level
(SL)=1**

i=0	keyFragment[0] : keyFragment[242]
i=1	keyFragment[243] : keyFragment[485]
i=27	keyFragment[6318] : keyFragment[6560]

```
buffer = hash(keyFragment[0-6560])
digests[0] = buffer[0]
digests[1] = buffer[1]
:
digests[242] = buffer[242]
```



security level=2

i=0	keyFragment[0] : keyFragment[242]
i=1	keyFragment[243] : keyFragment[485]
i=27	keyFragment[6318] : keyFragment[6560]

```
buffer = hash(keyFragment[0-6560])
digests[243] = buffer[0]
digests[244] = buffer[1]
:
digests[485] = buffer[242]
```



CALCULATE DIGESTS

security
level=3

i=0	keyFragment[0] : keyFragment[242]
i=1	keyFragment[243] : keyFragment[485]
i=27	keyFragment[6318] : keyFragment[6560]

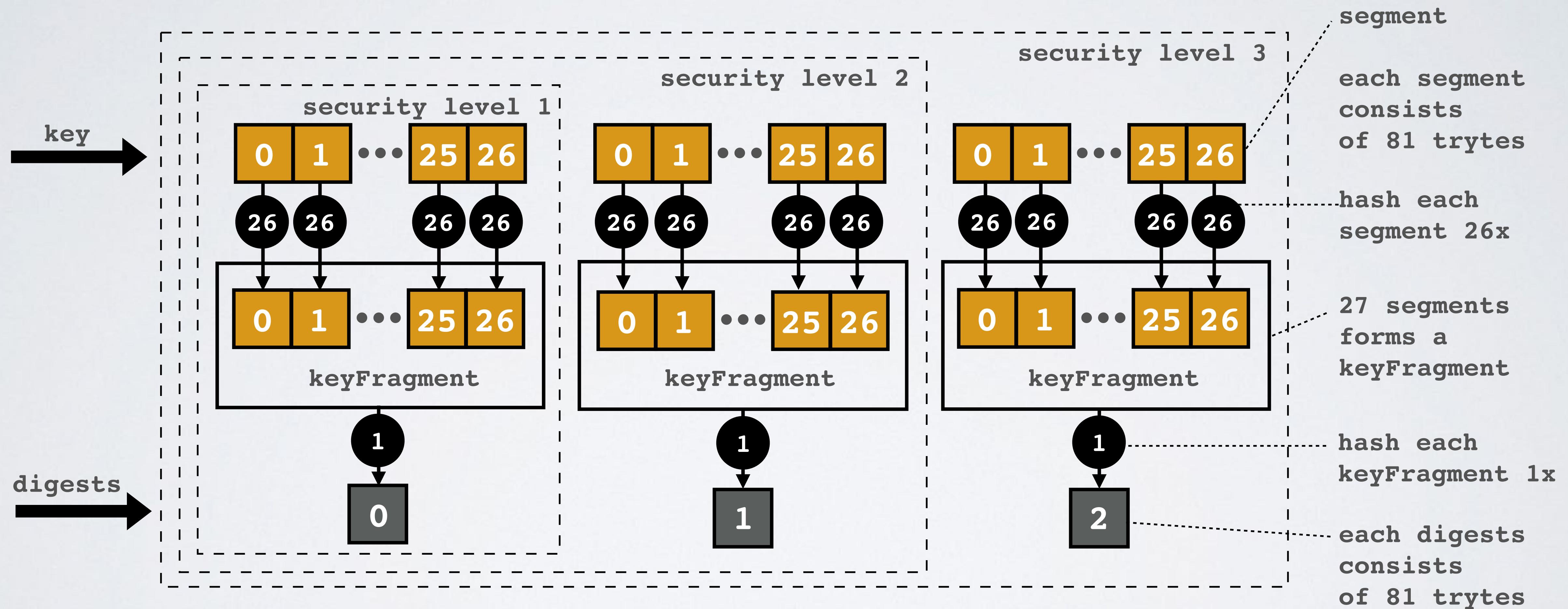
```
buffer = hash(keyFragment[0-6560])  
  
digests[486] = buffer[0]  
digests[487] = buffer[1]  
:  
digests[728] = buffer[242]
```

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

