A Terse Description of Two New Hash-Based DRBGs
John Kelsey, NIST, January 2004

1 Notation

Both DRBGs call the hash function as hash(inputString).

N is the number of bytes of output of the hash compression function output. (For SHA1, N=20) N = hash_outlen

M is the number of bytes of message block input in the compression function. (For SHA1, M=64) M = hash_inlen

The claimed security level of the DRBG is the number of bits in the hash function output.

X||Y is concatenation

Integers are assumed to be encoded in network byte order when they're hashed.

X[a:b] is bytes a..b-1 of byte string X

X[a:] is all of X from byte a forward.

X[:a] is the leftmost a bytes of X

2 HMAC_DRBG

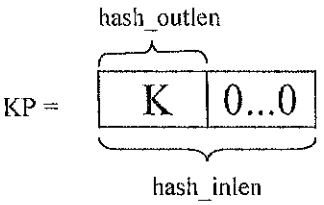
HMAC_DRBG has the following working state:

X (N bytes)	X consists of hash_outlen bytes
K (N bytes)	Y consists of hash_outlen bytes

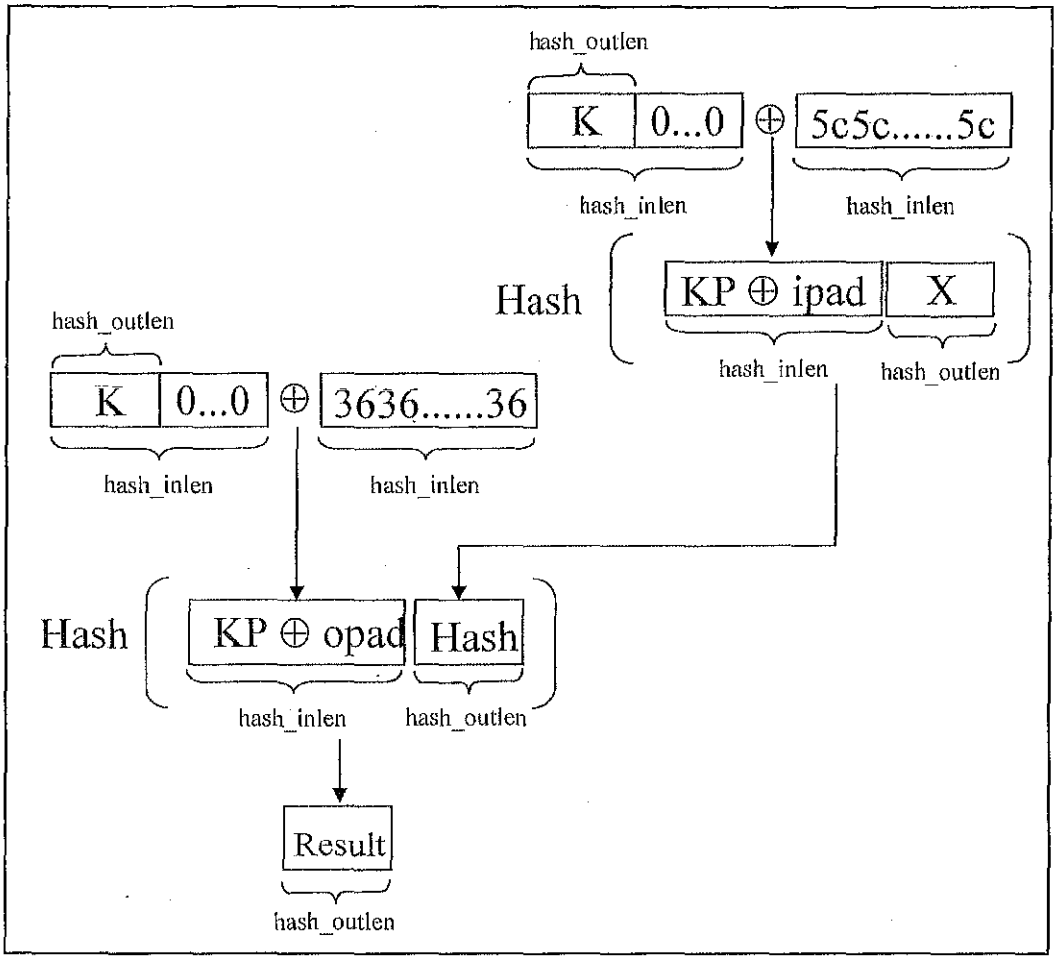
It uses one external function besides the hash function:

HMAC(K,X):

PAD = 0x00 0x00 ... 0x00 (M-N bytes)
opad = 0x36 0x36 ... 0x36 (M bytes)
ipad = 0x5c 0x5c ... 0x5c (M bytes)
KP = K || PAD



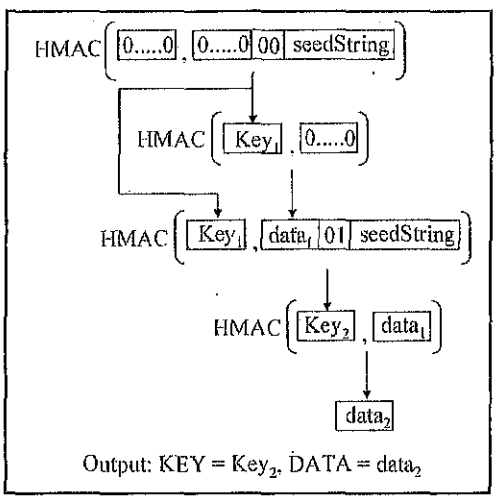
return hash(KP xor opad || hash(KP xor ipad || X))



It supports three public functions:

Initialize(seedString): Note: seedString is a function of the entropy bits and the personalization string

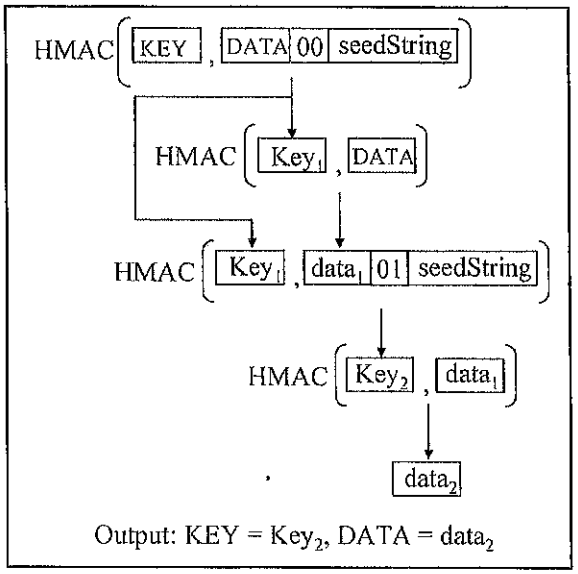
data = 0x00 0x00 ... 0x00 (hash_outlen bytes)
Key = 0x00 0x00 ... 0x00 (hash_outlen bytes)
Key_1 = HMAC(Key, data || 0x00 || seedString)
$data_1$ = HMAC(Key_1, data)
key_2 = HMAC(Key_1, $data_1$ || 0x01 || seedString)
$data_2$ = HMAC(Key ,data)



Reseed(seedString) Note: seedString is a function of the entropy bits and the personalization string

Key = HMAC(Key, data || 0x00 || seedString)
data = HMAC(Key, data)

Key = HMAC(Key, data || 0x01 || seedString)
data = HMAC(Key, data)



Generate(bytes,optionalString): Note: optionalString is the optional additional data?
if bytes>2^{32}: raise error condition

if optionalString exists:
Key = HMAC(Key, data || 0x00 || optionalString)
data = HMAC(Key, data)
Key = HMAC(Key, data || 0x01 || optionalString)
data = HMAC(Key, data)

} Reseed (optionalString)

tmp = ""
while len (tmp) <bytes:
DATA = HMAC(KEY, DATA)
tmp = tmp || DATA

if optionalString exists: Note: this step provides backtracking resistance
Key_1 = HMAC(KEY, DATA || 0x00 || optionalString)
$data_1$ = HMAC(Key_1, DATA)

KEY = HMAC(Key_1, $data_1$ || 0x01 || optionalString)
DATA = HMAC(KEY, $data_1$)
else:
KEY = HMAC(KEY, DATA || 0x00)
DATA = HMAC(KEY, DATA)

return tmp[:bytes] Note: return the leftmost bytes

3 KHF_DRBG

KHF_DRBG has the following working state:

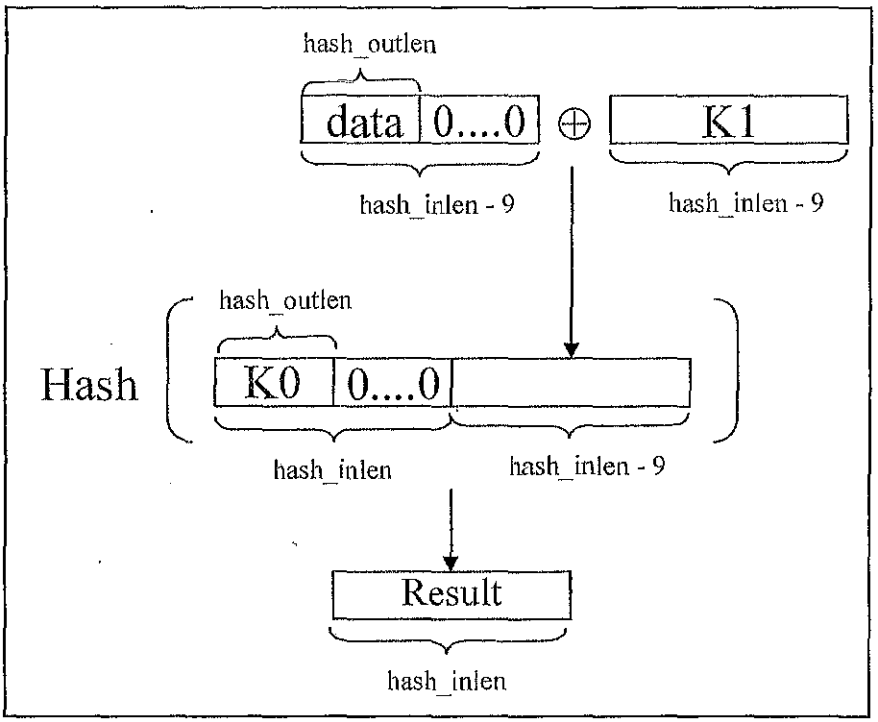
K0 (N bytes) K0 is hash_outlen bytes long
K1 (M-9 bytes) K1 is hash_inlen - 9 bytes long
data (N bytes) data is hash_outlen bytes long

KHF_DRBG uses two external functions besides the hash function:

hash_df(seed,bytes):

```
        tmp = ""
        i = 0       Is this a string?
        while len(tmp)<bytes:
            tmp = tmp || hash( bytes || i || seed )
    return tmp[:bytes]

KHF(K0,K1,X):
        PAD_0 = 0x00 0x00 ... 0x00 (M-N bytes)
        PAD_1 = 0x00 0x00 ... 0x00 (M-N-9 bytes)
        return hash(K0 || PAD_0 || (X || PAD_1) xor K1 )
```



KHF_DRBG supports three public functions:

Initialize(seedString): Note: seedString is a function of the entropy bits and the personalization string

```
        K0 = 0x00 0x00 ... 0x00 (hash_outlen bytes)
        K1 = 0x01 0x01 ... 0x01 (hash_inlen - 9 bytes)
        data = 0x02 0x02 ... 0x02 (hash_outlen bytes)
        T = ""
        while len(tmp)<hash_outlen + hash_inlen - 9:
            data = KHF(K0,K1,data)
            T = T || data
        T = T[:N+M-9] ⊕ hash_df(hash_inlen + hash_outlen - 9,seedString)
        K0 = T[:N]
        K1 = T[N:]
        DATA = KHF(K0,K1,data)
```

Reseed(seedString): Note: seedString is a function of the entropy bits and the personalization string

```
        T = ""
        while len(tmp)<N+M-9:
            DATA = KHF(K0,K1,DATA)
            T = T || DATA
        T = T[:N+M-9] ⊕ hash_df(hash_inlen + hash_outlen - 9,seedString)
        K0 = T[:N]
```

```
        K1 = T[N:]
        DATA = KHF(K0,K1,DATA)

Generate(bytes,optionalString):
        if bytes>2^{32}: raise error condition

        if optionalString exists:     Note: This is a reseed using optionalString
            T = ""
                while len(tmp)<N+M-9:
                    DATA = KHF(K0,K1,DATA)
                        T = T || DATA
            T = T[:N+M-9] ⊕ hash_df(hash_outpen + hash_inlen - 9,optionalString)
                K0 = T[:N]
                K1 = T[N:]
                DATA = KHF(K0,K1,DATA)

    tmp = ""
        while len(tmp)<bytes:
            DATA = KHF(K0,K1,DATA)
                tmp = tmp || DATA

    if optionalString exists
            T = ""
                while len(tmp)<N+M-9:
                    DATA = KHF(K0,K1,DATA)
                        T = T || DATA
            T = T[:N+M-9] ⊕ hash_df(hash_outpen + hash_inlen - 9,optionalString)

                K0 = T[:N]
                K1 = T[N:]
                DATA = KHF(K0,K1,DATA)
        else:
            T = ""
                while len(tmp)<hash_outlen + hash_inlen - 9:
                    DATA = KHF(K0,K1,DATA)
                        T = T || DATA
            T = T[:N+M-9] ⊕ hash_df(hash_outpen + hash_inlen - 9,,"")
                K0 = T[:N]
                K1 = T[N:]
                DATA = KHF(K0,K1,DATA)

        return tmp[:bytes]
```