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)

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

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)
K (N 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):
        X = 0x00 0x00 ... 0x00 (N bytes)
        K = 0x00 0x00 ... 0x00 (N bytes)
        K = HMAC(K,X || 0x00 || seedString)
        X = HMAC(K,X)
        K = HMAC(K,X || 0x01 || seedString)
        X = HMAC(K,X)

Reseed(seedString)
        K = HMAC(K,X || 0x00 || seedString)
        X = HMAC(K,X)
        K = HMAC(K,X || 0x01 || seedString)
        X = HMAC(K,X)

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

        if optionalString exists:
                K = HMAC(K,X || 0x00 || optionalString)
                X = HMAC(K,X)
                K = HMAC(K,X || 0x01 || optionalString)
                X = HMAC(K,X)

        tmp = ""
        while len(tmp)<bytes:
                X = HMAC(K,X)
                tmp = tmp || X

        if optionalString exists:
                K = HMAC(K,X || 0x00 || optionalString)
                X = HMAC(K,X)

                K = HMAC(K,X || 0x01 || optionalString)
                X = HMAC(K,X)
        else:
                K = HMAC(K,X || 0x00)
                X = HMAC(K,X)

        return tmp[:bytes]
```

3 KHF_DRBG

KHF_DRBG has the following working state:

```
K0 (N bytes)
K1 (M-9 bytes)
X  (N bytes)
```

KHF_DRBG uses two external functions besides the hash function:

```
hash_df(seed,bytes):
        tmp = ""
        i = 0
        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):
        K0 = 0x00 0x00 ... 0x00 (N bytes)
        K1 = 0x01 0x01 ... 0x01 (M-9 bytes)
        X = 0x02 0x02 ... 0x02 (N bytes)
        T = ""
        while len(tmp)<N+M-9:
              X = KHF(K0,K1,X)
              T = T || X
        T = T[:N+M-9] xor hash_df(M+N-9,seedString)
        K0 = T[:N]
        K1 = T[N:]
        X = KHF(K0,K1,X)

Reseed(seedString):
        T = ""
        while len(tmp)<N+M-9:
              X = KHF(K0,K1,X)
```

```
                T = T || X
        T = T[:N+M-9] xor hash_df(M+N-9,seedString)
        K0 = T[:N]
        K1 = T[N:]
        X = KHF(K0,K1,X)

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

        if seedString exists:
                T = ""
                while len(tmp)<N+M-9:
                        X = KHF(K0,K1,X)
                        T = T || X
                T = T[:N+M-9] xor hash_df(N+M-9,seedString)
                K0 = T[:N]
                K1 = T[N:]
                X = KHF(K0,K1,X)

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

         if seedString exists
                T = ""
                while len(tmp)<N+M-9:
                        X = KHF(K0,K1,X)
                        T = T || X
                T = T[:N+M-9] xor hash_df(N_M-9,seedString)
                K0 = T[:N]
                K1 = T[N:]
                X = KHF(K0,K1,X)
        else:
                T = ""
                while len(tmp)<N+M-9:
                        X = KHF(K0,K1,X)
                        T = T || X
                T = T[:N+M-9] xor hash_df(N+M-9,"")
                K0 = T[:N]
                K1 = T[N:]
                X = KHF(K0,K1,X)

        return tmp[:bytes]
```