9.1.2 Keyless Hash Function DRBG[1] using an Approved Hash Function (keylessHashDRBG)

[Comment: This is the new RBG that may be used with any of the hash functions.]

9.1.2.1 Properties

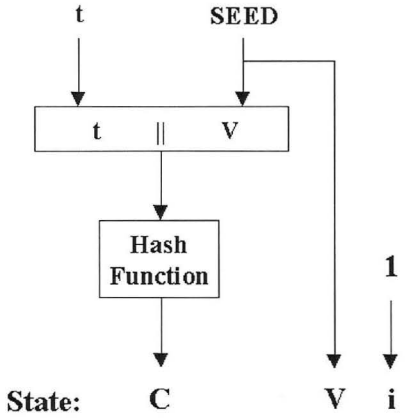

Figure 4: Keyless Hash DRBG Initialization

Figures 4 and 5 present a keyless deterministic RBG that uses any Approved hash function.

keylessHashDRBG employs an Approved hash function that produces pseudo random bits using a seed (*SEED*) and an application specific constant (*t*). Optional user input (*UserInput*) may be provided during each access of **keylessHashDRBG**.

keylessHashDRBG has been designed to meet different security levels (see Annex D). A number *N* is assigned to each hash function, where *N* = the output block size of the hash function.

The **keylessHashDRBG** generator requires the use of a hash function at three points in the process (which includes the initialization process); see Figures 4 and 5. The same hash function shall be used at all three points.

The length of the seed (*seedlen*) shall be at least equal to *N*, and the seed shall have at least *N* bits of entropy (i.e., at least *N* bits shall be selected randomly). Further requirements for the seed are provided in Section 7.2.1. [Do we want to require that the seed is this large?]

The application specific constant (*t*) shall be *N* bits in length.

When the optional user input (*UserInput*) is used, *UserInput* shall be at least *N* bits in length and have at least *N* bits of entropy (i.e, at least *N* bits of the *UserInput* shall be selected randomly).

Figures 6 and 7 depict the insertion of test input for the seed (*SEED*), the application specific constant (*t*) and the user input values (*UserInput*). The tests shall be run on the output of the generator. Validation and operational testing are discussed in Section 10. Detected errors shall result in a transition to the error state.

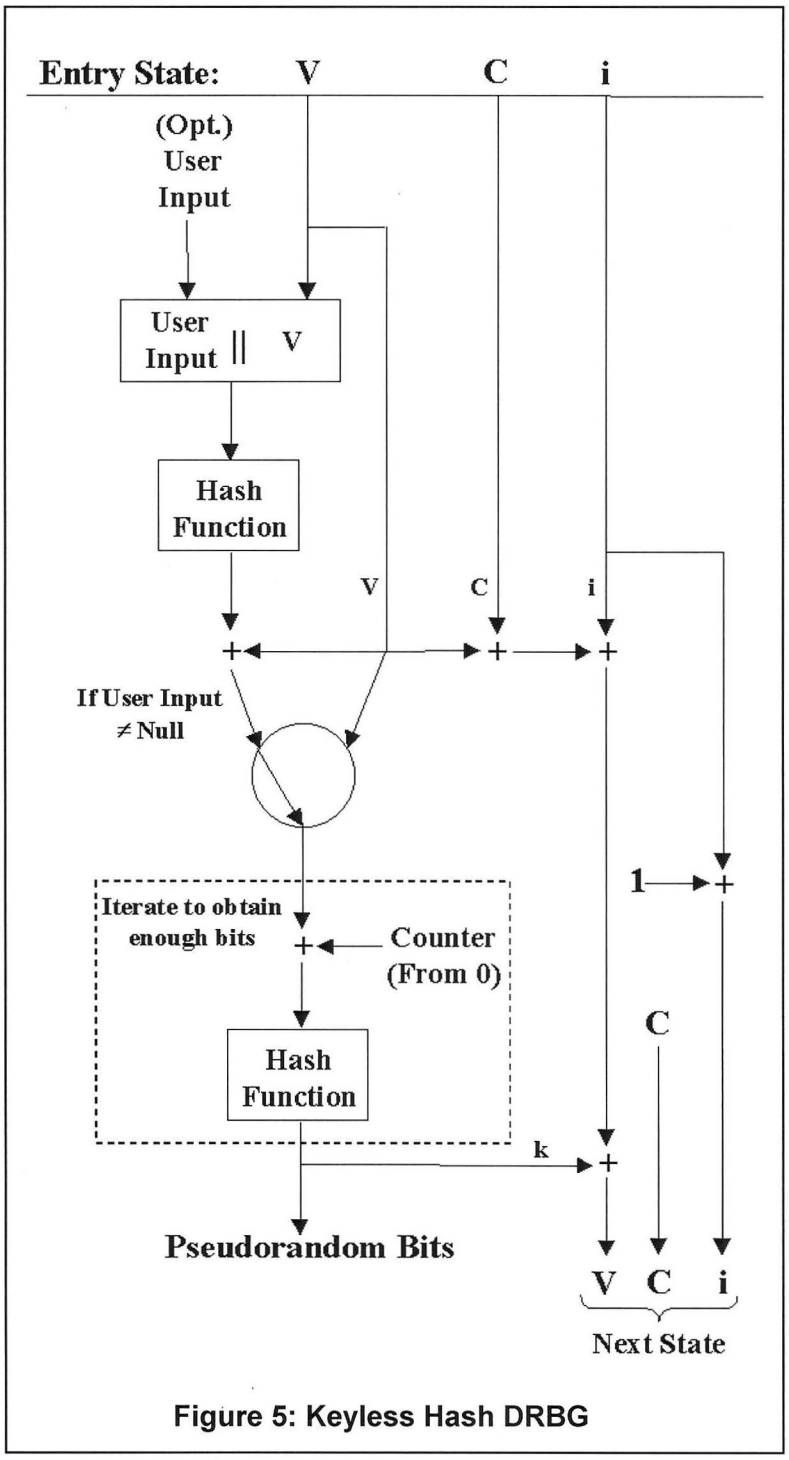

Figure 5: Keyless Hash DRBG

9.1.2.2 Description

The state of **keylessHashDRBG** consists of a counter (*i*) and two *N*-bit numbers (*V* and *C*). The state shall be protected in accordance with the sensitivity of the data protected by the consuming application.

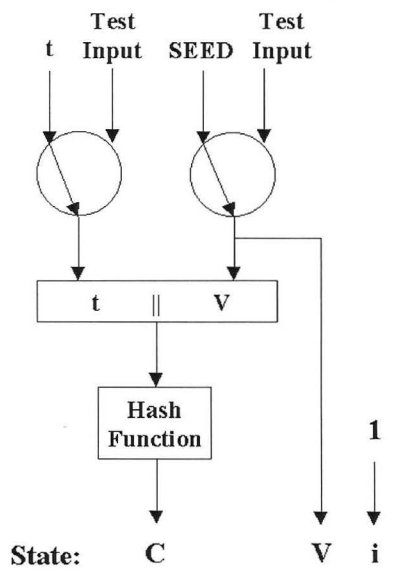

Figure 6: Keyless Hash DRBG Initialization (With Tests)

The variables used in the description of **keylessHashDRBG** are:

C An *N*-bit constant that is calculated during the initialization process.

i A counter that is used to update the state of **keylessHashDRBG** and may be used to determine the number of times that the generator has been accessed.

Hash(*a*) A hashing operation on data *a* using an appropriate Approved hash function.

m The number of iterations of the hash function needed to obtain the requested number of pseudorandom bits.

M The data to be hashed to obtain the pseudorandom bits

n The number of bits to be generated. *n* may be longer than the length of the hash function (*outlen*).

N A number that is determined by the desired security level. For this standard, *N* may be 160, 224, 256, 384 and 512 (see Annex D). *N* is used to determine the hash function to be used.

NRBG(*a*) A function that acquires an *a*-bit string from an Approved non-deterministic RBG or an Approved deterministic RBG (or chain of Approved deterministic RBGs) that is seeded by an Approved non-deterministic RBG.

outlen The output length of the hash function.

p The pseudorandom bits produced by the generator.

State The state of **keylessHashDRBG** that is carried between calls to the generator. The state consists of the current values of {*V*, *C*, *i*}.

t An application-specific *N*-bit constant.

UserInput Optional user input.

V A value that is initially set to the value of *SEED*, but assumes new values based on optional user input (*UserInput*), the pseudorandom bits produced by the generator (*p*), the constant (*C*) and the iteration count (*i*).

w, W Intermediate values.

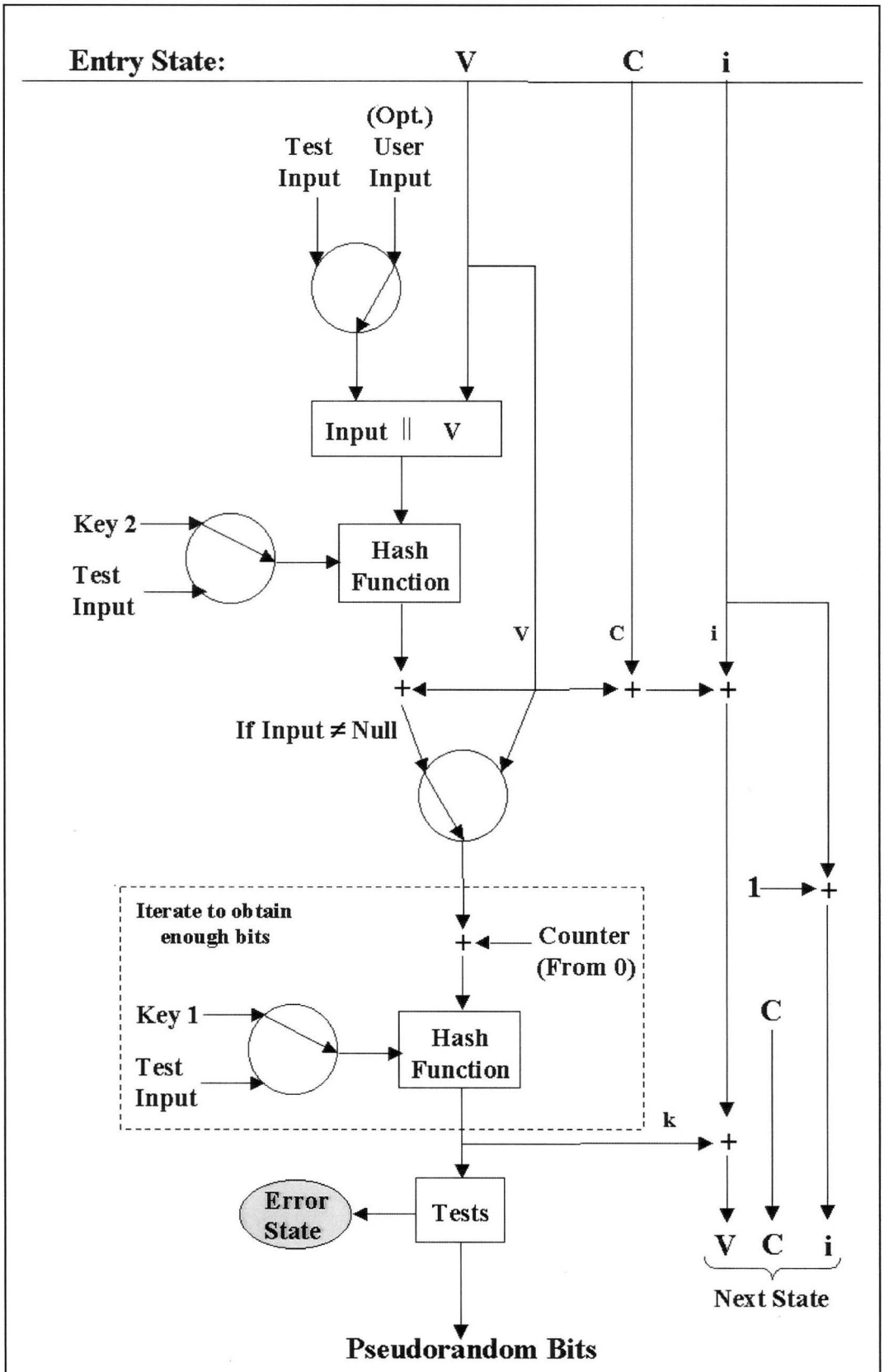

Figure 7: Keyless Hash DRBG (With Tests)

The algorithm shall proceed as follows:

Initialization:

1. t = the application specific constant
2. i = 1
3. $V = SEED =$ NRBG(N)
4. C = Hash(t || V) mod 2^N
5. $State = \{V, C, i\}$

keylessHashDRBG (n, $State$, [$UserInput$]):

1. If $UserInput \neq$ Null:
   a. w = Hash($UserInput$ || V)
   b. $V = (V + w) \bmod 2^N$
2. p = Hashgen(n, V)
3. $V = (V + p + C + i) \bmod 2^N$
4. $i = i + 1$
5. $State = \{V, C, i\}$
6. Return (p, $State$)

Hashgen(n, V)

1. $m = \lceil n/outlen \rceil$
2. $M = V$
3. W = the null string
4. For i = 0 to m-1
   a. w_i = Hash(M)
   b. $W = W \| w_i$
   c. $M = M + 1$ [Note that in Figures 5 and 7, this step is shown a bit differently; a suggestion for reconciliation is welcome.]
5. Return the leftmost n bits of W.

9.1.2.3 Reconstruction

If the input from keylessHashDRBG needs to be reconstructed at a later time (e.g., to perform domain parameter validation for discrete log algorithms), the following information is required:

- The hash function used,
- *SEED,*
- the application-specific constant *t,*

- the iteration counter *i*, and
- the *UserInput* value.

9.1.2.4 Generator Strength

The strength of the **keylessHashDRBG** generator is equal to *N*. [Note: The strength of the generator is determined by the amount of entropy in the seed, up to N. Section 9.1.2.1 states that "The length of the seed (*seedlen*) shall be at least equal to *N*, and the seed shall have at least *N* bits of entropy (i.e., at least *N* bits shall be selected randomly)."]

9.1.2.5 Period of the Generator

[To be determined]

9.1.2.6 Reseeding

A new seed shall be generated to reseed the generator [How often?]

9.1.4 Keyed-Hash Function DRBG[2] using an Approved Hash Function (keyedHashDRBG)

[Comment: This is the same as the keylessHashDRBG that is specified in Section 9.1.2.]

9.1.4.1 Properties

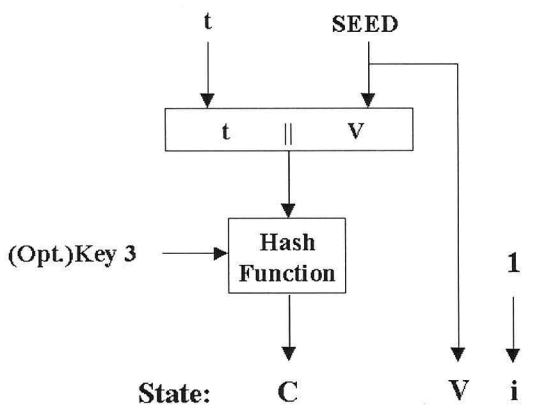

Figure 10: Keyed-Hash DRBG Initialization

Figures 10 and 11 present a keyed-hash deterministic RBG that uses any Approved hash function.

keyedHashDRBG employs an Approved hash function that produces pseudorandom bits using a seed (*SEED*), an application-specific constant (*t*) and one or more keys. Optional user input (*UserInput*) may be provided during each access of **keyedHashDRBG**.

keyledHashDRBG has been designed to meet different security levels (see Annex D). A number *N* is assigned to each hash function, where *N* is the output block size of the hash function.

The hash function used for each instance of **keyedHashDRBG** has an output block size that is *N* bits in length. The generator requires the use of a hash function at three points in the process (which includes the initialization process) (see Figures 10 and 11). The same hash function shall be used at all three points.

The length of the seed (*seedlen*) shall be at least equal to *N*, and the seed shall have at least *N* bits of entropy (i.e., at least *N* bits shall be selected randomly). Further requirements for the seed are provided in Section 7.2.1. [Question: Do we really need N bits of entropy when only less than N bits of strength are required (e.g., 80 bits)?]

The application specific constant (*t*) shall be *N* bits in length.

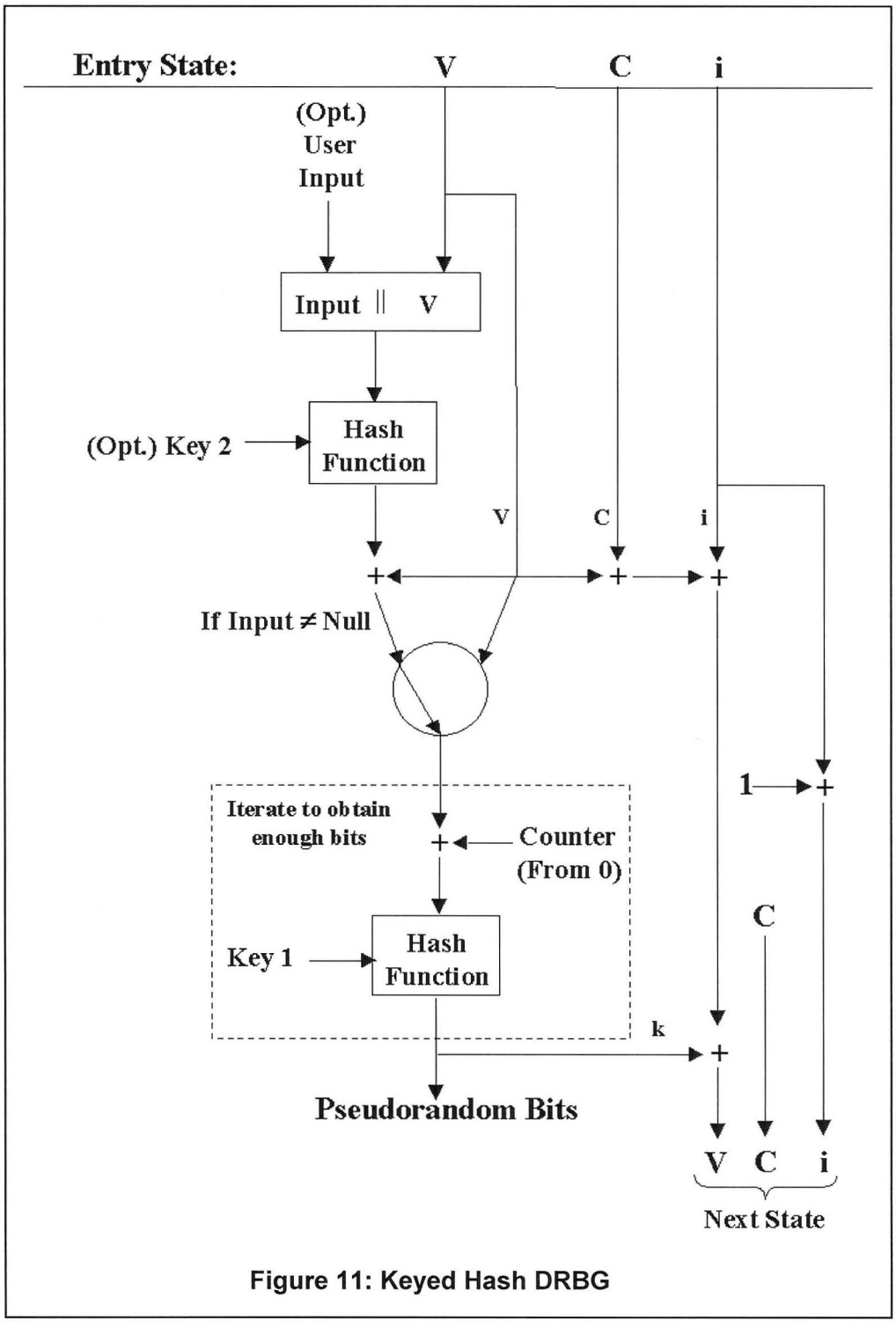

Figure 11: Keyed Hash DRBG

One or more keys are used by the **keyedHashDRBG** generator to key the hash function(s). A key (*Key* 1) for the final hash function (see Figure 11) shall be used. *Key* 2 and *Key* 3 are optional (see Figures 10 and 11). The minimum key length (*minkeylen*) is half the length of the output block (i.e., minkeylen ≤ *outlen*/2 ≤ *N*/2). The maximum key length is the difference

between the length of the input block (*inlen*) and the length of he seed (*seedlen*) (i.e., (*outlen*/2) ≤ *keylen* ≤ (*intlen* - *seedlen*). All keys shall have the same key length. The key (*Key*) shall have at least *outlen*/2 bits of entropy (i.e., at least *outlen*/2 bits shall be selected randomly). Depending on consuming application requirements and risks, the key may be fixed or may be replaced when seeds are replaced. [Question: Does this seem reasonable?] Further requirements for the key are specified in Section 7.2.1.

When the optional user input (*UserInput*) is used, *UserInput* shall be at least *N* bits in length and have at least *N* bits of entropy (i.e, at least *N* bits of the *UserInput* shall be selected randomly).[Same question as before: Are N bits of entropy really necessary if less than N bits of strength are required?]

Figures 12 and 13 depict the insertion of test input for the seed (*SEED*), the application specific constant (*t*), the user input values (*UserInput*) and any keys used. When a key is fixed, the ability to test different values for the key need not be present. The tests shall be run on the output of the generator. Validation and operational testing are discussed in Section 10. Detected errors shall result in a transition to the error state.

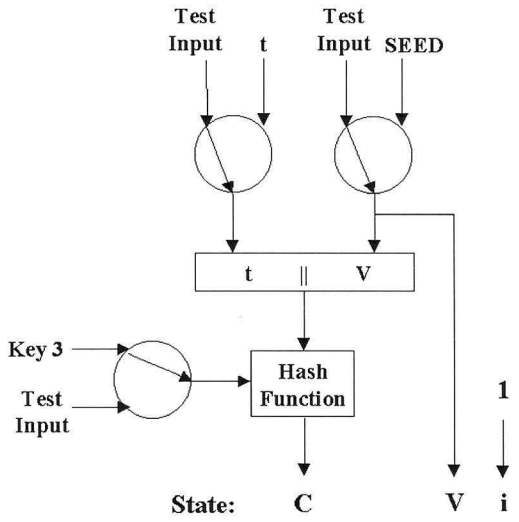

Figure 12: Keyed-Hash DRBG Initialization (With Tests)

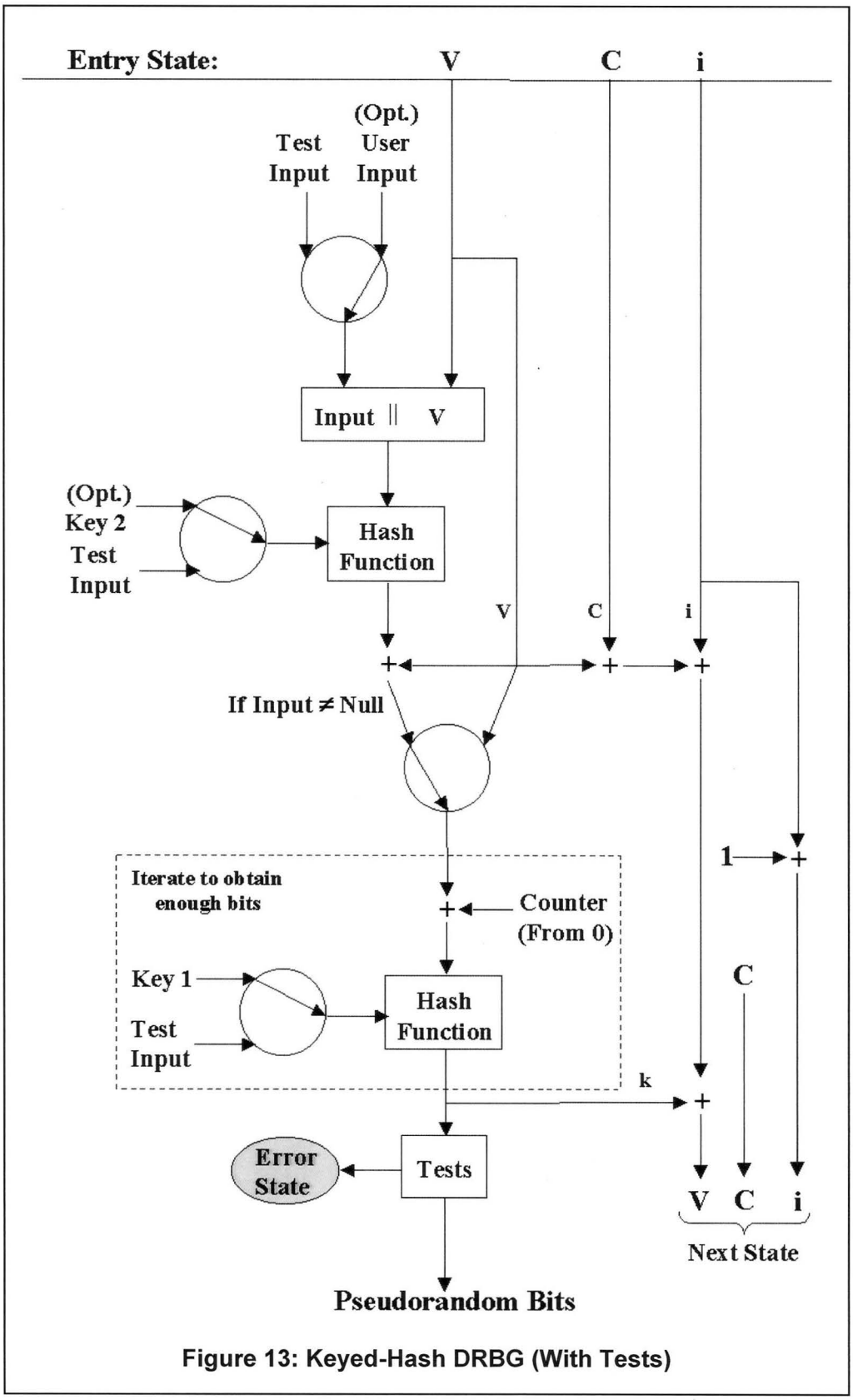

Figure 13: Keyed-Hash DRBG (With Tests)

9.1.4.2 Description

The state of **keyedHashDRBG** consists of a counter (i) and two N-bit numbers (V and C). If the generator uses one or more non-fixed keys, new values for these keys may also be obtained during initialization. The state shall be protected in accordance with the sensitivity of the data protected by the consuming application.

The variables used in the description of **keyedHashDRBG** are:

C	An N-bit constant that is calculated during the initialization process.
i	A counter that is used to update the state of **keyedHashDRBG** and may be used to determine the number of times that the generator has been accessed.
Hash(b)	A hashing operation on data b using an appropriate Approved hash function.
keylen	The length of the key(s) used by the generator, where (*outlen*/2) $\leq$ *keylen* $\leq$ (*inlen* - *seedlen*).
m	The number of iterations of the hash function needed to obtain the requested number of pseudorandom bits.
M	The data to be hashed to obtain the pseudorandom bits
n	The number of bits to be generated. n may be longer than the length of the hash function (*outlen*).
N	A number that is determined by the desired security level. For this standard, N may be 160, 224, 256, 384 and 512 (see Annex D). N is used to determine the hash function to be used.
NRBG(a)	A function that acquires an a-bit string from an Approved non-deterministic RBG or an Approved deterministic RBG (or chain of Approved deterministic RBGs) that is seeded by an Approved non-deterministic RBG.
outlen	The output length of the hash function.
p	The pseudorandom bits produced by the generator.
State	The state of **keyedHashDRBG** that is carried between calls to the generator. The state consists of the current values of $\{V, C, i\}$.
t	An application-specific N-bit constant.
UserInput	Optional user input.
V	A value that is initially set to the value of *SEED*, but assumes new values based on optional user input (*UserInput*), the pseudorandom bits produced by the generator (p), the constant (C) and the iteration count (i).
w, W	Intermediate values.

The algorithm shall proceed as follows:

Initialization:

1. t = the application specific constant
2. $i = 1$

3. $V = SEED =$ NRBG(N)
4. (Opt.) *Key1* = NRBG(*keylen*)
5. (Opt.) *Key2* = NRBG(*keylen*)
6. (Opt.) *Key3* = NBRG(*keylen*)
7. $C = \text{Hash}([Key3] \| t \| V) \bmod 2^N$

   Note: *Key*3 is used in this step if it has been fixed, or if it has been acquired using step 5.
8. *State* = {V, C, i, [*Key1*], [*Key2*], [*Key3*]}

keylessHashDRBG (n, *State*, [*UserInput*]):

1. If *UserInput* ≠ Null:
   a. $w = \text{Hash}([Key2] \| UserInput \| V)$

      Note: *Key*2 is used in this step if it has been fixed, or if it has been acquired during initialization.
   b. $V = (V + w) \bmod 2^N$
2. $p = \text{Hashgen}(n, V)$
3. $V = (V + p + C + i) \bmod 2^N$
4. $i = i + 1$
5. *State* = {V, C, i}
6. Return (p, *State*)

Hashgen(n, V, [Key1])

Note: *Key*1 is required as a parameter only if it was acquired during initialization; otherwise it is fixed and would not be a parameter.

1. $m = \lceil n/outlen \rceil$
2. $M = V$
3. W = the null string
4. For i = 0 to m-1
   a. $w_i = \text{Hash}(Key1, M)$
   b. $W = W \| w_i$
   c. $M = M + 1$

   [Note that in Figures 11 and 13, this step is shown a bit differently; a suggestion for reconciliation of the figure with this step is welcome.]
5. Return the leftmost n bits of W.

9.1.4.3 Reconstruction

If the output from **keyedHashDRBG** needs to be reconstructed at a later time (e.g., to perform domain parameter validation for discrete log algorithms), the following information is required:

- the hash function used,
- Seed,
- the application-specific constant *t*,
- the counter *i*,
- the *UserInput* values used and the associated value of *i* when each was used,
- *Key*1,
- *Key*2, if used, and
- *Key*3, if used.

9.1.4.4 Generator Strength

The maximum strength of the keyedHashDRBG generator is equal to the amount of entropy provided in the seed (i.e., the strength ≤ *N*).

9.1.4.5 Period of the Generator

[Editor's note: Say something about the period of this generator? How is this determined? A function of *xseed*, block size *n* and what?]

9.1.4.6 Reseeding and Rekeying

A new seed shall be generated to reseed the generator [How often?]

New keys may be generated to rekey the generator when a new seed is generated. For high risk applications, new keys (i.e., a new *Key*1 and, optionally, *Key*2 and *Key*3) shall be generated whenever new seeds are generated.