Deterministic RNG Based on cSHAKE and KMAC

April 16, 2024

Abstract

This specification defines a simple deterministic random number generator (DRNG) which can be used to generate cryptographically secure random bit strings for various use cases including symmetric and asymmetric key generation services. The DRNG is based on either cSHAKE or KECCAK Message Authentication Code (KMAC) and is intended to support a wide range of applications and requirements, and is conservative in its resource consumption. A reference implementation of the algorithm is given with leancrypto.

Contents

List of Figures

1 Deterministic RNG Based on cSHAKE and KMAC

A deterministic random number generator (DRNG), also called a pseudo-random number generator (PRNG), is one of the pillars of cryptographic systems. Its goal is to consume input data that is believed or defined to contain entropy and to generate random bit streams from this seed that are indistinguishable from perfect random numbers.

This specification defines a simple deterministic random number generator (DRNG) which can be used to generate cryptographically secure random bit strings named cSHAKE DRNG and KMAC DRNG, respectively. The DRNG is based on the customizable extendable output functions of cSHAKE and KMAC which in turn are based on the KECCAK algorithm as specified in [4].

The cSHAKE / KMAC algorithms are a customizable version of SHAKE which in turn is based on KECCAK with its sponge construction of the absorb phase and squeeze phase. The cSHAKE/KMAC DRNG uses the absorb phase to insert the cSHAKE/KMAC DRNG key data believed to contain entropy. In a second step, the KECCAK squeeze phase is applied to generate a random bit stream of the size requested by the consumer. The KECCAK squeeze phase generates a pseudorandom bit stream of the desired length. The number of the output bits depend on the specific cryptographic algorithms for which the random bit stream is needed.

The cSHAKE/KMAC key is generated with the seed operation that consumes the currently used cSHAKE/KMAC key and the seed data with a cSHAKE/KMAC operation to generate a new cSHAKE/KMAC key. The use of the cSHAKE/KMAC operation to process seed data ensures that input data without a uniform distribution is converted into a cSHAKE/KMAC key to be uniformly distributed. This allows inserting seed data that only partially contains entropy, including the insertion of nonces, personalization strings or other data in any order the caller desires. Thus, the goal of the seed operation is to compress the possibly dispersed entropy of the input data into a cryptographically strong cSHAKE/KMAC key which is used to generate random bit streams from.

The DRNG state management applies the fast-key-erasure mechanism as defined in [2] to ensure "backtracking resistance" in NIST terminology (also called "forward secrecy"). To ensure a key is only used for a limited amount of generated random bits, the fast-key-erasure mechanism is applied at least after generating of twice the cSHAKE256/KMAXOF256 block size number of random bits.

The state to be maintained for the life-time of the cSHAKE/KMAC DRNG is only its key as the cSHAKE/KMAC operation is transient in nature.

The cSHAKE/KMAC DRNG conceptually is very similar to the extract and expand approach of the HKDF algorithm specified in [3].

This specification is closely related to the specification [5], but was developed independently. The proposal of cSHAKE and KMAC DRNG defined in this document complies with the definition in [5] as well.

1.1 cSHAKE-based Deterministic Random Number Generator (cSHAKE-DRNG)

1.1.1 Notation

The cSHAKE-hash denotes the cSHAKE256 function [4]. The cSHAKE-hash has 4 arguments: the main input bit string `X`, the requested output length `L` in bits, a function-name bit string `N`, and an optional customization bit string `S`.

The inputs to the cSHAKE-hash function are specified with references to these parameters.

1.1.2 Encoding

The encoding function is derived from [5] Appendix B to be compliant with this specification.

Let $|\alpha|/8 \in 0, ..., 84$. I.e., the additional input α is a sequence of bytes, and it is at most 84 bytes long. Then, the following encoding unambiguously encodes the inputs while adding only a single byte of stretch:

$$encode(S, \alpha, n) = (S \parallel \alpha \parallel (n * 85 + |\alpha|/8)8)$$

where $(...)_8$ indicates an 8-bit (i.e., single-byte) encoding of a value in $0, 255$. Thus, $|encode| = 8$, i.e., the stretch is constantly one byte.

1.1.3 Seeding

```
cSHAKE-Seeding(K(N), seed, personalization string) ->
    K(N + 1)
```

Inputs:

- `K(N)`: The current cSHAKE DRNG key used by the current instance of the cSHAKE DRNG. If the cSHAKE DRNG is initialized and therefore no current key exists, a zero string of 512 bits size is used.
- `seed`: The caller-provided seed material that contains entropy.
- `personalization string`: An arbitrary string that may be used to achieve domain separation. This string has an arbitrary length and is allowed to be `NULL`.

Output:

- `K(N + 1)`: A new cSHAKE DRNG key that is used for instantiating the cSHAKE hash during the next generate or seed phase.

The seeding of the cSHAKE DRNG is performed as follows:

```
encoded string = encode(personalization string)
K(N + 1) = cSHAKE(N = "cSHAKE-DRNG seed",
                  X = seed || encoded string,
                  L = 512
                  S = K(N))
```

1.1.4 Generating One Block of Random Bit Stream

```
cSHAKE-Generate(K(N), additional input, length) ->
    K(N + 1), random bit stream
```

Inputs:

- `K(N)`: The current cSHAKE DRNG key of 512 bits size.
- `additional input`: The optional additional input may be used to further alter the generated random bit stream.
- `length`: The length of the random bit stream to be generated in bits. The length must be smaller or equal to 2 times the cSHAKE rate size minus 512 (equals to 1,576 bits). This ensures that the entire maximum of data to be squeezed from KECCAK equals to a multiple of full cSHAKE rate blocks.

Outputs:

- `K(N + 1)`: A new cSHAKE DRNG key that is used for instantiating the cSHAKE hash during the next generate or seed operation.
- `random bit stream`: Random bit stream of the requested length.

The generation of one random bit stream block is performed as follows:

```
T(0) = 512 left-most bits of R
T(1) = all right-most bits of R starting with the 512th bit
K(N + 1) = T(0)
random bit stream = T(1)
```

where:

```
encoded string = encode(additional input)
R = cSHAKE(N = "cSHAKE-DRNG generate",
           X = encoded string,
           L = 512 + length,
           S = K(N))
```

1.1.5 Generating Random Bit Stream of Arbitrary Length

Input:

- `K(N)`: The cSHAKE key of 512 bits size generated with the previous generate or seed operation.
- `additional input`: The optional additional input may be used to further alter the generated random bit stream.
- `length`: The length of the random bit stream to be generated in bits.

Output:

- `K(N + 1)`: A new cSHAKE DRNG key that is used for instantiating the cSHAKE hash during the next generate or seed operation.
- `random bit stream`: Random bit stream of the requested length.

The generation of the random bit stream is performed as follows:

```
B = 1088 * 2 - 512
N = ceil(length / B)
TMP_K(0) = K(N)
R = R(1) || R(2) || R(3) || ... || R(N)
random bit stream = first length bits of R
K(N + 1) = TMP_K(N)
```

where:

```
(TMP_K(1), R(1)) = cSHAKE-Generate(TMP_K(0),
                                   additional input, B)
(TMP_K(2), R(2)) = cSHAKE-Generate(TMP_K(1),
                                   additional input, B)
...
(TMP_K(N), R(N)) = cSHAKE-Generate(TMP_K(N - 1),
                                   additional input, B)
```

1.1.6 Rationale

The cSHAKE DRNG key size of 512 bits is chosen based on the following considerations:

- During instantiation of cSHAKE the given key size allows the limitation of KECCAK operations to one: The KECCAK operation is caused by the cSHAKE initialization considering that the length of the key, the cSHAKE256 customization string and the cSHAKE initialization encoding bytes together are less than the block size of cSHAKE256. This limits the number of KECCAK operations required based on the input to the absolute minimum possible based on the cSHAKE specification.

- cSHAKE256 has a security strength of 256 bits. Thus a key size of 256 bits would be sufficient. Yet, considering that due to the fast-key-erasure mechanism the key is hashed to generate a new key, over time the repeated hash operation will decrease the amount of entropy in the key. To allow callers to insert more entropy than the security strength of cSHAKE256 for offsetting this loss of entropy, the key size is set to 512 bits.

The selection of cSHAKE as a DRNG is based on the statement in [4] declaring Keccak is usable as a pseudorandom function.

1.2 KMAC-based Deterministic Random Number Generator (KMAC-DRNG)

1.2.1 Notation

The KMAC-hash denotes the KMACXOF256 function [4] instantiated with cSHAKE 256 [1]. The KMAC-hash has 4 arguments: the key `K`, the main input bit string `X`, the requested output length `L` in bits, and an optional customization bit string `S`.

The inputs to the KMAC-hash function are specified with references to these parameters.

1.2.2 Encoding

See section 1.1.2.

1.2.3 Seeding

```
KMAC-Seeding(K(N), seed, personalization string) ->
    K(N + 1)
```

Inputs:

- `K(N)`: The current KMAC DRNG key used by the current instance of the KMAC DRNG. If the KMAC DRNG is initialized and therefore no current key exists, a zero string of 512 bits size is used.
- `seed`: The caller-provided seed material that contains entropy.
- `personalization string`: An arbitrary string that may be used to achieve domain separation. This string has an arbitrary length and is allowed to be `NULL`.

Output:

- `K(N + 1)`: A new KMAC DRNG key that is used for instantiating the KMAC hash during the next generate or seed phase.

The seeding of the KMAC DRNG is performed as follows:

```
encoded string = encode(personalization string)
K(N + 1) = KMAC(K = K(N),
                X = seed || encoded string,
                L = 512
                S = "KMAC-DRNG seed")
```

1.2.4 Generating One Block of Random Bit Stream

```
KMAC-Generate(K(N), additional input, length) ->
    K(N + 1), random bit stream
```

Inputs:

- `K(N)`: The current KMAC DRNG key of 512 bits size.
- `additional input`: The optional additional input may be used to further alter the generated random bit stream.
- `length`: The length of the random bit stream to be generated in bits. The length must be smaller or equal to 2 times the cSHAKE rate size minus 512 (equals to 1,576 bits). This ensures that the entire maximum of data to be squeezed from KECCAK equals to a multiple of full cSHAKE rate.

Outputs:

- `K(N + 1)`: A new KMAC DRNG key that is used for instantiating the KMAC hash during the next generate or seed operation.
- `random bit stream`: Random bit stream of the requested length.

The generation of one random bit stream block is performed as follows:

```
T(0) = 512 left-most bits of R
T(1) = all right-most bits of R starting with the 512th bit
K(N + 1) = T(0)
random bit stream = T(1)
```

where:

```
encoded string = KMAC-Encode(additional input)
R = KMAC(K = K(N),
         X = encoded string,
         L = 512 + length,
         S = "KMAC-DRNG generate")
```

1.2.5 Generating Random Bit Stream of Arbitrary Length

Input:

- `K(N)`: The KMAC key of 512 bits size generated with the previous generate or seed operation.
- `additional input`: The optional additional input may be used to further alter the generated random bit stream.
- `length`: The length of the random bit stream to be generated in bits.

Output

- `K(N + 1)`: A new KMAC DRNG key that is used for instantiating the KMAC hash during the next generate or seed operation.
- `random bit stream`: Random bit stream of the requested length.

The generation of the random bit stream is performed as follows:

```
B = 1088 * 2 - 512
N = ceil(length / B)
TMP_K(0) = K(N)
R = R(1) || R(2) || R(3) || ... || R(N)
random bit stream = first length bits of R
K(N + 1) = TMP_K(N)
```

where:

```
(TMP_K(1), R(1)) = KMAC-Generate(TMP_K(0),
                                 additional input, B)
(TMP_K(2), R(2)) = KMAC-Generate(TMP_K(1),
                                 additional input, B)
...
(TMP_K(N), R(N)) = KMAC-Generate(TMP_K(N - 1),
                                 additional input, B)
```

1.2.6 Rationale

The KMAC DRNG key size of 512 bits is chosen based on the following considerations:

- During instantiation of KMAC the given key size allows the limitation of KECCAK operations to 2: The first KECCAK operation is due to the cSHAKE256 initialization. The second KECCAK operation is caused by the KMAC initialization considering that the length of the key, the cSHAKE256 customization string and the KMAC initialization encoding bytes together are less than the block size of cSHAKE256. This limits the number of KECCAK operations required based on the input to the absolute minimum possible based on the KMAC specification.

- KMAC256 has a security strength of 256 bits. Thus a key size of 256 bits would be sufficient. Yet, considering that due to the fast-key-erasure mechanism the key is hashed to generate a new key, over time the repeated hash operation will decrease the amount of entropy in the key. To allow callers to insert more entropy than the security strength of KMAC256 for offsetting this loss of entropy, the key size is set to 512 bits. The selection of KMAC as a DRNG is based on the statement in [4] declaring Keccak is usable as a pseudorandom function.

1.3 Comparison cSHAKE And KMAC DRNG

The cSHAKE DRNG is completely identical with the exception that the cSHAKE DRNG uses cSHAKE256 and the KMAC DRNG uses KMACXOF256 as central functions. The difference of the customization string is irrelevant to the cryptographic strength of both.

The handling of the key is also very similar:

- The cSHAKE DRNG sets the key as part of the N input - the N and X input are concatenated and padded by cSHAKE to bring the entire string into multiples of a cSHAKE block. This data is inserted into the SHAKE algorithm which implies that the insertion triggers as many KECCAK operations as cSHAKE blocks are present based on the input. The cSHAKE DRNG data implies that only one cSHAKE block is present and thus one KECCAK operation is performed.

- The KMAC DRNG sets the key compliant to the KMAC definition. KMAC sets two well-defined strings as part of the cSHAKE initialization. The cSHAKE initialization concatenates and pads the input strings to bring the entire string into multiples of a cSHAKE block. This data is inserted into the SHAKE algorithm which implies that the insertion triggers as many KECCAK operations as cSHAKE blocks are present on the input. The KMAC DRNG data implies that only one cSHAKE block is present and thus one KECCAK operation is performed. In addition, KMAC pads the key data into a string that is also multiples of a cSHAKE block in size. Again, this data is inserted into the SHAKE algorithm which again triggers as many KECCAK operations as cSHAKE blocks are present with the key-based input. The KMAC DRNG specification implies again, that only one KECCAK operation is performed.

The rationale shows that for both, the cSHAKE DRNG and the KMAC DRNG the data believed to hold entropy, the key, is inserted into the SHAKE state. The additional data inserted with the KMAC operation does not contain any entropy and only mixes the SHAKE state further without affecting the existing entropy. Therefore, with respect to the entropy management, the cSHAKE DRNG and the KMAC DRNG are considered equal.

Considering that the cSHAKE DRNG requires only one KECCAK operation during initialization whereas the KMAC DRNG requires two operations, the cSHAKE DRNG requires in total only 2 KECCAK operations for generating a random bit stream of 1088 - 512 = 576 bits (or less). When comparing this to the KMAC DRNG, in total 3 KECCAK operations are required for generating the same 576 bits (or less). This implies that the cSHAKE DRNG requires only 2/3 of the processing time compared to a KMAC DRNG. It is expected that the majority of all requests will be less than 576 bits, e.g. commonly 256 bits for symmetric keys.

Thus, the cSHAKE DRNG has a higher performance with a equal entropy management comparing to the KMAC DRNG.

References

[1] *FIPS PUB 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.* NIST, August 2015.

[2] Dan Bernstein. *Fast-key-erasure random-number generators.* July 23, 2017. Available at https://blog.cr.yp.to/20170723-random.html.

[3] P. Eronen H. Krawczyk. *RFC5869 HMAC-based Extract-and-Expand Key Derivation Function (HKDF).* May, 2010.

[4] Ray Perlne John Kelsey, Shu-jen Chang. *NIST Special Publication 800-185 SHA-3 Derived Functions: cSHAKE, CSHAKE, TupleHash and ParallelHash.* December, 2016.

[5] John Kelsey, Stefan Lucks, and Stephan Müller. Xdrbg: A proposed deterministic random bit generator based on any xof. *IACR Transactions on Symmetric Cryptology*, 2024(1):5–34, Mar. 2024.