HMAC Cryptographic Generator – HCG 1.0

Revision 1.0, October 20, 2024

John G. Underhill – john.underhill@protonmail.com

This document is an engineering level description of the HCG 1.0 deterministic random bit generator (DRBG).

1. Introduction

The HMAC-based Cryptographic Generator (HCG) is a deterministic random byte generator (DRBG) built using HMAC with SHA-512. It uses an internal state and caches random bytes for high throughput. HCG is primarily used for applications requiring cryptographically secure random numbers, such as key generation and cryptographic operations.

The generator is designed to support both deterministic and non-deterministic random bit generation through the optional use of the predictive resistance function, which injects random entropy at regular intervals (1MB) to enhance security.

2. Protocol Description

The HCG protocol consists of several functions to initialize, generate, update, and dispose of the random generator's state.

2.1 Initialization (qsc_hcg_initialize)

Purpose: Initializes the HCG state with a seed and optional personalization string. The generator can also be set to inject random entropy periodically through a predictive resistance mechanism.

Inputs:

- ctx: The context state structure.
- seed: The seed used to initialize the generator.
- seedlen: The byte length of the seed.
- info: The optional personalization string.
- infolen: The length of the personalization string.
- predres: A boolean flag enabling predictive resistance.

Function Logic:

- The seed is used to initialize the internal HMAC state.
- If a personalization string is provided, it is absorbed into the state.
- Predictive resistance, if enabled, injects random entropy at 1MB intervals using the qsc_acp_generate function.
- The internal cache is pre-filled with pseudo-random data generated by the HMAC.

2.2 Random Byte Generation (qsc_hcg_generate)

Purpose: Generates the requested number of pseudo-random bytes by accessing and refilling the internal cache as needed.

Inputs:

- ctx: The context state structure.
- output: The output buffer.
- otplen: The requested number of bytes.

Function Logic:

- Bytes are fetched from the internal cache if available. If more bytes are required, the internal state is updated using HMAC and more random bytes are generated.
- When predictive resistance is enabled, it ensures that new entropy is injected at periodic intervals (1MB by default).

2.3 State Update (qsc_hcg_update)

Purpose: Updates the HCG state with new seed material, ensuring the internal state remains unpredictable.

Inputs:

- ctx: The context state structure.
- seed: The new seed material.
- seedlen: The length of the seed.

Function Logic:

- New seed material is absorbed into the internal HMAC state to refresh the generator.
- After updating, the cache is re-filled with pseudo-random bytes from the updated state.

2.4 State Disposal (qsc_hcg_dispose)

Purpose: Securely disposes of the HCG state, wiping all sensitive data from memory.

Inputs:

- ctx: The context state structure.

Function Logic:

- All sensitive state information is securely wiped from memory, including the internal cache and HMAC state.

3. Mathematical Description

The **HCG** generator is built on the **HMAC** construction, which uses **SHA-512** as the underlying hash function. The mathematical foundation of HMAC is based on the following equation:

Where:

- k is the internal key.
- m is the salt or info string.
- n is the nonce.
- p is an internally generated seed
- s is the input seed.
- G is the pseudo-random generator.
- HMAC is the keyed hash function; HMAC(SHA2-512).
- Init is the initialization function.
- Update is the update function.

Initialize:

The initialization function keys the HMAC with the user supplied input seed s, and adds the info string m to the MAC state. If the predictive resistance option is used, a seed p is generated and added to the MAC state. The HMAC state is finalized, to create an internal key k, is stored and used to key the HMAC generator.

$$k = \mathrm{HMAC}(s, m \,\|\, p)$$

Generate:

A monotonic byte counter, the nonce n is incremented every time the HMAC is finalized. During the generation phase, the HMAC is keyed with the key k, and the info string m, the nonce n, and if predictive resistance is enabled, a new seed p is generated and added at 1MB intervals to the MAC message.

$$\mathrm{HMAC}(k, m \,\|\, n \,\|\, p)$$

When generating output, the HMAC is run consecutively until the output request length is generated, each time incrementing the nonce to produce a different output hash. This hash is added to the output array as pseudo-random bytes, with the cycle repeating until the request is fulfilled.

If T is the output from the generator and i is the output index of the generated segement:

$$T_i = \mathrm{HMAC}(k, \mathit{info} \,\|\, n_i \,\|\, p\,),\ n_i \leftarrow n_i + 1$$

The output of HMAC-SHA2-512 is used to generate pseudo-random bytes, which are used to fill the output array on a generation request

The generator uses an HKDF type construction, where a nonce counter is incremented after each call to HMAC. This counter along with the state, are permuted by the HMAC function to fill the output buffer. After each call to HMAC, the nonce is incremented and added to the state processed by HMAC.

5. Security Analysis

The HCG generator inherits its security from the HMAC construction and SHA-512, providing strong resistance to cryptographic attacks.

Brute-Force Resistance: The seed and SHA-512's 512-bit output make brute-force attacks computationally infeasible.

Predictive Resistance: Predictive resistance, when enabled, injects random entropy at regular intervals, protecting the generator from state compromise and ensuring unpredictability even after significant output.

Quantum Resistance: While HMAC-SHA-512 is not quantum-proof, it still provides security against quantum attacks, with Grover's algorithm reducing the security by only half (from 512 bits to 256 bits).

Side-Channel Attack Resistance: Sensitive data, such as the internal state and cache, are securely wiped after use, reducing the risk of side-channel attacks.

6. References

1. NIST FIPS 180-4: **SHA2 Standard**
   http://csrc.nist.gov/publications/fips/fips180-4
2. RFC 2104: **HMAC: Keyed-Hashing for Message Authentication** -
   http://tools.ietf.org/html/rfc2104
3. **SP 800-90C**: Recommendation for Random Bit Generator (RBG) Constructions
   https://csrc.nist.gov/publications/detail/sp/800-90c/final
4. **SP 800-22 Rev. 1a**: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications
   https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final

Conclusion

HCG is a secure and efficient deterministic random bit generator (DRBG) that uses the strength of HMAC-SHA-512 to generate high-quality pseudo-random numbers. With its ability to cache random bytes and support predictive resistance through periodic entropy injection, HCG is well-suited for cryptographic applications requiring both performance and security.

Key Strengths:

1. **Cryptographic Security**: HCG leverages HMAC-SHA-512, ensuring that the generated random bytes are resistant to brute-force attacks and provide strong diffusion properties for secure cryptographic pseudo-random generation.
2. **Predictive Resistance**: The ability to inject random entropy at regular intervals adds a layer of security, ensuring that future outputs remain unpredictable, even if previous state was compromised.
3. **Scalability and Performance**: The internal caching mechanism allows HCG to quickly generate large volumes of pseudo-random data, making it efficient in environments requiring fast, secure randomness, such as key generation in enterprise environments.
4. **Secure Memory Management**: By securely wiping sensitive internal state and cache after use, HCG minimizes the risk of data leakage through side-channel attacks.