Custom Shake Generator – CSG 1.0

Revision 1.0, October 20, 2024

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

This document is an engineering level description of the Custom SHAKE Generator (CSG) deterministic random byte generator (DRBG).

1. Introduction

The Custom Shake Generator (CSG) is a deterministic random byte generator (DRBG) that utilizes the SHAKE XOF function from the SHA-3 standard to generate secure, deterministic or non-deterministic pseudo-random byte stream. CSG provides a flexible, high-security mechanism for generating random numbers critical for cryptographic applications, ensuring forward secrecy, resistance to various attacks, and quantum resilience. The protocol is designed to offer scalable performance while maintaining the strong cryptographic properties of the Keccak sponge construction.

2. Protocol Description

The CSG Protocol consists of the following core operations:

2.1 Initialization (qsc_csg_initialize)

Purpose: Initializes CSG with a seed and optional personalization string. Optionally, predictive resistance can be enabled.

Inputs:

- ctx: The state context structure.
- seed: The primary generator seed.
- seedlen: The seed array length (32 or 64 bytes).
- info: Optional personalization string.
- infolen: The personalization string length.
- predres: Boolean flag enabling predictive resistance.

Function Logic:

- **Seed Initialization**: Determines if the protocol uses cSHAKE-256 (32-byte seed) or cSHAKE-512 (64-byte seed). Clears the internal state and cache to remove any previous data.
- **Predictive Resistance Handling**: If enabled, pseudo-random entropy is generated using the entropy provider and added to the SHAKE state periodically.

- **State Initialization**: The Keccak sponge is initialized with the seed and personalization string.
- **First Cache Fill**: Fills the cache with the first block of pseudo-random bytes.

2.2 Random Byte Generation (qsc_csg_generate)

Purpose: Generates the requested number of pseudo-random bytes.

Inputs:

- ctx: The state context structure.
- output: The buffer to store the random bytes.
- otplen: The number of bytes to write to the output array.

Function Logic:

- **Cache Check**: If enough random bytes are in the cache, write them to the output array. Otherwise add the bytes remaining in the cache to the output, refill the cache, and write the remaining bytes to the output array.
- **Cache Refill**: Calls csg_fill_buffer to squeeze random bytes from the Keccak sponge into the cache.
- **Reseed Check**: If predictive resistance is enabled and more than 1MB of data has been written to output, reseed the generator.

2.3 State Update (qsc_csg_update)

Purpose: Updates the CSG's internal state with new seed material.

Inputs:

- ctx: The state context structure.
- seed: The primary generator seed.
- seedlen: The seed array length.

Function Logic:

- **State Update**: Absorbs the new seed into the internal Keccak state, ensuring forward security.
- **Cache Refill**: After updating the state, the cache is refreshed with new pseudo-random data.

2.4 State Disposal (qsc_csg_dispose)

Purpose: Securely disposes of the internal state and clears all sensitive data.

Inputs:

- ctx: The state context structure.

Function Logic:

- **State Clear**: Securely wipes the Keccak state using qsc_keccak_dispose**.**
- **Memory Clear**: Clears the cache and state counters to prevent leakage of sensitive information.

3. Mathematical Description

CSG is based on the Keccak sponge construction, a flexible and secure cryptographic primitive that absorbs inputs and produces outputs through two phases: absorption and squeezing.

3.1 Absorption Phase:

The seed and optional personalization string are absorbed into the internal state ***S***. The Keccak permutation function **P** is applied to ensure that the input mixes thoroughly with the internal state:

$$S \leftarrow \mathrm{P}(S \oplus M)$$

3.2 Squeezing Phase:

After absorption, the state is squeezed to produce the random output ***Z***. The squeezed data is cached for future use:

$$Z = \mathrm{P}(S)$$

For each block of output, the Keccak permutation is applied to the state, ensuring that every byte of output is cryptographically secure.

3.3 Predictive Resistance:

If predictive resistance is enabled, additional entropy is periodically injected into the state at regular intervals (every 1MB of output), ensuring that attackers cannot predict future outputs based on previous ones.

4. Security Analysis

The **security analysis** of CSG focuses on its resistance to a variety of cryptographic attacks, including brute-force, side-channel, and quantum-based attacks. CSG leverages the inherent security properties of the SHAKE XOF function and the Keccak sponge to ensure a strong defense against modern and future threats.

4.1 Brute-Force Resistance:

Due to the cryptographic strength of SHAKE, it is computationally infeasible for an attacker to guess the internal state of the generator, even if they have access to a large number of random outputs.

4.2 Side-Channel Attack Defenses:

CSG implements secure memory handling to prevent side-channel attacks that could extract sensitive information like seeds or intermediate states.

Timing attacks are mitigated by using constant time functions across operations, reducing the ability of an attacker to infer the internal state based on timing information.

4.3 Quantum Resistance:

CSG is resistant to quantum-based attacks thanks to the properties of the SHAKE function. While quantum computers could potentially break weaker DRBGs, the SHAKE XOF function remains secure against known quantum algorithms.

4.4 Comparison with Other DRBGs:

Unlike traditional DRBGs that may rely on AES or other symmetric algorithms, CSG provides stronger post-quantum security due to its reliance on the Keccak sponge. This makes CSG a more future-proof alternative, particularly for high-security environments.

5. References

- **SHA-3 Standard**: NIST FIPS 202 - SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.
  https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

- **Keccak Sponge Function**: Keccak Team. (2020). The Keccak Sponge Function Family.
  https://keccak.team/sponge_duplex.html

- **SP 800-90C**: Recommendation for Random Bit Generator (RBG) Constructions
  https://csrc.nist.gov/publications/detail/sp/800-90c/final

- **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

CSG is a robust and efficient deterministic random byte generator (DRBG) that leverages the cryptographic strength of the SHAKE XOF function from SHA-3. By using the Keccak sponge construction, CSG provides a highly secure and scalable solution for randomness generation, which is critical in modern cryptographic systems.

Key Strengths of CSG:

- **Cryptographic Security**: CSG offers strong resistance to a wide range of attacks, including brute-force, preimage, second-preimage, side-channel, and quantum-based attacks. Its reliance on SHAKE ensures that random outputs are unpredictable and secure.
- **Post-Quantum Readiness**: As quantum computing becomes a reality, the security of many traditional cryptographic systems will be challenged. CSG's use of the SHA3 SHAKE functions makes it inherently resistant to quantum-based threats, making it a future-proof solution for generating secure random numbers.
- **Scalability and Flexibility**: CSG's ability to generate variable-length random outputs makes it adaptable to a wide range of cryptographic applications. Additionally, the sponge's flexible design allows for efficient absorption of entropy and random byte extraction, enabling CSG to scale well in high-performance environments.
- **Secure Memory Management**: The protocol ensures that sensitive cryptographic material, such as seeds and internal states, is securely handled and wiped after use, reducing the risk of side-channel attacks.

Recommendations:

CSG is well-suited for applications that require highly secure, deterministic random byte generation. It can be effectively deployed in cryptographic protocols, financial services, secure communications, and post-quantum systems. Given its quantum resistance and strong security properties, CSG is a viable and future-proof alternative to traditional DRBGs.

In conclusion, CSG provides a comprehensive solution for secure random byte generation, offering both current and future protection against advanced cryptographic threats. Its design makes it an ideal candidate for high-security environments where randomness plays a critical role in maintaining system integrity and confidentiality.