The Design and Formal Analysis of RCS: A Quantum-Resilient AEAD Scheme

John G. Underhill

Quantum Resistant Cryptographic Solutions Corporation

Abstract. RCS, the Rijndael Cipher Stream, is a wide-block authenticated encryption scheme that combines a 256-bit Rijndael permutation with cSHAKE derived round keys and a KMAC-based authentication layer. This paper provides a formal cryptanalysis of the construction, presenting a complete game-based security treatment and an engineering specification aligned with the reference implementation. We begin by defining the precise AEAD syntax, key and nonce requirements, and the operational behavior of RCS, including a pseudo-code description derived directly from the implementation. Using this model, we establish confidentiality through a sequence of hybrid games that reduce the security of the key-stream generator to the PRF security of cSHAKE and the PRP security of the Rijndael-256 permutation under independently derived subkeys. Integrity is shown to follow from the existential unforgeability of KMAC, which authenticates associated data, ciphertext, the nonce, and encoded length information under domain-separated customization parameters. Combining these results with a standard Encrypt-then-MAC composition theorem yields IND-CCA security under unique nonce usage. The cryptanalytic evaluation examines differential, linear, algebraic, and related-key attacks on wide-block Rijndael variants and demonstrates that the 22-round (RCS-256) and 30-round (RCS-512) configurations lie well beyond all published attack boundaries.

Independent cSHAKE derived round keys eliminate structural weaknesses associated with classical Rijndael key schedules, while the sponge-based authentication layer avoids the linearity and nonce-misuse issues inherent to polynomial MAC constructions. We also analyze post-quantum considerations, showing that RCS retains its intended security levels under standard quantum query models. Finally, we document the operational assumptions required for correctness and discuss the limits of the construction, including the fact that RCS is not misuse-resistant and relies critically on nonce uniqueness. The resulting analysis provides a rigorous and implementation-aligned foundation for evaluating RCS as a symmetric primitive suitable for high-assurance and post-quantum-transition environments.

1 Introduction

RCS, the *Rijndael Cipher Stream*, is a wide-block authenticated encryption construction that integrates a 256-bit Rijndael permutation with cSHAKE-based subkey generation and a KMAC authentication layer. The design seeks to provide deterministic and structurally simple authenticated encryption while maintaining strong resilience to classical and post-quantum adversaries. Unlike modes that derive both confidentiality and integrity from a single duplex state, RCS separates its encryption and authentication components, employing counter-mode encryption over a wide permutation and an independent sponge-based MAC. This modular layout simplifies analysis, enhances domain separation, and allows each component to be treated under its respective idealized model.

The RCS construction builds directly on two well-established families of primitives. The wide-block Rijndael permutation extends the AES design to a 256-bit state, enabling deeper diffusion and reducing exposure to structural attacks on reduced-round variants. The cSHAKE and KMAC functions, derived from the Keccak sponge framework, provide domain-separated key expansion and message authentication without relying on polynomial structures that are fragile under nonce misuse. Together these yield an authenticated encryption scheme with clean composition properties and conservative cryptanalytic margins.

This document refines and strengthens the original RCS analysis by giving a complete, formally grounded treatment of its security. Earlier drafts presented high-level reasoning and informal reductions; here we replace those with full game-based proofs of confidentiality, integrity, and AEAD security, each aligned with standard definitions used in modern cryptographic literature. In addition, we justify the chosen parameters through a broader cryptanalytic survey of differential, linear, algebraic, and related-key attacks on wide-block Rijndael and sponge-based constructions. Finally, we introduce an engineering-level description of the scheme derived directly from the reference implementation, ensuring that the formal model and the deployed system are fully aligned.

1.1 Background and Motivation

The motivation for RCS arises from the need for an authenticated encryption primitive that is simultaneously easy to analyze, domain-separated at all interfaces, and powered by primitives whose security arguments remain stable in the post-quantum era. Conventional block-cipher-based AEAD schemes, including popular polynomial modes, rely on key schedules or linear MAC structures that introduce analytical challenges or misuse vulnerabilities. Sponge-based designs address some of these limitations but often entwine encryption and authentication within a single state, complicating proofs and side-channel hardening.

RCS instead adopts a hybrid model: Rijndael-256 is used exclusively for key-stream generation in counter mode, while a separate KMAC instance authenticates the associated data and its length, the nonce, the ciphertext, and a final encoded length value derived from the number of processed bytes. The cSHAKE key schedule removes the linear recurrences that enable related-key and slide attacks against the classical AES schedule, and the wider permutation increases the difficulty of reduced-round differential and linear trails. These design choices make RCS a useful case study in integrating well understood primitives into a conservative, post-quantum-ready AEAD construction.

1.2 Contributions of this Work

This formal cryptanalysis advances the RCS analysis in four specific ways:

- **Engineering-Level Definition.** We introduce a precise operational description of RCS, including a pseudo-code specification derived directly from the reference

implementation. This ensures consistent interpretation across implementations and proofs.

- **Complete Game-Based Reductions.** We present full hybrid-game proofs for confidentiality (IND–CPA), integrity (INT–CTXT), and composed authenticated encryption security (IND–CCA), reducing these properties to the PRP security of the wide-block Rijndael permutation and the PRF/EUF–CMA security of cSHAKE and KMAC.
- **Cryptanalytic Justification of Parameters.** We analyze the resistance of the 22-round (RCS–256) and 30-round (RCS–512) configurations against published differential, linear, algebraic, and related-key attacks, demonstrating conservative margins in each case.
- **Clarification of Operational Assumptions and Limits.** We document the required conditions for security, including nonce uniqueness, and provide a clear explanation of the consequences of misuse, such as confidentiality failure under nonce reuse.

1.3 Organization of the Paper

Section 2 provides a complete engineering-level description of RCS, including a pseudo-code specification that aligns with the reference implementation. Section 3 introduces the notation, primitive assumptions, and adversarial model used throughout the analysis. Section 4 formalizes the RCS construction within this model. Section 5 presents the security definitions for confidentiality, integrity, and authenticated encryption. Section 6 contains the full game-based reductions establishing these guarantees. Section 7 evaluates the cryptanalytic strength of RCS against major attack families and discusses the rationale behind its parameters. Section 8 describes operational assumptions, limitations, and misuse considerations. Section 9 analyzes implementation conformance and side-channel aspects. Section 10 concludes with a summary of results and directions for further refinement.

2 Engineering Description of RCS

This section provides an implementation-aligned description of the RCS authenticated cipher stream. The presentation follows the behavior of the reference implementation `rcs.c` and `rcs.h` and serves as the canonical definition of the scheme. All notations introduced here are later used in the formal specification and security proofs.

2.1 Interface and Parameters

RCS is an authenticated encryption with associated data (AEAD) construction with fixed parameter sets RCS–256 and RCS–512. Each parameter set defines the sizes of the cipher key and authentication tag, while the block size and nonce size are fixed across variants.

Key Size. RCS–256 uses a 256-bit key (`QSC_RCS256_KEY_SIZE` = 32 bytes). RCS–512 uses a 512-bit key (`QSC_RCS512_KEY_SIZE` = 64 bytes).

Block Size. The permutation operates on a fixed 256-bit block (`QSC_RCS_BLOCK_SIZE` = 32 bytes).

Nonce. Each encryption requires a 256-bit nonce of length `QSC_RCS_NONCE_SIZE` = 32 bytes. Nonce uniqueness for each key is mandatory.

Authentication Tag. RCS–256 outputs a 32-byte tag (`QSC_RCS256_MAC_SIZE`). RCS–512 outputs a 64-byte tag (`QSC_RCS512_MAC_SIZE`).

Associated Data. Associated data is an arbitrary byte string that is authenticated but not encrypted. The caller provides associated data via `qsc_rcs_set_associated` before calling the transform routines.

API Surface. The implementation exposes the following public functions:

- `qsc_rcs_initialize(state, keyparams, encrypt)` initializes the cipher, derives round keys and the MAC key, loads the nonce, and sets the mode.
- `qsc_rcs_set_associated(state, data, length)` optionally adds AEAD data to the authentication function after initialization and before the transform.
- `qsc_rcs_transform(state, out, in, length)` encrypts or decrypts a message of length `length` bytes and applies or verifies a tag.
- `qsc_rcs_extended_transform` provides identical functionality for multi-part messages.

Table 1: RCS parameters for the two supported variants.

Parameter	RCS–256	RCS–512
Key length	32 bytes	64 bytes
Tag length	32 bytes	64 bytes
Rounds (R)	22	30
Round key size	32 bytes × (R+1)	32 bytes × (R+1)
Keccak rate (r)	136 bytes	136 bytes
Keccak capacity (c)	512 bits	512 bits
Domain string	`rcs256`	`rcs512`

2.2 Internal State Layout

The internal state structure `qsc_rcs_state` contains all data needed for the permutation, key schedule, counter generation, and authentication:

- **Cipher Type** (`ctype`) distinguishing RCS–256 or RCS–512.
- **Round Key Array** An array of 128-bit or 32-bit elements (depending on platform), containing the expanded subkeys for all rounds. The total number of rounds is:

$$R_{256} = 22, \qquad R_{512} = 30.$$

- **Keccak State** (`kstate`) The internal sponge state used for key expansion and MAC key extraction.
- **Nonce Storage** A 32-byte buffer containing the initialization nonce.
- **Byte Counter** A 64 bit counter that accumulates the total number of message bytes processed by the transform routines. It is used only in the final length encoding absorbed by the MAC and does not drive the counter values used by the Rijndael 256 encryption layer.
- **Mode Bit** (`encrypt`) Determines whether the transform function performs encryption or decryption.

The 256-bit Rijndael state is represented internally as two independent 128-bit halves, each stored as a 4×4 byte array. Round keys are 256-bit values applied to the full state each round.

2.3 Background on Rijndael–256

Rijndael–256 is a wide–block variant of the Rijndael family, operating on a 256–bit state arranged as two parallel 128–bit arrays. Each array is organized as a 4×4 byte matrix and processed using the standard Rijndael round functions: nonlinear byte substitution, row shifting, column mixing over $\mathbb{GF}(2^8)$ and round–key addition. The two state halves are updated in lockstep, and the round keys span the entire 256–bit block.
The nonlinear layer is identical to AES, and the linear diffusion layer is generalized to 256 bits through extended column mixing. The widened state increases the number of active S–boxes per round and raises the algebraic degree of the permutation more rapidly than AES–128 or AES–256. These properties sharply constrain differential and linear trail construction and increase the minimum data complexity required for distinguishers.
In RCS, this permutation is used exclusively in counter mode. Each key-stream block is produced by applying the Rijndael–256 permutation to a 256–bit counter value, which is incremented modulo 2^{256} at each block. The master key does not feed an internal key schedule; instead, independently distributed round keys are derived from a cSHAKE–based expansion. Under the assumption that the per–round keys behave as independent 256–bit values, the permutation is modeled as a pseudo-random permutation (PRP) in the security analysis.
The RCS–256 and RCS–512 variants use 22 and 30 rounds, respectively, providing significant margin over known reduced–round analyses of Rijndael–256. These round counts place the construction well outside the range of known differential, linear, algebraic and structural cryptanalytic techniques.

2.4 Key Expansion via cSHAKE

Key expansion is performed using the cSHAKE extendable-output function. The initialization absorbs:

1. the input cipher key,
2. the optional user-defined information (tweak) string `info`,
3. a fixed, scheme-specific domain string:
   - `rcs256_name` for RCS–256,
   - `rcs512_name` for RCS–512.

Once initialized, the cSHAKE instance produces a stream of pseudo-random bytes. This stream is used as follows:

- first, a contiguous block of bytes is squeezed and mapped into the array of (R + 1) 256-bit round keys,
- then, after at least one additional permutation of the sponge state, a further squeeze yields the MAC key (32 bytes in RCS–256 or 64 bytes in RCS–512).

The domain strings guarantee that RCS key expansion is isolated from other cSHAKE applications and is resistant to related-key and cross-domain collisions.

2.5 Counter Mode Encryption Layer

Encryption uses the wide-block Rijndael permutation in counter mode.

Counter Initialization. Counter Initialization. The initial counter block is the 32-byte nonce interpreted as a 256-bit little-endian integer. For block index i, the counter value is $\mathsf{CTR}_i = (N + i) \bmod 2^{256}$, and this value is used directly as the permutation input for that block.

key-stream Generation. For each block index i:

1. Form the counter block CTR_i as

$$\mathsf{CTR}_i = (N + i) \bmod 2^{256}.$$

2. Apply the Rijndael-256 round function with the expanded round keys:

$$KS_i = E_{RK}(CTR_i).$$

3. Increment the counter.

In the reference implementation the counter blocks are realized by treating the 32 byte nonce buffer as a 256 bit little endian integer and incrementing it in place after each block with qsc_intutils_le8increment; the byte counter ctx.counter is used only by the MAC layer.

Encryption and Decryption. The key-stream block KS_i is XORed with the plaintext block P_i to form ciphertext C_i, and conversely during decryption.

2.6 KMAC Authentication Layer

RCS authenticates ciphertext using a KMAC instance derived from the same cSHAKE state but extracted from an independent domain-separated portion of its output.
The authenticated input to KMAC consists of a deterministic transcript that includes:

- the associated data AD,
- a fixed-width encoding of $|AD|$,
- the current 32-byte nonce N,
- the ciphertext C, and
- a final fixed-width encoding of the total number of bytes processed during the transform stage (which depends on the nonce block size and the accumulated ciphertext length).

Tag Generation. Upon finalizing encryption:

1. all associated data is absorbed, followed by a fixed-width encoding of its length, via `qsc_rcs_set_associated`,
2. immediately before processing a message chunk, the current 32-byte nonce value is absorbed,
3. each ciphertext chunk is absorbed during the transform, and a running byte counter is updated,
4. at finalization, a fixed-width encoding of the total number of processed bytes (nonce block plus ciphertext and the length field itself) is absorbed, and
5. a tag of length 32 bytes in RCS–256 or 64 bytes in RCS–512 is output.

Tag Verification. During decryption, a tag is recomputed and must match the provided tag in constant time. If verification fails, decryption returns failure and no plaintext is released.

Table 2: Components of the authenticated transcript absorbed by KMAC.

Component	Description
AD	Associated data
encode_len($\|AD\|$)	Fixed-width encoding of AD length
N	32-byte nonce absorbed per transform call
C	Ciphertext bytes processed so far
encode_len(total_bytes)	Encoded count of total bytes absorbed

2.7 Background on Keccak, cSHAKE and KMAC

RCS uses two components from the Keccak family: cSHAKE for key expansion and KMAC for message authentication. Both are derived from the Keccak sponge construction, which absorbs input blocks into a large permutation state and later squeezes output blocks from the same state. The permutation used here is Keccak–f[1600], which updates a 1600–bit internal state through 24 rounds of nonlinear and linear mixing.

The sponge construction is parameterized by a rate r and capacity c, with $r + c = 1600$. RCS uses the 256–bit security configuration, giving a rate of $r = 1088$ bits (136 bytes) and a capacity of $c = 512$ bits. Each call to the squeeze function yields up to 136 bytes before requiring another permutation of the Keccak state.

The cSHAKE variant provides domain separation through a function–name string and an optional customization string. In RCS, the master key, the optional user–defined `info` value and a fixed function–name string (`rcs256` or `rcs512`) are absorbed to initialize the state. The output stream is interpreted as a sequence of pseudo-random bytes from which RCS slices $(R + 1)$ independent 256–bit round keys. At least one additional permutation is applied before squeezing the MAC key to ensure independent derivation relative to the round–key material.

KMAC is a message authentication code built on cSHAKE, using a sponge–based absorb–then–squeeze interface. In RCS, KMAC authenticates a transcript that includes the associated data and its length encoding, the 32–byte nonce, the ciphertext, and a final encoding of the total processed–byte count. Under the standard PRF and EUF–CMA assumptions for sponge–based MACs with capacity $c = 512$, this construction provides strong binding of all inputs to the tag.

2.8 Positioning RCS as a Post-Quantum Successor to AES-GCM and ChaCha20-Poly1305

Authenticated encryption in current network protocols is dominated by two constructions: AES-GCM and ChaCha20-Poly1305. Both achieve high performance and broad hardware support, but their long-term viability in a post-quantum environment is limited by structural properties of their authentication mechanisms and by security bounds tied to classical number-theoretic assumptions.

AES-GCM combines AES in counter mode with the GHASH polynomial authenticator. GHASH is linear over $\mathbb{GF}(2^{128})$ and its security reduces to the hardness of forging evaluations of a fixed polynomial. This structure inherits quantum speedups via multi-evaluation and parallel quantum queries, reducing the effective tag security to the 64-bit range. In addition, the polynomial authentication layer is fundamentally tied to finite-field

multiplication, which is known to be susceptible to further algebraic and quantum-assisted attacks.

ChaCha20-Poly1305 inherits similar limitations because Poly1305 is also a polynomial MAC evaluated modulo a large integer, and its post-quantum strength is bounded by the algebraic degree and collision properties of the underlying polynomial evaluation. In both cases, the tags are 16 bytes, limiting the birthday-bound security of the MAC even in fully classical settings.

RCS avoids these structural weaknesses. The authentication layer is KMAC, constructed from the Keccak sponge with a 512-bit capacity, providing strong post-quantum collision and forgery resistance under generic attacks. The tag size (32 or 64 bytes, depending on the variant) raises the classical and post-quantum forgery bounds far above those achievable by 16-byte GCM or Poly1305 tags. The use of cSHAKE for domain-separated key expansion produces independent round keys and a MAC key without relying on algebraic key schedules or operations with number-theoretic structure. The encryption layer is a wide-block Rijndael permutation used in pure counter mode, eliminating dependence on finite-field multiplication and removing the structural assumptions that both GCM and Poly1305 rely on.

Operationally, RCS also preserves existing hardware acceleration investment. Systems equipped with AES-NI benefit from the Rijndael structure underlying RCS, as the S-box and linear layer align with the operations accelerated by AES-NI, allowing hardware-supported implementations without requiring new instructions. Deployments can therefore transition to post-quantum authenticated encryption without discarding existing hardware optimizations.

Taken together, these properties position RCS as a viable post-quantum successor to the mainstream AEAD constructions in current protocols. It maintains the operational profile of AES-CTR-based encryption while replacing polynomial MACs with a sponge-based authenticator that preserves security margins in both classical and quantum models. Importantly, the RCS 512-bit key option restores the effective post-quantum security level that 256-bit keys lose under Grover's algorithm, ensuring that the scheme retains its intended security strength even in the presence of quantum adversaries.

2.9 pseudo-code Specification

The following pseudo-code mirrors the behavior of `qsc_rcs_initialize` and `qsc_rcs_transform`.

Algorithm 1 RCS_INITIALIZE

Require: Context structure `ctx`, key parameters `params`, flag `encrypt`
Ensure: Initialized RCS state in `ctx`

// Load key parameters
$K \leftarrow$ `params.key`
$info \leftarrow$ `params.info`
$N \leftarrow$ `params.nonce`
// Initialize context
`ctx.encrypt` $\leftarrow$ `encrypt`
`ctx.nonce` $\leftarrow N$
`ctx.counter` $\leftarrow 0$
`ctx.ctype` $\leftarrow$ `params.ctype`
// Initialize cSHAKE with key, domain string, and optional info
$S \leftarrow$ CSHAKE_INIT(K, `rcs_name`, $info$)
// Derive round keys as a single squeezed byte stream
$rk\_bytes \leftarrow$ SPONGE_SQUEEZE(S, $(R+1) \cdot 32$)
for $i = 0$ **to** R **do**
    `ctx.RK[`i`]` $\leftarrow$ SLICE(rk_bytes, $32 \cdot i$, 32)
end for
// Derive MAC key
`ctx.mac_key` $\leftarrow$ SPONGE_SQUEEZE(S, `MAC_LEN`)
// Initialize KMAC state
KMAC_INIT(`ctx.mac`, `ctx.mac_key`)

SpongeSqueeze(S, rlen) denotes a contiguous squeeze of rlen bytes from the cSHAKE sponge, implemented internally by repeated calls to the rate-sized block function as in the reference code.

Algorithm 2 RCS_TRANSFORM

Require: Context structure `ctx`, output buffer `output`, input buffer `input`, length `length`, flag `finalize`

Ensure: On success, `output` contains ciphertext or plaintext and the function returns `true`

```
// Update processed byte counter
ctx.counter ← ctx.counter + length
if ctx.encrypt = true then
    // Encryption branch
    KMAC_UPDATE(ctx.mac, ctx.nonce)
    RCS_CTR_TRANSFORM(ctx, output, input, length)
    KMAC_UPDATE(ctx.mac, output[0..length-1])
    if finalize = true then
        KMAC_UPDATE(ctx.mac, ENCODE_LEN(ctx.counter))
        tag ← KMAC_FINAL(ctx.mac)
        copy tag into output[length..length+MAC_LEN-1]
    end if
    return true
else
    // Decryption branch
    KMAC_UPDATE(ctx.mac, ctx.nonce)
    KMAC_UPDATE(ctx.mac, input[0..length-1])
    if finalize = true then
        KMAC_UPDATE(ctx.mac, ENCODE_LEN(ctx.counter))
        code ← KMAC_FINAL(ctx.mac)
        // Select expected tag length from cipher type
        tagpos ← length
        if ctx.ctype = RCS256 then
            tagpos ← length
            ok ← VerifyEqual(code, input[tagpos ... tagpos + 32 − 1])
        else
            tagpos ← length
            ok ← VerifyEqual(code, input[tagpos ... tagpos + 64 − 1])
        end if
        if ok = 0 then
            // MAC check failed, skip decryption
            return false
        end if
    end if
    // Only reached if MAC check passed or finalize is false
    RCS_CTR_TRANSFORM(ctx, output, input, length)
    return true
end if
```

3 Preliminaries

This section introduces the notation, idealized models, and adversarial framework used throughout the security analysis. All definitions follow standard conventions in symmetric cryptography and AEAD security proofs.

3.1 Notation

A bit string X of length n is written $X \in \{0,1\}^n$, and $|X|$ denotes its length in bits. Concatenation is written $X \parallel Y$. The bitwise exclusive-or operator is denoted $X \oplus Y$. For a positive integer q, we write $[q] = \{1, 2, \ldots, q\}$.

Random Sampling. The notation $x \xleftarrow{\$} S$ denotes sampling a value x uniformly at random from a finite set S. The notation $x \xleftarrow{\$} \{0,1\}^n$ denotes choosing a uniform n-bit string.

Probability. For a probabilistic experiment Exp and event E, the probability that E occurs is written $\Pr[\mathsf{Exp} \Rightarrow E]$. All probabilities are taken over the randomness of the experiment and the adversary.

Negligible Functions. A function $\mu(\lambda)$ is negligible in the security parameter λ if for every polynomial $p(\cdot)$ there exists N such that $\mu(\lambda) < 1/p(\lambda)$ for all $\lambda > N$.

Encoding. Encoding. The MAC input uses fixed width little endian encodings. The associated data length is encoded as a 32 bit value, and the final byte count is encoded as a 64 bit value. We write $\texttt{encode\_len}(\cdot)$ for this fixed width encoding, with the understanding that the width depends on the field (32 bits for associated data length, 64 bits for the final processed byte count).

3.2 Underlying Primitives

The RCS construction relies on three cryptographic components: a wide-block Rijndael permutation, a cSHAKE-based key expansion function, and a KMAC-based message authentication function. Each is analyzed under a standard idealized model.

Rijndael-256 as a PRP. The wide-block Rijndael permutation is modeled as a pseudo-random permutation (PRP) over 256-bit blocks under independent round keys. Formally, for a key K and input $X \in \{0,1\}^{256}$, the function $E_K(X)$ is assumed to be indistinguishable from a uniformly chosen permutation over $\{0,1\}^{256}$ when keyed with independently distributed subkeys. This idealization is used in the confidentiality proof.

cSHAKE as a PRF. The key expansion function is modeled as a pseudo random function (PRF) from the master key, an optional info string, and a domain separated customization string to an unbounded pseudo-random output stream. Given the indifferentiability of the Keccak sponge from a random oracle, and the domain separation of the RCS customization strings, this model captures the claimed behavior of cSHAKE for key derivation and round-key expansion.

KMAC as a PRF and EUF–CMA MAC. The KMAC instance used by RCS is modeled both as:

1. a PRF keyed by a cSHAKE-derived MAC key for the purposes of confidentiality proofs involving hybrid transitions, and
2. an existentially unforgeable MAC (EUF–CMA) for the integrity and AEAD proofs.

The inputs to KMAC include domain-separated customization strings, ensuring that the MAC key is never correlated with the round keys generated during key expansion.

Sponge and Random-Oracle Idealization. Where appropriate, we use the indifferentiability of Keccak's sponge construction from a random oracle to justify the PRF behavior of both cSHAKE and KMAC, following the standard security arguments for Keccak-family XOFs.

3.3 Adversarial Model

We consider probabilistic polynomial-time (PPT) adversaries interacting with the RCS construction through oracle interfaces defined by the standard AEAD security experiments. Unless stated otherwise, all adversaries operate in the classical query model. Quantum-query considerations are addressed separately in the post-quantum subsection of the security analysis.

Adversary Classes. Throughout this work, we consider:

- IND–CPA adversaries with access to an encryption oracle,
- IND–CCA adversaries with access to both encryption and decryption oracles, where decryption returns $\perp$ on invalid tags,
- INT–CTXT adversaries attempting to forge a ciphertext–tag pair,
- adversaries attempting to exploit related-key or structural properties, modeled under the PRF idealization of cSHAKE.

Query Bounds. Let q_E and q_D denote the number of encryption and decryption queries, respectively, and let ℓ denote the maximum total message length across all queries. Advantage bounds throughout the proofs are expressed as functions of (q_E, q_D, ℓ) and the security parameter $\lambda \in \{256, 512\}$.

Oracle Semantics. The encryption oracle returns ciphertext and tag pairs produced by the real algorithm, while the decryption oracle outputs the decrypted plaintext if and only if the supplied tag is correct. Outputs are suppressed on tag failure, preventing side-channel oracles that could bias the adversary.

Nonce Conditions. All security definitions assume that nonces are never repeated for a given key. The security model does not provide guarantees under nonce reuse in counter mode, and this limitation is explicitly documented in the misuse analysis.

Post-Quantum Considerations. We adopt the standard square-root degradation model for symmetric primitives under quantum adversaries; reductions in the post-quantum analysis quantify the expected security loss in terms of $\sqrt{2^\lambda}$ generic search bounds.

4 Formal Specification of RCS

This section defines the RCS authenticated encryption scheme in mathematical terms. The description abstracts the engineering-level construction of Section 2 into a concise formal model suitable for security analysis.
Let $\lambda \in \{256, 512\}$ denote the security parameter corresponding to the key size. All bit strings and operations follow the notation established in the Preliminaries.

4.1 Syntax of the Scheme

RCS is an authenticated encryption with associated data (AEAD) scheme consisting of three algorithms:

$$\mathsf{RCS} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}).$$

Key Generation. A key K is a uniform bit string of length λ. Nonces are 256-bit strings and are supplied externally by the caller. RCS does not generate nonces internally.

$$K \xleftarrow{\$} \{0,1\}^{\lambda}, \qquad N \in \{0,1\}^{256}.$$

Encryption. Given a key K, nonce N, associated data $A \in \{0,1\}^*$, and plaintext $P \in \{0,1\}^*$, the encryption algorithm outputs a ciphertext C and tag T:

$$(C, T) \leftarrow \mathsf{Enc}(K, N, A, P).$$

Decryption. Given (K, N, A, C, T), the decryption algorithm outputs either the plaintext P or the failure symbol $\perp$:

$$P \leftarrow \mathsf{Dec}(K, N, A, C, T).$$

Decryption returns P if and only if the tag verifies correctly.

4.2 Encryption and Decryption Algorithms

We describe the algorithms using compact mathematical definitions that correspond to the pseudo-code in Section 2.6.

Round-Key and MAC-Key Derivation. Let:

$$(RK_0, RK_1, \ldots, RK_R) \leftarrow \mathsf{cSHAKE}(K, \mathsf{info}, d_{\mathsf{RCS}})$$

denote the cSHAKE output stream, domain separated by d_{RCS}, partitioned into (R + 1) independent 256-bit round keys followed by a MAC key, where the MAC key is obtained only after at least one additional permutation step on the sponge state. The number of rounds is:

$$R = \begin{cases} 22 & \text{for RCS–256,} \\ 30 & \text{for RCS–512.} \end{cases}$$

Counter-Mode Encryption. Counter-Mode Encryption. Define the initial counter block $CTR_0 = N$, and for each $i \geq 0$ let $CTR_i = (N + i) \bmod 2^{256}$, where the addition is on the 256-bit little-endian representation of N. For each block P_i of the plaintext,

$$KS_i = E_{RK}(CTR_i), \qquad C_i = P_i \oplus KS_i,$$

where E_{RK} is the Rijndael-256 round function using round keys $(RK_0, \ldots, RK_R)$.
The ciphertext C is the concatenation of all C_i.

Tag Generation. Let $\ell = |P|$ denote the plaintext length in bytes. The authentication tag is generated via KMAC as a function of a transcript that includes A, its encoded length, the nonce N, the ciphertext C, and a final encoded length value derived from the total processed bytes. In particular, we write this abstractly as

$$T = \mathrm{KMAC}(mkey,\, \tau(A, N, C)),$$

where $\tau(A, N, C)$ is the deterministic transcript corresponding to the sequence of `rcs_mac_update` and `rcs_mac_finalize` calls in the implementation.

Full Encryption Algorithm. In summary,

$$\mathsf{Enc}(K, N, A, P) = (C, T)$$

where C and T are computed as above.

Decryption Algorithm. To decrypt (C, T):

1. Recompute the MAC key mkey and round keys via the same cSHAKE call.
2. Compute
$$T' = \mathrm{KMAC}(\mathrm{mkey},\, \tau(A, N, C)),$$
where $\tau(A, N, C)$ is the deterministic transcript consisting of the associated data and its fixed width length encoding, the current 32 byte nonce, the ciphertext blocks processed by the transform, and the final encoded total byte count, exactly as realized by the sequence of `qsc_rcs_set_associated`, `rcs_mac_update`, and `rcs_mac_finalize` calls in the implementation.
3. If $T' \neq T$, return $\perp$.
4. Otherwise, derive $KS_i = E_{RK}(CTR_i)$ and recover $P_i = C_i \oplus KS_i$ for each block.

Correctness follows from the fact that both encryption and decryption use identical counter values and identical key-stream blocks.

4.3 Key and Nonce Requirements

RCS imposes the following requirements on its inputs:

- **Key Length.** Keys must be uniformly sampled from $\{0,1\}^{256}$ or $\{0,1\}^{512}$.
- **Nonce Length and Uniqueness.** The nonce must be a 256-bit string and *must not* repeat for a given key. RCS provides no security guarantees under nonce reuse.
- **Tag Length.** The tag length is fixed by the parameter set:
$$\mathsf{taglen} = \begin{cases} 32 \text{ bytes} & \text{(RCS–256)}, \\ 64 \text{ bytes} & \text{(RCS–512)}. \end{cases}$$
- **Associated Data.** Any string supplied as A is authenticated but not encrypted and must be provided in full before encryption or decryption proceeds.

These conditions reflect the operational requirements of the reference implementation and define the scope in which the security proofs apply.

5 Security Definitions

This section defines the confidentiality and integrity notions used in the analysis of RCS. All definitions follow standard AEAD security frameworks, including the IND–CPA, IND–CCA, and INT–CTXT notions. These definitions form the basis for the game-based proofs developed in Section 6.
Let $\lambda \in \{256, 512\}$ denote the security parameter corresponding to the key size. All adversaries are probabilistic polynomial-time (PPT) algorithms unless otherwise stated.

5.1 IND-CPA and IND-CCA Experiments

Confidentiality is expressed through chosen-plaintext (IND–CPA) and chosen-ciphertext (IND–CCA) indistinguishability experiments. Each experiment allows the adversary to interact with oracles that implement the AEAD interface of RCS.

IND–CPA Experiment. The IND–CPA experiment $\mathsf{Exp}^{\mathsf{ind\text{-}cpa}}_{RCS}(A)$ proceeds:

1. A key $K \xleftarrow{\$} \{0,1\}^\lambda$ is sampled.
2. The adversary A may adaptively query an encryption oracle:
$$\mathcal{O}_{\mathsf{enc}}(A, P_0, P_1, N)$$
where P_0, P_1 are equal-length plaintexts and N is a nonce chosen by A.
3. A hidden bit $b \xleftarrow{\$} \{0,1\}$ is sampled.
4. The oracle returns
$$(C, T) \leftarrow \mathsf{Enc}(K, N, A_{\mathsf{assoc}}, P_b),$$
where A_{assoc} denotes the associated data supplied by A.
5. A outputs a bit b'.

The adversary's advantage is:

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{RCS}(A) = \left|\Pr[b' = b] - \tfrac{1}{2}\right|.$$

IND–CCA Experiment. The IND–CCA experiment extends the IND–CPA experiment by allowing access to a decryption oracle, with tag-verification failures suppressed:

1. A key K is generated as above.
2. The encryption oracle is identical to the IND–CPA experiment.
3. The adversary may also query a decryption oracle:
$$\mathcal{O}_{\mathsf{dec}}(A, N, A_{\mathsf{assoc}}, C, T),$$
which returns $\mathsf{Dec}(K, N, A_{\mathsf{assoc}}, C, T)$ unless (C, T) was previously returned by the encryption oracle.
4. Adversary outputs a guess b'.

The IND–CCA advantage is:

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{RCS}(A) = \left|\Pr[b' = b] - \tfrac{1}{2}\right|.$$

These definitions assume that nonces never repeat under the same key, since counter-mode encryption does not provide confidentiality under nonce reuse.

5.2 Ciphertext Integrity (INT-CTXT)

Ciphertext integrity captures the adversary's inability to produce a valid ciphertext–tag pair not previously output by the encryption oracle. The INT–CTXT experiment $\mathsf{Exp}^{\mathsf{int\text{-}ctxt}}_{RCS}(A)$ is defined:

1. A key K is chosen uniformly at random.
2. A may query an encryption oracle $\mathcal{O}_{\mathsf{enc}}(A, P, N)$, receiving (C, T) for chosen (P, N).

3. Eventually A outputs a tuple (N^*, A^*, C^*, T^*).
4. A wins if:
   (a) (C^*, T^*) was not the output of any encryption query,
   (b) $\mathsf{Dec}(K, N^*, A^*, C^*, T^*) \neq \perp$.

The integrity advantage is:

$$\mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{RCS}(A) = \Pr\left[\mathsf{Exp}^{\mathsf{int\text{-}ctxt}}_{RCS}(A) = 1\right].$$

This definition captures forgeries, truncation attempts, block reordering, and bit-flip attacks, since any modification of C results in tag failure with all but negligible probability under the EUF–CMA security of KMAC.

5.3 Related-Key and Misuse Notions

RCS uses cSHAKE to derive independently distributed round keys from the master key, nonce, and domain-separated customization strings. This prevents the algebraic and linear relations exploited by classical related-key attacks on AES-like key schedules.

Related-Key Security. For completeness, we consider adversaries whose queries may involve keys drawn from a related-key oracle:

$$K' = f(K),$$

where f is an efficiently computable key-derivation function chosen by the adversary.
In the RCS model, all round keys and MAC keys are derived from cSHAKE under independent domain-separated inputs; under the PRF assumption for cSHAKE, related-key attacks reduce to distinguishing cSHAKE from a random oracle, and are therefore out of scope for the permutation-based part of the cipher.

Misuse Considerations. This work does *not* analyze RCS under nonce reuse or nonce-misuse scenarios. In counter mode, reuse of a nonce immediately leads to key-stream reuse, enabling trivial disclosure of plaintext relations:

$$C_1 \oplus C_2 = P_1 \oplus P_2.$$

Thus, RCS is *not* an MRAE (misuse-resistant authenticated encryption) scheme. Nonce uniqueness is mandatory for confidentiality guarantees.

5.4 Adversarial Advantages and Parameters

Security bounds in the proofs are expressed using standard advantage notation:

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{RCS}(A), \qquad \mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{RCS}(A), \qquad \mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{RCS}(A).$$

These depend on the following parameters:

- q_E: number of encryption queries,
- q_D: number of decryption queries (IND–CCA only),
- ℓ: total length of all encrypted messages,

- λ: security parameter (256 or 512),
- R: number of rounds (22 or 30),
- taglen: tag length (32 or 64 bytes).

All advantages are functions of $(q_E, q_D, \ell, \lambda)$ and are negligible in λ for the intended usage of RCS.

6 Provable Security Analysis

In this section we relate the security of RCS to the assumed security of its underlying components: the Rijndael 256 permutation, the cSHAKE based key expansion, and the KMAC based authentication layer. All adversaries are probabilistic polynomial time unless otherwise specified, and all advantages are defined as in Section 5.

6.1 Confidentiality Reduction (IND-CPA)

We prove that RCS achieves IND CPA confidentiality under unique nonce usage, assuming that cSHAKE behaves as a PRF and Rijndael 256 behaves as a PRP under independent round keys. The proof is given as a sequence of hybrid games.

Theorem 6.1 (IND CPA security of RCS). Let A be any IND CPA adversary against RCS that makes at most q_E encryption queries of total length at most ℓ bits. Then there exist adversaries B_{prf} and B_{prp} such that

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{RCS}(A) \le \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{cSHAKE}}(B_{\mathsf{prf}}) + \mathsf{Adv}^{\mathsf{prp}}_{\mathsf{Rijndael}}(B_{\mathsf{prp}}) + \varepsilon_{\mathsf{ctr}}(q_E, \ell),$$

where $\varepsilon_{\mathsf{ctr}}$ is negligible in the key size and accounts for the probability of counter collisions under unique nonces.

Proof. We consider a sequence of hybrids $H_0, \ldots, H_3$ and bound the distinguishing advantage between successive games.

Game H_0 (Real RCS). This is the real IND CPA experiment for RCS. The key K is sampled uniformly. For each encryption query (N, A, P_0, P_1) of equal-length messages, the challenger samples a hidden bit b, derives the MAC key and round keys from cSHAKE, derives key-stream blocks by applying the Rijndael 256 permutation in counter mode, and returns the ciphertext and tag for P_b.
By definition,

$$\mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{RCS}(A) = \left|\Pr[H_0(A) = 1] - \tfrac{1}{2}\right|.$$

Game H_1 (Replace cSHAKE by a PRF). In H_1, the challenger replaces the cSHAKE based key expansion by a family of independent uniformly random functions. Concretely, for each key K and nonce N the challenger programs an oracle that, when first queried on (K, N, info), samples:

$$\mathsf{mkey}, RK_0, \ldots, RK_R \xleftarrow{\$} \{0,1\}^{\lambda} \times (\{0,1\}^{256})^{R+1},$$

and stores them in a table. Future queries with the same (K, N, info) return the same tuple.
The view of A in H_1 is identical to its view in H_0 unless A can distinguish cSHAKE from a random function. Thus there exists an adversary B_{prf} such that

$$|\Pr[H_0(A) = 1] - \Pr[H_1(A) = 1]| \le \mathsf{Adv}^{\mathsf{prf}}_{\mathsf{cSHAKE}}(B_{\mathsf{prf}}).$$

Game H_2 (Replace Rijndael by a random permutation). In H_2, the challenger replaces the Rijndael 256 permutation E_{RK} by an ideal random permutation π over $\{0,1\}^{256}$ keyed implicitly by the table of round keys. For each block encryption, the challenger evaluates π on the current counter value and uses the result as the key-stream block.
By the PRP assumption on Rijndael 256 there exists an adversary B_{prp} such that

$$|\Pr[H_1(A) = 1] - \Pr[H_2(A) = 1]| \leq \mathsf{Adv}^{\mathsf{prp}}_{\mathsf{Rijndael}}(B_{\mathsf{prp}}).$$

Game H_3 (Replace the PRP based key-stream by a random function). In counter mode with unique nonces, each query to the encryption oracle uses a fresh sequence of counter blocks that never repeats across the experiment. When the block cipher is replaced by a random permutation, the map

$$(K, N, i) \mapsto \pi(CTR_i)$$

is indistinguishable from a random function from the domain of counters to $\{0,1\}^{256}$, except with probability bounded by the chance that two counters collide. Under unique nonces and bounded message lengths, this collision probability is at most $\varepsilon_{\mathsf{ctr}}(q_E, \ell)$ and is negligible for the intended parameters.
In H_3 we therefore model the key-stream as sampled uniformly and independently for each distinct counter value. As a result, for any fixed choice of b, the ciphertexts in H_3 are one time pad encryptions of P_b under fresh random key-stream blocks. Hence A has no information about b beyond random guessing, and

$$\Pr[H_3(A) = 1] = \tfrac{1}{2}.$$

Bounding the overall advantage. By a telescoping sum over the four hybrids,

$$\begin{aligned}\left|\Pr[H_0(A) = 1] - \tfrac{1}{2}\right| \leq\ & |\Pr[H_0(A) = 1] - \Pr[H_1(A) = 1]| \\ & + |\Pr[H_1(A) = 1] - \Pr[H_2(A) = 1]| \\ & + |\Pr[H_2(A) = 1] - \Pr[H_3(A) = 1]|\,.\end{aligned}$$

Substituting the bounds for each transition yields the stated inequality. □

6.2 Integrity Reduction (INT-CTXT)

We now show that any successful ciphertext forgery against RCS can be converted into a forgery against the underlying KMAC instance. The reduction relies on the fact that the tag input is fully determined by the associated data and its encoded length, the nonce, the ciphertext, and the final encoded length value produced by the MAC finalization.

Theorem 6.2 (INT CTXT security of RCS). Let A be any INT CTXT adversary against RCS making at most q_E encryption queries. Then there exists an adversary B_{mac} such that

$$\mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{RCS}(A) \leq \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{KMAC}}(B_{\mathsf{mac}}),$$

where $\mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{KMAC}}$ denotes the standard existential unforgeability advantage for KMAC.

Proof sketch. We construct B_{mac} that uses A as a subroutine and interacts with a KMAC oracle. The goal of B_{mac} is to output a fresh input string X^* together with a valid tag T^* such that (X^*, T^*) was not previously returned by the oracle.

Simulation of RCS for A. The challenger for the MAC game provides B_{mac} with oracle access to either a real KMAC under a hidden key mkey or a random function. The adversary B_{mac} simulates the RCS encryption oracle for A as follows:

1. For each encryption query (N, A, P) from A, B_{mac} samples a fresh key-stream deterministically using the public Rijndael permutation and the counter construction, which does not require knowledge of mkey.
2. It computes $C = P \oplus KS$ blockwise and forms the MAC input $X = A \parallel C \parallel \mathrm{encode}_{64}(|P|)$.
3. It queries its KMAC oracle on X to obtain T and returns (C, T) to A.

Since the simulation uses the real RCS key-stream generation and delegates tag computation to the KMAC oracle, the view of A is identical to that in the real INT CTXT experiment when the oracle is real KMAC.

Extracting a forgery. Eventually A outputs a candidate forgery (N^*, A^*, C^*, T^*) that satisfies the INT CTXT winning conditions: it was not obtained from the encryption oracle and it verifies under RCS. Verification computes

$$X^* = A^* \parallel C^* \parallel \mathrm{encode}_{64}(|P^*|),$$

for the implied plaintext length, and accepts if and only if

$$\mathsf{KMAC}(\mathsf{mkey}, X^*) = T^*.$$

If A succeeds, then B_{mac} outputs (X^*, T^*) as its MAC forgery. By construction, (X^*, T^*) was never returned by the MAC oracle during the simulation of encryption, because any such pair would correspond to an earlier ciphertext returned to A. Thus any successful INT CTXT forgery yields a valid MAC forgery.
The probability that B_{mac} wins in the MAC game is at least the probability that A wins in the INT CTXT game, up to negligible differences due to failure events of the key-stream simulation. This gives the claimed bound. □

6.3 AEAD Security (IND-CCA)

We now argue that the combination of IND CPA confidentiality and INT CTXT integrity yields IND CCA authenticated encryption for RCS, provided that the decryption oracle never returns plaintext on invalid tags. This follows the standard Encrypt then MAC composition paradigm.

Theorem 6.3 (IND CCA security of RCS). Suppose RCS is IND CPA secure and INT CTXT secure as established above. Then for any IND CCA adversary A against RCS there exist adversaries B_{cpa} and B_{int} such that

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{RCS}(A) \le \mathsf{Adv}^{\mathsf{ind\text{-}cpa}}_{RCS}(B_{\mathsf{cpa}}) + \mathsf{Adv}^{\mathsf{int\text{-}ctxt}}_{RCS}(B_{\mathsf{int}}).$$

Proof sketch. The result is an application of the generic theorem that IND CPA security combined with ciphertext integrity yields IND CCA security for Encrypt then MAC schemes, under the condition that decryption rejects before revealing any plaintext when tag verification fails.
We outline the reduction. Let A be an IND CCA adversary for RCS. We construct B_{cpa} that runs A and simulates the IND CCA environment using only access to an IND CPA encryption oracle and an internal INT CTXT adversary B_{int}.

- The encryption oracle in the IND CCA game is simulated directly by the IND CPA oracle, since both return real ciphertexts and tags.
- The decryption oracle must reject any query whose tag does not verify. Any decryption query that would be accepted corresponds to a valid ciphertext tag pair. If such a pair did not come from the encryption oracle, it would constitute an INT CTXT forgery.

Thus either A distinguishes the IND CCA game from IND CPA, which yields a distinguishing advantage for B_{cpa}, or A manages to exploit the decryption oracle in a way that implies an INT CTXT forgery, yielding advantage to B_{int}. Combining these cases gives the inequality above. □

6.4 Post-Quantum Considerations

We briefly summarize how the above bounds degrade in the presence of a quantum adversary endowed with superposition access to the underlying primitives. The high level impact follows the standard square root rule for generic quantum search.

Symmetric Key Length. For a block cipher or PRF with key size λ, the best known generic quantum attack achieves complexity approximately $2^{\lambda/2}$. In the context of RCS, this affects both Rijndael 256 and the cSHAKE based key expansion. Thus:

- RCS 256, with $\lambda = 256$, offers roughly 2^{128} post quantum security against exhaustive key search.
- RCS 512, with $\lambda = 512$, offers roughly 2^{256} post quantum security under the same model.

Sponge Based Components. The security of cSHAKE and KMAC depends on the capacity of the underlying Keccak sponge. Quantum attacks can reduce the effective collision and preimage security exponents by a factor of two, but for the capacity choices used by RCS the resulting bounds remain above the target key strengths. The indifferentiability based arguments for sponge constructions carry over to the quantum setting with similar square root losses.

Resulting Bounds. The classical bounds from Theorems 6.1 and 6.2 can therefore be interpreted in the quantum setting by replacing each term of size approximately $2^{-\lambda}$ with a term of size approximately $2^{-\lambda/2}$, plus the usual factors in q_E, q_D, and ℓ. For both parameter sets this yields margins that remain well within the intended security levels, provided the underlying primitives retain their expected post quantum properties.

7 Cryptanalytic Evaluation of RCS

This section surveys the resistance of RCS to classical cryptanalytic techniques applied to the underlying wide-block Rijndael permutation and to the sponge-based components used for key expansion and authentication. The goal is not to provide new attacks but to situate the chosen parameters within the landscape of known results, and to justify that the 22-round (RCS–256) and 30-round (RCS–512) configurations sit comfortably beyond current analytical frontiers.

7.1 Wide-Block Rijndael Structure

The core permutation of RCS is a 256-bit Rijndael instance operating on two independent 128-bit halves. Each half is arranged as a 4×4 byte array, and each round applies the standard sequence of Rijndael operations: byte-wise S-box substitution, row rotations, column mixing over $\mathsf{GF}(2^8)$, and addition of a 256-bit round key derived from cSHAKE. The two halves are processed in lockstep, and the round keys span the full 256-bit state. Relative to AES, which fixes a 128-bit state and uses 10, 12, or 14 rounds, Rijndael-256 increases the diffusion radius and the size of the permutation space. The wider state raises the birthday bound for generic attacks and alters the structure of possible differential and linear trails. In particular, each application of the linear mixing layer affects a larger number of bytes per round, increasing the minimum number of active S-boxes over any multi-round trajectory.
In RCS, the 256-bit permutation is used exclusively in counter mode: the state input for each block is a counter derived from the 32-byte nonce and an internal block index, and the output is treated as key-stream. This usage pattern eliminates complex feedback paths that appear in certain block-cipher modes and simplifies the analysis to that of a key-stream generator built from a wide-block PRP.

7.2 Differential and Linear Cryptanalysis

Differential and linear cryptanalysis measure how input differences and linear approximations propagate across the S-box and linear layers of the permutation. The security of Rijndael-256 under these techniques is governed by the number of active S-boxes and the probability of the best differential or linear trails.

Existing analyses of Rijndael-type ciphers show that:

- nontrivial differentials and linear approximations are confined to significantly reduced round counts,
- the minimum number of active S-boxes grows rapidly with the number of rounds, leading to exponential decay in trail probabilities,
- the transition from reduced-round distinguishers to full-round attacks requires a substantial increase in complexity that quickly exceeds practical limits.

RCS sets the round counts to

$$R_{256} = 22, \qquad R_{512} = 30,$$

which are substantially higher than the reduced-round regimes targeted by published attacks on Rijndael-256 and AES-like instances. The design rationale is to keep the best known differential and linear distinguishers at a distance of several rounds from the operational configurations, and to ensure that any extension of existing trails would require work factors comparable to or greater than exhaustive key search.
From the perspective of the stream cipher usage, the relevant question is whether an adversary can distinguish the key-stream from random by exploiting correlations in the underlying permutation. With 22 and 30 rounds and the widened state, the probability of mounting such a distinguisher with complexity below 2^{λ} is negligible for the intended parameters.

7.3 Algebraic and Structural Attacks

Algebraic cryptanalysis aims to represent the cipher as a system of Boolean or multivariate equations and to solve or approximate this system more efficiently than brute force. For

SPN ciphers like Rijndael, two aspects are especially relevant: the algebraic degree of the round functions and the growth of that degree as rounds are iterated.
For Rijndael-256, the nonlinear S-box and linear mixing layer cause the algebraic degree of the overall permutation to grow quickly with each round. After a modest number of rounds the algebraic degree approaches its maximum, and the resulting systems for full-round instances become too complex for current equation-solving techniques to exploit in practice.
Structural attacks such as slide attacks, meet-in-the-middle techniques, and invariant subspace attacks typically require regularities in the round structure or key schedule. In RCS, each round key is derived independently from the cSHAKE-based key schedule, which breaks the kind of linear recurrences and self-similar patterns that underlie many structural cryptanalytic constructions.
The choice of 22 and 30 rounds ensures that any algebraic representation of the full permutation would involve a very high degree and a large number of variables, placing known algebraic methods well beyond feasible bounds for both parameter sets.

7.4 Related-Key and Key-Schedule Attacks

Classical Rijndael key schedules are known to admit related-key patterns and structures that enable certain attacks on reduced-round variants. These attacks rely on linear recurrences in the key expansion, which create predictable relations between round keys derived from related master keys.
RCS replaces the classical key schedule with a cSHAKE-based expansion that treats the master key, optional information string, and a fixed domain-separation string as input to a Keccak sponge. The resulting output stream is then sliced into a MAC key and a sequence of independent 256-bit round keys. Under the PRF assumption for cSHAKE, the round keys are indistinguishable from independently uniform random values, even when master keys are related by efficiently computable transformations.
From the point of view of related-key security, any adversary that exploits relations between round keys across different master keys would induce a distinguishing attack on cSHAKE. The formal model in Section 5 therefore treats related-key attacks on the permutation as subsumed by the PRF security of the key-expansion function. Within this model, standard related-key and slide attacks against Rijndael do not apply to RCS.

7.5 Summary of Cryptanalytic Margins

The cryptanalytic evaluation can be summarized as follows:

- **Differential and linear attacks.** Known distinguishers and key-recovery attacks on Rijndael-type ciphers reach only significantly reduced round counts and do not extend to the 22- and 30-round configurations used by RCS. The widened 256-bit state further increases the diffusion per round and the number of active S-boxes.
- **Algebraic and structural attacks.** Algebraic degree growth and the absence of exploitable structure at the chosen round counts place RCS beyond the reach of current equation-based and structural techniques, given realistic resource bounds.
- **Related-key attacks.** The cSHAKE-based key schedule yields round keys that behave as independent random values under standard assumptions, eliminating the linear relations exploited by classical related-key attacks on Rijndael.
- **Overall margin.** For both RCS–256 and RCS–512, the best known attacks against the underlying permutation remain well below the configured round counts and do not offer any advantage over generic key search at the intended security levels. Within

the adopted models for Rijndael, cSHAKE, and KMAC, RCS therefore maintains conservative cryptanalytic margins for its intended deployment scenarios.

These conclusions are consistent with the reductionist view developed in Section 6: the security of RCS reduces to the PRP behavior of the wide-block Rijndael permutation under independent keys and the PRF and EUF–CMA properties of the Keccak-based components, all of which enjoy substantial analytical support in the existing literature.

Table 3 summarizes the effective security margins of RCS against the primary attack families considered in the cryptanalytic evaluation. The margins are expressed in rounds and represent the gap between the best published reduced-round analyses of Rijndael-256 and the full-round configurations used in RCS-256 (22 rounds) and RCS-512 (30 rounds).

Table 3: Cryptanalytic margins for RCS.

Attack	**Best Known**	**256 Margin**	**512 Margin**
Differential	$\leq$ 12–13 rounds	9–10	17–18
Linear	$\leq$ 11–12 rounds	10–11	18–19
Algebraic	$\sim$10 rounds	12	20
Related-key	n/a	n/a	n/a
Structural/slide	n/a	n/a	n/a

8 Misuse Resistance and Operational Limits

This section addresses the operational assumptions under which the security guarantees of RCS hold, with particular emphasis on nonce handling in counter mode and limitations that arise when these assumptions are violated. These considerations define the boundary between the provable guarantees established in previous sections and practical misuse scenarios that fall outside the formal model.

8.1 Nonce Reuse and its Consequences

RCS is *not* misuse resistant with respect to nonce reuse. As a counter mode construction, RCS requires that the 256-bit nonce N be *unique for each encryption under a fixed key.* If a nonce is ever reused with the same key, the confidentiality of both messages encrypted under that nonce collapses immediately.
Let P_1 and P_2 be two plaintexts encrypted under the same key–nonce pair, and let KS_i denote the key-stream blocks derived from the fixed counter sequence associated with that nonce. The ciphertexts satisfy

$$C_1 = P_1 \oplus KS, \qquad C_2 = P_2 \oplus KS,$$

implying

$$C_1 \oplus C_2 = P_1 \oplus P_2.$$

Thus an adversary observing both ciphertexts learns the XOR of the plaintexts, which generally suffices to recover both messages whenever either plaintext possesses known structure or redundancy. This attack is independent of the number of rounds of the permutation and does not rely on any weakness in Rijndael or cSHAKE.
Nonce reuse therefore constitutes a *catastrophic* confidentiality failure. RCS must be deployed with strict nonce management, ensuring that no nonce–key pair is ever repeated. This requirement is explicitly assumed in all security definitions and reductions.

8.2 Modeling Assumptions and Out-of-Scope Attacks

The formal analysis of RCS relies on a number of explicit modeling assumptions that bound the adversarial capabilities considered. The following scenarios fall outside the guarantees provided by the proofs:

Incorrect or Predictable Nonce Generation. If nonces are generated deterministically without safeguarding uniqueness, or if an adversary can predict or influence nonce values, then the confidentiality guarantees do not hold. RCS does not provide nonce-misuse-resistant encryption, forward secrecy under reuse, or synthetic IV behavior.

Weak or Compromised Keys. The analysis assumes that keys are sampled uniformly from $\{0,1\}^{\lambda}$ and remain secret for the duration of their use. If the master key is poorly generated, predictable, or exposed to the adversary, the security guarantees are void.

Side-Channel Leakage. The proofs treat all underlying primitives as black-box idealizations and do not cover timing, power, cache, electromagnetic, or fault-injection side channels. Although the reference implementation is written to avoid obvious timing leaks, formal side-channel resistance is out of scope.

State Rollback or Incomplete Zeroization. The model assumes correct implementation behavior in which the internal Keccak state, round keys, counters, and MAC keys are destroyed at the end of an operation. If an implementation fails to erase state or allows state rollback (e.g., via persistence or virtualization artifacts), confidentiality or integrity may be compromised.

Associated-Data Misuse. Associated data is authenticated but not encrypted. The model assumes that the caller supplies all associated data before encryption or decryption, and that the associated data is treated consistently across both operations. Misuse of associated data invariants is outside the formal guarantees.

8.3 Reduced-Rounds Configuration

The reference implementation of RCS supports an optional compile-time reduced-rounds configuration, enabled via the macro `QSC_RCS_REDUCED_ROUNDS`. When this option is selected, the wide-block Rijndael permutation is instantiated with 14 rounds for the RCS–256 parameter set and 21 rounds for RCS–512, instead of the standard 22 and 30 rounds, respectively. In addition, the KMAC authentication component is configured to use a reduced-round Keccak-f[1600] permutation with 12 rounds per permutation call, rather than the full 24-round instance.

The motivation for this configuration is performance optimization in constrained or latency-sensitive environments, while preserving a cryptographically sound structure relative to the design properties of RCS. In particular, RCS does not employ the classical Rijndael key schedule. All round keys are derived independently via a cSHAKE-based expansion, eliminating the linear recurrences and related-key structures that underpin many reduced-round attacks against AES-style constructions, including boomerang-type, rectangle, and differential switching attacks that rely on key schedule regularity. Under the PRF assumption for cSHAKE, the reduced-round Rijndael permutation is therefore analyzed as a wide-block substitution-permutation network with independent round keys, significantly altering the applicability of known reduced-round cryptanalytic techniques.

For the authentication layer, the reduced-round KMAC configuration leverages the substantial security margin of the Keccak-f[1600] permutation. No known cryptanalytic

results substantially weaken the security of Keccak when the number of rounds is reduced from 24 to 12 in the context of message authentication, particularly for usage models in which tags are verified immediately after transmission and are not relied upon for long-term cryptanalytic exposure. The authenticated transcript in RCS is unchanged under this option, and full domain separation is preserved.
This reduced-rounds configuration is not intended to replace the standard parameter sets analyzed elsewhere in this paper. Rather, it represents an explicit performance and security tradeoff made available to implementers who understand their operational threat model. Deployments requiring maximal conservatism or long-term security guarantees should use the full-round configurations analyzed in Sections 6 and 7. The formal security reductions in this work apply directly to the standard configuration, while the reduced-round variant rests on the same structural assumptions with reduced quantitative margins.

Out-of-Scope Cryptanalytic Models. The analysis does not address:

- attacks assuming related-message structures across multiple keys,
- multi-user or multi-target amplification attacks beyond standard bounds,
- attacks exploiting physical co-location or shared hardware behavior,
- deliberate weakening of cSHAKE customization parameters.

Within the formal scope defined by the AEAD syntax, unique nonce usage, correct implementation of cSHAKE and KMAC domain separation, and the PRP/PRF properties of the underlying primitives, the RCS construction achieves the confidentiality and integrity guarantees established in the preceding sections. The conditions listed here specify the assumptions that must be upheld by any correct implementation or deployment of RCS.

9 Implementation Conformance and Side Channels

This section relates the formal RCS model to the reference implementation and records the assumptions required for side channel resistance and randomness quality. The intent is to ensure that the deployed code actually realizes the scheme analyzed in the previous sections.

9.1 Mapping Between Model and Reference Code

The public interface in `rcs.h` corresponds directly to the AEAD syntax

$$(K, N, A, P) \mapsto (C, T)$$

formalized in Sections 3 and 4. The main structures and functions align with the model as follows.

State and Parameters. The structure `qsc_rcs_state` contains:

- the cipher type flag, which selects RCS-256 or RCS-512 and fixes the key size, tag size and round count,
- the round key array, which stores $(R + 1)$ 256-bit round keys used by the Rijndael permutation,
- the Keccak sponge state used for cSHAKE and KMAC operations,
- the stored nonce, a 32 byte buffer that represents N,

- an internal byte counter used to derive successive counter blocks,
- mode flags that indicate encryption or decryption.

These fields implement the abstract state variables specified in the formal definition of RCS.

Initialization and Key Expansion. The function `qsc_rcs_initialize` instantiates the scheme for a given key, nonce and optional information string. It performs the following steps:

1. chooses the correct parameter set (RCS-256 or RCS-512) based on the provided key size,
2. initializes the cSHAKE instance with the master key, the optional info tweak string, and the fixed RCS domain string,
3. squeezes the MAC key and then the sequence of round keys from the cSHAKE output stream,
4. copies the nonce into the state and initializes the internal counter.

This implements the key expansion procedure of Section 4.2, in which

$$(\mathsf{mkey}, RK_0, \ldots, RK_R)$$

are derived from cSHAKE under a fixed domain string.

Transform Functions. The functions `qsc_rcs_transform` and `qsc_rcs_extended_transform` implement the encryption and decryption algorithms described in Section 4.

They:

- construct counter blocks from the stored nonce and internal counter,
- apply the Rijndael 256 round function to obtain key-stream blocks,
- XOR key-stream blocks with input blocks to produce output blocks,
- absorb associated data, ciphertext and length into the KMAC state,
- output or verify the authentication tag.

When the mode flag selects encryption, the function produces (C, T) from P. When the mode flag selects decryption, the function recomputes the tag, compares it in constant time, and only then releases the recovered plaintext. This behavior matches the Encrypt then MAC model required by the IND-CCA reduction.

9.2 Constant Time Behavior and Leakage

The security proofs treat RCS as an idealized black box. To approach this model in practice, the implementation must avoid control flow and memory access patterns that depend on secret data.

At a minimum, the following conditions should hold:

- **Key schedule.** The cSHAKE based key expansion runs on fixed length inputs and uses only public loop bounds. The number of calls to the Keccak permutation and the memory access pattern within the sponge do not depend on the key bits.

- **Permutation rounds.** The Rijndael 256 permutation applies a fixed sequence of S-box lookups, row shifts and column mixes for each round. The round count and order of operations are independent of key and data values.
- **Counter mode.** Counter increment operations act on public counters and do not branch based on secret bits. The mapping from block index to counter value is deterministic and public.
- **Tag comparison.** Tag verification uses a constant time comparison that scans every byte and aggregates differences without early exit. No timing variation should reveal whether a prefix of the tag matches.
- **Error handling.** On tag failure, the implementation should avoid emitting partial plaintext or performing data dependent cleanup. The time to reject invalid tags should depend only on public parameters such as message length.

These properties do not guarantee resistance to all microarchitectural attacks, but they are necessary preconditions for any realistic constant time implementation. Deployment in high assurance environments may require additional hardening against cache, branch predictor and speculative execution side channels.

9.3 Random Number Generation Requirements

The formal model assumes that both keys and nonces are generated in a way that prevents adversarial prediction and reuse. This implies the following operational requirements.

Key Generation. Master keys must be generated from a cryptographically secure random number generator that provides at least λ bits of entropy. Keys must not be derived from passwords or other low entropy sources unless they are first processed by a suitable key derivation function whose security matches or exceeds that of RCS.

Nonce Generation and Management. Nonces must be unique for each encryption under a given key. Acceptable strategies include:

- a strictly increasing counter per key, stored in non-volatile memory and updated atomically for each encryption,
- a random nonce generator with collision probability bounded well below the security level, combined with rejection or retry on detected reuse,
- a hybrid scheme that combines a per-device identifier and a local counter.

Reusing a nonce under the same key causes key-stream reuse, which breaks confidentiality as discussed in Section 7. Implementations must therefore treat nonce uniqueness as a hard requirement, not as a best effort property.

Associated Randomness. If RCS is composed with higher level protocols that derive keys or nonces from other primitives (such as KDFs or key exchange mechanisms), those components must provide security at least comparable to the RCS parameter set in use. The formal results in this paper do not cover failures in upstream randomness sources. Under these conditions, the behavior of the deployed implementation conforms to the assumptions made in the provable security analysis.

10 Conclusion

The analysis presented in this paper establishes a rigorous and implementation aligned security foundation for the RCS authenticated cipher stream. By connecting the engineering specification directly to a formal model and by applying standard game based reasoning, we obtain confidentiality and integrity guarantees that reduce cleanly to the well studied properties of the underlying Rijndael and Keccak primitives.

10.1 Summary of Results

We summarize the principal contributions and findings:

- **Formalization of the Scheme.** The RCS construction was defined precisely through an engineering level specification and a mathematical abstraction that captures the behavior of the reference implementation. This provides a single authoritative model for analysis and future standardization.
- **Confidentiality Proof.** Through a four step hybrid argument, we reduced IND CPA security of RCS to the PRF security of cSHAKE and the PRP security of the 256 bit Rijndael permutation under independently derived round keys. Under unique nonce usage, the resulting key-stream is indistinguishable from uniformly random strings.
- **Integrity Proof.** Ciphertext integrity was shown to reduce directly to the EUF–CMA security of the KMAC function applied to the associated data and its length, the nonce, the ciphertext and the final encoded length value that summarizes the processed bytes. Any successful forgery against RCS yields a forgery against KMAC.
- **AEAD Composition.** By combining confidentiality and integrity, and using the fact that RCS rejects invalid tags before releasing plaintext, we proved IND CCA security for the overall AEAD construction following the Encrypt then MAC paradigm.
- **Cryptanalytic Margins.** We reviewed differential, linear, algebraic and related key attacks on Rijndael type ciphers, and concluded that the 22 round (RCS 256) and 30 round (RCS 512) configurations remain well beyond the reach of published techniques. The use of cSHAKE derived independent round keys eliminates key schedule based attacks.
- **Post Quantum Considerations.** We argued that the classical bounds degrade gracefully under quantum query models, preserving roughly 2^{128} and 2^{256} security for RCS 256 and RCS 512, respectively, in line with the usual square root attack heuristics.

Together, these results show that RCS is a conservative, transparent and well structured AEAD primitive whose security reduces to well understood assumptions on widely analyzed cryptographic components.

10.2 Limitations and Future Work

Although the analysis covers the core security claims of the RCS design, certain limitations remain and motivate possible refinement:

- **Nonce Misuse.** RCS does not provide misuse resistance under nonce reuse. While this is inherent to counter mode designs, formalizing RCS variants with synthetic IV or deterministic nonce generation could mitigate operational risks.

- **Side Channel Resistance.** The proofs treat the permutation and sponge components as idealized black boxes. Further work could include formal leakage models, masked implementations, or provable bounds under bounded leakage assumptions.
- **Reduced Round Analysis.** Although the chosen round counts lie well above known cryptanalytic thresholds, exploring higher fidelity bounds, automated trail search, and algebraic degree analyses specific to the 256 bit Rijndael variant could provide quantitative insight into long term safety margins.
- **Quantum Adversary Models.** A more complete treatment of cSHAKE and Rijndael under quantum superposition queries, or the implications of quantum chosen ciphertext attacks, may refine the understanding of post quantum behavior beyond the square root heuristic.
- **Multi User and Compositional Security.** Extending the proofs to multi user settings, key rotation processes, or compositional frameworks for large scale systems may support integration of RCS into complex protocols.

RCS demonstrates that combining a wide block Rijndael permutation with Keccak based key derivation and authentication yields a simple and analyzable AEAD primitive with conservative margins and clean provable guarantees. Further refinement of the model and deeper cryptanalytic investigation may strengthen its suitability for standardization and high assurance deployment.

References

1. Daemen, J., Rijmen, V. *The Design of Rijndael: AES, the Advanced Encryption Standard.* Springer, 2002. Available at: https://doi.org/10.1007/978-3-662-04722-4
2. Daemen, J., Mennink, B., Van Assche, G. *The Keccak Reference.* Submission to the NIST SHA3 Competition, 2011. Available at: https://keccak.team/files/Keccak-reference-3.0.pdf
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G. *Keccak Specifications and Implementations.* NIST, 2012. Available at: https://keccak.team/files/Keccak-specifications.pdf
4. National Institute of Standards and Technology (NIST). *FIPS 197: Advanced Encryption Standard (AES).* U.S. Department of Commerce, 2001. Available at: https://doi.org/10.6028/NIST.FIPS.197
5. National Institute of Standards and Technology (NIST). *FIPS 202: SHA3 Standard.* U.S. Department of Commerce, 2015. Available at: https://doi.org/10.6028/NIST.FIPS.202
6. National Institute of Standards and Technology (NIST). *SP 800-185: SHA3 Derived Functions, cSHAKE, KMAC, TupleHash, and ParallelHash.* U.S. Department of Commerce, 2016. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf
7. Daemen, J., Rijmen, V. *Rijndael Reference Code and Test Vectors.* AES Submission Archive. Available at: https://csrc.nist.gov/Projects/block-cipher-techniques/aes-development Direct package: https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/aes-development/aes-rijndael.pdf
8. Keccak Team. *Keccak Reference Software and Documentation.* Available at: https://keccak.team/
9. QRCS Corporation. *RCS Technical Specification* Quantum Resistant Cryptographic Solutions Corporation, 2024. Available at: https://www.qrcscorp.ca/documents/rcs_specification.pdf
10. QRCS Corporation. *RCS Implementation Analysis* Quantum Resistant Cryptographic Solutions Corporation, 2025. Available at: https://www.qrcscorp.ca/documents/rcs_analysis.pdf
11. QRCS Corporation. QSC Cryptographic Library Implementation (GitHub Source Code Repository). https://github.com/QRCS-CORP/QSC