KITTEN
Internet-Draft
Intended status: Standards Track
Expires: 27 May 2026

A. Bouchez
Tranquil IT SAS

23 November 2025

SCRAM with Modular Crypt Format (SCRAM-MCF)
draft-bouchez-scram-mcf-01

Abstract

This document specifies SCRAM-MCF, an extension to the Salted Challenge Response Authentication Mechanism (SCRAM) family of SASL mechanisms ([RFC5802]) and HTTP Digest extensions ([RFC7616], [RFC7677]).

The extension replaces the PBKDF2-specific iteration count attributes i= and s= in the server-first-message with a generic Modular Crypt Format (MCF) descriptor f=. This allows servers to use modern memory-hard key derivation functions such as Argon2, SCrypt, or bcrypt while preserving the full security properties and message flow of SCRAM.

The change is fully backward compatible: servers can continue sending i= and s= for legacy clients and only send f= (or both) when the client advertises support.

This document is intended to be discussed and potentially adopted by the KITTEN working group. Feedback from the KITTEN WG is welcome on the kitten@ietf.org mailing list.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 May 2026.

Table of Contents

1. Introduction

SCRAM as defined in [RFC5802] and its SHA-256 variant in [RFC7677] is widely deployed (PostgreSQL, MongoDB, Kafka, SASL libraries, etc.). Its only key derivation function is PBKDF2-HMAC-SHA-1 or PBKDF2-HMAC-SHA-256. PBKDF2 is no longer considered state-of-the-art against GPU/ASIC-based password cracking.

Modern password hashing algorithms (Argon2 winner of the 2015 Password Hashing Competition, SCrypt, bcrypt) are memory-hard and far more resistant to parallel brute-force attacks. However, replacing SCRAM entirely with a new mechanism is unnecessary: the core proof-of-possession construction of SCRAM is excellent. Only the key derivation step needs to become negotiable.

This document defines the smallest possible standards-compliant extension that achieves exactly that.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The term "Modular Crypt Format" (MCF) refers to the de-facto standard string format used by crypt(3), Passlib, Spring Security, and most modern password hashing libraries. Examples:

$argon2id$v=19$m=65536,t=2,p=4$z8e0jsE2kz7z6...$7KX4Wm6Q7f...
$scrypt$ln=16,r=8,p=1$z8e0jsE2kz7z6uL0m4zZ6Q$7KX4Wm6Q7...
$2a$06$m0CrhHm10qJ3lXRY.5zDGO3rS2KdeeWLuGmsfGlMfOxih58VYVfxe

3. SCRAM-MCF Extension

3.1. Client Capability Advertisement

A client that implements SCRAM-MCF MUST include the attribute f=y in the reserved extension field of the client-first-message when using any SCRAM mechanism that supports this extension.

Example (client-first-message):

n,,n=user,r=fyko+d2lbbFgONRv9qkxdawL,f=y

Clients that do not send f=y MUST be treated exactly as in the original SCRAM algorithm (legacy clients). They MUST NOT support the f= attribute if they did not send f=y first.

The f= attribute name - as in "format" - was selected because [RFC5802] states that names are single US-ASCII letters, and that m= is already reserved.

3.2. Server-first-message Changes

A server that receives f=y and wishes to use a modern KDF MUST respond with the attribute f= instead of i= and s=.

The value of f= is the full MCF identifier string up to but not including the final stored hash part, encoded as base-64. In other words: everything up to and including the base64-encoded salt and its trailing $ character (if any), then converted to base-64. The MCF identifier MUST be converted to base-64, to ensure no commas signs appear in the attribute value.

Valid examples of MCF identifiers:

```
$argon2id$v=19$m=65536,t=2,p=4$z8e0jsE2kz7z6uL0m4zZ6Q$
$scrypt$ln=16,r=8,p=1$z8e0jsE2kz7z6uL0m4zZ6Q$
$2a$06$m0CrhHm10qJ3lXRY.5zDGO
```

Values transmitted as f= attribute, after base-64 encoding:

```
f=JGFyZ29uMmlkJHY9MTkkbT02NTUzNix0PTIscD00JHo4ZTBqc0Uya3o3ejZ1...
f=JHNjcnlwdCRsbj0xNixyPTgscD0xJHo4ZTBqc0Uya3o3ejZ1TDBtNHpaNlEk
f=JDJhJDA2JG0wQ3JoSG0xMHFKM2xYUlkuNXpER08=
```

Backward compatibility fallback: If the server cannot or does not want to use MCF, it simply omits f= and sends the normal i= and s= attributes.

3.3. Client Processing Rules

* If the server-first-message contains f=, the client MUST ignore any i= and s= attributes and MUST use the specified MCF string, after proper base-64 decoding, to derive SaltedPassword exactly as if it were calling the standard password-to-key function of that algorithm with the cleartext password and the MCF parameters/salt.

* The full encoded output of this MCF KDF, including its identifier, parameters, salt and checksum, MUST become the SaltedPassword as in [RFC5802] Section 3.

* The rest of the protocol (ClientKey, ServerKey, AuthMessage, ClientProof, etc.) is unchanged.

* The H() and HMAC() functions used to compute the client and server proofs MUST follow the SASL negotiation, e.g. SHA-256 and HMAC-SHA-256 for "SCRAM-SHA-256" or SHA-512 and HMAC-SHA-512 for "SCRAM-SHA-512".

* If the client does not recognize or support the MCF identifier, it MUST respond with a SASL/HTTP error (e.g., "invalid-parameters").

3.4. SCRAM-MCF Algorithm Overview

The following is a description of the algorithms used in a full, uncompressed SASL SCRAM-MCF authentication exchange.

```
// SCRAM-MCF password KDF
SaltedPassword = MCF(Password, identifier, parameters, salt)
// using MCF prefix identifier, parameters and salt
// e.g. "$2a$06$m0CrhHm10qJ3lXRY.5zDGO3rS2Kd..."

// initial server storage
ClientKey       := HMAC(SaltedPassword, "Client Key")
StoredKey       := H(ClientKey)
ServerKey       := HMAC(SaltedPassword, "Server Key")
persist the MCF prefix, StoredKey and ServerKey in the DB

// client
ClientKey       := HMAC(SaltedPassword, "Client Key")
ClientSignature := HMAC(StoredKey, AuthMessage)
return ClientProof := ClientKey XOR ClientSignature

// server
ClientSignature      := HMAC(StoredKey, AuthMessage)
CandidateClientKey   := ClientProof XOR ClientSignature
Checks: H(CandidateClientKey) == StoredKey
return ServerProof := HMAC(ServerKey, AuthMessage)

// client verifies
ServerSignature := ServerProof XOR ClientSignature
Checks: ServerSignature == HMAC(ServerKey, AuthMessage)
```

The rest of these messages is defined in [RFC5802] Section 7.

3.5. Server Storage Recommendation

Servers MUST store the MCF prefix string (identifier + parameters + salt) for each user. For security reasons, the MCF checksum part MUST NOT be persisted; instead, the SCRAM-derived StoredKey and ServerKey MUST be stored.

Servers MAY store and accept other SCRAM hashes (e.g., SCRAM-SHA-1 or SCRAM-SHA-256) for backward compatibility. But restricting to the safest algorithms (like SCRAM-MCF) is strongly recommended.

4. Formal ABNF Changes (augmented from RFC 5802)

```
server-first-message = ( scram-attr-val "," )* "r=" nonce
                       [ "," "s=" base64 ]
                       [ "," "i=" posint ]
                       [ "," "f=" mcf-base64 ]
                       *( "," scram-extension )

mcf-base64     = base64( mcf-descriptor )
mcf-descriptor = "$" mcf-id "$" mcf-params "$" base64-salt [ "$" ]
mcf-id         = "argon2i" / "argon2d" / "argon2id" /
                 "scrypt" / "2b" / other registered identifiers
```

5. Security Considerations

5.1. Preservation of the SCRAM Proof Construction

In respect to the standard SCRAM mechanism, this SCRAM-MCF extension could be evaluated as such:

* The extension preserves channel binding, proof-of-possession, and mutual authentication exactly as in SCRAM.

* Memory-hard KDFs dramatically increase the cost of offline dictionary attacks.

* Because the KDF identifier and parameters are sent in the clear (as in standard SCRAM with i=), no new information is leaked to an eavesdropper.

* MCF algorithms MUST use a cryptographically random per-user salt.

* Clients and servers MUST reject weak MCF parameters (see "Minimum Acceptable" column in Section 5.2).

* Thanks to the MCF registration mechanism, this SCRAM-MCF pattern was designed to be future-proof.

* Weaker "SCRAM-SHA-1" MUST be rejected when used with the f= extension.

5.2. Minimum and Recommended MCF Parameters Levels

MCF parameters levels are informed by NIST SP 800-63B [NIST.SP.800-63B] (preferring Argon2id with memory-hard properties) and OWASP guidelines [OWASP.PasswordStorage].

Tune iteratively based on client load; aim for ~500ms-1s per verification to balance security and usability. Note that the server load is not affected by the MCF parameters, since during SCRAM authentication, the password derivation is done on the client side. For performance and Denial Of Service (DoS) mitigation, consider rate-limit derivations, e.g. testing with RFC 5802 [RFC5802] tools.

KDF	Minimum Acceptable	Recommended (2025+)	Notes
Argon2id (preferred)	Memory: 19 MiB t=1, p=1, m=19× 1024	Memory: 64 MiB t=3, p=4, m=64× 1024	Memory-hard; primary recommendation in NIST and OWASP. Salt >= 16 bytes.
bcrypt	cost=10 salt >= 128 bits	cost=1416 salt >= 192 bits	Still acceptable when Argon2 is unavailable. 72-byte password limit.
scrypt	N=2^17 (131072), r=8, p=1	N=2^20 (1M), r=8, p=4	Memory-hard; good alternative when Argon2 cannot be used.
PBKDF2 (FIPS-only fallback) (for plain SCRAM)	>= 10 000 iterations HMAC-SHA-256, salt >= 128 bits	>= 600 000 iterations HMAC-SHA-256, salt >= 256 bits	Approved by [NIST.SP.800-132] but vulnerable to GPU attacks; use only when FIPS compliance is required.
PBKDF2 KDF is listed only as reference, and makes little sense to appear as a MCF algorithm, since it is already used by the original SCRAM.			

Table 1

5.3. Using MCF-Derived Outputs as HMAC Keys

As defined above, ClientKey and ServerKey are computed using HMAC() on the MCF output. The length and structure of this value differ from PBKDF2, as used in classic SCRAM.

HMAC constructions (including HMAC-SHA-256 and HMAC-SHA-512 refered by this document) are provably secure even when the supplied key is long or non-uniform. Per Bellare, Canetti, and Krawczyk (1997; 2006), HMAC guarantees pseudorandomness provided that:

* keys longer than the hash function block size are reduced by hashing (as mandated by HMAC), and

* keys of arbitrary internal structure are permitted.

Thus, MCF outputs — even if longer than a SHA-2 block — are acceptable and safe HMAC keys. Because memory-hard KDFs generate outputs indistinguishable from random to an attacker lacking the password, the resulting SaltedPassword is at least as strong as the PBKDF2-derived SaltedPassword in classic SCRAM.

Therefore, replacing PBKDF2 output with memory-hard KDF output does not weaken the HMAC-based authentication proofs of SCRAM.

6. IANA Considerations

This document requests IANA registration of the SASL/GS2 extension attribute f=y and the server-first-message attribute f.

No new SASL mechanism name ("SCRAM-MCF-*") needs to be registered. The existing names "SCRAM-SHA-256", "SCRAM-SHA-256-PLUS", "SCRAM-SHA-512", etc. continue to identify the underlying H() and HMAC() functions used for the proof calculation.

7. SCRAM Authentication Exchange

This is a simple example of a SCRAM-SHA-256 authentication exchange when the client doesn’t support channel bindings (username ’user’ and password ’pencil’ are used, with SCrypt MCF hashing):

```
C: n,,n=user,r=rOprNGfwEbeRWgbNEkqO,f=y
S: r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFIlj)hNlF$k0,f=
JHNjcnlwdCRsbj00LHI9OCxwPTEkUU54NE40NTRwcE1lS21Eanh5cmhzaDdRL1BZQlF3JA==
C: c=biws,r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFIlj)hNlF$k0,
   p=LyMcKMrzBJUoDhfO7e8aD0BYBLqe3wBrCoJMTIEhJEg=
S: v=2c7tBGYYY8tx/oLZNhZVJnjQKWeku17ZDaV0Jh3Hvmo=
```

The corresponding MCF prefix string is "$scrypt$ln=4,r=8,p=1$QNx4N454ppMeKmDjxyrhsh7Q/PYBQw$", which is base-64 encoded in the f= attribute above.

8. Acknowledgments

The idea of using Modular Crypt Format inside a SCRAM-like flow was first implemented by the author in the Synopse mORMot 2 Framework in 2025 and standardized as the wire-level extension described in this document.

The author would like to thank the PostgreSQL, MongoDB, and SASL communities for their prior art and feedback on modernizing password hashing in authentication protocols. A special thanks to Simon Josefsson and Thilo Molitor for their very valuable feedback to our initial proposal.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC5802] Newman, C., Menon, A., Hildebrand, J., and K. Zeilenga, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", July 2010, <https://www.rfc-editor.org/rfc/rfc5802>.

[RFC7677] Hansen, T. and A. Melnikov, "SCRAM-SHA-256 and SCRAM-SHA-256-PLUS SASL Mechanisms", November 2015, <https://www.rfc-editor.org/rfc/rfc7677>.

[RFC7616] Fielding, R. and J. F. Reschke, "HTTP Digest Access Authentication", September 2015, <https://www.rfc-editor.org/rfc/rfc7616>.

[NIST.SP.800-132] Turan, M. S., Barker, E., Burr, W., and L. Chen, "Recommendation for Password-Based Key Derivation: Part 1: Storage Applications", NIST Special Publication 800-132, December 2010, <https://doi.org/10.6028/NIST.SP.800-132>.

[NIST.SP.800-63B]
Grassi, M. L., Richer, M. E., Winters, S. P., Lefkovitz, K. B., Light, J. M., Burr, W. E., Nadeau, S., Gruenberg, M. A., McKay, D. A., and J. A. Sigmon, "Digital Identity Guidelines: Authentication and Lifecycle Management", NIST Special Publication 800-63B, June 2017, <https://doi.org/10.6028/NIST.SP.800-63b>.

[OWASP.PasswordStorage]
Foundation, OWASP., "Password Storage Cheat Sheet", November 2025, <https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html>.

Author's Address

Arnaud Bouchez
Tranquil IT SAS
12 Avenue Jules Verne
44230 Saint-Sebastien-sur-Loire
France
Email: abouchez@tranquil.it
URI: https://tranquil.it