S3 uses an access key ID and secret key because they are designed as long-lived, *programmatic* credentials that can be safely used to sign API requests and be tightly scoped by IAM policies, unlike typical interactive usernames and passwords used for console logins. They integrate directly with the S3 signing process (Signature Version 4), where each request is cryptographically signed with the secret key to prove authenticity and integrity.

Credential purpose

- **Username/password** in AWS are meant for humans logging into the web console, often with MFA and session-based access, not for automated API/CLI traffic.
- **Access key ID/secret key** are explicitly intended for SDKs, CLIs, and applications accessing services like S3 programmatically.

How S3 authenticates requests

- Each S3 API request is signed: a canonical representation of the request is hashed with the **secret key**, producing a signature sent along with the **access key ID** in headers.
- The service uses the access key ID to look up the principal, then recomputes the signature with the stored secret key to verify both identity and that the request was not modified in transit.

Security and IAM integration

- Access keys are tightly bound to IAM users or roles, so fine-grained policies can allow "this key can only access this bucket, in this account, with these operations."
- You can rotate access keys, have at most two active per IAM user, and disable or delete them without affecting the user's console password or other credentials.

Why not username/password for S3 APIs?

- Username/password alone do not provide a built-in request signing mechanism; sending them on every API call would be less secure and harder to protect at rest in configs or code.
- Access keys are treated as the "API equivalent" of username/password but with better lifecycle controls, programmability, and alignment with cryptographic signing flows used by S3 and other AWS APIs.

Modern best practice

- Long-lived access keys are now considered a legacy pattern; where possible, use IAM roles and temporary credentials (STS) so that even these keys are short-lived.
- When access keys are unavoidable (e.g. third-party backup tools), best practice is to create a dedicated IAM user with least-privilege S3 permissions and store the keys securely (env vars, credential stores, not in code).

⁂

1. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
2. https://docs.aws.amazon.com/IAM/latest/UserGuide/security-creds.html
3. https://docs.hetzner.com/storage/object-storage/faq/s3-credentials/
4. https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-management.html
5. https://themedev.net/blog/aws-s3-bucket-access-key-id-and-secret-access-key/
6. https://docs.snowflake.com/en/user-guide/data-load-s3-config

7. https://stackoverflow.com/questions/59864321/difference-between-credentials-and-accesskey
8. https://kirkpatrickprice.com/blog/secure-aws-access-keys/
9. https://beta.docs.unified-streaming.com/documentation/vod/cloud/amazon/aws_s3_authentication.html
10. https://www.reddit.com/r/aws/comments/1002b9a/using_an_s3_bucket_as_a_backup_destination/
11. https://www.reddit.com/r/aws/comments/14h540t/how_are_iam_access_keys_stored_for_programmatic/
12. https://stackoverflow.com/questions/74975775/best-practice-aws-access-key-and-secret-for-applications
13. https://community.netapp.com/t5/Network-and-Storage-Protocols/S3-Bucket-Key-Authentication/m-p/450513