Engineer

Encryption

making data unreadable without the key

What it is

Encryption, which some NIST publications call enciphering, turns readable data, the plaintext, into ciphertext by running it through a published algorithm together with a secret value called a key. Decryption, given the right key, returns the plaintext byte for byte. The algorithm is public on purpose: AES, the cipher most systems reach for, won an open NIST competition in 2000 and is published as a free standard. Only the key is secret.

Ciphers come in two families. A symmetric cipher uses the same key in both directions. AES, specified in FIPS 197, encrypts data in blocks of 128 bits with a key of 128, 192 or 256 bits. ChaCha20, from RFC 8439, is a stream cipher about three times as fast as AES on hardware without AES instructions. Symmetric ciphers are fast enough for a whole disk, but both ends need the same key, and getting it there safely is the hard part.

Public-key cryptography answers that with a pair of keys, one published and one kept private. Two parties can each combine their own private key with the other's public key and arrive at the same shared secret without ever sending it. Public-key operations are slow, so real systems use both: a public-key exchange agrees a fresh symmetric key, and the symmetric cipher carries the data. TLS, the protocol behind every `https://` address, works this way.

The modern idea that matters most is authentication. A bare cipher hides content but leaves it open to change: in counter mode, flipping one bit of the ciphertext flips the same bit of the plaintext, and the receiver decrypts the altered message as if it were genuine. Authenticated encryption with associated data, AEAD, adds a tag the receiver checks before trusting a byte. AES-GCM and

ChaCha20-Poly1305 are the common pair, and TLS 1.3, published as RFC 8446 in 2018, kept AEAD ciphers alone and made every public-key exchange forward-secret, so a server key stolen later opens no recorded session.

Use the concept in a decision

Put “What is encryption?” to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

Data rarely stays on hardware its owner controls. It crosses other people’s networks, sits in someone else’s data centre and leaves the building on laptops. Encryption is what lets it pass through those places safely: a stolen disk, a tapped cable or a leaked bucket yields ciphertext, and ciphertext without the key is noise.

That moves the problem rather than removing it: once data is encrypted, its safety is the safety of the key. NIST’s key-management recommendation, SP 800-57, compares keys to the combination of a safe: an adversary who knows the combination walks past the strongest safe, and poor key management can easily compromise a strong algorithm. Most of the real work is therefore key work: storing keys apart from the data, limiting how much each one protects, replacing them on a schedule and guarding the recovery copy as closely as the original.

Regulators draw the same line. The GDPR lists the pseudonymisation and encryption of personal data among the security measures in article 32. Article 34 lets a controller skip telling each affected person about a breach when the leaked data was unintelligible to anyone not authorised to see it, and encryption is the example it gives.

Encryption is applied in two places. Encryption in transit, mostly TLS, protects data on the wire. Encryption at rest protects it in storage. The moment of use sits outside both: a server that decrypts a record to work on it holds the plaintext and the key at once, so whoever controls that server reads what it reads.

What it looks like in practice

In transit, the daily work is keeping certificates valid. When a hundred Docker applications needed SSL/TLS certificates, an ACME workflow created every certificate, renewed each before it lapsed and deployed it to Ubuntu hosts running Apache and Nginx. The aim was to take the person out of the loop, because the person is the part that forgets, and an expired certificate takes an app down as surely as a broken cipher.

At rest, the useful question is who holds the key. The company's encrypted off-host backup implementation was designed to extend recovery beyond a provider snapshot. It uses restic, which encrypts what it stores with AES-256 in counter mode and authenticates it with Poly1305-AES, and its scripts support sending snapshots over SFTP to storage at another provider. In that design the provider receives ciphertext and the company holds the passphrase in a protected file. That describes the implemented controls, not their operational activation: the record dated 2026-10-27 says the timers were disabled without a configured repository and records no completed live restore. Encryption is a property of the proposed storage path; proving that the backup can be recovered requires running the separate restore test.

Even the route the passphrase takes mattered. It is read from a file, because the supervisor processes escapes in environment values, and a passphrase containing a backslash would have silently differed from the one that created the repository. The implemented monthly restore-test workflow is designed to restore a known file from the repository and compare it with the expected result. Because the test requires the passphrase, a successful run would also verify that the key still opens the data.

When the platform was about to gain a second host, the same reasoning decided how every operational secret was split per host. A backup repository string identifies a location. Separate restic repositories can have distinct master keys even when they use the same passphrase. Someone who obtains that passphrase and can read both repositories' encrypted data and key files can unlock both, so sharing it couples their exposure. Each secret became a map keyed by host, with the argument written beside the configuration. SP 800-57 makes the general case: limiting what one key protects limits the damage when it leaks.

Where it goes wrong

The key exists in one place. A strong cipher treats an owner who forgot the password exactly like an attacker. The restic documentation puts it plainly: once the password is lost, so is the data, with no way to recover it. Every encrypted store needs a second copy of its key, kept where whatever destroys the first cannot reach it.

The key sits beside the data. A database encrypted with a key in the same server's configuration, or a backup whose passphrase lives in the same bucket, stops only a thief who takes half the files. Disk encryption is the quieter version: it protects a drive pulled from a machine that is switched off, while a running server has already unlocked its disk, and an attacker on that server reads plaintext.

Secrecy is taken for integrity. A system built on a bare cipher mode hides content and still lets an attacker alter it. Use an AEAD construction and discard any message whose tag fails before acting on a byte of it.

A nonce repeats. AES-GCM and ChaCha20-Poly1305 take a nonce, a value that must stay unique for every message under one key. RFC 8439 calls this the most important security consideration: two messages with the same nonce share a keystream and a one-time authentication key, which reveals the XOR of the two plaintexts. A counter is a safe way to make nonces, as long as one counter owns the key: two processes that share a key and each count from zero produce exactly the repeat the RFC warns about.

A password stands in for a key. People choose passwords; ciphers need random keys of a fixed length. A passphrase goes through a key derivation function first, as restic does with scrypt, which RFC 7914 designs to be memory-hard so that guessing in bulk on custom hardware stays expensive. Stored login passwords are a separate case: they belong in such a function as hashes, because a server that can decrypt them can also leak them.

The metadata stays visible. Encryption hides content, while size, timing and the addresses at either end remain in view. TLS 1.3 lets a sender pad records to hide their size, and padding is optional, so length often says more than the owner expects.

Further reading

- NIST Glossary: encryption — every definition NIST publications use, each with its source document.
- FIPS 197: Advanced Encryption Standard (AES) — the AES specification, with its block size and three key lengths.
- NIST SP 800-57 Part 1 Rev. 5: Recommendation for Key Management — how keys are made, stored, used and retired, and what follows when one leaks.
- RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 — the protocol behind `https://`, with AEAD-only records and forward-secret key exchange.
- RFC 8439: ChaCha20 and Poly1305 for IETF Protocols — a stream cipher, an authenticator and their AEAD pairing, with the nonce rules.
- restic: References — how a real backup tool derives keys from a password, then encrypts and authenticates its data.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is encryption?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/encryption/