

Week 13 – Security Requirements

Date	11.12.2026
Learning outcomes	LO.5, 7
Duration	3 hours
Prerequisites	Asset, threat and countermeasure from Week 1; evaluation and evidence concepts from Week 12; what the S sections of the term project are for
Labs	code/week-13 – 2 demos; build once in the <code>code</code> folder, then run from <code>bin/linux</code> (<code>bin\windows</code> on Windows)



This week's working demo

`code/week-13/01-uyum-matrisci` – Compliance matrix validator: flags "met" rows with no evidence as findings (S17). · `code/week-13/02-gereksinim-kalite` – Requirement quality checker: flags vague/unverifiable requirements.

Run it: build once in the `code` folder with `./build.sh` (`.\build.ps1` on Windows), then from `bin/linux` (`bin\windows` on Windows) inside the demo folder. Step-by-step commands are in the box below. Fully synthetic and safe; it does not harm the student's computer.

 Run the demo yourself — step by step (copy-paste)

The initial build is explained in `code/README.en.md`. From the `code` folder:

```
# Windows (PowerShell)
.\build.ps1
cd week-13\01-uyum-matrisi
.\bin\windows\uyum.exe
cd ../02-gereksinim-kalite
.\bin\windows\gereksinim_kalite.exe
```

```
# WSL / Linux
./build.sh
cd week-13/01-uyum-matrisi && ./bin/linux/uyum
cd ../02-gereksinim-kalite && ./bin/linux/gereksinim_kalite
```

Expected output: The first demo reads a **compliance matrix** row by row and flags "**met**" rows that have **no evidence** with a red flag. The second demo scans example requirements; it flags **vague/unmeasurable** wording such as "should be secure" or "appropriately" as **WEAK**, and the **exit code equals the number of weak requirements**.

 By the end of this week you will be able to

1. Tell a good **security requirement** apart from a bad one; build a **traceability matrix** that links a requirement to design, control, test, and evidence.
2. Use requirement statuses (**met / deferred / not met**) correctly; write down the requirements that a software component **defers** to the operating system and to the parent application.
3. Explain the core concepts of **Common Criteria** (ISO/IEC 15408) — TOE, ST, PP, SFR, SAR, EAL — and the security levels of **FIPS 140-3**.
4. Compare what the requirement sets published by ETSI, GSMA, EMVCo, and PCI for mobile and embedded systems ask for.
5. Using the course's requirement families, write a **compliance matrix** (S17) and a **deferred requirements** section (S14) for your own project.

Class schedule (timing plan for the instructor)



Time	Section	What happens
0:00–0:20	1	What is a requirement, how do you write a good one?
0:20–0:50	2–3	Traceability from requirement to evidence; the requirement-block pattern and deferred requirements
0:50–1:00	Break	
1:00–1:30	4	Common Criteria: TOE, ST, PP, SFR/SAR, EAL
1:30–1:50	5	FIPS 140-3 and cryptographic module validation
1:50–2:00	Break	
2:00–2:25	6	ETSI, GSMA, EMVCo, PCI MPoC, OWASP MASVS
2:25–2:50	7–8	Carrying requirements into the plan and asset management; the course's requirement families — compliance matrix activity
2:50–3:00	9+	Project step (S14, S17), self-check

About this week's sources

The requirement texts of the card schemes are proprietary to the institution and shared under a non-disclosure agreement. So this week uses **open standards** (Common Criteria, FIPS 140-3, ETSI EN 303 645, OWASP MASVS) and the course's own **requirement ids** adapted from them, instead of the schemes' texts. The structures and methods are the same as the documents used in the field.

0. Basic concepts (from scratch)

This section **assumes no prior knowledge**. We define, from scratch, the terms we will use for the rest of the week. If you don't know a term, read this section first; the following sections build on these.

What is a requirement?

- **Requirement:** a condition the system **must meet**.
- A security requirement: a security condition.
- A good requirement is **verifiable** (testable).

Three types of requirement

- **Functional:** what security function will exist? (e.g., data is protected with AEAD)
- **Assurance:** how will we trust that it was done correctly? (e.g., a test report)
- **Process:** how must the organisation operate? (e.g., every change is reviewed)

Good vs. bad requirement

- **Bad:** "The application must be secure." (unverifiable)
- **Good:** "The release build must be produced with a stack canary, PIE, and full RELRO." (measurable)

Traceability

- **Traceability:** linking each requirement to a **control**, a **test**, and **evidence**.
- "Where was this requirement met, how was it verified?"

Compliance matrix

- **Compliance matrix:** a requirement → status → section → verification → evidence table.
- The project's **S17** section.
- The first thing an evaluator looks at.

Requirement statuses

- **Met:** the product satisfies this requirement (with evidence).
- **Deferred:** another party satisfies it (to whom, why, how).
- **Not met:** not yet satisfied.

Deferred requirement

- If a component cannot meet a requirement, it **defers** it to the parent application/OS.
- The guide states **to whom**, **why**, and **how** it will be met.
- The project's **S14** section.

Common Criteria (CC)

- **Common Criteria (ISO/IEC 15408):** the international standard for product security evaluation.
- Concepts: TOE, ST, PP, SFR, SAR, EAL (shortly).

CC · basic terms

- **TOE:** the product being evaluated.
- **ST (Security Target):** this product's security-target document.
- **PP (Protection Profile):** a common requirement set for a product **class**.

CC · SFR, SAR, EAL

- **SFR**: security **functional** requirements.
- **SAR**: security **assurance** requirements.
- **EAL**: the **depth** level of the evaluation (EAL1–EAL7).

FIPS 140-3

- **FIPS 140-3**: the standard for validating cryptographic **modules**.
- Security levels (1–4).
- It covers only the module, not the whole application.

Sector standards

- **ETSI EN 303 645**: baseline IoT security.
- **GSMA, EMVCo, PCI**: mobile/payment.
- **OWASP MASVS**: mobile application requirements.

Why are there so many terms? (rationale)

The certification world uses its own vocabulary because an evaluator reads your **document**, not your project. If what you call a "requirement" in the document is actually a "control," the evaluator conflates the two and asks the wrong question: instead of "Where is this requirement's control?" they should be asking "Why is this requirement written like a control, what does it satisfy?" If the terms are not used consistently, the traceability chain (Section 2) cannot be built; each word maps to a different box (requirement, control, or evidence), and if those boxes get mixed up the matrix becomes meaningless. That is why the discipline this course requires is, first, using the right word in the right place; the technical solution is the second step.

Common mistake: confusing a requirement with a control

Beginners often write: "Requirement: AES-256-GCM must be used." This is actually a **control** (a solution), not a requirement. The requirement should have been: "Sensitive data at rest must be protected with authenticated encryption (AEAD)." The control is **how** you meet this requirement: "With AES-256-GCM, as described in S7.2." A requirement is fixed (it comes from a standard or a threat model); a control can vary from project to project — one project may meet the same requirement with AES-GCM, another with ChaCha20-Poly1305; both correctly meet the same requirement.

Consequence: A compliance matrix that writes a solution in the requirement column gives the evaluator the impression that "you've already made the decision, you didn't consider alternatives"; also, because the requirement is detached from the standard, which standard clause it corresponds to is lost, and the traceability chain breaks.

✓ Rule

Always write **what** is being asked for (a condition coming from a standard or threat model) in the requirement column; always write **how** you meet it (which algorithm, which code, which library) in a separate "control" column or sentence. If a sentence contains a product name, a library name, or an algorithm name, that sentence is most likely a control, not a requirement.

Let's see the concepts through a single asset (mini example)

To see how the terms fit together, let's quickly walk through a single asset; we'll see the fully worked-out version end to end in Section 2.

1. **Asset** (from Week 1): "the user's session token"; marked with confidentiality class **C** in the asset table.
2. **Threat** (from Week 1): an attacker listening on the network can capture this token and act as the user.
3. **Requirement**: "The session token must only be carried over an encrypted channel." This states **what** the system must do against the threat; it does not say **how**.
4. **Status**: Does the team meet this itself, or **defer** it to another party? Say the product sets up TLS itself: the status is **met**.
5. **Traceability**: In which section of the guide is this requirement explained (control), how is it verified (test), and where is its evidence? The answers to these three questions form one row of the **compliance matrix**.

Let's follow the same asset to see the "deferred" status too: if the product were a library and the parent application set up the TLS connection, the requirement would be **deferred** to the parent application, and S14 would read "to whom: the parent application; why: the library does not open network connections itself; how: the parent application must connect using TLS 1.2+ and certificate validation."

The difference between a 'requirement family' and a 'standard'

This course uses two different things coming from two different sources, and confusing them is a common mistake:

- **Standard**: an external document the course did not write (e.g., ETSI EN 303 645, Common Criteria, FIPS 140-3, OWASP MASVS). It has its own clause numbers and its own issuing body.
- **The course's requirement family**: the course's own short list, adapted from these standards, with ids of the form `CEN429-<family>-<number>` (Section 8). Its purpose is to gather the common denominator of different standards into a single workable list.

When writing a requirement, "which standard it is based on" (source) and "which of the course's families it belongs to" (id) are two separate questions; you state both separately, in the S1 (sources) and S17 (compliance matrix) sections.

Quick self-check (before finishing Section 0)

- **Q**: Is "The release build must be produced with PIE" a requirement or a control? **A**: A requirement — a measurable condition; it doesn't yet say "how it is met" (which compiler flag).
- **Q**: A row in a guide reads "met" but has no file name next to it. What does this row mean? **A**: It proves nothing; the traceability chain is considered broken (Section 2).
- **Q**: If the product never connects to the network, what status is written for "data in transit" requirements? **A**: "Not applicable," together with its rationale.

- **Q:** Why isn't "Keys must be well managed" singular? **A:** "Managing well" vaguely packs several conditions — generation, storage, use, and destruction — into one sentence; each should be its own requirement.

The difference between 'not met' and 'not applicable'

These two statuses are often confused but mean very different things; using both correctly is just as important to the matrix's credibility as the requirement statuses themselves:

- **Not met:** the requirement **applies** to the product but is **not yet satisfied**. This is a **gap**; it is written into the residual risk and goes into the next release plan.
- **Not applicable:** the requirement **does not apply** to the product at all (e.g., if the product does not use network connectivity, "data in transit" requirements are not applicable). This is not a gap, but if written without a rationale, the evaluator treats it as "the requirement was skipped, not checked."

Rule: whenever you write "not applicable," always add one sentence stating **why** it does not apply. A "not applicable" with no rationale is just as untrustworthy as a "met" with no evidence — in the evaluator's eyes, both can mean "not checked."

The six status/decision words we've seen in this section

Let's gather the status and decision words we've seen throughout this section; these are the words you'll use over and over when filling in S17 and S14 in your term project:

Word	When to use it	What must accompany it
Met	The product satisfies the requirement itself	Control + verification + evidence (Section 2)
Deferred	Another party satisfies it	To whom + why + how (this section, Section 3)
Not met	Applies but not yet satisfied	Residual risk + planned fix
Not applicable	Does not apply to the product at all	Rationale
must (MUST)	Mandatory	A direct finding if not met
should (SHOULD)	Strong recommendation	A written rationale if not met

Keeping this table in mind will give you a reference point as you read every example in the sections ahead.

Now we're ready

Terms:

requirement (functional/assurance/process) · traceability · compliance matrix · status (met/deferred/not met) · deferred · CC (TOE/ST/PP/SFR/SAR/EAL) · FIPS 140-3 · ETSI/GSMA/EMVCo/PCI/MASVS

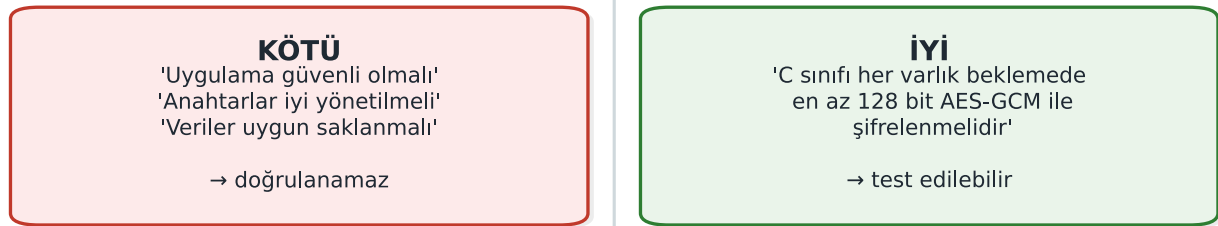
Now: how do you write a good requirement?

1. What is a security requirement? How do you write a good one?

Throughout the term we linked every control to a threat: "we protect this asset against this attacker." In the certification world, the same link extends one step further: every control is linked to a **requirement**. A requirement states **what** the product must do; it does not say how. Design and code answer the "how"; the evaluator instead asks "has the requirement been met, and where is the evidence?"

Kötü gereksinim → iyi gereksinim

fark: ölçülebilir dayanak ve tekil ifade



Zayıf gereksinim test edilemez; test edilemeyen gereksinim karşılandığı kanıtlanamaz.

İyi gereksinimin ölçütleri

bu ölçütleri karşılamayan gereksinim test edilemez

TEKİL	bir cümle, bir gereksinim
DOĞRULANABİLİR	nasıl test edileceği bellidir
İZLENEBİLİR	kaynağı ve kanıtı vardır
ÖLÇÜLEBİLİR	sayı ya da somut teknik terim içerir
'NE' odaklı	'nasıl' yapılacağını dayatmaz

'Uygulama güvenli olmalı' bunların HİÇBİRİNİ karşılamaz.

A short history: how did security requirements become standardized?

- **1985** — **TCSEC** (the "Orange Book") formalizes security **requirement levels** for the first time.
- **1994** — **FIPS 140** cryptographic module requirements (today **140-3**, 2019).
- **1999** — **Common Criteria (ISO/IEC 15408)**: the **PP/ST**, **SFR/SAR**, and **EAL** concepts come from here (Week 12's process).
- **2010s** — sector-specific sets: **OWASP MASVS/MASTG** for mobile, **ETSI EN 303 645** for consumer IoT, **EMVCo/PCI** for payments.

The unchanging principle: a good requirement must be **measurable** and **traceable**; a "met" with no evidence is invalid.

Three types of requirement

Type	Question	Example
Functional security requirement	Which security function must the product perform?	"The application must protect sensitive data in the local database with authenticated encryption."
Assurance requirement	How will we trust that the product performs this function correctly?	"The developer must provide the source code's static-analysis report and the results of the security tests."
Process requirement	How must the organisation that produces the product operate?	"Every change must be reviewed by at least one other developer before being merged."

Bad and good requirements

Bad	Why is it bad?	Good
"The application must be secure."	Unverifiable	"The application's release build must be compiled with a stack canary, PIE, and full RELRO enabled."
"Data must be encrypted."	Which data, in what state, at what strength?	"Every asset marked C in the asset table must be encrypted at rest with an AEAD algorithm providing at least a 128-bit security level."
"Keys must be protected and well managed."	Two requirements at once, no criteria	(1) "Every key's purpose, cryptoperiod, and destruction time must be documented." (2) "Keys must be erased from memory once their job is done."
"The application should try to detect attacks."	"Should try" is unmeasurable	"The application must refuse sensitive operations and report the event when a debugger is attached."

The properties of a good requirement: **singular** (one sentence, one requirement), **verifiable** (it can be shown to be met with a test or review), **traceable** (it has an id and is linked to a threat or a standard), **feasible** (technically possible), and **what, not how** (it doesn't dictate a solution, but does give criteria such as a security level).

Final form: CEN429-DR-01 — "Sensitive class-C data at rest must be protected with authenticated encryption (AEAD)."
(Threat: T-03, "an attacker who compromises the device reads/modifies the database file," Week 1's threat model.)

Each of the six steps eliminated one kind of ambiguity: **which data** (asset table), **against which threat** (threat model), **which goal** (confidentiality/integrity), **which tool class** (AEAD — the class, not the solution itself), **which state** (at rest), and **which obligation** (MUST). If any one of these six is missing, the requirement remains open to debate and the evaluator sends it back.

Second example: a speed test of the process

To apply the process faster, let's go through the same six steps again in a much shorter example; this time as a table:

Step	Question	In this example
1	Which word is vague?	"The application must be fast and secure." → both "fast" and "secure" are unmeasurable, and there are two separate requirements in one sentence.
2	Which asset/function?	Which function does "secure" cover? Say, the authentication flow.
3	Which goal?	That authentication cannot be bypassed .
4	Which technical criterion?	Server-side validation; client-side checking alone is not enough.
5	Which state/obligation?	On every login attempt, MUST.
6	Id and threat?	CEN429-ID-04 , threat: "an attacker who bypasses the client-side check"

Final form: CEN429-ID-04 — "Every login attempt must be validated server-side; client-side checking alone must not be considered sufficient."

The word "fast" is handled as a separate performance requirement (outside this course's scope) — this is another face of the singularity rule: mixing security and performance requirements in the same sentence blurs which one is met and which isn't; the two can progress at different paces.

Common mistake: mistaking 'how' for 'what' and deciding too early

Teams who write "AES-256-GCM must be used" directly in Step 4, instead of "AEAD must be used," lock the requirement to a single library version. If the project switches to ChaCha20-Poly1305 the following year, the guide still says "AES-256-GCM," so either the guide update is forgotten, or the requirement mistakenly appears "not met."

Consequence: unnecessary guide revisions and false "not met" findings.

✓ Rule

Write the **tool class** (AEAD, CSPRNG, TLS 1.2+) in the requirement text; put a specific library/version name not in the requirement but in the **control** description (the guide section). That way, even if the tool changes, the requirement text stays fixed and only the control is updated.

? How does an evaluator test this?

When an evaluator reads through a requirements list, they ask every sentence three questions: "Does this sentence contain a single condition?", "Which test would I run to verify this?", "Who proves this sentence was met, and how?" If even one of the three questions can't be answered clearly, the requirement must be rewritten.

Reinforcing requirement keywords with an example

Word	Meaning	Example sentence	What happens if not met?
must (MUST)	Mandatory, no exceptions	"The session token must be carried over TLS 1.2+."	A direct finding; written into residual risk
should (SHOULD)	Strong recommendation, can be skipped with a rationale	"Keys should be stored in a hardware secure element (recommended)."	If not met, a written rationale for why it wasn't is required
may (MAY)	Optional	"The application may offer an additional PIN lock."	Not a finding even if not met

The concrete consequence of confusing these three words: if you write a SHOULD requirement as if it were MUST and fail to meet it, you unnecessarily put yourself in a "not met / residual risk" situation when a rationale would have sufficed. The reverse also happens: if you soften a MUST into a SHOULD and don't meet it, the evaluator treats it as a serious gap.

Third example: improving a process requirement

Alongside functional requirements, process requirements demand the same discipline. The bad version:

"Code review must be performed."

- Which change? (Every change, or only big ones?)
- Who reviews it? (Does the author themselves count?)
- When? (Before merging, or after?)

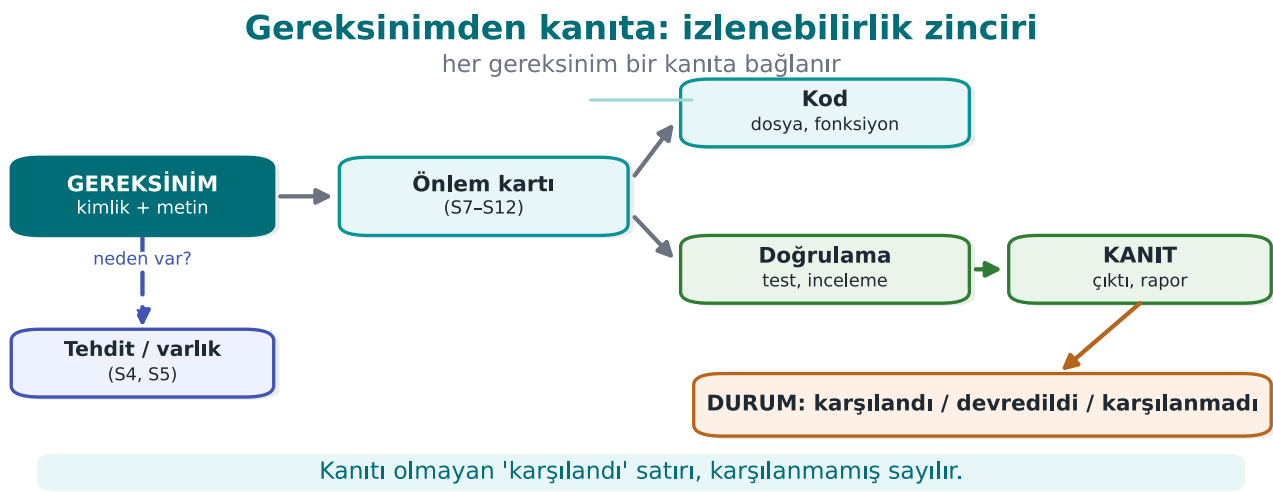
The good version: "Every change must be reviewed by at least one person other than the developer who wrote it, before being merged into the main branch (main/master), and the review approval must be recorded in the version control system." This sentence is now verifiable: you check the version control history to see whether every pull request has at least one approval record — it can even be audited automatically.

Writing a requirement isn't enough: the next question is "how do I verify it?"

In this section we learned to write a requirement **well**; but even a well-written requirement proves nothing on its own. The next question is always: "By what concrete step do I show that this requirement has been met?" This question takes us to the traceability chain in Section 2.

2. From requirement to evidence: traceability

A requirement is only considered met **together with its evidence**. In certification, the following chain is built for every requirement:



The tabular form of this chain is called the **compliance matrix**:

Id	Requirement	Status	Control (guide section)	Verification	Evidence
CEN429-DR-01	Sensitive data at rest must be encrypted with AEAD	Met	S7.2 "Vault file encryption"	Unit test: decryption is rejected when a single byte is changed	test_butunluk.c output
CEN429-AP-03	The release build must not contain debug logging	Met	S12.1	strings scan	Scan output, CI record
CEN429-AP-07	The application must be installed and updated securely	Deferred	S14.2 (to the distribution platform)	—	Rationale
CEN429-DT-04	The server certificate must be pinned	Not met	—	—	Residual risk: S16.4

The matrix has two directions: **forward traceability** (does every requirement lead to a control and a test?) and **backward traceability** (does every control rest on a requirement or a threat, or is it unnecessary?). The evaluator reads the forward direction first: a row claimed to be met but with no evidence is the first finding.

Worked example: from an asset to evidence, end to end

Let's fill in how the chain is built, from start to finish, through a single asset.

1. Asset (from Week 1's asset table).

Field	Value
Asset	The user's session token
Location	Client memory; carried in the HTTP header on every request to the server
Class	C (confidentiality) — if captured, the attacker acts as the user
Lifecycle	Generated at login → used on every request → invalidated at logout or timeout

2. Threat (from Week 1's threat model). An attacker positioned on the network (e.g., on the same public Wi-Fi) can listen on an unencrypted connection, capture the token, and use it in their own requests to act as the user (STRIDE: **Spoofing + Information Disclosure**, Week 1 terminology).

3. Requirement (this week's subject). We write what the system must do against the threat; we briefly reapply the six steps from Section 1: asset (session token) → threat (network eavesdropping) → goal (confidentiality + identity) → tool class (encrypted channel + server authentication) → state (in transit) → obligation (MUST).

CEN429-DT-01 — "The session token must be carried over a channel using TLS 1.2 or higher, with the server certificate validated."

4. Control (which section of the guide, what is done). In the guide's S10.3 "Transport security" section: "The client performs a TLS 1.2+ handshake on every connection to the server; the certificate chain and hostname are validated (see Week 10). No request is sent before the channel is established."

5. Verification (how it was tested). The security testing team tries to intercept the connection with a MITM (man-in-the-middle) tool: it verifies that (a) presenting an invalid/self-signed certificate gets the connection **rejected**, and (b) attempting a plaintext (non-TLS) connection gets **rejected**.

6. Evidence (where, produced by whom). The test tool's log output (`mitm_test_2026-11-03.log`), the CI run record ([#617](#)), and the TLS handshake packet capture taken during the connection (`handshake.pcapng`) are stored in the project repository's `evidence/week13/` folder, and are referenced by these file names in the compliance matrix.

Once these six steps are complete, the row in the compliance matrix fills in like this:

Id	Requirement	Status	Control	Verification	Evidence
CEN429-DT-01	The session token must be carried over TLS 1.2+ with certificate validation	Met	S10.3 "Transport security"	MITM test: an invalid certificate and a plaintext attempt are rejected	<code>mitm_test_2026-11-03.log</code> , CI #617 , <code>handshake.pcapng</code>

What would happen if **one** of the chain's six links were missing?

- If the asset were undefined → the requirement would have no answer to "which data."
- If it weren't linked to a threat → the requirement would look arbitrary; there'd be no answer to "why does this requirement exist?"
- If no control were written → the "met" claim would be unsupported.
- If no verification were defined → there would be no answer to "how was it tested?"
- If there were no evidence → the row would be the evaluator's first finding (see the rule below).

Backward traceability: finding a superfluous control

We saw forward traceability above. **Backward traceability** asks the opposite: does every control written in the guide rest on a requirement? Say the guide has this sentence: "The application applies a random delay at startup." Which requirement does this control link to? If the team cannot show that it rests on CEN429-AP-05 ("a defined response must be given when a debugger or hook is detected"), then either (a) the threat model is missing a threat (why was the delay needed, which attack does it slow down?), or (b) the control is **unnecessary** and should be removed from the guide. Backward traceability is the way to clean out "seemed like a good idea" controls that pad the guide but meet nothing; every page is a page the evaluator must read, and an unnecessary control lengthens the evaluation time.



Common mistake: skipping the chain's last link (evidence)

Teams often write the requirement, the control, and the verification method nicely, but leave the evidence column blank or wave it away with a generic phrase like "tested." "Tested" is not evidence to an evaluator; you need to answer **which** test, **when**, and **with what result**, with a concrete file/record/screenshot. A "met" row with no evidence is, as noted in the class-schedule box too, the first kind of finding the demo tool (01-uyum-matrisi) flags.



Rule

Every "met" row must have a **findable, named** reference in the evidence column: a file name, a CI run number, a test report section. Phrases like "exists," "tested," "checked" are not evidence, they are a **promise** of evidence; a compliance matrix demands evidence, not a promise.

The link between traceability and Week 12's assessment

The assessment process you saw last week (Week 12) is, in fact, the way this matrix is **read**: when an evaluator examines your project, they don't read random lines of code; they open the compliance matrix first, select the "met" rows, request evidence for each, and **independently verify** that evidence (Week 12's "independent verification" principle). The better your matrix is built, the faster and with fewer question marks the assessment proceeds; a poorly built matrix makes the evaluator stop at every row and ask "how do we know this?", lengthening the process.

Worked example: the full chain of the 'not met' status

The chain is built not only for "met" rows but for "not met" rows too — the difference is in the **last two links**.

Link	The "met" example (above)	The "not met" example
Asset	Session token	Server certificate
Threat	Network eavesdropping	MITM with a forged server certificate (if there's no certificate pinning)
Requirement	CEN429-DT-01	CEN429-DT-04 "The server certificate must be pinned"
Status	Met	Not met
Control	S10.3	— (not yet)
Verification	MITM test passed	—
Evidence	Test log	Residual risk: written into S16.4 as "certificate pinning not yet implemented, MITM risk open; planned for v1.1"

This is the expanded version of the `CEN429-DT-04` row we saw in the compliance-matrix table at the start of the section. In the "not met" state, the control and verification columns can stay blank, but the **evidence** column cannot — there, instead of real evidence, you write **where the residual risk is documented**. Leaving it blank gives the impression that the risk was forgotten.

Why 'matrix,' not 'list'?

A checklist only says "done/not done." The **compliance matrix** is called a "matrix" because it holds six dimensions together (id, requirement, status, control, verification, evidence): every row is a requirement, every column is a link in the traceability chain. If a cell is missing, it means that row is **unverifiable** in that dimension — a list can't show this, a matrix can.

The five finding types an evaluator writes most often

An evaluator reading a compliance matrix usually writes one of these five finding types; check these five before submitting your own matrix:

1. **A "met" with no evidence.** The most common finding; this section's main rule.
2. **Inconsistent ids.** The same requirement is given different ids in different places in the guide (e.g., `CEN429-DR-01` in one place, `DR01` in another) — traceability can no longer be checked automatically with software tools.
3. **An unlinked control (backward traceability).** The guide has a control but doesn't state which requirement it meets.
4. **A "not applicable" with no rationale.** The mistake covered in this week's Sections 0 and 7.
5. **Generic/copy-pasted evidence.** The mistake seen in the worked example in Section 8.

The demo tool `01-uyum-matrisci` automatically scans only for the first type (a "met" with no evidence); the other four are, for now, found manually, by reading like an evaluator.

Carrying traceability into the code level too

The compliance matrix is usually kept as a separate document (a table), but it is also possible — and common in large projects — to carry traceability **into the source code itself**: adding a comment line like "this function meets CEN429-DR-01" above a critical function lets the next developer reading the code (or yourself, a year later) instantly answer "why is this line here?" This is a small, code-side reflection of the requirement block from Section 3:

Traceability comment in code (example)

```
/* CEN429-DR-01: the vault file at rest is encrypted with AEAD. */  
int kasa_dosyasi_sifrele(const unsigned char *anahtar, ...) {  
    ...  
}
```

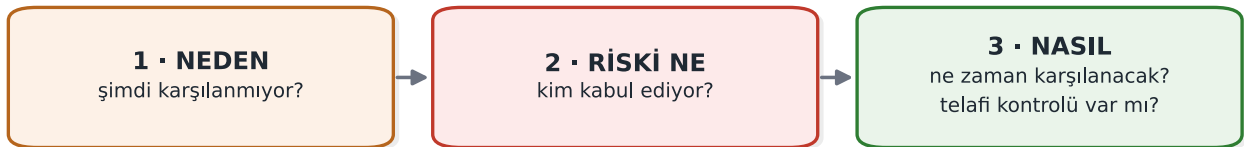
A comment like this does not replace the evidence column (the evidence must still be a test/record); but it visually links the requirement to the code. Someone doing a code review can scan for these comments when looking for the code that meets a requirement.

3. The requirement-block pattern and deferred requirements

The requirement block in the guide

Devrederken yanıtlanacak üç soru

devretmek serbest, SESSİZ devretmek değil



Kayıt ve onay olmadan atlanan gereksinim = gizli açık, sahipsiz risk.

Gereksinim bloğunun alanları

her gereksinim bu altı alanla yazılır

Kimlik (ID)	CEN429-DR-01 gibi benzersiz
Ölçülebilir ifade	ne yapılacak
Kaynak / gerekçe	hangi tehdit ya da standart
İlgili varlık	neyi koruyor (S5)
Doğrulama yöntemi	hangi test kanıtlayacak (S16)
Durum	karşılandı / devredildi / karşılanmadı

Alanlardan biri boşa gereksinim izlenebilir değildir.

In security guides that pass certification, every section opens with the requirements that fall under it. Every requirement is written in this pattern:

```
[Standard/scheme] Requirement id – STATUS
The requirement's text (from the standard).
A description of how the product meets this requirement; reference to the relevant control cards and sections.
```

In the field, in a mobile payment library's guide, every section – "data at rest," "data in use," "data in transit," "reporting," "crypto methods and keys" – opens with the two card schemes' requirements on that topic, and each one is marked **met** or **deferred to the parent application**. The strength of this pattern is that the evaluator can go from requirement to evidence in a single glance.

Deferred requirements: the boundary of responsibility

A software component (e.g., a library embedded inside other applications) cannot meet every requirement on its own. Installing the application, updating it, showing a notification to the user, or protecting its own source code is the job of the application that uses the component, not the component itself. These requirements are **deferred** and written explicitly in the guide's "security expected from external components" section.

Deferred to whom?	Typical requirement	How is it written in the guide?
Parent application	Secure installation and update	"The application's distribution and update mechanism is the parent application's responsibility."
Parent application	Reporting a compromise to the backend and the user	"The library reports the attack it detects to the parent application (method: ...); relaying it to the backend and the user is the parent application's job."
Parent application	Hiding and protecting its own source code	"The library only protects its own code; the parent application must take its own protective measures."
Parent application	Protecting assets passed to the library as parameters	Parameters such as the server address, certificate, and pinning values are listed one by one
Operating system	Application isolation, the minimum supported version	"The minimum supported operating system version is ...; there is no security expectation from the operating system beyond this."
Hardware	Secure element, TEE	If not used, "there is no expectation from hardware" is stated explicitly

⚠ Deferring is not the same as ignoring

For every deferred requirement, you write **to whom**, **why** it was deferred, and **how** the other party can meet it. Leaving responsibility vague turns into a gap where both sides assume "the other one is doing it." In the field, a library is seen weighing two options for reporting a compromise to the parent application (wiping all data and writing a flag file, or letting the parent application delete its own file), discussing their pros and cons, and choosing one with a rationale; that discussion is written into the guide.

For your project: S14 "Assumptions and deferred requirements"

Even if your project is a standalone application, it has assumptions: the supported operating systems, that the user is not an administrator, that the build environment is trustworthy, that the server is protected by a separate team. Writing these down **explicitly** helps the evaluator correctly understand the scope, and answers the question "why is this threat out of scope?" in advance.

Worked example: filling in the requirement block from start to finish

Let's fill in the pattern with a real example; this is the **written-up-in-the-guide** form of the CEN429-DR-01 chain from Section 2:

```
[CEN429-DR] CEN429-DR-01 – Met
Requirement: Class-C sensitive data at rest must be protected with authenticated encryption (AEAD).
Compliance: The local vault file is encrypted with AES-256-GCM (see S7.2 "Vault file encryption").
The encryption key is derived from the device's secure storage unit (Android Keystore / iOS
Keychain) and is never stored as plaintext outside memory. Verification: a unit test
(test_butunluk.c) checks that changing a single byte of the encrypted file and then decrypting it is
rejected. Evidence: CI run record #482, test output evidence/week13/test_butunluk_ci482.log.
```

Every line in the block serves a purpose: the **header line** ([Family] Id – Status) tells the evaluator's eye quickly where to go; the **Requirement** line repeats the "what" coming from the standard or the threat model (copy-paste, not reinterpreted); the **Compliance** paragraph describes the "how," with concrete file and section names; the **Verification** and **Evidence** sentences complete the last two links of the chain from Section 2.

Now let's fill in the same pattern for a **deferred** requirement — this time the parent application meets it, not the product itself:

```
[CEN429-AP] CEN429-AP-07 – Deferred (to the parent application)
Requirement: The application must be installed and updated securely.
Compliance: The library does not have its own distribution/update mechanism; this is the
responsibility of the parent application that packages the library. The recommended method for the
parent application's developer: (1) use the official app store's signature verification, (2) if a
custom update channel is used, sign update packages with the signing scheme described in Week 6.
To whom: the parent application's developer. Why: the library does not access installation at the
network/file-system level, it is only called as an API. How: one of the two methods above, detailed
in the guide's "Security expected from external components" section.
```

Notice that the "deferred" block also has a **Compliance** field — but this field answers "how must the other party do it, and why aren't we doing it ourselves" rather than "how did we do it." Deferring does not mean leaving that field blank; on the contrary, it demands **more** explanation, because you have to prove where the boundary of responsibility lies.

Three assumptions commonly overlooked when filling in S14

Projects usually forget these three assumptions; check for them when adding to S14:

1. **Is the build environment trustworthy?** The assumption "an attacker cannot access the build server" is usually not written down, but it is a foundation every project rests on (Week 6, supply-chain security).
2. **Is the user's device not rooted/jailbroken?** If it is, which controls become invalid (e.g., secure storage may no longer be secure) must be stated explicitly.
3. **Is the server side protected by a separate team?** If so, it must be stated clearly which requirements (e.g., server-side access control) are **out of scope** for this project; otherwise the evaluator will expect them from this project too.



Common mistake: filling the compliance text with marketing language

A sentence like "Our product uses industry-standard encryption and protects your data securely" is a typical mistake written in the compliance field that makes nothing **verifiable**. Which file, which section, which test — without any of these the sentence looks nice but is useless to the evaluator; the matrix demo tool (02-gereksinim-kalite) flags exactly this kind of "vague/unverifiable" wording as WEAK for precisely this reason.



Rule

The compliance text must always end with a **concrete reference**: a file name, a code section, a guide subsection, a test name. If a sentence has marketing adjectives ("strong," "industry-standard," "best practice") but no reference behind them, rewrite that sentence.

Why deferring is not a bad thing (rationale)

New developers perceive the word "deferring" as a weakness and try to meet everything in their own component — this usually leads to the component trying to take on tasks it **cannot** do (installation, OS-level isolation, UI warnings), or worse,

silently skipping them without handling them at all. In the certification world, **drawing boundaries explicitly** always scores better than **leaving boundaries vague**: an evaluator trusts a guide that knows who was deferred what; they do not trust a guide that says "we handle everything ourselves" but cannot produce evidence.

Worked example: the requirement block in the 'not met' state

Some requirements are neither met nor deferred yet — the project has consciously decided to carry this risk. The block's form in this state:

```
[CEN429-DT] CEN429-DT-04 - Not met
Requirement: The server certificate must be pinned (certificate pinning).
Status: Not currently implemented; only standard TLS certificate chain validation is performed
(CEN429-DT-01 is met, but pinning is absent). Residual risk: if an attacker manages to obtain a
fraudulent certificate from a trusted root certificate authority, MITM may become possible (see
S16.4). Planned fix: pinning will be added in v1.1 (see project plan, task #217).
```

The "not met" block fills in the **Status** and **Residual risk** fields instead of **Compliance**; this honestly states that the requirement will be met in the future but is not met right now. Not writing it at all (removing the row from the matrix) is worse: the evaluator finds the requirement themselves and asks "why did you skip this?"

A hard deferral decision: when does hardware take on a requirement, and when doesn't it?

"If there is no hardware secure element (secure element/TEE), a requirement cannot be deferred to hardware" — this simple rule is sometimes applied incorrectly. Take, for example, the requirement "the key must be protected against physical attack":

- If the device **has** a secure element: the requirement can be deferred to hardware; the guide states "key generation and storage are deferred to the device's secure element, the application only calls the API."
- If the device **does not have** a secure element: the requirement cannot be deferred, because there is no party to defer it to. In this case the product itself must implement a software-based control (e.g., white-box cryptography, code obfuscation), or mark the requirement "not met" and write the residual risk.

A common mistake is to write "deferred to hardware" in the second case too — since there is no concrete party to defer to, this is an **invalid deferral** and is rejected by the evaluator.



Common mistake: not asking whether the other party in a deferral actually exists

A team that writes "deferred to the operating system" or "deferred to hardware" may not check whether that version of the operating system or that hardware actually has a feature that meets this requirement. E.g., saying "application isolation is deferred to the operating system" is an empty claim if the minimum supported operating system version does not provide this isolation.



Rule

Before deferring a requirement, verify that the other party (parent application, operating system, hardware) can **actually** meet it — even for the minimum supported version/hardware. Deferring is not "hoping someone will do it"; it is "knowing who will do it, with what concrete mechanism."

? How does an evaluator test this?

An evaluator specifically looks for "not met" rows; these are the project's **indicator of honesty**. A matrix with no "not met" or "not applicable" rows at all, where everything looks "met," raises suspicion in an experienced evaluator – real projects almost always have at least a few gaps or deferred requirements.

4. Common Criteria (ISO/IEC 15408)

Common Criteria (CC) is the international standard for the security evaluation of IT products. The evaluation methodology is defined in a separate document (CEM, ISO/IEC 18045). Thanks to the Common Criteria Recognition Arrangement (CCRA), a certificate issued in one country is also recognised in the other countries that are party to the arrangement.

Ortak Kriterler kavramları

EAL4+ her zaman 'daha güvenli' demek DEĞİLDİR

PP (Protection Profile)	ürün SINIFI için standart gereksinim kümesi
ST (Security Target)	BELİRLİ ürünün güvenlik hedefi
SFR	ürün NE yapacak (fonksiyonel)
SAR	buna NE KADAR güvenilecek (güvence)
EAL	güvence DERİNLİĞİ — güvenlik miktarı değil

Kapsam (ST/TOE) darsa yüksek EAL az şey kapsar. Kapsamı bilmeden EAL karşılaştırılmaz.

Basic concepts

Concept	Meaning	Its counterpart in this course
TOE (Target of Evaluation)	The product being evaluated and its documents; uniquely identified	The target of evaluation (Week 1, S0/S1)
ST (Security Target)	A product-specific security-target document: threats, assumptions, security objectives, requirements	Your security guide
PP (Protection Profile)	A common requirement set for a product class (e.g., mobile device, firewall); products claim to conform to it	The course's requirement families
SFR (Security Functional Requirement)	Functional security requirements; chosen from a standard catalogue (e.g., FCS cryptographic support, FDP user data protection, FIA identification and authentication)	Functional requirements
SAR (Security Assurance Requirement)	Assurance requirements (e.g., ADV development, ATE testing, AVA vulnerability assessment)	Assurance requirements
EAL (Evaluation Assurance Level)	Predefined SAR packages: from EAL1 to EAL7	—

EAL levels

Level	Name	Typical use
EAL1	Functionally tested	Situations with low threat
EAL2	Structurally tested	Basic assurance
EAL3	Methodically tested and checked	Medium level
EAL4	Methodically designed, tested, and reviewed	The most common upper level for commercial products (operating systems, network devices)
EAL5	Semi-formally designed and tested	Smart cards and secure elements (often EAL5+ / EAL6+)
EAL6	Semi-formally verified design	High-risk environments
EAL7	Formally verified design	Very high-risk, small systems

A higher EAL does not mean the product is "more secure"; it means the evaluation was carried out **more deeply**. An EAL4 product can be less secure than an EAL2 product; what matters is the scope of the security objectives and threats in the ST. The "+" mark (EAL4+) shows that additional assurance components were added to the package; the most commonly added one increases the depth of the vulnerability assessment.

Vulnerability assessment and attack potential

CC's vulnerability-assessment component (AVA_VAN) determines what **attack potential** the product must withstand from an attacker: Basic, Enhanced-Basic, Moderate, and High. Attack potential is scored in the CEM with five factors: **elapsed time, expertise, knowledge of the product, window of opportunity, and equipment**. The sum of the scores shows which potential level a successful attack requires; the product must withstand attackers below the level it claims in the ST. Adapted versions of the same idea are used in the evaluation of payment products as well.

Worked example: filling in an ST skeleton for your project

Writing a security target (ST) from scratch can look intimidating; it is actually a **reorganization** of the documents you've gathered throughout the term. Below, we fill one in for a sample mobile-payment component in three steps.

Step 1 — Define the TOE (what we are evaluating, what we are not).

TOE: the "CEN429-Pay" library, version 1.0, Android ARM64 build only. **Out of scope:** the parent application's user interface, server-side components, the operating system kernel.

When defining the TOE, writing what's **out of scope** is just as important as writing what's in scope — it connects directly to the "deferred requirements" idea from Section 3: everything left out of scope reappears in the ST as an "assumption."

Step 2 — List the threats and assumptions (from Week 1's threat model).

Code	Threat/Assumption
T.EAVESDROP	An attacker positioned on the network eavesdrops on traffic
T.TAMPER	An attacker with physical access to the device modifies the application's memory/files
A.PLATFORM	The operating system correctly enforces application isolation (assumption, not proven)

Step 3 — Map the security objectives to the corresponding SFR families.

Threat	Security objective	SFR family	Course requirement
T.EAVESDROP	Data in transit must stay confidential and intact	FCS (cryptographic support), FTP (trusted path/channel)	CEN429-DT-01
T.TAMPER	The application must check its own integrity	FPT (protection of the TSF)	CEN429-AP-04
—	The user's/server's identity must be verified	FIA (identification and authentication)	CEN429-ID-02

This table is the **heart** of the ST: every row is a bridge between "why this requirement exists" (threat) and "what evidence is requested" (the SFR family, later tested against a SAR). Choosing the target EAL is the last step: for this project the team can choose, as a **target**, an assurance depth resembling EAL4, "the most common upper level for commercial products" (without actually undergoing a real CC evaluation, just to build their own internal discipline to that level).

⚡ Common mistake: leaving the TOE boundary vague

A definition like "TOE = our application" leaves open the evaluator's question "what does your application include, is the server included, are third-party libraries included?" If the boundary is vague, two teams can read the same ST and understand different scopes; this blocks the evaluation process from the very start.

✓ Rule

A TOE definition always lists **both what it includes and what it explicitly excludes**; ideally it is backed by a boundary diagram (which components this product covers, and through which interfaces it talks to the outside).

? How does an evaluator test this?

The evaluator first checks that every threat in the ST links to at least one security objective, and that every objective links to at least one SFR (they want a mapping like the table above). A threat with no link, or an SFR with no counterpart, is the first question asked.

The origin of CC: why do several countries use the same standard?

Before Common Criteria, the US, Europe, and Canada each had their own separate evaluation criteria (the Orange Book/TCSEC in the US, ITSEC in Europe, CTCPEC in Canada). Because each country certified against its own criteria, a product certified in one country had to be **re-evaluated** in another. Common Criteria merged these three approaches into a single international standard (ISO/IEC 15408); thanks to the CCRA (Common Criteria Recognition Arrangement), a certificate from one country is recognised in the other party countries **without** needing re-evaluation. This is much like different countries switching from different electrical-outlet standards to a single common one: the cost of compliance drops, and comparison becomes easier.

Why SFRs are not invented, but chosen from a catalogue

The relevant part of Common Criteria has a predefined SFR catalogue (families such as FCS, FDP, FIA, FAU, FMT, FPT, FTP, and their subcomponents). When writing an ST, you **choose from this catalogue** instead of inventing a new SFR; if a need has no exact counterpart in the catalogue, it is separately defined and justified as an "extended component." The reason for this is **consistency**: if two different products' STs use the same catalogue, an evaluator can compare them; if every product invented its own terminology, comparison would be impossible.

? How does an evaluator test this?

If they see an SFR in an ST that doesn't belong to the catalogue – an "invented," non-standard code that doesn't fit the catalogue – the evaluator checks whether it was properly defined as an extended component (dependencies, rationale, evaluation method); an improperly "invented" component can invalidate the ST.

Choosing an EAL: a practical decision guide

Students often ask "which EAL should we write for our project?" Even if you're not going to undergo an actual CC certification (and you won't, for the term project), it's useful to use the EAL concept as an **internal discipline target**:

Your project's situation	Suitable target	Why
Small, single-developer, course project	EAL1–EAL2-like discipline (functional test + structural test)	No comprehensive documentation infrastructure
Team project, has code review and CI	EAL3–EAL4-like discipline (methodical design, testing, review)	The processes you built throughout this course (Week 6 CI, Week 12 review) correspond exactly to this level
A high-risk component such as payment/authentication	EAL5+–like discipline (semi-formal design)	In field examples, smart-card and secure-element products are evaluated at this level

This table does not promise a CC certificate; it only offers a rough comparison for the question "how rigorous a documentation and testing process do we need."

Reusing a PP for an ST

If a PP already exists for a product class (e.g., the "mobile device" or "firewall" PPs), a team developing a new product does not write its ST from scratch; it takes the PP's requirements, adapts them to its own TOE, and says "we claim conformance to this PP." This is the same logic as the course's requirement families in Section 8: a PP is a **template**, an ST is that template **applied to a specific product**. For your own project, the course's requirement families can be thought of as an informal "mini PP."

Comparing the CC process with Week 12's assessment process

The independent-assessment steps you saw last week (Week 12) — document review, code review, repeating tests, vulnerability analysis — are, in fact, the evaluation activities CC defines in the CEM, **scaled down** to this course's level: in CC, ADV (development) documentation is reviewed, ATE (testing) results are **independently repeated**, and AVA (vulnerability assessment) analyses attack potential. Your one-session assessment last week imitates, on a small scale, a process an accredited lab spends months on — the same discipline, a different scale.

5. FIPS 140-3: validating cryptographic modules

FIPS 140-3 is the US NIST's security standard for cryptographic modules; its content is based on the international ISO/IEC 19790 standard. Validation is performed by accredited laboratories under the **Cryptographic Module Validation Program** (CMVP), jointly run by NIST and Canada. It replaces FIPS 140-1/140-2, mentioned in the textbook's Recipe 11.18; as of 2026, FIPS 140-2 certificates are being moved to the historical list.

FIPS 140-3 güvence düzeyleri

yukarı çıktıkça fiziksel koruma beklentisi artar



'FIPS doğrulanmış kütüphane kullanıyoruz' demek yetmez: doğru kip ve yapılandırma şart.

Security levels

Level	Standout requirement	Typical module
1	Approved algorithms, production-grade components; no physical protection required	Software crypto libraries
2	Tamper evidence (seals, coatings), role-based authentication	Secure tokens, some hardware modules
3	Tamper resistance and response (erasing the key), identity-based authentication	Network-attached HSMs
4	Full protection against environmental (voltage, temperature) attacks	Environments where physical attack is expected

What a module must do under FIPS 140-3 (summary)

- Use only **approved algorithms** (AES, SHA-2/3, HMAC, RSA, ECDSA, approved random generators; post-quantum algorithms from 2024 onward) in **approved mode**; each algorithm must also have a certificate from the algorithm validation program (CAVP).
- Perform **self-tests** on power-up and conditionally (known-answer tests, integrity checks); on failure, enter an error state and provide no cryptographic service.
- Define **roles and services** (user, operator); document which critical security parameter (key) each service accesses.
- Generate, input/output, and zeroize** critical security parameters securely once their job is done.
- Publish a **security policy** document.

**What does FIPS validation NOT mean?**

An application "using a FIPS-validated library" does not mean the application itself has been validated. Misusing the library (nonce reuse, leaving the key in memory, turning off validation) brings back all the mistakes we saw in Weeks 3 and 10. If your project says "we use OpenSSL's FIPS provider," write down separately which requirement this satisfies and which it does not.

Worked example: auditing a module against FIPS 140-3 Level 1

Let's apply the "summary" list above to a real (but simplified) project. Say your project's crypto layer works like this: it uses OpenSSL's standard `EVP` interface, encrypts with AES-256-GCM, generates keys from `getrandom()` (Week 3), and silently **falls back to a default key** on error. Let's check off the Level 1 expectations one by one:

Level 1 expectation	Status in this project	Why
Only approved algorithms, in approved mode	Partial	AES-GCM is an approved algorithm/mode, but this cannot be claimed if the OpenSSL build in use has no CAVP certificate
Power-up and conditional self-tests	Not met	The project doesn't run its own self-tests, it only trusts the library
Roles and services defined	Not met	The code does no role separation at all
Secure input/output and zeroization of critical parameters	Not met	"Falling back to a default key" on error is the exact opposite behaviour: using a predictable key instead of zeroizing
Security policy document	Not met	No such document exists

This table shows the project is far even from FIPS 140-3 Level 1, and that the most critical gap is "silently falling back to a default key on error" — this is the exact **opposite** of the zeroization principle, and a violation of the "if the random generator fails, stop, don't continue" rule we saw in Week 3.

**Common mistake: saying 'we're on the FIPS-approved algorithm list,' without a certificate**

AES and SHA-2 **being** FIPS-approved algorithms does not mean the **build** you use has been validated. In CMVP, every module has its own certificate number and scope (which operating system, which compiler, which version). The sentence "we use AES, so we comply with FIPS" collapses the moment you're asked which concrete certificate it rests on.

**Rule**

Behind every FIPS claim there must always be a **CMVP certificate number and scope**; if there isn't, honestly narrow the claim to "we use FIPS-approved algorithms (not a validated module)."

? How does an evaluator test this?

The evaluator first compares the version and build flags of the crypto library used against CMVP's public list; then, in the source code, they look for a power-up self-test call, a stop-on-error behaviour, and key zeroization (memory-wiping) calls. If even one is missing, the claim is considered invalid.

The steps of FIPS validation (roughly)

Getting a cryptographic module validated under FIPS 140-3 is not a one-time approval, it is a **step-by-step process**:

1. The **developer** prepares the module with approved algorithms, together with a security policy document.
2. **CAVP** (Cryptographic Algorithm Validation Program) tests that each algorithm (AES, SHA-2, HMAC, RSA...) is correctly implemented, **separately**, and issues an algorithm certificate.
3. An **accredited laboratory** tests the module against FIPS 140-3's physical, logical, and process requirements.
4. **CMVP** reviews the laboratory's report and publishes the module certificate; the certificate states the module's **exact version and the platform it runs on**.
5. If the module or platform changes (e.g., a new operating system version), the certificate may require **revalidation**.

This sequence shows why the sentence "we use AES" is insufficient as a FIPS claim: Step 2 covers only the algorithm, while Steps 3–4 cover the **whole module** (algorithm + self-tests + roles + zeroization). The certificate is not issued until Step 4 is complete.

What does a CMVP certificate state?

A real CMVP certificate generally states: the module's name and version, which operating system/hardware it was tested on, which security level it was validated at, and the CAVP certificate numbers of the validated algorithms. When a project says "we use the FIPS-validated library X," it must actually check that the **exact version and platform match** in that certificate is identical to how it's used in their own product; if the certificate was issued for Linux x86-64, an Android ARM build of the same library does **not** automatically inherit that certificate.

Don't confuse FIPS levels with CC EAL

Because both are expressed with small numbers (1 to 4, or 1 to 7), students confuse these two scales:

	FIPS 140-3 level (1–4)	CC EAL (1–7)
What does it measure?	The physical/logical protection of a cryptographic module	The evaluation depth of the whole product
Scope	Only the crypto module	The whole TOE the product defines in the ST
What changes as the number goes up?	More resistance to physical attack	A deeper, more rigorous evaluation (not "more secure")

A product can claim both "FIPS 140-3 Level 2" and "CC EAL4" at the same time — these are not mutually exclusive, they measure different dimensions. Neither substitutes for the other.

**Common mistake: not noticing a platform/version mismatch**

A team says "OpenSSL's FIPS provider is validated, and we use OpenSSL," but their own compiled version, build flags, or target platform are **not the same** as what's in the certificate. FIPS validation is tied to the exact tested binary; even with identical source code, a differently compiled binary is considered unvalidated.

**Rule**

When writing a FIPS claim, verify that the certificate's **module version, platform, and build configuration** match your project's usage exactly; if they don't match, don't write the claim.

Three questions to ask before claiming FIPS compliance

Before writing a FIPS claim in your project, ask these three questions in order:

1. **Which module** has the certificate — the operating system's crypto library, or a separate library embedded by the application?
2. Does this certificate cover the **exact version and platform** you use (the "platform/version mismatch" mistake above)?
3. Does **every** algorithm call you use run in the module's "approved mode," or are you using one of the module's non-approved helper functions?

If you cannot answer "yes" to all three questions, narrow your claim down to a smaller, more honest sentence such as "we use FIPS-approved algorithms" (as seen in the tip box at the start of this section).

The connection to this week's demo

This week's `01-uyum-matrisi` demo automatically finds "met" rows with no evidence; writing a FIPS claim without evidence is a mistake of the same class. If you write a sentence like "we comply with FIPS 140-3" in your own project, treat it as a **separate row** in your compliance matrix, and write the certificate number in the evidence column (or the honest "no certificate, we only use approved algorithms").

6. ETSI, GSMA, EMVCo, PCI, and OWASP: sector-specific requirement sets

ETSI EN 303 645: the consumer internet of things

Sektöre özgü gereksinim setleri

hepsi aynı soruyu sorar: hangi gereksinim, hangi kanıtla karşılandı?

OWASP MASVS / MASTG	mobil uygulama — bu dersin projesine en yakın
ETSI EN 303 645	tüketici IoT
EMVCo / PCI	ödeme kartı ekosistemi
GSMA	mobil operatör / SIM

Projenizde MASVS seçin: somut, kontrol edilebilir maddeler + MASTG test rehberi.

This standard from the European Telecommunications Standards Institute (ETSI) defines a **baseline security line** for internet-connected consumer devices, and has become the basis for regulations in Europe. Its thirteen provisions read like a brief summary of the topics we've covered throughout the term:

#	Provision	Its counterpart in this course
5.1	No universal default passwords	Week 1, authentication
5.2	Implement a means to manage reports of vulnerabilities	Week 12
5.3	Keep software updated	Week 1, update signing, Week 10
5.4	Securely store sensitive security parameters	Weeks 3, 10, 11
5.5	Communicate securely	Weeks 3, 10
5.6	Minimise exposed attack surfaces	Week 1, Week 5 minimization
5.7	Ensure software integrity	Week 6
5.8	Ensure that personal data is protected	Week 3, masking
5.9	Make systems resilient to outages	Availability
5.10	Examine system telemetry data	Week 2, audit logging
5.11	Make it easy for users to delete user data	Week 3, destruction
5.12	Make installation and maintenance of devices easy	Psychological acceptability
5.13	Validate input data	Weeks 4, 5

GSMA

GSMA, the association of mobile operators, runs the **IoT Security Guidelines** for IoT devices and services, the **Security Accreditation Scheme (SAS)** for SIM/eSIM manufacturing and subscription-management facilities, and an assessment scheme for network equipment (**NESAS**). SAS is an example of auditing the **manufacturing and management facility**, not the product: physical security, personnel, processes, key management.

EMVCo and PCI: payments

Document	Scope	Standout requirement families
EMVCo Software-Based Mobile Payment security requirements	Payment solutions that don't use the phone's secure element (emulating the card in software)	Application protection, identification/authentication and device binding, asset protection, data at rest/in use/in transit, reporting, crypto and keys, the development process
PCI MPoC	Using commercial off-the-shelf phones and tablets as payment-acceptance terminals	The application and the backend together; monitoring and attestation services
PCI DSS	Environments that process card data	Data protection, access control, logging, testing and scanning
PCI Software Security Framework	Secure development of payment software	Secure lifecycle, the software's own security

Notice the EMVCo family list above: throughout the term, every week's topic corresponded to one of these families. The course's requirement families (Section 8) are adapted from this structure.

OWASP MASVS: a practical checklist

Although it doesn't issue a formal certificate, OWASP's **Mobile Application Security Verification Standard** (MASVS) is the most widely used open checklist for mobile applications. Its control groups: storage (MASVS-STORAGE), crypto (MASVS-CRYPTO), authentication (MASVS-AUTH), network (MASVS-NETWORK), platform (MASVS-PLATFORM), code (MASVS-CODE), resilience (MASVS-RESILIENCE), and privacy (MASVS-PRIVACY). Testing methods are given in a separate guide (MASTG). The resilience group covers the topics of Weeks 4, 5, 6, and 9.

Worked example: the same requirement's counterpart across four standards

Although the standards were written independently of one another, they generally express the **same security need** in different words. Let's trace two requirements across four standards.

Example 1 — Encryption of sensitive data at rest

Standard	This need's counterpart
Common Criteria	SFR family FCS (cryptographic support, e.g., FCS_COP cryptographic operation) + FDP (user data protection) components, defined specifically for this data in the ST
FIPS 140-3	The Level 1 requirements of the module performing the encryption: approved algorithm, self-test, key zeroization
ETSI EN 303 645	Provision 5.4 "Securely store sensitive security parameters" (in the table at the start of this week)
OWASP MASVS	The MASVS-STORAGE group (protection of local storage)
The course's family	CEN429-DR-01

Example 2 — Preventing default/weak authentication

Standard	This need's counterpart
Common Criteria	SFR family FIA (identification and authentication) components (e.g., authentication failure handling)
FIPS 140-3	The role-based authentication that Level 2 requires
ETSI EN 303 645	Provision 5.1 "No universal default passwords"
OWASP MASVS	The MASVS-AUTH group
The course's family	CEN429-ID-01 / CEN429-ID-02

What these two tables show is this: if a product has already prepared the evidence to meet the "data at rest encryption" requirement for an EMVCo/PCI document, most of that same evidence can be **reused** in a MASVS-STORAGE audit too — because both ask for the same underlying need (encrypted, zeroizable, non-leaking storage). But "reusable" doesn't mean "automatically met"; each standard's own criteria (e.g., MASVS also requires the key to be protected via the operating system's secure-storage API) must be verified **separately**.

How do you choose and prioritise among standards?

A project does not try to comply with **all five** of these sources (CC, FIPS, ETSI, GSMA/EMVCo/PCI, MASVS) at once; which standard applies depends on the product's **type** and **market**:

- If you're selling a consumer IoT device → ETSI EN 303 645 (a legal requirement in some markets).
- If you're developing a mobile application → OWASP MASVS (the de facto industry standard, even without a certificate requirement).
- If you're a payment-processing application/library → EMVCo and/or PCI documents (the card schemes' requirement).
- If you're selling to a government/defence buyer → a Common Criteria certificate (often a tender requirement).
- If you're selling a cryptographic module/library → FIPS 140-3 (mandatory for US federal buyers).

The course's requirement families (Section 8) extract the common denominator of these five; you're expected to determine which sector your own project falls into and state explicitly, in the S1 (sources) section, which standards you're basing it on.



Common mistake: automatically counting compliance with one standard as compliance with another

The sentence "we use a FIPS-validated module, so we also comply with MASVS-CRYPTO" is dangerous. FIPS only validates that the **algorithm** is correctly implemented; MASVS-CRYPTO also asks **where** the keys are stored, whether the application hard-codes its own key, and whether key generation uses the platform's secure random generator. Even if a module itself is correct, its **usage** can be wrong (just as in the "What does FIPS validation NOT mean?" box in Section 5).



Rule

Use the overlap between standards to **speed up evidence gathering** (reference the same test/document in more than one matrix), but check off each standard's checklist **separately**. Do not turn a compliance-matrix row into an unevidenced "met" by saying "we already met it under standard X."

Worked example: applying ETSI EN 303 645 provision 5.13 to your project

Provision 5.13 says "validate input data"; let's map this one-to-one to the input validation/sanitization topic we saw in Weeks 4 and 5:

1. **Provision:** "Device software must prevent the corruption of data or the crashing of the device by validating data coming from user interfaces or the network."
2. **The course's requirement:** This provision can be adapted to the course's `CEN429-AP` family or to a new input-validation family: "Every input coming from the network or the user must pass a length, type, and range check before being processed (see Weeks 4–5)."
3. **Control:** The guide lists which parsing functions perform boundary checking.
4. **Verification:** Fuzzing or boundary-value testing shows that invalid input is safely rejected (links to Week 4's fuzzing demo).
5. **Evidence:** The fuzz-test run report, a summary showing the crash count is zero.

This example shows how the ETSI provision's abstract sentence turns into a concrete requirement, control, and test for the course — just like the traceability chain in Section 2.

A short note on GSMA NESAS and PCI DSS levels

- **GSMA NESAS** (Network Equipment Security Assurance Scheme) evaluates mobile network equipment manufacturers' secure development processes and their products' security tests; unlike SAS (which is facility-focused), it is both **process-** and **product-**focused.
- **PCI DSS** divides organisations that process card data into levels based on transaction volume (the highest-volume organisations are subject to the strictest audits); we don't go into level details within this course, but the idea that "audit frequency/depth changes with size" is similar to CC's EAL logic: the bigger the risk, the deeper the assurance must be.

A comparative summary table of the sector sets

Set	Which body	Which product type	Is there a formal certificate?
ETSI EN 303 645	ETSI (Europe)	Consumer IoT devices	A legal declaration of conformity in some countries
GSMA IoT Security Guidelines / SAS / NESAS	GSMA	IoT device/service, SIM facilities, network equipment	Yes for SAS/NESAS, no for the guidelines
EMVCo software-based mobile payment	EMVCo	Software payment solutions emulating a card	Yes (EMVCo's list)
PCI MPoC / DSS / SSF	PCI SSC	Payment acceptance, card data processing, payment software	Yes
OWASP MASVS	OWASP (non-profit community)	Mobile applications	No (de facto standard, no formal certificate)
Common Criteria	ISO/IEC, multinational (CCRA)	Any kind of IT product	Yes (internationally recognised)
FIPS 140-3	NIST (US)	Cryptographic modules	Yes (CMVP)

This table is this week's answer to the question we asked at the very start of the course — "who certifies what, and how?" — and serves as a summary closing out Section 6.

Can a product belong to more than one set at once?

Yes, and in fact most real products are. For example, a mobile payment application can simultaneously target: the card scheme's EMVCo requirements (because it works with card data), OWASP MASVS (because it's a mobile application), and, indirectly through the crypto library it uses, FIPS 140-3 (because that library may be FIPS-validated). A Common Criteria certificate is generally **not needed** for this product, because CC is requested mostly by government/defence buyers and is rarely required for consumer payment applications. The same analysis applies to your own project: the answer to "which market are we selling into, and which standard does that market ask for?" determines which standards you write into S1 — you don't have to write all of them, only the **relevant** ones.

7. Carrying requirements into the software plan and asset management

Reading a requirement set is the easy part. The real work is **turning it into the product's plan**. The steps:

Uyum matrisi: her satır bir kanıt ister

S17'nin iskeleti

Kimlik	Gereksinim	Durum	Önlem	Kanıt
DR-01	Beklemede AEAD	Karşılandı	S7.2	test çıktısı
AP-07	Güvenli güncelleme	Devredildi	S14.2	gerekçe
DT-04	Sertifika sabitleme	Karşılanmadı	—	kalan risk

Kanıt sütunu boş olan 'Karşılandı' satırı, KARŞILANMAMIŞ sayılır.

- Applicability:** for every requirement in the set, ask "does this apply to this product?" Those that don't apply are marked with their rationale (e.g., "the product doesn't use network connectivity, data-in-transit requirements don't apply").
- Responsibility:** applicable requirements are split into "we meet it" and "we defer it" (Section 3).
- Linking to assets:** every requirement is linked to the relevant assets in the asset table. The requirement "crypto keys must be erased once their job is done" requires the "creation → erasure" column to be filled in for **every** key in the asset table.
- Linking to threats:** every requirement must correspond to at least one threat; if it doesn't, either the threat model is missing something, or the requirement is unnecessary for this product.
- Control and verification:** a control card and a verification method (test, review, analysis) are planned for every requirement.
- Release plan:** which requirement will be met in which release is written into the project plan; those that cannot be met are written into residual risk.

Gereksinimi projeye aktarma kararı

her gereksinim için aynı soruları sor



Devredilen gereksinim de yazılır: kime, neden, karşı taraf nasıl karşılayacak.



How it's done in the field

In a library's guide, the asset table is a direct reflection of the requirements: for every asset, its size, origin, creation and destruction time, default value, and protection class (C, I, I+, N) are written down, because the requirements say "the confidentiality and integrity need for every asset must be defined" and "assets must be protected while being processed, transferred, and stored." An evaluator goes from the requirement to the asset table, from there to the security-shell matrix, and to the control cards.

Worked example: applying the six steps to a single requirement

Let's apply the six steps from the start of this section (applicability → responsibility → linking to assets → linking to threats → control and verification → release plan) end to end, through the `CEN429-CR-03` requirement ("keys must be erased once their job is done").

- 1. Applicability.** Does the project use a crypto key? Yes — there's a session key and a vault-file key. The requirement **applies**.
- 2. Responsibility.** The code that generates and uses the key is the project's own code; there's no reason to defer. The team decides **we meet it ourselves**.
- 3. Linking to assets.** The asset table (S5) has two key rows: "session key" and "vault-file key." Both need their "creation → use → erasure" column filled in; right now the "vault-file key" row has an empty erasure time. This **gap** is noticed right here.
- 4. Linking to threats.** Which threat requires this requirement? Week 1's threat model has the threat "an attacker with physical access to the device takes a memory dump"; if the key stays in memory after its job is done, this threat can be realized. The requirement **links** to this threat; if it couldn't be linked, either the threat model was missing something or the requirement was unnecessary for this product (item 4 above).
- 5. Control and verification.** Control: `sifreleme_bellek_sil()` is called immediately after the key is used (overwriting with zeros; it's made sure the compiler doesn't optimise this call away — linking to Week 9's protection rules). Verification: a memory-analysis tool (e.g., taking a memory dump and searching for the key's bytes) verifies that the key is **not** present in memory once its job is done.
- 6. Release plan.** For the session key, this control already exists in v0.9 (met). For the vault-file key it's missing; the team adds it to the v1.0 release plan and, until that day, writes the status in the compliance matrix as **not met**, with the residual risk "the vault-file key may remain in memory after its job is done; fix in v1.0."

At the end of the six steps, both a compliance-matrix row **and** a task in the project plan (zeroizing the vault-file key in v1.0) come out of it — this is the difference between "reading a requirement" and "turning a requirement into the project."



Common mistake: leaving the 'not applicable' decision without a rationale

A team marks every requirement in the "data in transit" family "not applicable" with a single sentence, "our product doesn't use the network," but provides **no evidence** to back up that sentence (e.g., a report from a code scan showing there's no network API call). The evaluator questions this claim: "Is there really no network connection at all, or did you simply not check?"

✓ Rule

Every "not applicable" decision must be written together with **evidence** supporting its rationale (a code scan, an architecture diagram, a dependency list). A "not applicable" with no rationale is just as untrustworthy as a "met" with no evidence — both can mean "not checked."

Second example: applying the same six steps to a process requirement

The steps work the same way not only for technical requirements but for process requirements too. Let's go quickly through CEN429-DV-02 ("every change must be reviewed"):

1. **Applicability:** Does the project use a version control system? Yes → applies.
2. **Responsibility:** The team meets it through its own process (there's no party to defer to).
3. **Linking to assets:** It's not linked to an "asset" directly, but to the **development process**; it's documented not in S5, but in the guide's "development lifecycle" section.
4. **Linking to threats:** Threat: "a malicious or faulty change entering the main branch unnoticed" (Week 6, supply-chain security).
5. **Control and verification:** Control: a pull-request rule, at least one approval mandatory. Verification: it's checked that the last 20 merges in the version-control history have at least one approval record.
6. **Release plan:** This process is already in effect; it's marked "met," noted as a rule requiring **continuous monitoring** (not a one-time thing, it applies to every merge).

This second example shows that the six steps work the same way not just for "crypto requirements," but for **every** type of requirement (functional, assurance, process).

Writing it into the release plan: a small example

The sixth step ("which requirement will be met in which release") turned into a concrete project-plan row:

Requirement	Target release	Owner	Status (this week)
CEN429-CR-03 (vault-file key erasure)	v1.0	Ahmet	In progress
CEN429-DT-04 (certificate pinning)	v1.1	Aslı	Planned
CEN429-DV-02 (code review)	Continuous	Team	Met

This table is the project-management-side counterpart of the "not met" rows in S17 (the compliance matrix); whoever updates S17 should update this table in the same session — otherwise the two documents **drift apart** and one contradicts the other.

? Who should own these steps?

On a small team, a single person (e.g., the team lead) should keep the compliance matrix up to date, but every developer should note **which requirement their own control meets, while they're writing that feature**. Trying to remember this weeks later usually leads to a wrong or incomplete mapping.

Why skipping these six steps is tempting but wrong

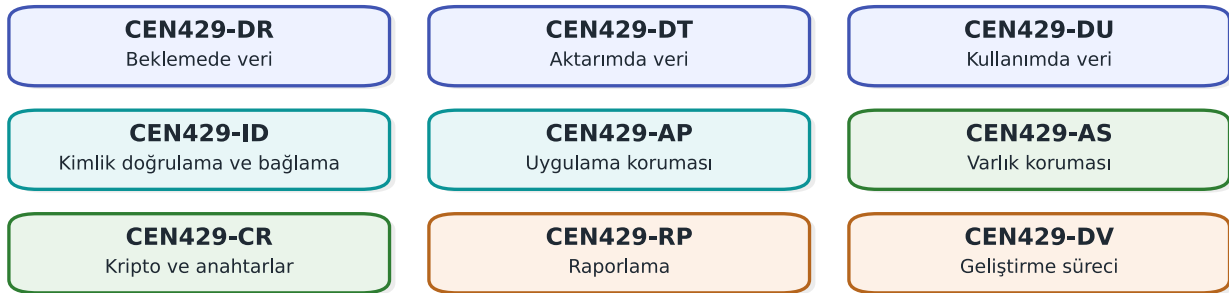
Reading a requirements list and directly saying "everything's met" is the fastest way to skip the six steps – and exactly for this reason it's the most common mistake. Teams under time pressure skip Step 3 (linking to assets) and Step 4 (linking to threats), because these look like "paperwork." But once these two steps are skipped, the remaining "met" claim rests on **nothing**; this is why Section 9's project checklist asks you to review S5 (the asset table) and S17 (the compliance matrix) **together**.

8. The course's requirement families

For your term project, we use a requirement set adapted from open standards (ETSI EN 303 645, OWASP MASVS, NIST SSDF, Common Criteria catalogues) and the structure of sector documents. Ids take the form CEN429-`<family>-<number>`.

Dersin gereksinim aileleri

her gereksinim bir aileye, her aile bir varlığa bağlanır



Kimlik numarası olmayan gereksinim izlenemez; izlenemeyen gereksinim denetlenemez.

Family	Topic	Example requirements (summary)
CEN429-AP	Application protection	AP-01 the release build must be produced with compiler protections enabled · AP-02 sensitive code sections must be obfuscated · AP-03 the release must contain no debug logging · AP-04 the application must check its own integrity at runtime · AP-05 a defined response must be given when a debugger or hook is detected · AP-06 version rollback must be rejected · AP-07 installation and update must be done securely
CEN429-ID	Authentication and binding	ID-01 every party the application authenticates must be listed · ID-02 mutual authentication with the server must be performed · ID-03 sensitive data must be bound to the device and the version
CEN429-AS	Asset protection	AS-01 the location, lifecycle, and C/I/I+ class of every asset must be defined · AS-02 compromising a single copy must not affect other copies (no shared static key) · AS-03 sensitive constants must be protected against static analysis
CEN429-DR	Data at rest	DR-01 sensitive data must be encrypted with AEAD · DR-02 data must be securely erased once no longer needed and upon compromise
CEN429-DU	Data in use	DU-01 sensitive data must be kept in the clear only for the duration of the operation · DU-02 session data must be erased from memory immediately after use
CEN429-DT	Data in transit	DT-01 transport must use at least TLS 1.2, with certificate and hostname validation · DT-02 data must pass only through defined interfaces · DT-03 sensitive data must be protected at the message level in addition to TLS
CEN429-RP	Reporting	RP-01 sensitive data must not be written to logs in the clear · RP-02 error codes must not give information that helps an attacker · RP-03 the application must not run in debug or test mode
CEN429-CR	Crypto and keys	CR-01 only standard algorithms with a secure configuration must be used · CR-02 every key's hierarchy, purpose, and cryptoperiod must be defined · CR-03 keys must be erased once their job is done · CR-04 random numbers must come from a source with sufficient entropy
CEN429-DV	Development process	DV-01 source code must be kept in version control, with protected branches · DV-02 every change must be reviewed · DV-03 every release must be uniquely identified · DV-04 dependencies must be tracked with an SBOM · DV-05 security tests must be run in continuous integration



Compliance matrix activity (25 minutes, teams)

1. From the families above, choose at least **15** requirements that apply to your project; mark the ones that don't apply, with their rationale.
2. For each one, write the status (met / deferred / not met), the guide section, and the verification method.
3. For every row you call "met," show **where** the evidence is **right now**. Change the status of any row you cannot show it for.
4. Have two teams swap matrices and have each question three of the other's rows "like an evaluator."

Worked example: filling in a compliance-matrix row from start to finish, mistakes and all

Let's follow how a team filled in its matrix row for CEN429-AP-04 ("the application must check its own integrity at runtime"), from the **first draft** to the corrected version; this is Section 1's "bad to good" idea applied to a compliance-matrix row.

First draft (inadequate):

Id	Requirement	Status	Control	Verification	Evidence
CEN429-AP-04	The application must check its own integrity at runtime	Met	There's an integrity check	Tested	—

This row has three problems: (1) "There's an integrity check" is not a description of a control, it's a **restatement of the requirement**; which mechanism (a checksum? a signature check? debugger detection?) is not stated. (2) "Tested" is not a verification method, it's a **claim** that verification exists. (3) The evidence column is **empty**.

The fix, step by step:

1. The team makes the control concrete: the application computes a hash of its own code section at startup and compares it with the expected hash embedded at build time; if they don't match, it stops running. This is described in the guide's S12.3 section.
2. The verification method is clarified: a test tool changes one byte of the binary and verifies that the application **refuses to start**.
3. Evidence is added: the CI job where this test runs automatically (`ci-integrity-check`), its latest run record, and the output log.

The corrected row:

Id	Requirement	Status	Control	Verification	Evidence
CEN429-AP-04	The application must check its own integrity at runtime	Met	S12.3: a hash of the code section is computed at startup and compared with the embedded expected hash; execution stops on mismatch	Automated test: a byte of the binary is changed and it's verified that startup is refused	CI job <code>ci-integrity-check</code> , run #391 , <code>integrity_test.log</code>

The difference isn't in the row's **length**, it's in the **concreteness** of every cell. The first draft also looked "filled in"; but none of its cells could be **independently** verified.



Common mistake: pasting the same evidence into every row

Some teams paste a single general test report ("see general-test-report.pdf") as evidence into **every** row of the matrix. When the evaluator opens the report and can't find a test specific to that requirement, the credibility of the **whole matrix** is called into question — one copy-pasted piece of evidence makes the rest of the matrix look suspicious too.

 Rule

Every row's evidence must be **specific to that row**: a specific test name, a specific CI run number, a specific log file. If you're referencing a general report, also state **which section/page** of the report it is.

The logic behind the families: why nine families?

The eight families (AP, ID, AS, DR, DU, DT, RP, CR) plus the process family (DV) correspond to nine questions that recur in real guides across the industry: "How does the application protect itself (AP)? Who does it trust (ID)? What does it protect (AS)? Where does that protected thing sit — on disk (DR), in memory (DU), or on the network (DT)? What does it say/not say if something goes wrong (RP)? Which cryptographic tools does it do all this with (CR)? And how does the team that produced it work (DV)?" These nine questions overlap almost one-to-one with the section headings in EMVCo's software-based mobile payment document (as seen in Section 6), because nearly every security guide in the industry asks the same nine questions in a different order.

A common mistake when choosing a family: writing to the wrong one

Putting a requirement in the wrong family (e.g., writing "the key must be erased once its job is done" under DU instead of CR) isn't a big mistake, but it breaks **consistency**: when looking for the same kind of requirements, an evaluator checks a specific family; if the requirement sits in the wrong family, it can be missed.

 Common mistake: forcing a new requirement into one of the existing families

A requirement may come up in your project that doesn't fit neatly into any of the families (e.g., an authorization rule needing multi-user support). Forcing it into CR or ID anyway breaks both the id numbering and the matrix's readability.

 Rule

If none of the existing nine families fit, define your own short family prefix (e.g., CEN429-AZ for authorization) and explain in one sentence in S1 what this new family covers; don't stretch an existing family to fit.

The difference between the AS family and the DR/DU/DT families

Beginners often ask "isn't AS (asset protection) the same thing as DR (data at rest)?" The difference is this: **AS** is about the asset **itself** (which asset exists, how it's classified, what happens if a copy is compromised); **DR/DU/DT** is about which **state** (at rest/in use/in transit) that same asset is in. The same asset (e.g., a crypto key) can be the subject of both an AS requirement (e.g., "the location and lifecycle of every key must be defined") and a DU requirement (e.g., "session data must be erased from memory immediately after use") — one asks for the asset's **inventory**, the other for the asset's state **at a given moment**.

Remembering the nine families at a glance

The order AP-ID-AS-DR-DU-DT-RP-CR-DV can be hard to remember; remembering this chain of questions can help: "Does it protect itself (AP)? Who does it trust (ID)? What does it protect (AS)? Where is that thing — at rest (DR), in use (DU), in

transit (DT)? What does it say if something goes wrong (RP)? With which tool (CR)? Who produced it, and how (DV)?" This is the "asset → threat → control" chain we've followed since Week 1, split into nine pieces.

9. Term project: this week

- ✔ **S17 – Compliance matrix:** all requirements from the course's families that apply to your project; every row has status, section, verification, and evidence.
- ✔ **S14 – Assumptions and deferred requirements:** supported platforms, out-of-scope components, user and environment assumptions; for every deferred requirement, to whom, why, and how.
- ✔ **S1 – Sources:** state which standards you're basing it on (ETSI EN 303 645, OWASP MASVS, NIST SSDF, CERT).
- ✔ Review your asset table (S5) against the requirements: is every asset's lifecycle and protection class filled in?

How to prepare this week's deliverable in 45 minutes

S17 and S14 are the documents an evaluator opens **first** (as stated in the info box at the start of this week too); leaving them messy creates a bad first impression even if the rest of the project is good. The suggested order:

1. Mark the requirements that apply to your project **from the family table in Section 8** (copy-paste, then instead of deleting the ones that don't apply, leave them as "not applicable, rationale: ..." – the rule from Section 7).
2. Fill in the **six columns from Section 2** (id, requirement, status, control, verification, evidence) for each one; don't write "met" without filling in the evidence column.
3. Move every row you marked "deferred" into S14; write the to-whom/why/how triple in **full sentences**, like the example in Section 3.
4. Open your asset table (S5); find the row of the asset each requirement links to. If it can't be linked, either the asset is missing or the requirement doesn't apply to this product.
5. Write which standards you're basing it on into S1 (sources) – see the "how do you choose standards" guide in Section 6.



Common mistake: filling in S17 the last night and never updating S5

Teams that write the compliance matrix in a single sitting right before the deadline skip **linking** requirements to the asset table (Step 3 of Section 7). Result: the matrix looks "done," but no row rests on a real asset; the evaluator can't establish the link the first time they question it.



Rule

Keep S5 open in every session where you update S17; whenever you add a new requirement, check whether the corresponding asset row exists, and add it if it doesn't.

How you should organise your evidence folder

A simple folder layout is recommended so every evidence reference in the compliance matrix is **actually findable**:

```
evidence/  
  week13/  
    test_butunluk_ci482.log  
    mitm_test_2026-11-03.log  
    handshake.pcapng  
    integrity_test.log  
  README.md  <- lists, in one line each, which requirement every file is evidence for
```

The `README.md` file is the **mirror** of the evidence column in the compliance matrix: every row in the matrix that says "evidence: file X" must actually be findable in this folder. If the folder is empty or the file is missing, the matrix is considered to have no evidence — just as described in the rule in Section 2.

If you're short on time: the priority order

There's a lot to deliver this week; if time is limited, proceed in this order (the order that earns the most points for the least time):

1. First, produce an S17 skeleton with **at least 15 requirements** (id + requirement + status columns filled in, control/verification/evidence can be blank) — this is far better than an empty S17.
2. Then, complete the rows you called "met" for which you actually **have** the evidence; honestly turn the ones without evidence into "not met."
3. Then, move the deferred rows to S14.
4. Finally, fill in the control/verification columns of the remaining rows.

This order rests on the principle that a "small but honest" matrix always scores better than a "large but evidence-free" one (see the evidence rules in Sections 2 and 8).

Why you shouldn't rush this week

S17 and S14 should become a document that grows a little **every week**, not just in the last week of the term project. If you note, throughout the term, which requirement each new control you add (in the "this week's project step" sections) satisfies, then by the time you reach this week S17 will already be **half full**. This week's real work then becomes only completing the missing rows and checking consistency.

10. Work on your own

? Exercise 1 — Easy: fix the requirement



Rewrite these requirements to meet the criteria of a "good requirement": "The system must be resilient to attacks," "Passwords must be stored securely," "The application must be fast and secure."

? Exercise 2 — Easy: ETSI mapping



Choose the ones among ETSI EN 303 645's thirteen provisions that apply to your project, and write which control in your project corresponds to each one.

? Exercise 3 – Medium: deferred requirements



Think of your project as a library embedded inside another application. Which requirements would you need to defer to the parent application? Prepare a one-paragraph instruction for the parent-application developer for each one.

? Exercise 4 – Medium: a security-target skeleton



Produce a Common Criteria security target (ST) skeleton for your project: the TOE definition, assumptions, threats, security objectives, and the corresponding requirements. Which sections of your security guide correspond to which ST section?

? Exercise 5 – Hard: FIPS 140-3 gap analysis



Examine your project's crypto code against FIPS 140-3's expectations of "approved algorithm, self-test, role and service, zeroization." If your project were a cryptographic module, what would be missing for Level 1?

Why you should solve these exercises on your own

The worked examples you saw in the sections above (bad→good requirement, the end-to-end traceability chain, the ST skeleton, the FIPS audit, the cross-standard mapping, the compliance-matrix row) were all explained through **other** requirements and **other** assets. The goal was to show the **method**; now you need to apply the same method to your **own** project's assets and requirements. You learn a method not only by watching it, but by applying it yourself once – especially in Exercises 1 and 3, **you** must notice which word is vague; this skill only comes with practice.

💡 If you get stuck, which section to go back to

Exercise 1 → reread the six-step example in Section 1. Exercise 2 → look at the mapping tables in Section 6. Exercise 3 → look at the deferred-requirement example in Section 3. Exercise 4 → look at the ST skeleton example in Section 4. Exercise 5 → look at the FIPS audit table in Section 5.

⚡ Common mistake: 'solving' the exercise in a single sentence

Replacing "The system must be resilient to attacks" with another vague sentence like "The system must be secure" looks like the exercise is solved, but repeats the same mistake. A requirement is not "good" unless **all** six steps from Section 1 are applied.

✓ Rule

After finishing each exercise answer, ask yourself: "Can I verify this sentence with a test/review?" If the answer is "no" or "maybe," the requirement isn't ready yet.

A mini checklist for evaluating your own answer

After finishing each exercise, check this list of five (it's the inverted form of the "five finding types an evaluator writes most often" list from Section 2):

- ✓ Does my answer have a "met" with no evidence?
- ✓ Are the ids consistent (the same id for the same requirement everywhere)?
- ✓ Is every control I wrote linked to a requirement?
- ✓ Is there a rationale everywhere I said "not applicable"?
- ✓ Did I write a generic sentence as evidence, or a concrete reference?

If you can answer "no" (i.e., no problem) to all five items, your answer is close to evaluator standard.

A starting template for Exercise 1

If you still don't know where to start, use this fill-in-the-blank template (the compressed form of the six steps from Section 1):

"[Which asset], [against which threat], [with which goal: confidentiality/integrity/availability], [with which tool class], [in which state: at rest/in use/in transit], [MUST/SHOULD] be protected."

Once you fill in every bracket, you'll have a requirement resembling the examples in Section 1. If one of the blanks can't be filled in (e.g., you can't answer "against which threat"), this is a sign the requirement isn't mature yet — go back to your threat model (Week 1).

11. Self-check

? 1. What is the difference between a functional security requirement and an assurance requirement? ✓

A functional requirement defines which security function the product will perform; an assurance requirement defines how we will trust that this function is performed correctly (review, testing, documentation).

? 2. What are the five properties of a good requirement? ✓

Singular, verifiable, traceable, feasible, and focused on "what" rather than "how."

? 3. What do forward and backward traceability ask in a compliance matrix? ✓

Forward: does every requirement lead to a control and a verification? Backward: does every control rest on a requirement or a threat?

? 4. When is the 'deferred' status used? How must it be written? ✓

When another party (the parent application, the operating system) can meet the requirement instead of the component. To whom and why it was deferred, and how the other party will meet it, must be written explicitly.

? 5. Is an EAL4 product always more secure than an EAL2 product? ✓

No. EAL shows the depth of the evaluation; security depends on the threats and objectives in the security target.

? 6. What is the difference between ST and PP? ✓

A PP is a common requirement set for a product class; an ST is a specific product's security target, and it can claim conformance to a PP.

? 7. Which factors score attack potential in the CEM? ✓

Elapsed time, expertise, knowledge of the product, window of opportunity, and equipment.

? 8. What is the difference between Level 2 and Level 3 in FIPS 140-3? ✓

Level 2 requires tamper evidence and role-based authentication; Level 3 requires tamper resistance and response (erasing the key) and identity-based authentication.

? 9. Why isn't 'we use a FIPS-validated library' enough? ✓

The application can use the library incorrectly (nonce reuse, turning off validation, leaving the key in memory); the validation covers the module, not the application itself.

? 10. What do ISO/IEC 27001 and Common Criteria each certify? ✓

ISO/IEC 27001 certifies the organisation's information security management system; Common Criteria certifies a product.

? 11. Why is each of the 'to whom, why, how' triple needed for a deferred requirement? ✓

To whom: it must be clear who the responsibility shifts to. Why: the reason the component cannot meet it must be explained so the evaluator can question the rationale. How: a concrete way the other party can actually meet it must be shown; if any of the three is missing, deferring becomes "leaving responsibility hanging."

? 12. What is the relationship between S17 (compliance matrix) and S14 (assumptions and deferred requirements)? ✓

Every row marked "deferred" in S17 is explained in S14 with the to-whom/why/how detail; the two complement each other and must be consistent.

? 13. What is the CEM (Common Evaluation Methodology), and how does it relate to CC? ✓

ISO/IEC 18045; the methodology document that defines how Common Criteria (ISO/IEC 15408) is applied in an evaluation, and which steps an evaluator follows.

? 14. What is the AVA_VAN component used for? ✓

For vulnerability assessment; it determines what attack potential (Basic, Enhanced-Basic, Moderate, High) the product must withstand from an attacker.

? 15. What is zeroization, and why is it required in FIPS 140-3? ✓

The secure erasure of critical security parameters (such as keys) from memory/storage once their job is done; an attacker who gains physical or logical access to the device must not be able to find old keys.

? 16. What kind of products is ETSI EN 303 645 written for? ✓

Consumer internet of things (IoT) devices; it defines a baseline security line for internet-connected consumer /home products.

? 17. Does GSMA SAS audit a product or a facility? ✓

A facility: it audits the physical security, personnel, processes, and key management of facilities that manufacture SIMs/eSIMs and manage subscriptions.

? 18. What is the main difference between EMVCo's software-based mobile payment requirements and PCI MPoC? ✓

EMVCo covers the application-level security of payment solutions that don't use the phone's secure element (emulating the card in software); PCI MPoC covers using commercial phones/tablets as payment-acceptance terminals, the application and the backend together, with monitoring and attestation services.

? 19. Which of OWASP MASVS's control groups covers 'resilience under attack' topics such as debugger detection and code integrity? ✓

MASVS-RESILIENCE.

? 20. What must be written when a requirement is marked 'not applicable'? ✓

The rationale for why the product doesn't fall under that requirement's scope (e.g., if there's no network connection, data-in-transit requirements don't apply) must be documented explicitly; a "not applicable" with no rationale is treated as a finding.

? 21. What must be done if a requirement using the SHOULD keyword (should, recommended) is not met? ✓

The rationale for not meeting it must be documented in writing; otherwise the evaluator treats it as a gap.

? 22. What does it mean for a good requirement to be 'singular,' and why does it matter? ✓

A sentence must contain only one condition; if multiple conditions are combined (e.g., "must be fast and secure"), it becomes unclear what the status is when one is met and the other isn't.

? 23. If the same security need has a counterpart in more than one standard, does complying with one automatically comply with the other? ✓

No. The overlap makes gathering evidence easier, but each standard's own criteria must be verified separately; otherwise an unevidenced assumption is made, like "we complied with one standard, so we also meet the other."

? 24. Why must the evidence column in a compliance matrix be filled in with a file/test name, not a generic phrase like 'there's a test report'? ✓

The evaluator must be able to find the evidence and verify it independently; a generic phrase breaks traceability and effectively leaves the row without evidence.

? 25. What is the difference between CAVP and CMVP? ✓

CAVP tests that individual algorithms (such as AES, SHA-2) are implemented correctly; CMVP evaluates the whole cryptographic module (algorithms + self-tests + roles + zeroization + security policy) and issues the module certificate.

? 26. Why does the platform/version information on a FIPS certificate matter? ✓

Validation is tied to the exact binary tested; if the same source code is compiled on a different platform or build configuration, that build does not automatically inherit the same certificate.

? 27. How does GSMA NESAS differ from GSMA SAS? ✓

SAS audits facilities (manufacturing/management), while NESAS evaluates both the development process and the product itself for mobile network equipment manufacturers.

? 28. What is the fundamental difference between the AS (asset protection) family and the DR/DU/DT families? ✓

AS is about the asset itself (its inventory, its classification); DR/DU/DT is about which state (at rest/in use/in transit) that same asset is in. The same asset can be the subject of both kinds of requirement.

? 29. Which countries had their own separate evaluation criteria before Common Criteria? ✓

The US's TCSEC (Orange Book), Europe's ITSEC, and Canada's CTCPEC; CC merged these three into a single international standard and, with the CCRA, provided recognition across countries.

? 30. When time is limited, in what priority order should you fill in S17? ✓

First the skeleton (id/requirement/status), then completing the rows you actually have evidence for, then moving deferred rows to S14, and finally filling in the remaining control/verification columns; a "small but honest" matrix is better than a "large but evidence-free" one.

? 31. Does adding a comment like 'this line meets CEN429-DR-01' in the code replace the evidence column? ✓

No. Such a comment only visually links the requirement to the code, making it easier to read; the evidence must still be a separate test/record/log file.

? 32. Can a requirement belong to both the AS (asset protection) and DU (data in use) families? ✓

Yes. AS asks about the asset's inventory/classification, DU asks about the same asset's state while in use; the same asset (e.g., a key) can be the subject of both kinds of requirement.

? 33. What is the concrete risk of not stating a version/year when claiming compliance with a standard? ✓

The evaluator can't know which checklist to compare against; since clause numbers and groupings can change between old and new versions of a standard, the claim becomes unverifiable.

12. Resources and further reading

Textbook — J. Viega, M. Messier, *Secure Programming Cookbook for C and C++*, O'Reilly, 2003: Recipe 11.18 (statistical randomness tests and the FIPS 140 context; today, FIPS 140-3 and NIST SP 800-90B/22).

Open standards and sources

- ISO/IEC 15408 (Common Criteria) and ISO/IEC 18045 (CEM); the protection profiles at commoncriteriaportal.org.
- FIPS 140-3, ISO/IEC 19790 and 24759; the NIST SP 800-140 series; CMVP and CAVP.
- ETSI EN 303 645 and ETSI TS 103 701 (conformity assessment).
- GSMA IoT Security Guidelines; GSMA SAS.
- EMVCo software-based mobile payment documents (general structure); PCI MPoC, PCI DSS v4.0, PCI Secure Software Framework.
- OWASP MASVS and MASTG; NIST SP 800-218 (SSDF).
- RFC 2119 / RFC 8174 (requirement keywords).

Why we chose these sources this way

This week's source list is deliberately made up of **open and freely accessible** standards (except for the card schemes' own texts, since those are shared under a non-disclosure agreement — see the info box at the top of the page). The goal is to offer sources you can still access after this course, and can reference directly in your project; when you say "standard X says this" on an exam or in the project, we want to make sure that standard is genuinely accessible.



Common mistake: not stating which version of the standard you're looking at

Standards get updated over time (FIPS 140-2 → 140-3, MASVS v1 → v2, ETSI EN 303 645's first version → later revisions). Saying "according to OWASP MASVS..." without stating which version leaves the evaluator not knowing which checklist was used; clause numbers and groupings can change between two versions.



Rule

Add a **version and/or year** to every standard reference in your guide and in the S1 (sources) section (e.g., "OWASP MASVS v2.0," "FIPS 140-3 (2019)," "PCI DSS v4.0"). This lets the evaluator compare against the correct checklist.

How you format sources in S1 (example)

S1 – Sources (example format)

- [1] ETSI EN 303 645 V2.1.1, "Cyber Security for Consumer Internet of Things: Baseline Requirements."
- [2] ISO/IEC 15408-1:2022, Common Criteria for Information Technology Security Evaluation.
- [3] OWASP Mobile Application Security Verification Standard (MASVS) v2.0.
- [4] NIST FIPS 140-3 (2019), Security Requirements for Cryptographic Modules.

Listing every source with its version/date information is a direct application of the rule at the start of this section; the same format can also be used when referencing a standard in the requirement blocks (Section 3) of your guide.

This week's short summary

This week we saw how to write a good requirement from scratch (Section 1), how to trace it all the way to evidence (Section 2), how to write it correctly into the guide (Section 3), and how the major standard families (Common Criteria, FIPS 140-3, ETSI/GSMA/EMVCo/PCI/MASVS) express the same needs in different words (Sections 4–6). In Sections 7–8 we covered how to carry these into your project, and in Section 9, how to prepare this week's concrete deliverable step by step. The asset table you've gathered since the start of the term (Week 1), the protection rules (Week 9), and the assessment process (last week) all come together this week in a single document (the compliance matrix) – in the coming weeks, this document will be the first and most important page you present to the evaluator for your project.



Glossary



Term	Turkish	Short definition
Compliance matrix	Uyum matrisi	Requirement → control → verification → evidence → status table
Deferred requirement	Devredilen gereksinim	A requirement another party must meet
TOE	Değerlendirme hedefi	The product being evaluated and its documents
Security Target (ST)	Güvenlik hedefi	A product-specific threat, objective, and requirement document
Protection Profile (PP)	Koruma profili	A common requirement set for a product class
SFR / SAR	İşlevsel / güvence gereksinimi	CC requirement types
EAL	Değerlendirme güvence düzeyi	CC's assurance packages (1–7)
Attack potential	Saldırı potansiyeli	The score of resources needed for a successful attack
Zeroization	Sıfırlama	The secure erasure of critical parameters