



Whitebox Cryptography

CEN429 Secure Programming — Week 11

Asst. Prof. Dr. Uğur CORUH · 27.11.2026

Today's Plan (3 Hours)

Hour	Section	Topic
1	0–1	Basic concepts · black/grey/white box · WBC attacker · naive solutions
2	2	Table-based WBC (Chow AES) step by step · cost
3	3–4	Attack history · countermeasures · layered placement · hardware · project

A Brief History — Whitebox Cryptography

- **1883** — **Kerckhoffs**: security must lie in the **key**, not in the secrecy of the system
- **2002** — Chow et al. publish the first **whitebox AES/DES** (for DRM) — the birth of WBC
- **2004** — the **BGE attack** breaks the first WB-AES
- **2016** — **DCA** (Bos et al.): hardware DPA moves into software, breaks it automatically
- **2017–2024** — **WhibOx**: every published pure-software candidate is broken

Today's rule: WBC is a **delaying layer**; prefer **hardware** (TEE/SE/HSM) where possible.

Where Does This Week Fit?

- **Week 9:** code obfuscation — we said "hiding doesn't protect the key"
- **This week (11):** so **how** do we protect the key in software? → whitebox
- **Week 10:** correct use of cryptography, the key lifecycle

Today we tackle the question: "what do we do while the key is in the attacker's hands?"

Learning Outcome

This week is about **LO.2 / LO.4** (cryptography, secure communication).

By the end, you will be able to:

- Distinguish the black/grey/white box models
- Explain **what whitebox does and why it isn't enough**
- Make the right protection decision for a key

A Warning, Up Front

Every **pure-software** whitebox design published to date has **been broken**.

So today we won't say "whitebox = secure key."

We'll say: whitebox is a layer that **delays** key extraction — and it is not sufficient on its own.



0. Basic Concepts (From Scratch)

What Is Encryption?

- **Encryption:** turning readable data (**plaintext**) into unreadable data (**ciphertext**) using a **key**.
- **Decryption:** reversing it with the key.

Şifreleme nedir? İki yönlü, anahtara bağlı dönüşüm

aynı anahtar hem kilitler hem açar (simetrik şifreleme)



Kerckhoffs ilkesi: algoritma herkesçe bilinebilir; güvenlik yalnız anahtarın gizliliğindedir.

What Is a Key?

- **Key:** the secret number (a byte sequence) that governs encryption.
- Same algorithm + different key = different result.
- **Security depends on the secrecy of the key**, not the secrecy of the algorithm (Kerckhoffs's principle).

Symmetric and Asymmetric

- **Symmetric:** encryption and decryption use the **same** key (e.g. **AES**). Fast.
- **Asymmetric:** a public/private key pair (e.g. RSA). Slow but easy key distribution.

This week we'll mostly work with **AES** (symmetric).

What Is AES? (From a High Level)

- **AES**: the most widely used symmetric encryption algorithm.
- It works on 16-byte **blocks**.
- It mixes data in **rounds**; each round: byte substitution, row/column mixing, key addition.

The details won't be on this week's exam; the **idea** is enough.

What Is an S-box?

- **S-box (substitution box):** a **fixed table** that replaces one byte with another.
- This is AES's "byte substitution" step.
- Example: a 256-entry mapping such as `S-box[0x53] = 0xED`.

Keep this table in mind; it's the heart of whitebox.

Lookup Table

- **Lookup table:** an array that gives "output for a given input".
- $T[x]$ = the precomputed result for x .
- Instead of computing, **you read from the table**; it's fast.

Whitebox buries computation inside tables.

XOR Reminder

- **XOR** ($\wedge$): 1 if the bits differ, 0 if they're the same.
- $a \wedge b \wedge b == a \rightarrow$ reversible.
- In AES, the "key addition" step is a XOR: $state \wedge key$.

Bit Security Level

- "n-bit security" = roughly 2^n attempts are needed to break it.
- AES-128 → 128 bits; considered unbreakable today.
- Higher number = stronger.

What Is a Side Channel?

- **Side channel:** an attack that uses not the algorithm's math but the **physical/indirect information it leaks** while running.
- Examples: elapsed **time**, **power** consumption, electromagnetic emission.

Classic on smart cards; shortly we'll see whitebox's software equivalent.

What Is DPA?

- **DPA (Differential Power Analysis):** a side-channel attack that applies statistics to a device's **power consumption** measurements to extract the key.
- It doesn't look inside the device; it measures from the outside.

Keep this idea in mind; whitebox's strongest attack (DCA) is the software form of exactly this.

Bijection (a One-to-One, Onto Mapping)

- **Bijection:** a transformation that maps every input to exactly one output and is **invertible**.
- Example: $f(x) = x \wedge 0x5A$ is a bijection (its own inverse).

Whitebox wraps tables in **secret bijections**.

TEE and Secure Element (SE)

- **TEE (Trusted Execution Environment):** a phone processor's secure region, **isolated** from the operating system.
- **Secure element (SE):** a separate, tamper-resistant **hardware** chip that stores keys.

These are the strongest protection for a key; whitebox is for the case **when they aren't available**.

HSM and SoftHSM

- **HSM (Hardware Security Module)**: dedicated hardware on a server that stores/processes keys; the key **never leaves**.
- **SoftHSM**: a **software emulation** of an HSM; same interface (PKCS#11), but no hardware protection (for development/testing).

Now We're Ready

Terms we know:

encryption/decryption · key · symmetric/asymmetric · AES · S-box · lookup table · XOR · bit security · side channel · DPA · bijection · TEE/SE · HSM/SoftHSM

Now to the real question: **what do we do while the key is in the attacker's hands?**



1. Three Attacker Models

Why a "Model"?

Before designing a defence, we must determine **what the attacker can see**.

This is called the **attacker model**.

There are three models: black, grey, white box.

Black Box

- The attacker sees **only input and output**.
- They cannot see the inside (key, intermediate values).
- Typical setting: a **remote server** (the code is yours, the attacker connects over the network).

AES/RSA's security proofs assume this.

Grey Box

- In addition to input/output, the attacker sees **side channels** (time, power, EM).
- They can't read the inside directly, but they measure it **indirectly**.
- Typical setting: a **smart card** or an embedded device in the attacker's hands.

White Box

- The attacker sees **everything**: code, memory, every intermediate value.
- And they can **modify it**.
- Typical setting: **software** running on the attacker's own device.

This is the real situation for a mobile/desktop application.

Three Models Side by Side

Model	Sees / does	Environment
Black	Input/output	Remote server
Grey	+ side channel	Smart card
White	Everything + modifies	Software, own device

Crypto was designed for **black box**; we're in **white box**.

The Three Attacker Models — What Do They See?

Üç saldırgan modeli: ne görüyor?

soldan sağa görünürlük artar



WBC en zor modelde anahtar çıkarmayı GECİKTİRMEYE çalışır; imkânsız kılmaz.

Here's the Problem

AES is mathematically strong. But:

- In white box, the **key sits in memory** somewhere
- The attacker can read memory
- So what good is AES's strength?

A strong lock is useless if you hang the key next to the door.

The Idea of Whitebox Cryptography (WBC)

- **WBC:** performing encryption so that the key **is never explicitly in memory**.
- The key is embedded inside precomputed **lookup tables**.
- There is **no** byte sequence in memory you could point to and say "that's the key."

WBC Is Not a Definition, It's an Attacker Model

- WBC was defined in 2002 by Chow and colleagues.
- It's really an **attacker model**: "how do you encrypt if the attacker can see and modify everything?"

What Can the WBC Attacker Do? (1)

Observes:

- Accesses the instructions being executed at that moment
- Watches the algorithm's flow
- **Sees the memory** being used

So they see every intermediate value the program sees.

What Can the WBC Attacker Do? (2)

Modifies:

- Tamperers with the program's memory
- Runs **only a single round** of the algorithm
- Alters `if` conditions
- Alters loop counters
- **Injects a fault** (corrupts a value)

The Source's Compliance Clause

The technical guide opens this section with the following requirement:

"Cryptographic keys shall be **secured and/or hidden** to protect confidentiality and integrity."

But It Immediately Adds

The guide says that WBC is **not the only** thing protecting against these attacker capabilities:

What protects the application and its assets is **code hardening (Week 9) and RASP (Week 6)** methods.

WBC is never presented **alone**, from the very first sentence.

Why Is the Problem So Hard?

The core tension:

- The program **must access** the key to encrypt
- The attacker sees everything the program sees
- So at some point the key is **somewhere the attacker can also see**

WBC tries to get past this by "baking the key into tables."

This Isn't Secrecy, It's Cost

WBC doesn't **move** secrecy away from the crypto.

It **spreads** the key across tables with a mathematical transformation.

The goal is the same as in Week 9: make extracting the key back out **much more expensive**.

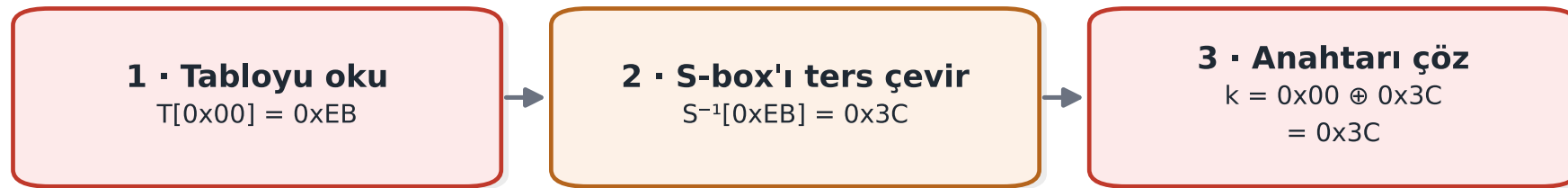


Why Do Naive Solutions Collapse?

How the Key Leaks From a Table — Diagram

Naif tablo anahtarı nasıl sızdırır?

$T[x] = S\text{-box}[x \oplus k]$ — tek girdi yeter, 256 deneme bile gerekmez



'Anahtar kodda görünmüyor' demek, 'anahtar korunuyor' demek DEĞİLDİR.

Why Do We Study What Collapses?

To see why WBC is needed.

Every failed "easy" solution turns into a secure-programming **rule**.

Naive 1 · Embed the Key in an Array

```
static const uint8_t k[16] = { 0x2B, 0x7E, ... }; /* KÖTÜ */
```

The easiest way is the weakest way.

Naive 1 · How Is It Broken?

1. `strings` / byte scan: the 16-byte block is visible
2. **Entropy scan**: a high-entropy block says "the key is here" (the Shamir–van Someren observation)
3. Debugger: confirms this block is used in the encryption call

Time: minutes.

Naive 1 · Rule

A plainly embedded key is **not** key protection.

This is exactly Week 9's rule: "string hiding doesn't store the key."

Naive 2 · "Scramble" With XOR

```
static const uint8_t k_gizli[16] = ...;    /* k ^ maske */  
/* çalışırken: k = k_gizli ^ maske */
```

"I'll XOR the key with a mask and store that."

Naive 2 · Why Does It Collapse?

- The **mask is also** in the binary
- If the program can compute `k`, the attacker can too
- Just one extra step; not a real barrier

Whatever the program can do, the white-box attacker can do too.

Naive Threat 3 · Code Lifting

- Say WBC baked the key into tables.
- The attacker copies the encrypting code fragment (tables + interpreter) **as is, without ever extracting** the key.
- They use it in their own program: they **steal the function** without knowing the key.

Code Lifting — Diagram

Code lifting: anahtarı hiç çıkarmadan

WBC'nin en sinsi saldırısı: anahtar gizli kalsa bile işlev çalınır

Saldırı

whitebox rutinini/tablosunu
olduğu gibi KOPYALA

kendi programında kullan
anahtar hiç bilinmez

Önlem

dış kodlama (F, G)
cihaz bağlama
sunucu tarafı denetim

taşınan rutin işe yaramaz

Bu yüzden WBC tek başına değil, bağlama ve sunucu denetimiyle birlikte kullanılır.

Code Lifting · Rule

WBC saying "I hid the key" isn't enough.

Countering code lifting needs **device/version binding** (Section 3, shortly): the tables should only be meaningful on one specific device/version.

Naive Solutions: Summary

Naive solution	Broken by
Key embedded in an array	<code>strings</code> + entropy + gdb
XOR/whitening	the mask is inside too
(even WBC)	code lifting → needs device binding

Section 1 — Quick Check

1. The difference between black, grey, white box?
2. AES is strong, so what's the problem in white box?
3. Why isn't embedding the key in an array protection?

Section 1 — Answers

1. **Black:** input/output only · **Grey:** + side channels (time/power) · **White:** everything + modifies.
2. The key sits **in memory** at some point; the white-box attacker sees it. AES's strength rests on the **black box** assumption.
3. `strings` + **entropy** scan + gdb finds it in minutes; the program has to use it.



2. Table-Based WBC (Step by Step)

What Will We Learn?

Chow and colleagues' 2002 AES design is the classic of table-based WBC.

- Its **idea** (how it works)
- Its **cost** (table size, speed)
- Its **limit** (why it isn't enough)

No math on the exam; **the idea and the limit** are.

Reminder: Two Steps of an AES Round

1. **XOR** the state byte with the round key: $x \oplus k$
2. Pass the result through the **S-box**: $S\text{-box}[x \oplus k]$

Can these two steps be combined for a fixed k ? Yes.

Step 1 · "Bake" the Key Into a Table

```
Normal:  çıktı = S-box[ x ^ k ]  
WBC:    T[x]  = S-box[ x ^ k ]  (256 girişli tablo)
```

There's no longer a `k` variable in the code. `k` is baked **into** the table.

Step 1 · What Did We Gain?

- There's no byte sequence named `k` in memory
- There's only the `T` table (256 entries)

The key seems to have disappeared. But...

Step 1 · Why Is This Insecure on Its Own?

- T is the S-box **shifted** by $x \wedge k$
- The attacker compares T with the known S-box
- They **recover** k from the shift amount

This is why the tables need to be **encoded**.

Step 2 · Grow the Tables (T-box)

- Instead of a single-byte mapping, combine the S-box with AES's **mixing** step (MixColumns)
- Result: larger tables (**T-box**) that take 8 bits in and give **32 bits** out
- The XORs between bytes also become small **XOR tables** (4 bits)

Step 2 · Result

- The whole round turns into a **network of lookup tables**
- No explicit arithmetic key operation is visible anywhere
- But the tables can still be examined on their own → encoding is required

Step 3 · Internal Encodings

- Apply a secret bijection to each table's **output**
- Apply its inverse at the **input** of the next table
- Consecutive encodings **cancel out** → the result is still correct

But **intermediate values look scrambled**.

Internal Encoding — Diagram

İç kodlama ne değiştirir?

gizli bijeksiyon G , ara değeri maskeler

Kodlamasız tablo

$$T[x] = \text{S-box}[x \oplus k]$$

S^{-1} uygulanabilir
→ k geri hesaplanır

İç kodlamalı tablo

$$T^1[x] = G(\text{S-box}[x \oplus k])$$

G bilinmeden S^{-1} yok
→ adım 2 kırılır

Ardışık tabloların kodlamaları birbirini götürür: sonuç doğru, ara değer karışık.

Step 3 · What Does It Prevent?

It aims to stop an attacker examining a table **on its own**.

Because that table's input/output has been scrambled by a secret encoding.

Step 4 · Mixing Bijections

- On top of the internal encodings, linear **mixing matrices** over $\text{GF}(2)$ (8×8 , 32×32)
- These **spread** information across tables
- Information leaking from a single table alone doesn't mean much

($\text{GF}(2)$ = the math of working with bits and XOR arithmetic; no detail needed.)

Step 4 · Table Types

- In Chow's design these encoded/mixed tables have **types** (types I–IV in the literature).
- Each type plays a different role.
- For this course, it's enough to know: "the tables aren't all one type, they're varied."

Step 5 · External Encodings F and G

- On the outermost layer, the whole cipher is wrapped with two secret bijections: **F** and **G**
- Source guide: "the whitebox AES operation is encapsulated with two randomly chosen non-singular matrices called F and G"
- So the network is no longer plain AES:

Dış kodlama: en güçlü koruma, en büyük sınır

çıktı artık STANDART AES değildir



Karşı taraf F ve G'yi bilmeden sonuç anlamsız → yalnız iki ucu da sizin olduğu protokolde.

Step 5 · Why Is External Encoding Strong?

- What the attacker sees at input/output is **not standard AES's input/output**
- It makes many attacks (statistical/algebraic) that look only at the tables much harder

Step 5 · But This Is Also the Biggest Limit

$G \circ AES \circ F^{-1}$ is not standard AES.

- The other side (server/other endpoint) gets a **meaningless** result unless it knows and undoes F and G
- So externally-encoded WBC is only usable in a protocol where **you control both ends**
- It may not be usable in a protocol like EMV that **expects standard AES**

Five Steps · Summary

1. Bake the key into a table
2. Grow the tables (T-box + XOR tables)
3. Internal encodings
4. Mixing bijections
5. External encodings (F, G)

Result: no explicit key; everything lives in encoded tables.



The Cost of WBC

WBC Isn't Free (1)

The price of baking the key into tables: **huge tables**.

Metric	Approximate (order of magnitude)
Table size	Hundreds of KB (~770 KB)
Comparison	Standard AES key: 16–32 bytes

WBC Isn't Free (2)

Metric	Approximate (order of magnitude)
Table lookups per round	Thousands (~3,000)
Speed	~50× slower than standard AES

A Sense of Scale: a Table-Size Calculation

Example: 288 tables $\times$ 1 KB $\approx$ **294,912 bytes** (~288 KB).

- The point isn't the exact number, it's the **order of magnitude**
- This is why WBC is applied only to **selected, critical** operations
- The same logic as Week 9's virtualization (K-10)

Section 2 — Quick Check

1. Why is $T[x] = S\text{-box}[x \wedge k]$ insecure on its own?
2. The difference between internal and external encoding?
3. What's the biggest limit of external encoding (F, G)?
4. Why is WBC applied only to selected operations?

Section 2 — Answers

1. **T** is the known S-box shifted by x^k ; comparing the two tables **recovers k**.
2. **Internal**: scrambles the intermediate values between tables · **External (F,G)**: wraps the whole cipher ($G \circ \text{AES} \circ F^{-1}$).
3. It's no longer **standard AES**; the other side has to know F/G — it may not be usable in standards like EMV.
4. Tables run to hundreds of KB + $\sim 50\times$ slower; the cost limits it to **small/critical** operations only.

3. Attack History and Countermeasures

WBC's Breaking History — Diagram

Whitebox kriptografi: kırılma tarihi

yayımlanmış hiçbir saf-yazılım tasarım ayakta kalmadı

Kerckhoffs:
güvenlik anahtarda

1883

BGE saldırısı
ilkini kırar

2004

WhibOx: tüm
adaylar kırıldı

2017-24

2002

Chow vd.
WB-AES / WB-DES

2016

DCA (Bos vd.)
otomatik kırma

Sonuç: WBC bir GECİKTİRME katmanıdır; mümkünse donanım tercih edilir.

Why Do We Look at History?

How much you should trust a protection is told by its **history of being broken**.

Every row is an **inference** for the defender.

2002 · Birth

- Chow and colleagues publish the WB-AES and WB-DES designs.
- The table-based approach is born.

Inference: the idea is promising; but untested.

2004 · The BGE Attack

- **BGE** (Billet–Gilbert–Ech–Chatbi): breaks Chow AES in practical time ($\sim 2^{30}$ operations).
- Solves the internal encodings.

Inference: internal encodings alone are **not enough**.

2007 · WB-DES Falls

- Goubin, Wyseur, and others break WB-DES and its externally-encoded variants.

Inference: DES-based WBC is abandoned.

2010s · The "More Tables" Attempt

- Karroumi's "dual AES" hardening is also broken ($\sim 2^{22}$).

Inference: more tables **is not** more strength.

2015 · DFA (Fault Injection)

- **DFA (Differential Fault Analysis):** injects a fault into a value while running and computes the key back from the corrupted output.
- "Unboxing the White-Box" (Black Hat EU).

Inference: it can be broken **without knowing** the design's internal math.

2016 · DCA (the Most Important One)

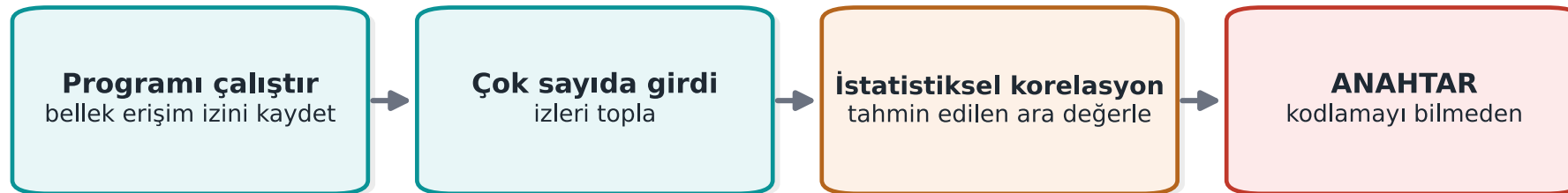
- **DCA (Differential Computation Analysis):** collects a **trace** of the intermediate values produced while running, and extracts the key with DPA-like statistics.
- It **ignores** internal encodings most of the time.

Inference: classic side-channel defences are also needed for WBC.

The DCA Attack — Diagram

DCA: donanım yan kanalının yazılım hâli

güç izi yerine YAZILIM izi kullanılır



Klasik DPA'nın kardeşi. İç kodlama zayıfsa korelasyon kodlamayı aşar.

2017–2024 · WhibOx Competitions

- Academia and industry submit their designs to a competition.
- **Every** submitted design gets **broken**.

Inference (today): there is **no** published pure-software WBC left standing.

Chronology · Summary

Year	Event	Inference
2002	Chow WB-AES/DES	Birth
2004	BGE	Internal encoding isn't enough
2007	WB-DES broken	DES abandoned
2010s	Karroumi broken	More tables $\neq$ strength
2015	DFA	Without knowing the math
2016	DCA	Side-channel defence needed
2017–24	WhibOx	All broken

Why Do DCA and DFA Also Hit New Designs?

Because neither one **tries to solve** the design's secret math:

- **DCA:** doesn't open the box; collects a **trace** while running, finds the key with statistics
- **DFA:** **injects a fault** while running, computes it from the corrupted output

Because they're general and **portable**, they threaten every design.

So What Do We Do? (Countermeasures)

Research hasn't stopped; but no countermeasure says "now it's secure."

They all **raise the cost** and are used **together**. In order:

Countermeasure 1 · Non-Linear Masking

- Tries to break the **linear correlation** DCA relies on (e.g. Biryukov–Udoenko 2018).
- Makes the statistical attack harder.

Countermeasure 2 · External Encoding (F, G)

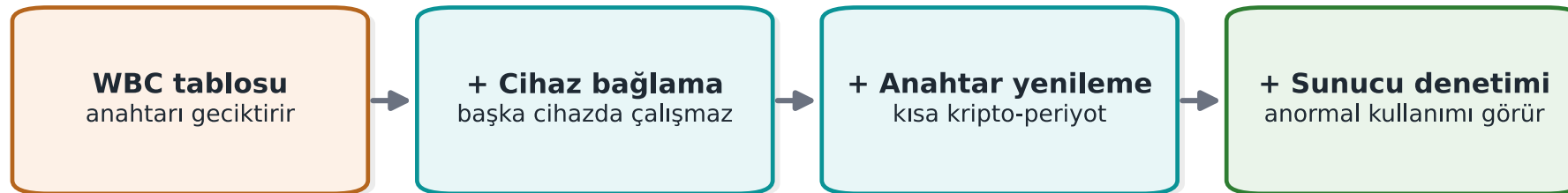
- Makes DCA/DFA/BGE harder.
- **But** we already saw the limit: it breaks standard-AES compatibility.

Countermeasure 3 · Layered Defence

- Week 9's **obfuscation** rules around WBC
- Week 6's **RASP** (anti-debug, tamper/integrity checking)
- Goal: make DCA's trace collection and DFA's fault injection **harder**

Layered WBC — Diagram

WBC'yi tek başına bırakmayın
her katman saldırının kazanç penceresini daraltır



Tek başına WBC = zaman kazandırır; katmanlı WBC = saldırıyı ölçeklenemez kılar.

Countermeasure 4 · Key Rotation

- If the key rotates often, extracting one key **is worth less**
- Week 10: **crypto-period** (a key's lifetime)

Countermeasure 5 · Device/Version Binding

- Tables are only meaningful on **one specific device/version**
- Blocks **code lifting**

Countermeasure 6 · Server-Side Risk Checking

- Whatever protection exists on the edge, the transaction is **also** evaluated on the **server**
- Rate limiting, anomaly detection, counter/ATC checking
- The **last and most reliable** defence

Countermeasure 7 · Move to Hardware

- If possible, **never put the key in software** at all
- TEE, secure element (SE), HSM

This is the subject of the next section.

The Core Rule (Again)

WBC is a **layer**, not a solution.

The right statement: "I'm delaying key extraction by this much; my real assurance comes from key rotation, device binding, and server-side risk checking."

Section 3 — Quick Check

1. How does "all of them were broken" affect your decision?
2. Which classic attack does DCA resemble?
3. Why don't internal encodings always stop DCA?
4. Which is the most reliable last line of defence?

Section 3 — Answers

1. WBC is **not** assurance on its own; combine with key rotation + device binding + server checking + obfuscation; move to **hardware** where possible.
2. **DPA** (power analysis) — DCA is its software form.
3. Internal encoding is a one-to-one mapping (bijection); statistical **correlation** often survives it.
4. **Server-side** risk checking (the last and most reliable).



4. Layered Placement, Hardware, Project

Protecting a Key: the Options

WBC isn't the only option. There are several ways to protect a key; strength and cost differ.

In order, from weakest to strongest.

Option 1 · Plain Key in a Fixed Array

- Key: in the software, **in the clear**
- Strength: **none**
- When: **never**

Option 2 · Hidden/Split Key

- Key: in the software, hidden
- Strength: very low
- When: only for low-value, short-lived secrets

Option 3 · WBC + Layered Defence

- Key: in the software, **embedded in tables**
- Strength: medium (delays)
- When: no hardware root available; **with key rotation + server checking**

Option 4 · TEE / Secure Element (SE)

- Key: **isolated by hardware**
- Strength: high
- When: **preferred** if the device supports it

Option 5 · HSM / SoftHSM

- Key: in a hardware module (or its software emulation)
- Strength: high (HSM); SoftHSM for testing only
- When: **server-side**; the key never leaves

Options · One Table

Option	Where's the key	Strength
Plain array	In the clear, in software	None
Hidden/split	Hidden, in software	Very low
WBC + layers	Embedded in tables	Medium
TEE / SE	Isolated by hardware	High
HSM / SoftHSM	Hardware module	High

The Decision Rule

If you **don't have to** put the key in software, **don't**.

- If a hardware root (TEE/SE/HSM) exists, **use it**
- WBC is a **trade-off** solution for when there's **no** hardware root
- Always: together with key rotation + device binding + server checking

SoftHSM and PKCS#11 (Server-Side Bridge)

- **PKCS#11**: the standard interface for talking to key modules
- The application never sees the key's **value**; it just says "sign/encrypt this"
- **SoftHSM**: a software emulation offering the same interface; **no** real hardware protection → for development/testing

WBC is a client-side (edge) problem; this is the answer to the server's key problem.



Flawed → Attack → Protection

1 · Flawed Design

```
/* İstemci uygulaması: veri şifreleme anahtarı gömülü */  
static const uint8_t k[16] = { ... }; /* KÖTÜ */
```

A client embeds a data-encryption key in a fixed array.

2 · Attack (Conceptual)

1. `strings` + **entropy scan** → a high-entropy 16-byte block
2. Debugger → confirms this block is used in the encryption call
3. The key comes out in **minutes**

3 · Protection (Layered) — a

- Move the key to **TEE/SE** if possible
- Otherwise, bake it into **WBC tables** (no plain byte array left)

3 · Protection (Layered) — b

- Wrap WBC with Week 9's **obfuscation** and Week 6's **RASP**
 - Collecting a trace for DCA and injecting a fault for DFA both get harder
- **Bind to device/version** → code lifting doesn't work

3 · Protection (Layered) — c

- Rotate the key often (short crypto-period)
- Apply risk checking on the server (rate limiting, anomalies)

4 · The Risk That Honestly Remains

- A sufficiently determined attacker can still extract the key **on a single device**
- But now:
 - Breaking one copy **doesn't open the whole system** (device binding)
 - The extracted key has a **short lifetime** (rotation)
 - The server catches abnormal use

Rules That Come Out of the Scenario

- Don't embed the key in plain form
- Use a hardware root if one exists
- **Never** leave WBC alone
- Device binding + rotation + server checking



Project and Closing

Project · S8 (Key Protection Rationale) — 1

Write a protection decision and its **rationale** for at least one sensitive key:

1. Where does the key live, how is it protected? (if not plain: TEE/SE, WBC, hiding — which one, **why**)

Project · S8 — 2

2. If the protection isn't sufficient (e.g. you're pure software), state this **explicitly** and list your compensating layers:

- key rotation period
- device/version binding
- server-side risk checking

Project · S8 — 3

3. If you didn't use WBC, give the **reason why not**:

- cost
- you have a hardware root
- the asset's value is low

A justified "we didn't use it" is a **complete** answer.

No Real Keys in the Repository

- Every key file, table, and example must be **synthetic**
- A real key in the repo at certification time = one of the most severe findings
- It counts the same way in the project

Glossary (1)

Term	Meaning
Black/grey/white box	What the attacker sees
WBC	Crypto that runs in the white box
S-box	Fixed table that substitutes a byte
Bijection	Invertible one-to-one mapping
Internal/external encoding	Inside a table / secret transform wrapping the whole cipher

Glossary (2)

Term	Meaning
Code lifting	Copying the code without extracting the key
DCA	DPA-like statistics applied to execution traces
DFA	Injecting a fault and computing the key
Crypto-period	A key's lifetime
TEE/SE/HSM	Hardware-based key protection

Self-Check (1–6)

1. Three attacker models; which one does the AES proof assume?
2. "Run only one round" and "inject a fault" set up which attacks?
3. Why isn't a key embedded in an array protection? (entropy)
4. What is code lifting, which countermeasure closes it?
5. Why is $T[x] = S\text{-box}[x^k]$ insecure on its own?
6. Internal vs. external encoding; the limit of external encoding?

Self-Check — Answers (1–6)

1. **Black box** (I/O only), **grey box** (+ side channel: power/time), **white box** (full access + modifies). The classic AES proof assumes **black box**.
2. "Run one round/observe an intermediate value" → **DCA**; "inject a fault" → **DFA**.
3. The key sits as a **high-entropy** block → found by an entropy scan/ `strings` /pattern (demo `--scan`). Moving its location isn't hiding.
4. **Code lifting**: copying a table/routine as-is and using it elsewhere. Closed by: **external encoding** (+ device binding/server checking).
5. `T[x]=S-box[x^k]` depends directly on k ; comparing the table with S-box **recovers k** . **Mask it with internal encoding**.
6. **Internal encoding** masks intermediate values with secret bijections; **external encoding** transforms the input/output (makes code lifting harder). Limit: **the caller must match too** → breaks standard AES compatibility.

Self-Check (7–12)

7. Table size/speed order of magnitude? Why only selected operations?
8. How does "all of them were broken" affect your decision?
9. Which hardware attack does DCA resemble?
10. Rank the key-protection options by strength/cost
11. What 3 layers should you add if you use WBC?
12. Write the S8 rationale for a key, in three sentences

Self-Check — Answers (7–12)

7. Tables run to **MB order of magnitude** and are tens-to-hundreds of times **slower** than the black box → applied only to **selected/critical** operations.
8. We don't count WBC alone as key protection; we use it as a **delaying layer**, together with rotation + binding + server checking (hardware where possible).
9. **DPA** (Differential Power Analysis); DCA replaces the power trace with a **software trace** (memory access/intermediate value).
10. (weak→strong) plain embedded key < encoded table (pure-software WBC) < WBC+binding/rotation < **TEE/SE** < **HSM/hardware**. Cost rises in the same direction.
11. **Key rotation · device/application binding · server-side checking/revocation** (+ RASP/integrity).
12. Example: "The DEK protects data at rest with AES-256-GCM. It is generated/stored in an HSM, never kept in plain memory. Hardware was chosen because pure-software whitebox has been broken and the asset's value is high."

Summary: This Week in One Sentence

Every published pure-software whitebox has been broken; WBC is a layer that **delays** key extraction, and it only gains meaning together with rotation, device binding, and server checking.

Sources

- **The secure programming technical guide** — the WBC attacker model, where whitebox AES sits in a product
- Chow, Eisen, Johnson, van Oorschot — WB-AES (SAC 2002), WB-DES (DRM 2002)
- Wyseur — "Hiding Keys in Software" (short introduction)
- BGE (SAC 2004); Bos et al. DCA (CHES 2016); "Unboxing the White-Box" (BH EU 2015)
- WhibOx competition results (2017–2024)

Next Week

Week 12 — Certification and Penetration-Test Planning

Measuring how much a protection "actually withstands" → the independent-evaluation process and reporting.



Appendix A · A Tiny Numeric Example

Why a Toy Example?

Real AES works with 256-entry tables; that won't fit on a board.

We'll see the idea with a tiny **4-value** example instead.

Goal: see by hand what "baking the key into a table" means.

Defining the Tiny S-box

Say we have 2-bit values (0,1,2,3) and this fixed S-box:

x:	0	1	2	3
S-box[x]:	3	2	0	1

This table is **public** (part of the algorithm, not a secret).

Key and Operation

- Key $k = 1$ (secret).
- Operation (a mini version of one AES round): $\text{output} = \text{S-box}[x \wedge k]$.

$\wedge$ = XOR. $x \wedge 1 : 0 \leftrightarrow 1, 2 \leftrightarrow 3$ (flips the last bit).

Step by Step: Normal Operation

```
x=0 → x^1=1 → S-box[1]=2  
x=1 → x^1=0 → S-box[0]=3  
x=2 → x^1=3 → S-box[3]=1  
x=3 → x^1=2 → S-box[2]=0
```

Here `k=1` is used **explicitly** in the code. The attacker sees it.

"Bake" the Key Into a Table

Let's precompute $T[x] = S\text{-box}[x \wedge 1]$ as a table:

x:	0	1	2	3
T[x]:	2	3	1	0

There's no `k` in the code anymore; only `T`. The key seems to have disappeared.

But the Key Leaks! (Without Encoding)

The attacker compares T with the known $S\text{-box}$:

```
S-box: 3 2 0 1  
T:      2 3 1 0
```

Knowing that $T[x] = S\text{-box}[x \wedge k]$, they try which k fits: $k=1$ fits.

Result: an unencoded table gives the key away. This is exactly why internal/external encoding exists.

The Idea of Encoding (Tiny)

Apply a secret mapping E to the output: $T'[x] = E(T[x])$.

$E: 0 \rightarrow 1, 1 \rightarrow 3, 2 \rightarrow 0, 3 \rightarrow 2$ (gizli bijeksiyon)

$T': E(2) E(3) E(1) E(0) = 0 2 3 1$

Now T' doesn't resemble $S\text{-box}$; a simple comparison doesn't give up k .

The Price of Encoding

- The next step must apply `E`'s **inverse** for the result to come out right.
- In real AES this is done by **chaining** tables (internal encodings).
- Every encoding means an extra table and more size → **cost**.

Even the tiny example shows the idea: secrecy isn't cheap.

The Lesson From the Tiny Example

1. Baking the key into a table doesn't make it **invisible** (encoding is required)
2. Encoding solves the problem but adds **size/complexity**
3. In reality the tables run to hundreds of KB

This is the **intuition** behind the five steps in Section 2.



Appendix B · Why Does DCA Work? (Intuition)

Intuition: What Is It Tracking?

- While WBC runs, it reads **intermediate values** from the tables.
- These intermediate values change **depending on** the secret key.
- DCA collects a trace of these values.

How Does Statistics Give Up the Key?

- The attacker makes a **guess** for one key byte.
- From the guess, they compute how the intermediate value **should** behave.
- Among the collected traces, the guess that **fits best** is the correct byte.

This is the **software** form of DPA (power analysis).

Why Doesn't Internal Encoding Stop DCA?

- Internal encoding **scrambles** the intermediate value but is a **one-to-one** mapping (bijection)
- Statistical correlation often **survives** it
- This is why WBC designs remained vulnerable to DCA after 2016

Countermeasure Intuition

- **Non-linear masking:** hides intermediate values in a way that breaks the correlation
- **External encoding:** takes the input/output the attacker sees out of being standard
- **RASP:** makes trace collection (attaching tools) harder

None is a definitive fix; all of them are **cost + delay**.



Appendix C · Decision Flow

"How Do I Protect This Key?" — 1

Question 1: Does the device have a TEE/secure element?

- **Yes** → put the key there. **Done** (strongest).
- **No** → continue.

"How Do I Protect This Key?" — 2

Question 2: Is this a server-side key?

- **Yes** → HSM/PKCS#11 (HSM in production, SoftHSM in testing).
- **No (client, no hardware)** → continue.

"How Do I Protect This Key?" — 3

Question 3: Does the asset's value justify the cost of protection?

- **Low** → light hiding + a short lifetime is enough.
- **High** → WBC + layered defence + rotation + device binding + server checking.

"How Do I Protect This Key?" — 4

In every case:

- Don't embed the key **plainly**
- Write the decision and its rationale **into S8**
- State the remaining risk explicitly

Decision Flow · at a Glance

Anahtar koruma seçenekleri

yukarıdan aşağı: güç azalır, maliyet de azalır

HSM / donanım

anahtar hiç çıkmaz — en güçlü, en pahalı

TEE / Güvenli öge (SE)

yalıtılmış işlemci bölgesi

WBC + bağlama + yenileme

geciktirir, tek başına değil

Saf yazılım WBC

yayımlananların hepsi kırıldı

Düz gömülü anahtar

koruma DEĞİL

Seçim varlığın değerine göre gerekçelendirilir ve S8'e yazılır.



Appendix D · Mini Case (Synthetic)

Case · Setup

A mobile app encrypts local data with a **session key** it downloads from the server.

- Some devices have a TEE, some don't.
- The key rotates once a day.

How do we protect it?

Case · Devices With a TEE

- Put the key **in the TEE**.
- The app never sees the key; encryption happens inside the TEE.
- The strongest solution; extra WBC is unnecessary.

Case · Devices Without a TEE

- Bake the key into **WBC tables** (no plain array).
- Wrap it with Week 9 obfuscation + Week 6 RASP.
- **Bind to the device:** the tables shouldn't work on another device.

Case · Shared Layers

- The key **rotates once a day** → a short lifetime for anything extracted.
- The **server** catches abnormal use (too many requests, an odd location).
- So even if one device gets broken, the damage is limited.

Case · Remaining Risk (Honest)

- On a device without a TEE, a determined attacker can extract the key.
- But: limited to one device + 24-hour lifetime + server checking.
- This trade-off is **written explicitly into S8**.

Case · Takeaway

There is no single "magic" protection.

Strength comes from the **combination of layers** and an **honest analysis of the remaining risk**.

This is exactly what the evaluator (Week 12) looks for.

Appendix — Closing Note

These appendices aren't separately asked on the exam; but:

- The tiny example gave the "baking + encoding" intuition
- The DCA intuition explained "why all of them were broken"
- The decision flow + case help you write S8

To **understand** WBC is to **put it in its right place**.



Appendix E · Solved Self-Check

Question 1

Distinguish the black, grey, and white box models in one sentence. Which one does the AES proof assume?

Answer 1

- **Black:** input/output only.
- **Grey:** + side channels (time, power).
- **White:** everything + can modify.

AES's security proof assumes **black box**; in deployment we're in white box.

Question 2

Which attacks do the WBC attacker's abilities to "run a single round" and "inject a fault" set up?

Answer 2

- Running a single round + collecting a trace → **DCA**.
- Injecting a fault → **DFA**.

Both work without knowing the design's internal math.

Question 3

Why isn't embedding the key in a fixed array protection? What does an entropy scan do here?

Answer 3

- The program uses the key while running; the block sits in memory/the binary.
- The **entropy scan** finds the high-entropy (random-looking) block → "the key is here."
- Confirmed in minutes with `strings` + gdb.

Question 4

What is code lifting? Which countermeasure closes it?

Answer 4

- The attacker copies the encrypting code (tables + interpreter) **without extracting** the key.
- They steal the function without knowing the key.
- Countermeasure: **device/version binding** — the tables are only meaningful on that device.

Question 5

Why is $T[x] = S\text{-box}[x \wedge k]$ insecure on its own?

Answer 5

- T is the known S -box shifted by $x \wedge k$.
- The attacker compares the two tables and recovers k from the shift amount.
- This is why internal/external **encoding** is needed.

Question 6

The difference between internal encoding and external encoding (F, G)? External encoding's biggest limit?

Answer 6

- **Internal:** between tables, scrambles the intermediate values.
- **External (F, G):** wraps the whole cipher ($G \circ \text{AES} \circ F^{-1}$).
- **Limit:** it's no longer **standard AES**; the other side has to know F/G → may not be usable in standards like EMV.

Question 7

Approximate table-size and speed cost? Why only on selected operations?

Answer 7

- Table: **hundreds of KB** (standard AES key is 16–32 bytes).
- Speed: **~50× slower**.
- This cost limits WBC to **small and critical** operations only.

Question 8

How does "every published pure-software WBC has been broken" affect your project decision?

Answer 8

- You can't count WBC as key assurance **on its own**.
- If you use it: with rotation + device binding + server checking + obfuscation/RASP.
- Move to hardware (TEE/SE) where possible.

Question 9

Which classic hardware attack does DCA resemble? Why don't internal encodings always stop it?

Answer 9

- Resembles **DPA** (power analysis) — its software form.
- Internal encoding scrambles the intermediate value but is a one-to-one mapping; statistical correlation often survives it.

Question 10

Rank the key-protection options along the strength/cost axis.

Answer 10

Plain array (none) → hidden/split (very low) → **WBC + layers** (medium) → **TEE/SE** (high) → **HSM** (high, server).

Rule: if you don't have to put it in software, don't.

Question 11

If you use WBC, what are at least three layers so it isn't left alone?

Answer 11

1. Obfuscation (Week 9) + RASP (Week 6)
2. Key rotation (short crypto-period)
3. Device/version binding + server-side risk checking

Question 12

Write the S8 rationale for a key in your project, in three sentences (example).

Answer 12 (Example)

"The session key is kept in WBC tables on devices without a TEE and wrapped with obfuscation+RASP. It's bound to the device against code lifting; the key rotates every 24 hours. Remaining risk: it can be extracted on a single device, but server-side risk checking and its short lifetime limit the damage."

Closing · Three Weeks Together

- **Week 9:** obfuscation rules (code)
- **Week 11:** whitebox (key) — today
- **Week 14:** automation (Tigress)

Common rule: protection is **not unbreakability, it's delay**; strength comes from layers and measurement.