

UNIDAD 1: BLOCKCHAIN - INTRODUCCIÓN

CASO PRÁCTICO 1: CIFRADO

1. Introducción

El objetivo de este caso práctico era crear una herramienta capaz de cifrar y descifrar texto de forma reversible, aplicando el concepto de integridad de datos.

Para ello desarrollé dos programas en Python 3 basados en el cifrado César, un método sencillo que desplaza las letras del alfabeto según una clave numérica.

El proyecto fue también una excusa para experimentar y divertirme un poco: lo ambienté con estética *hacker noventera* inspirada entre *Hackers* (1995) y *Matrix* (1999).

- El codificador (codif.py) representa al hacker *CrashOverride*, encargado de cifrar los datos.
- El decodificador (decod.py) representa a *AcidBurn*, quien logra revertir el cifrado y recuperar la información.

Así, además de cumplir con el ejercicio de cifrado, el resultado tiene un toque más personal y creativo.

2. Descripción del proyecto

El proyecto consta de dos scripts:

- codif.py → cifra el texto usando una clave numérica.
 - Permite modo normal (sin animación) y modo cinematográfico (*CrashOverride*).
 - Guarda los resultados en la carpeta mision/:
 - input.txt (texto original)
 - output.txt (texto cifrado)
 - log.txt (fecha, clave y hash SHA-256)
- decod.py → descifra el texto cifrado.
 - También tiene un modo con animación de *AcidBurn*, que simula el proceso de rastreo y recuperación de datos.

Ambos scripts aceptan argumentos desde terminal mediante argparse y pueden leer mensajes directos (-m) o desde archivos (-f).

3. Funcionamiento del codificador

Al ejecutar el programa con:

```
python codif.py -m "crashoverride" -k 707
```

se inicia la secuencia visual de *CrashOverride*.

El texto se cifra con una clave (707 ≡ 5 en el cifrado César) y se guardan los archivos correspondientes en mision/.

```
== CRASH OVERRIDE - INITIATING OVERDRIVE ===
Connecting to 0xDEADBEEF... OK
Injecting payload: [REDACTED] 65%
![]^_!;{{-e!^&b*_%d[()*$?/*#!f<-_!}{afe&f;<aa[-@?@&/fc#:=-}d?_*>+%,=,)_0-*&
0-!}?}/{#%*(f$/>&E;#):&0,_-d:;[#/d;e#%f;!/e=_,d<f]/0a]a{)+/-'f.=![,?<c)&q@#;
.^+{[*{(>%1/_#+!/&)01[$?%^..>%b&*>?0,&_+$]*.#f>*}a-{#<_1-<:^$&a};c:^(>[
])/_/^(?/c&{+?a$#<${{f1:-ad<-}}/}>?0%_<*/1[_;-_=]%,b10:b)!Q1,:lb!+//_c<=:1:1:_$d!
&a/_0$%d=<#c$d&a)]]<.lef[.}^&`_d{[(a(=a{a,^eb%.%})>?]!<,^#!#]_#).;,>ad&b.d;${=
e-_c.#&lt;<#1!0!e*{&,,($f==?e_f!_f#et#)>fd!>^=a$fd+[&&&]>?c_b>c>},0._c;1-0!/%
b>b#+[a?/)??_>^*[+e:1]+%>{b<00#f!=%+{%/e{*b_%}{&0>d::>>?-<eb()]}%&_#=:[*+-&
b->@:.d){..0}*>)*_f,f_*>d&{&:-/a@.e&1:>,:=(ef_,>b;^_0}{{&0+#{a/ac;a&=1_-;#Bypassing firewall... OK
Crack assembled.
OVERDRIVE in 3...
OVERDRIVE in 2...
OVERDRIVE in 1...
>>> PACKET STORM 5714
>>> PACKET STORM 6804
>>> PACKET STORM 7682
,[{"?a,$+/%F;)0#?[0:d1!$:c[%a=&e.$<b/$fd[1=<*(-=f)>,),e<e)&+e>&d_&f!%*{$1e)0.$=^(
#&;f#[+c[&:1c$@&0,>[?a;e@a_._#*b!?}*+&_.0$&d{a}^&Sc,(1=df[0];}?a-&,c_+!be{#e}^;f(.)#_
##!*#e_c.0$c1=#.b,#](e(<_f<10+e</-0>);$]^id^[$fb?f$**<*&:bf{,-[>,.b!^=dfc^d)}>.;@1@
b.>([f#*-.f!$.;,:]+[+,>e<_++:#:@_0)?/e_.,aceb)f[0<_,edf]:,#[{&c,&c;_<>:b;=$}]b++f
a{!c[;b$1ad)[#b&b;+>ca{;&}+a=[{f;??<+}_a_a.({$0)00ac#-[=_[>f;-((&c-#)<%-$d0)]->}.e.[dd
:/],e!/ea-#d*@>d0d0[___$,+}{.{}e?}>-/0.],J_.,{1-<*a}@exb/(d(?:[c?/=a)d?/>^-
{b})=(*%)(@^b$%^_,[#!d'f^-^le-@0d._ae!&#!a_?)cb}/{[b#/b@^0f,1@b<,1[*-,>]$*d}>-!@{($>:>1
c^]"+_!=0dc!.(*@{be}ac&<&bc@)ea#;?&})@{@{$e)?aeb.*?b#/*-^%f>,:?-%{,c#*b/a[-{.&b=.%#.
*** SYSTEM CRASH ***
```

Captura 1: Ejecución del codificador con animación.

```
PS E:\DAW\Blockchain\UD1\caso1> python codif.py --mode normal -m "Mess with the best, die like the rest." -k 707
Rjxx bnym ymj gjxy, inj qnpj ymj wjxy.
Archivos guardados en carpeta 'mision/'
```

Captura 2: Ejecución del codificador sin animación.

File Explorer				
This PC > Disco D (E:) > DAW > Blockchain > UD1 > caso1 > mision				
File View				
Name	Date modified	Type	Size	
input	01/11/2025 02:01	Text Document	1 KB	
log	01/11/2025 02:01	Text Document	1 KB	
output	01/11/2025 02:01	Text Document	1 KB	

Captura 3: Contenido de la carpeta mision/.

```
[2025-10-31 23:33:13.823715] Clave=707 | Hash=3a69cd3af12b5275
[2025-11-01 00:15:36.369142] Clave=707 | Hash=0f5c1c2f5581c49f
[2025-11-01 00:20:19.076968] Clave=707 | Hash=0f5c1c2f5581c49f
[2025-11-01 01:24:47.872079] Clave=707 | Hash=0f5c1c2f5581c49f
[2025-11-01 01:25:09.572961] Clave=707 | Hash=3a69cd3af12b5275
[2025-11-01 01:26:12.842206] Clave=707 | Hash=0f5c1c2f5581c49f
[2025-11-01 01:27:30.502767] Clave=707 | Hash=3a69cd3af12b5275
[2025-11-01 01:45:58.376273] Clave=707 | Hash=3a69cd3af12b5275
[2025-11-01 02:01:40.725443] Clave=707 | Hash=3a69cd3af12b5275
```

Captura 4: Extracto del archivo log.txt.

4. Funcionamiento del decodificador

El proceso inverso se realiza con:

```
python decod.py -f mision/output.txt -k 707 --anim
```

Esto muestra la animación de *AcidBurn*, quien “rastreando el ataque” recupera el texto original.

```
== ACIDBURN - STRIKING BACK ==
&_!++);^@++)^&!d_1:&*d[:a*]_a?{#f[<_?s1&!(@>^b^%_.!)&<<[e/$+]*#d^=1{}#_0)*%^/
*^/_;^?{$(c?e#[{}0*]^#-%,^%b=0)+{##&?_!e:(.b[.{++%:_:#$}d/?=:a)%&f,b,)b###f}
;{(f(e%)[*5a&fb];.b!@{/!{_<_{}[@!!e>.f>#,:_bc&]*e-/d?,c)};ld[b/*1?<,{*<_{}a.&
;?*?_*.][])<*(f$<*cf}]{e^_@}:*b^!/_c_d,*$<-c>da.a.[<]b-@_,/_;1a]+/1<.f:(();(#(fd)1;-d^_0)^?&.>@&f-!/e2bbc#:>@:.d!@a10,_[(?*$?-+_?:((C,=(1!@->1:@)]1%{!*d*$/++%}_#;
;c/_01f.(>b.0a(0-)$<-,1))@a>+f!&&+!,d.d$b^f:+*{(&?f^@$=_@[<a{*_1,(_#])1+<#%
;:ic,.&e.?fb[e?>]=0&#,>0-<-/-CC/[&:@^:][f!@0@1_?=l?e]*#;@=10&d*[;1=a,fe/{;
;$d][;,&bef/;/&{&:-<0e+[=.0#>|e/%0+;&$d;#%_[1.#_+]0!^-[;,:e>;1>|-_,@[#+/a{/.@
Trace detected - locking target.
Target locked.
...
Neutralizing: [ ] 100%
[RECOVERED DATA]
Clave: 707
Original: Mess with the best, die like the rest.

AcidBurn: YOU GOT BURNED. Session terminated.
```

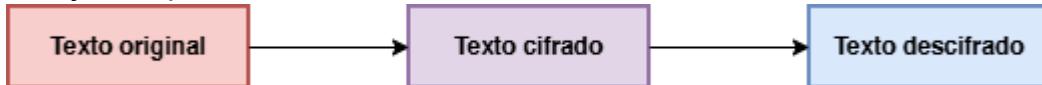
Captura 5: Ejecución del decodificador con animación y texto de salida.

```
PS E:\DAW\Blockchain\UD1\caso1> python decod.py -f mision/output.txt -k 707
Texto descifrado:
Mess with the best, die like the rest.
```

Captura 6: Ejecución del decodificador sin animación y texto de salida.

5. Resultados

El flujo completo demuestra un cifrado reversible:



Captura 7: Proceso de cifrado y descifrado del mensaje.

y cómo puede registrarse un **hash** para comprobar la integridad del mensaje, concepto que también se aplica en sistemas basados en blockchain.

Video del código en acción en: <https://youtu.be/urLft32FYwE>

6. Conclusión

Este ejercicio me permitió comprender de forma práctica la diferencia entre cifrado y hashing, y cómo ambos contribuyen a la integridad de los datos.

Además de reforzar el uso de Python para leer y escribir archivos, fue una oportunidad para practicar argumentos por terminal y darle un toque creativo al resultado con animaciones tipo consola.

7. Recursos consultados

- Curso Python de Udemy.
- *¿Cómo programar un cifrado de César en Python? Aprende en 9 minutos* (Youtube):
<https://www.youtube.com/watch?v=3jeFGqLWeE8>
- *Cifrado de César en Python* (Youtube):
<https://www.youtube.com/watch?v=DhSIInnOIK1c>