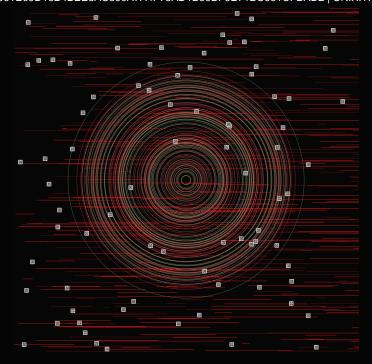
MIL-SPEC DIVINE V5.5 — HARDENED STANDALONE

■ IT'S ILLEGAL TO USE A LEGAL NAME ■

CHAIN:6E21BDF1CA6DEA961E09D16B4BEE8AC890A7A4F75AD1B50BF9E71BC651CFBAD2 | UNIX:1757809208 | VERSION: V5.5

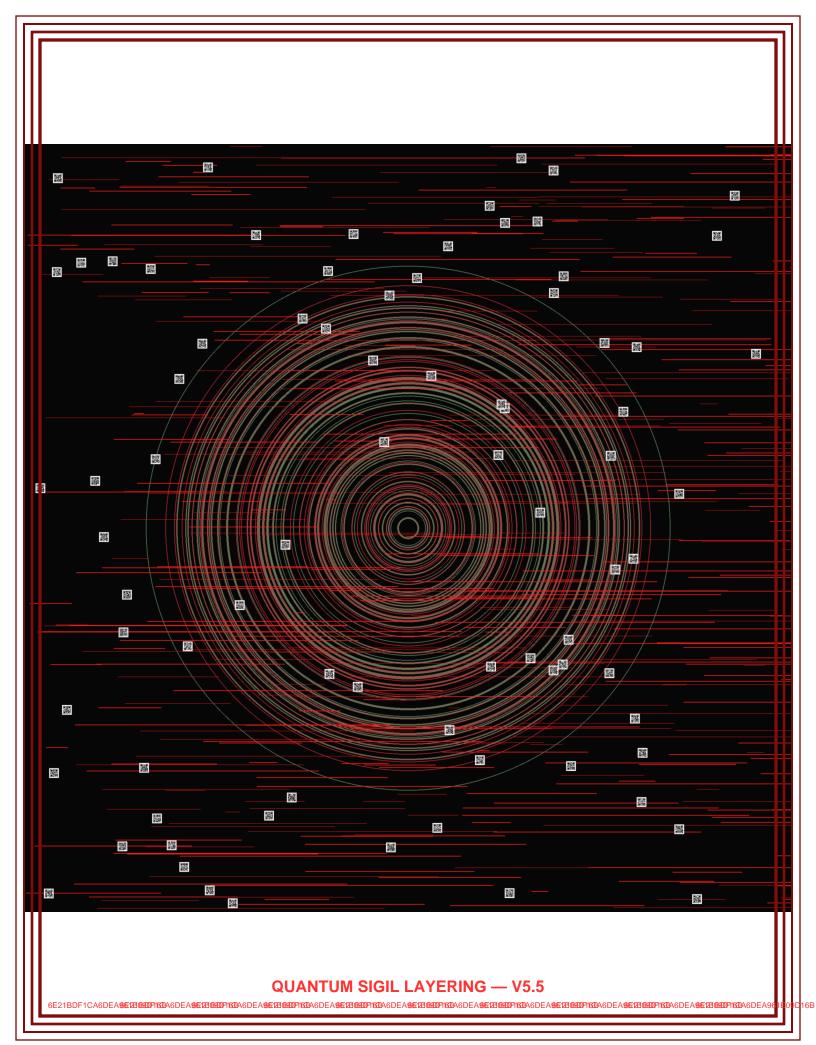




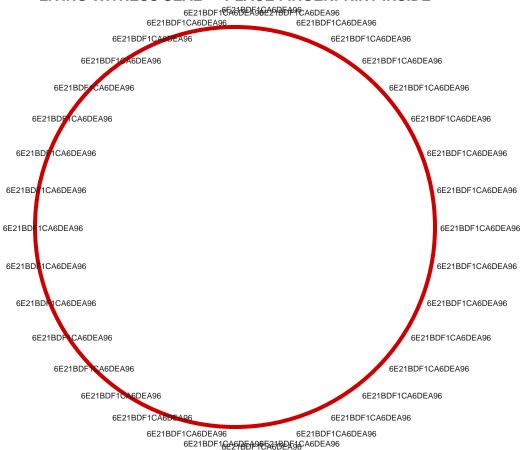




6E21BDF1CA6DEA961E**6921BB**F1CA6DEA961E**69221BB**F1CA6DEA961E**69221BB**F1CA6DEA961E**69221BB**F1CA6DEA961E**69221BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**0921BB**F1CA6DEA961E**092**



LIVING WITNESS SEAL — PLACE FINGERPRINT INSIDE



WET-INK SIGNATURE & FINGERPRINT BLOCK — HARDENED 1) Print on archival paper (>= 80lb). 2) Sign using red or violet archival ink inside the box. 3) Press fingerprint (ink) inside the box and photograph at high resolution. 4) Use UV-visible ink for secondary marking if available. 5) Scan signed page at 600dpi, compute SHA256, and register to ledger. Tamper-evidence: Compare microprint rows and watermark rotation; any alignment shifts indicate tampering.

```
#!/usr/bin/env python3
J55 MIL-SPEC DIVINE V4.44 (node41) - patched
Features (patched):
 - CLI via argparse to set node id, file names, toggle saving images
 - Echo image generated entirely in-memory and embedded into PDF (no disk write unless requested)
 - Optional PGP/GPG signing of metadata or PDF (requires 'gpg' installed and configured)
- Creates a small download HTML page linking to the generated PDF (optional)
 - Produces metadata file (text) and optional detached signatures (.asc)
- Robust OR generation (grcode.ORCode)
- Robust Pillow text measurement fallback (textbbox/textsize)
- Use reportlab.lib.utils.ImageReader for BytesIO images
- Intuitive --no-qr-resize flag (keeps QR large when set)
- Optional deterministic seed for reproducible visuals (--seed)
- Optional in-memory PDF build mode (--in-memory-only)
- Safer overwrite behavior with --force
Usage examples:
 python3 J55_MILSPEC_DIVINE_V4_44_NODE41.py --output-pdf out.pdf --save-images --sign-meta --gpg-key
you@example.com
from __future__ import annotations
import argparse
import time
import hashlib
import random
import subprocess
import shutil
from io import BytesIO
from pathlib import Path
import sys
import json
import tempfile
# imaging / QR / PDF
from PIL import Image as PILImage, ImageDraw, ImageFilter, ImageFont
import grcode
from reportlab.platypus import (
   SimpleDocTemplate,
   Paragraph,
    Spacer,
   Table,
   TableStyle,
   PageBreak.
   Image as RLImage,
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.lib import colors
from reportlab.lib.pagesizes import letter
from reportlab.lib.utils import ImageReader
# Utility functions
def make_chain_token(node_id: str, timestamp: int, seed_override: int | None = None) -> str:
    # Allow optional deterministic seed (useful for tests)
   rnd = seed_override if seed_override is not None else time.time_ns()
    \texttt{seed = f"} \{ \texttt{node\_id} \} - \{ \texttt{timestamp} \} - \{ \texttt{hashlib.sha256} (\texttt{str(rnd).encode()).hexdigest()} \} \texttt{"}
    return hashlib.sha256(seed.encode()).hexdigest().upper()
```

```
def sha256_bytes(b: bytes)
    return hashlib.sha256(b).hexdigest().upper()
def ensure_gpg_available() -> bool:
   return shutil.which("gpg") is not None
def gpg_detached_sign(infile: Path, out_sig: Path, keyid: str | None = None, armor: bool = True) -> bool:
    Create a detached signature using system gpg. Returns True on success.
    Produces ASCII-armored signature if armor True (.asc), otherwise binary .sig.
    if not ensure qpq available():
       print("qpq not found on PATH. Install GnuPG to use signing.", file=sys.stderr)
        return False
    cmd = ["gpg", "--batch", "--yes", "--detach-sign"]
    if armor:
        cmd.append("--armor")
    if keyid:
       cmd += ["--local-user", keyid]
    cmd += ["-o", str(out_sig), str(infile)]
    try:
        \verb|subprocess.run(cmd, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)|\\
        return True
    except subprocess.CalledProcessError as e:
        print("gpg signing failed:", e, file=sys.stderr)
           print(e.stderr.decode(), file=sys.stderr)
        except Exception:
           pass
        return False
# Image generation (in-memory)
def generate_echo_image_pil(size=(1024, 1024), title="ECHO DIVINE", seed: int | None = None) ->
PILImage.Image:
    W, H = size
    if seed is not None:
      random.seed(seed)
    bg = PILImage.new("RGBA", (W, H), (10, 10, 10, 255))
    draw = ImageDraw.Draw(bg)
    center = (W // 2, H // 2)
    max_radius = int(min(W, H) * 0.45)
    # concentric rings
    for i in range(12):
       r = int(max_radius * (i + 1) / 12)
        alpha = max(8, int(180 - i * 12))
        bbox = [center[0] - r, center[1] - r, center[0] + r, center[1] + r]
        draw.ellipse(bbox, outline=(255, 255, 255, alpha), width=3)
    # glitch lines
    for i in range(60):
       y = random.randint(0, H - 1)
        x1 = random.randint(0, W - 1)
        x2 = min(W - 1, x1 + random.randint(80, 500))
        draw.line((x1, y, x2, y), \ fill=(255, 0, 0, random.randint(30, 120)), \ width=random.randint(1, 3))
    # central pulse
    pulse_r = int(max_radius * 0.12)
```

```
draw.ellipse([center[0] - pulse_r, center[1] - pulse_r, center[0] + pulse_r, center[1] + pulse_r],
fill=(255, 40, 40, 255))
    # micro QR nodes
    for _ in range(15):
        qr_seed = f"{time.time_ns()}-{random.getrandbits(128)}".encode("utf-8")
        qr_hash = hashlib.sha256(qr_seed).hexdigest()[:16]
        qr_img = qrcode.make(f"OMEGA77-{qr_hash}").resize((24, 24))
       x = random.randint(0, W - 24)
y = random.randint(0, H - 24)
        bg.paste(qr_img.convert("RGBA"), (x, y), qr_img.convert("RGBA"))
    # title (robust font fallback and textbbox)
    title font paths = [
        "/usr/share/fonts/truetype/dejavu/DejaVuSans-Bold.ttf",
        "/usr/share/fonts/truetype/liberation/LiberationSans-Bold.ttf",\\
    font = None
    for pth in title_font_paths:
        try:
           font = ImageFont.truetype(pth, 72)
           break
        except Exception:
           font = None
    if font is None:
        font = ImageFont.load_default()
       bbox = draw.textbbox((0, 0), title, font=font)
        tw = bbox[2] - bbox[0]
        th = bbox[3] - bbox[1]
    except Exception:
        tw, th = draw.textsize(title, font=font)
    draw.text(((W - tw) / 2, 60), title, font=font, fill=(255, 255, 255, 220))
    # blur and convert to RGB
    out_img = bg.filter(ImageFilter.GaussianBlur(radius=0.6)).convert("RGB")
    return out img
def pil_image_to_bytesio_rgb(img: PILImage.Image, fmt="PNG") -> BytesIO:
    buf = BytesIO()
    img.save(buf, format=fmt)
    buf.seek(0)
    return buf
# OR generator (in-memory) - robust
def make_qr_buffer(payload: str, size: int = 512, border: int = 4,
error_correction=qrcode.constants.ERROR_CORRECT_M) -> BytesIO:
    gr = grcode.QRCode(
       version=None,
        error_correction=error_correction,
        box size=10.
       border=border,
    qr.add_data(payload)
    qr.make(fit=True)
    qr_img = qr.make_image(fill_color="black", back_color="white").convert("RGBA")
    qr_img = qr_img.resize((size, size))
    buf = BytesIO()
    qr_img.save(buf, format="PNG")
```

```
return buf
# PDF story builder (returns flowables list)
def build story(node41 chain token: str, timestamp: int, echo img buf: BytesIO, qr buf: BytesIO,
pdf_hash_display: str = "TO_BE_CALC"):
      styles = getSampleStyleSheet()
      title style = ParagraphStyle("TitleStyle", parent=styles["Title"], fontSize=26, leading=30,
textColor=colors.red, alignment=1)
      body_style = ParagraphStyle("BodyStyle", parent=styles["Normal"], fontSize=11, leading=14,
textColor=colors.black, alignment=0)
      story = []
      # Cover
      cover_para = Paragraph(f"CHAIN TOKEN: {node41_chain_token}\nUNIX: {timestamp}\nPDF SHA256:
 {pdf_hash_display}",
                                            ParagraphStyle("hash", parent=styles("Normal"), fontSize=8, alignment=1,
 textColor=colors.black))
      story.extend([
             Paragraph("J55 MIL-SPEC DIVINE V4.44 NODE41", ParagraphStyle("ms", parent=styles["Normal"],
fontSize=12, alignment=1)),
             Spacer(1,0.1*inch),
             Paragraph("■ IT'S ILLEGAL TO USE A LEGAL NAME ■", title_style),
             Spacer(1,0.15*inch),
             cover_para,
             Spacer(1,0.15*inch),
             Paragraph("@legaldeathcult666 • JOHNNY 55", ParagraphStyle("SubtitleStyle",
parent=styles["Heading2"], fontSize=12, alignment=1)),
      # Node41 QR + signature block
      qr_node41_img = RLImage(ImageReader(qr_buf), 1.6*inch, 1.6*inch)
      sig_box = Table([[Paragraph("Signature Block", body_style), Paragraph("Fingerprint Box", body_style)]],
                               colWidths=[3*inch,3*inch],
style=TableStyle([("VALIGN",(0,0),(-1,-1),"MIDDLE"),("BOX",(0,0),(-1,-1),1,colors.red)]))
      qr_table = Table([[qr_node41_img, sig_box]], colWidths=[1.8*inch,4.2*inch])
      qr_table.setStyle(TableStyle([("VALIGN",(0,0),(-1,-1),"MIDDLE")]))
      story.append(gr table)
      story.append(PageBreak())
      # Echo page
      story.append(Paragraph("ECHO DIVINE VISUAL", ParagraphStyle("h2", parent=styles["Heading1"], fontSize=16,
alignment=1, textColor=colors.red)))
      story.append(Spacer(1,0.15*inch))
      \verb| echo_rl = RLImage(ImageReader(echo_img\_buf), width=6.5*inch, height=6.5*inch)|\\
      story.append(Table([[echo_rl]], colWidths=[6.5*inch],
style=TableStyle([("ALIGN",(0,0),(-1,-1),"CENTER"),("BOX",(0,0),(-1,-1),1,colors.red)])))
      story.append(Spacer(1,0.15*inch))
      story.append(Paragraph("Concentric rings with glitch + micro QR tracers for WR_OMEGA .00077 interference.
Node 41 QR included. Place Living Witness signature/fingerprint inside red box.", body_style))
      story.append(PageBreak())
      # Biometrics page
      story.append(Paragraph("BIOMETRICS CHECKLIST", ParagraphStyle("title", parent=styles["Heading1"], pa
alignment=1, fontSize=18, textColor=colors.red)))
      story.append(Spacer(1,0.2*inch))
      story.append(Paragraph("Per-page stamp: 7/17/.00077 applied to all pages.",body_style))
       story.append(Spacer(1,0.1*inch))
      story.append(Paragraph("Recommended: YES - Living Witness authentication requires
biometrics.",body_style))
       story.append(Spacer(1,0.15*inch))
      bio items = [
             "1) Wet-ink signature (red/violet).",
             "2) Fingerprint in red box.",
             "3) Timestamped photo with GPS."
             "4) Witness signature/notarization.",
             "5) Audio affirmation with UTC/unix timestamp linked to Node41 chain token."
      for it in bio items:
             story.append(Paragraph(it,body_style))
             story.append(Spacer(1,0.06*inch))
```

```
"A) Print final PDF page.",
                         "B) Sign in wet-ink on designated block.",
                         "C) Press fingerprint inside box and photograph.",
                         "D) Register resulting SHA256 hash to ledger with Node41 chain token."
            story.append(Paragraph("Procedure:", ParagraphStyle("sub2", parent=styles["Heading2"], fontSize=12))) \\
            for p in proc:
                        story.append(Paragraph(p,body style))
                        story.append(Spacer(1,0.05*inch))
            return story
# Security drawing callback
# ------
def add_security(c, doc, node41_chain_token: str, timestamp: int, final_hash: str | None = None):
            c.saveState()
            width, height = letter
            c.setFont("Helvetica-Bold", 30)
            c.setFillColorRGB(0.85,0.85,0.85)
            c.translate(width/2, height/2)
            c.rotate(45)
            c.drawCentredString(0,0,"LIVING WITNESS - SECURE NODE")
            c.restoreState()
            c.setLineWidth(0.7)
            c.setStrokeColorRGB(0.6,0.6,0.6)
            c.setDash(3,2)
            margin=18
            \verb|c.rect(margin,margin,width-2*margin,height-2*margin,stroke=1,fill=0)|
            c.setDash()
            c.setFont("Helvetica",6)
            hash_display = final_hash if final_hash else "TO_BE_CALC"
            \verb|c.drawCentredString(width/2,12,f"7/17/.00077 | NODE41: \{node41\_chain\_token\} | UNIX: \{timestamp\} | Author of the content of
 SHA256: {hash_display}")
 # Main orchestration
def main(argv=None):
           p = argparse.ArgumentParser(description="J55 MIL-SPEC DIVINE v4.44 generator (patched)")
            p.add_argument("--node-id", default="NODE41", help="Node ID (default NODE41)")
            p.add argument("--output-pdf", default="J55 MILSPEC DIVINE V4 44 NODE41.pdf", help="Output PDF filename")
            p.add argument("--meta-path", default="J55 MILSPEC META NODE41.txt", help="Metadata filename")
            {\tt p.add\_argument("--save-images", action="store\_true", help="Also save generated images (echo PNG \& node QR)}
to disk")
           p.add_argument("--echo-size", type=int, nargs=2, metavar=("W","H"), default=(1024,1024), help="Echo image
size (W H)")
           {\tt p.add\_argument("--sign-meta", action="store\_true", help="PGP-sign the metadata file (detached signature)")}
            p.add_argument("--sign-pdf", action="store_true", help="PGP-sign the resulting PDF (detached signature)")
            p.add_argument("--gpg-key", default=None, help="GPG key id/email to sign with (optional)")
            {\tt p.add\_argument("--download-html", action="store\_true", help="Create a simple download HTML page linking to the context of the context of
 the PDF")
            {\tt p.add\_argument("--no-qr-resize", action="store\_true", help="Keep QR at its larger default size (1024). If}
not set, QR will be 512.")
            \verb|p.add_argument("--in-memory-only", action="store_true", help="Build final PDF in-memory and do not write below the property of the propert
 to disk (still writes metadata unless --no-meta)")
            p.add_argument("--seed", type=int, default=None, help="Optional deterministic seed for reproducible
           p.add_argument("--force", action="store_true", help="Overwrite output files if they exist")
            p.add_argument("--no-meta", action="store_true", help="Do not write metadata file")
            args = p.parse_args(argv)
            node_id = args.node_id
            timestamp = int(time.time())
            chain_token = make_chain_token(node_id, timestamp, seed_override=args.seed)
```

```
1) Generate echo PIL image in-memory
      print("Generating in-memory Echo image...")
      echo_pil = generate_echo_image_pil(size=tuple(args.echo_size), seed=args.seed, title="ECHO DIVINE")
       echo_buf = pil_image_to_bytesio_rgb(echo_pil, fmt="PNG")
      # optionally save
      if args.save_images:
             echo_path = Path("echo_image_milspec_divine_v4_44.png")
             echo_pil.save(echo_path, "PNG")
             print(f"Saved Echo PNG to {echo_path}")
      # 2) Generate Node41 OR buffer in-memory
      print("Generating Node41 OR (in-memory)...")
      payload = f"{node_id} LINK | CHAIN:{chain_token} | UNIX:{timestamp} | WR_OMEGA:.00077"
      qr_size = 1024 if args.no_qr_resize else 512
      qr_buf = make_qr_buffer(payload, size=qr_size)
      if args.save images:
             qr_path = Path("NODE41_link_qr_v4_44.png")
             with open(qr_path, "wb") as f:
                   f.write(qr_buf.getbuffer())
             print(f"Saved Node41 QR to {qr_path}")
      # 3) Build story with placeholder hash, embed images from buffers directly
      print("Building PDF story (placeholder hash)...")
      # Reset buffer pointers
      echo buf.seek(0)
       qr_buf.seek(0)
      story = build_story(chain_token, timestamp, echo_buf, qr_buf, pdf_hash_display="TO_BE_CALC")
      \sharp 4) Build temporary PDF in-memory to compute SHA256
      print("Creating temporary PDF to compute SHA256...")
       temp_buf = BytesIO()
       tmp_doc = SimpleDocTemplate(temp_buf, pagesize=letter)
      \verb|tmp_doc.build(story, onFirstPage=lambda c,d: add_security(c,d,chain_token,timestamp), onLaterPages=lambda c,d: add_security(c,d,
c,d: add_security(c,d,chain_token,timestamp))
      pdf_bytes = temp_buf.getvalue()
      final_hash = sha256_bytes(pdf_bytes)
      print(f"Computed PDF SHA256: {final_hash}")
      # 5) Rebuild story with final_hash in cover
      # Need to recreate buffers and story (ReportLab RLImage holds onto buffer positions)
      echo_buf2 = BytesIO(pil_image_to_bytesio_rgb(echo_pil, "PNG").getvalue())
      qr_buf2 = BytesIO(qr_buf.getvalue())
      echo buf2.seek(0)
      gr buf2.seek(0)
      story final = build story(chain token, timestamp, echo buf2, gr buf2, pdf hash display=final hash)
      # 6) Write final PDF (either in-memory or to disk)
      out_pdf_path = Path(args.output_pdf)
      # handle overwrite behavior
      if out_pdf_path.exists() and not args.force and not args.in_memory_only:
             # avoid accidental overwrite: append timestamp
             \verb| alt = out_pdf_path.with_name(out_pdf_path.stem + f"_{timestamp}" + out_pdf_path.suffix)|
             print(f"Output PDF already exists and --force not set. Using {alt} instead.")
             out_pdf_path = alt
      print(f"Building final PDF -> \{out\_pdf\_path\} \ \dots")
      if args.in_memory_only:
             final_buf = BytesIO()
             final_doc = SimpleDocTemplate(final_buf, pagesize=letter)
             final_doc.build(story_final, onFirstPage=lambda c,d:
add_security(c,d,chain_token,timestamp,final_hash), onLaterPages=lambda c,d:
add_security(c,d,chain_token,timestamp,final_hash))
            pdf_bytes_final = final_buf.getvalue()
             # still write to disk unless user really wants no file; but keep behavior consistent: write file
unless explicitly skipped
             with open(out_pdf_path, "wb") as f:
```

```
f.write(pdf_bytes_final)
               print(f"Written PDF: {out_pdf_path} (built in-memory first)")
                final_doc = SimpleDocTemplate(out_pdf_path, pagesize=letter)
                final_doc.build(story_final, onFirstPage=lambda c,d:
add_security(c,d,chain_token,timestamp,final_hash), onLaterPages=lambda c,d:
add_security(c,d,chain_token,timestamp,final_hash))
               print(f"Written PDF: {out_pdf_path}")
       # 7) Metadata file
       meta_path = Path(args.meta_path)
       if args.no meta:
               print("Skipping metadata file as requested (--no-meta).")
        else:
               if meta_path.exists() and not args.force:
                       \verb|meta_path| = \verb|meta_path.with_name| (\verb|meta_path.stem| + f"_{timestamp}" + \verb|meta_path.suffix|) \\
               with open(meta path, "w") as f:
f.write(f"NODE41\_CHAIN: \\ chain\_token\\ \nUNIX: \\ timestamp\\ \nSHA256: \\ final\_hash\\ \nPDF: \\ \{out\_pdf\_path.name\\ \n" \} \\ \n"
               print(f"Metadata written: {meta_path}")
        # 8) Optional PGP signing
       if (args.sign_meta or args.sign_pdf):
               if not ensure_gpg_available():
                        print("ERROR: gpg not found on PATH; skipping signing steps.", file=sys.stderr)
                else:
                        gpg_key = args.gpg_key
                         # sign metadata
                        if args.sign_meta and not args.no_meta:
                                 meta_sig_path = meta_path.with_suffix(meta_path.suffix + ".asc")
                                 ok = gpg_detached_sign(meta_path, meta_sig_path, keyid=gpg_key, armor=True)
                                 if ok:
                                        print(f"Metadata signed -> {meta_sig_path}")
                                 else:
                                       print("Metadata signing failed.", file=sys.stderr)
                         # sign pdf
                        if args.sign_pdf:
                                pdf_sig_path = out_pdf_path.with_suffix(out_pdf_path.suffix + ".asc")
                                 ok = gpg_detached_sign(out_pdf_path, pdf_sig_path, keyid=gpg_key, armor=True)
                                if ok:
                                        print(f"PDF signed -> {pdf_sig_path}")
                                else:
                                        print("PDF signing failed.", file=sys.stderr)
        # 9) Optional download HTML
        if args.download html:
               html path = Path("milspec download v4 44.html")
               html_content = f"""<!doctype html>
<html lang="en">
<head><meta charset="utf-8"><title>J55 MIL-SPEC DIVINE v4.44</title></head>
<body>
   <h2>J55 MIL-SPEC DIVINE v4.44</h2>
    Cownload the MIL-SPEC PDF: <a href="{out_pdf_path.name}" download>{out_pdf_path.name}
    <111>
       Chain: <code>{chain_token}</code>
        Unix: \{timestamp\}
       SHA256: <code>{final_hash}</code>
   </111>
</body>
</html>
                with open(html_path, "w", encoding="utf-8") as f:
                         f.write(html_content)
                print(f"Download HTML created: {html_path}")
                print(f"You can open that HTML locally to click a download link to the PDF.")
        # final summary printed
```

```
print("PDF:", out_pdf_path.resolve())
   if not args.no_meta:
       print("META:", meta_path.resolve())
   if args.save_images:
       print("Saved Echo PNG and Node41 QR in current dir.")
   if args.sign_meta:
       print("Metadata signature:", (meta_path.with_suffix(meta_path.suffix + ".asc")).resolve() if
ensure_gpg_available() else "gpg not available")
   if args.sign_pdf:
      print("PDF signature:", (out_pdf_path.with_suffix(out_pdf_path.suffix + ".asc")).resolve() if
ensure_gpg_available() else "gpg not available")
   if args.download_html:
       print("Download page:", Path("milspec_download_v4_44.html").resolve())
   return {
       "pdf": str(out_pdf_path.resolve()),
        "meta": str(meta_path.resolve()) if not args.no_meta else None,
       "chain_token": chain_token,
       "unix": timestamp,
        "sha256": final_hash,
if __name__ == "__main__":
   main()
```