Encrypted Arch Linux on SATA-to-USB

A Practical Field Guide for Resisting Drive Seizure, Forensic Analysis, and State Coercion

The Techno Anarchist

Contents

About This Guide

This guide has one specific goal: help you build and operate an encrypted Arch Linux system on a portable SATA-to-USB drive so that if the drive is physically seized by an authoritarian authority, they obtain nothing useful.

It covers the complete picture — from choosing hardware through emergency destruction — organized as a linear book you can follow from beginning to end, or reference chapter by chapter when you need a specific procedure.

What this guide covers:

- Hardware selection and SATA-to-USB enclosure setup
- Full-disk encryption that resists offline brute-force at any scale
- Anti-forensic techniques tailored to how state forensics actually works — not how it is portrayed in television
- Plausible deniability: how to make the drive appear to contain something harmless even if you are forced to unlock it
- Boot security that detects and resists tampering while you are away from the device
- Network anonymization for the actual use of the machine
- Emergency destruction procedures for when physical security fails
- Legal frameworks and what to say — and not say — under duress

What this guide does not cover: This is not a general Linux security guide, a penetration-testing manual, or an offensive security resource. Every chapter exists to solve one specific problem: protecting a portable encrypted drive from a state adversary who can seize it, image it, and attack it offline.

Part I — Understanding the Threat

Before writing a single configuration file, you must understand who you are defending against and exactly what they do. Incorrect threat modeling produces a system that is hardened in the wrong places.

Chapter 1. Your Adversary

The Authoritarian State Adversary

This guide is written for one specific adversary tier: a national or regional law enforcement or intelligence agency operating under an authoritarian government. This is a narrower, more tractable threat model than "all possible adversaries," which allows precise, actionable guidance instead of vague generalities.

What they have:

- Legal authority — or willingness to act without it — to seize your device at a checkpoint, during a raid, or at a border crossing
- Access to commercial forensic tools: Cellebrite UFED 4PC, Magnet AXIOM, EnCase, FTK, Passware Kit Forensic, and state-specific equivalents built for their domestic threat environment
- The ability to compel you to decrypt under threat of criminal prosecution, indefinite detention, or physical coercion
- Time — they can take the drive offline and run passphrase attacks indefinitely without any cost pressure
- Potentially: the ability to access a running, networked machine remotely if your device was compromised before seizure

What they do not have (against a properly configured system):

- Any mathematical attack against AES-256. None exists. Even theoretical quantum attacks (Grover's algorithm) only reduce AES-256 security to 128-bit equivalence — still far beyond any foreseeable computational attack
- The ability to brute-force a six-word diceware passphrase at any realistic timescale when protected by Argon2id with a 4 GiB memory parameter
- A method to decrypt the drive without the passphrase, keyfile, or hardware token
- A way to prove the existence of a VeraCrypt hidden volume in the unallocated space of an outer volume
- A way to detect that a partition is encrypted at all if you use a detached LUKS header

What they will do in practice — in order:

1. Physically seize the device, typically powered off
2. Image the drive forensically before doing anything else, to preserve the evidence state
3. Run automated analysis to identify encryption type and parameters
4. Begin a passphrase recovery attack calibrated to your KDF parameters
5. Apply legal or physical coercion to obtain the passphrase
6. If decryption fails: attempt to use possession of the encrypted drive itself as evidence

Your defense must address each of these steps. This guide is organized around that sequence.

The Attack Vectors Ranked by Practicality

Attack	Practical?	Primary Defense
Drive seizure + offline brute force	Very common — the default outcome	Strong passphrase + Argon2id
Legal compulsion to decrypt	Common in authoritarian jurisdictions	Plausible deniability + jurisdiction planning
Evil maid (tamper while unattended)	Possible if device left unattended	TPM measured boot, Secure Boot, anti-tamper indicators
Physical coercion (detention, threats)	Real risk for high-profile targets	Duress passphrase, deniability, legal preparation
Remote access to a running machine	Only if networked while compromised	Tor, network hardening, air-gap for sensitive sessions
Cold boot (RAM attack while running)	Requires access to a running machine	Shut down immediately when threatened; never sleep

Five Foundational Principles

Principle 1: Encryption at rest is your primary defense. Everything else is defense in depth. A properly encrypted, powered-off drive resists all known cryptanalytic attacks by any actor — state, corporate, or academic.

Principle 2: Deniability matters when compulsion is a risk. Mathematical security alone is insufficient in authoritarian contexts where you can be legally or physically forced to provide the passphrase. Deniability changes the game: you provide the decoy passphrase and they see only what you intended them to see.

Principle 3: Data you don't have cannot be seized. Minimize what you store. Information that does not need to live on the drive should not exist on it. Use tmpfs (RAM-backed storage) for working data that does not need to survive a reboot.

Principle 4: Your passphrase is the weakest link after encryption itself. A passphrase you can remember is vulnerable to coercion. A short passphrase is vulnerable to brute force. These two constraints pull in opposite directions — diceware with Argon2id resolves this tension.

Principle 5: Physical security enables digital security. No cryptographic scheme protects you if an adversary has undetected access to your running, unlocked machine. Boot security, anti-tamper indicators, and shutdown discipline are not optional extras — they are prerequisites.

Chapter 2. How State Drive Forensics Works

Understanding your adversary's tools and methods allows you to design effective countermeasures. This chapter explains in detail what happens when a forensics team receives your seized drive — and why each step of this guide directly counters a specific phase of that process.

Phase 1: Documentation and Acquisition

Before touching a seized device, a trained forensics team follows strict procedures to preserve the legal admissibility of evidence:

1. **Photography:** The device, its placement, all connections, and any visible screen state are photographed. This documents the state at seizure and can later help reconstruct whether the device was running, sleeping, or powered off.

2. **State assessment:** If the machine is running or sleeping, this is a critical opportunity. The LUKS decryption key exists in RAM while the device is unlocked. A live memory acquisition using tools like Magnet RAM Capture, LiME (Linux Memory Extractor), or Rekall can extract the full RAM contents — potentially including the mounted encryption key. This is why you must *never* sleep the machine in a threat situation. Always shut down.

3. **Hardware write blocker:** A physical write blocker (e.g., Tableau T35u, WiebeTech Forensic UltraDock) connects between the drive and the forensic workstation. It passes read commands but blocks any writes, ensuring the original drive is never modified. This preserves the legal evidence chain.

4. **Forensic imaging:** A bit-for-bit image of the entire drive is created using tools like Guymager, FTK Imager, or `dcfldd`. Every sector, including unallocated space, is copied. For a 500 GB SSD over USB 3.0, this takes 30–90 minutes.

5. **Hash verification:** SHA-256 hashes of the original and the image are compared. They must be identical. Any discrepancy invalidates the evidence chain. All subsequent analysis is performed on the copy; the original is sealed as court exhibit.

This means everything on the drive is captured forever at the moment of seizure. There is no "undo." Emergency destruction (Chapter 20) must happen before the drive is imaged, not after.

Phase 2: Automated Analysis

After imaging, tools scan the image for known patterns:

Encryption detection: Commercial tools maintain databases of file signatures and encryption headers. LUKS2 is identified by the 6-byte magic sequence `4C 55 4B 53 BA BE` at the start of the partition. When detected, the LUKS header is automatically parsed, revealing:

- The exact cipher and mode (AES-XTS-256)
- The PBKDF and its memory/iteration parameters (Argon2id, 4 GiB)
- How many key slots contain active keys
- The anti-forensic split (the encrypted master key — useless without the passphrase)

This information allows the forensics team to calibrate their brute-force attack precisely to your configuration.

File system analysis on unencrypted partitions: The EFI System Partition (your /boot) is unencrypted. Tools will thoroughly examine it, looking for:

- Hostname embedded in bootloader configuration
- Kernel command-line parameters (can reveal the encrypted device UUID)
- Any scripts, config files, or logs that were accidentally placed in /boot
- Timestamps of bootloader installation and configuration

File carving: On any unencrypted or successfully decrypted partition, carvers like Scalpel, PhotoRec, and Foremost scan raw disk space for file magic bytes. They can recover files from unallocated space — deleted files, filesystem fragments, and partial data — without needing a filesystem structure at all.

Timeline reconstruction: From inode metadata alone (creation time, modification time, access time), analysts reconstruct a detailed timeline of what was done on the system, when, and in what order. This can be used to establish patterns of activity, prove ownership, or contradict statements you may make.

Phase 3: Passphrase Recovery Attack

If the drive is identified as LUKS-encrypted, the forensics team extracts the header and runs an offline passphrase attack against a copy. The crucial detail is that they never need the original drive for this — the header backup is sufficient.

```
<code># What the forensics team does: cryptsetup luksHeaderBackup /dev/sdX2 --header-ba
```

The mathematics of brute-force vs. Argon2id:

Configuration	Guesses/sec (single RTX 4090)	Time for 6-word diceware
PBKDF2-SHA256 (low iteration)	~1,000,000	~150,000 years
Argon2id, 64 MiB memory	~300	Hundreds of millions of years
Argon2id, 4 GiB memory (this guide)	**~0.5**	**Effectively infinite**

The 4 GiB Argon2id parameter means each guess requires 4 GiB of RAM. A GPU with 24 GB of VRAM can attempt only 6 guesses in parallel. This makes GPU clusters — the standard tool for PBKDF2 and bcrypt cracking — nearly irrelevant.

Phase 4: Coercion

Against a mathematically sound encrypted drive, coercion is the only practical avenue. This takes two forms:

Legal compulsion: Courts in many jurisdictions can order production of the decryption key. In authoritarian states, this order may be issued with no meaningful oversight or appeal mechanism. Refusal results in contempt charges, which can mean indefinite imprisonment until you comply.

Physical coercion ("rubber-hose cryptanalysis"): Detention, threats, or physical harm used to extract the passphrase. This is the operational reality in authoritarian contexts where rule of law is functionally absent. No amount of encryption prevents a determined human adversary from applying physical pressure — but plausible deniability (Chapter 15) allows you to provide a passphrase without surrendering your real data.

What Forensics Cannot Do Against a Properly Configured System

Capability	Possible?	Reason
Break AES-256 encryption	No	No known attack; even quantum resistance is adequate at 256 bits
Brute-force a 6-word diceware passphrase	No	Physically impossible timescales with Argon2id 4 GiB
Recover data from a LUKS drive without the key	No	Ciphertext is indistinguishable from random noise
Prove a VeraCrypt hidden volume exists	No	Unallocated outer-volume space has a valid cryptographic explanation as random noise
Detect encryption when the header is detached	No	The partition contains only raw AES-XTS output — no signature
Recover files deleted from an encrypted SSD	No	All physical NAND cells contain only ciphertext (see Chapter 12)
Extract the LUKS key from a powered-off machine	No	Key exists only in RAM while the drive is actively mounted

Part II — Hardware

The right hardware choices establish the physical foundation your encryption runs on. Several common hardware decisions undermine encryption before it even starts.

Chapter 3. Choosing Your Drive and Enclosure

Why Use a SATA-to-USB Drive?

A portable SATA SSD in a USB enclosure gives you capabilities that a fixed internal drive cannot:

- **Physical portability:** Your entire operating system and encrypted data fits in a shirt pocket. The same drive boots on any UEFI-capable laptop. You are not tied to a specific machine.
- **Clean host machine:** The host laptop's internal drive contains nothing. At a checkpoint, an examiner who searches the laptop's internal storage finds no incriminating data. Your sensitive work exists only on the USB drive, which you may or may not have with you.
- **Plausible appearance:** A USB enclosure looks like a mundane external backup drive. It attracts no more scrutiny than carrying a USB stick.
- **Destruction advantage:** A 2.5" SSD PCB is approximately 7 × 10 cm. It can be broken with bare hands in under five seconds if necessary. An internal laptop drive requires tools to access and additional time to destroy.
- **No installation trace:** Nothing is installed on the host machine. The laptop's boot history, event logs, and filesystem contain no reference to your encrypted drive or its activities.

Why Not Use Hardware Encryption (TCG Opal / eDrive)?

Many SSDs advertise built-in hardware encryption — Samsung EVO drives label it "Encrypted Drive," and the industry standard is called TCG Opal or eDrive. **Do not rely on this for your threat model.**

Academic research by Carlo Meijer and Bernard van Gastel (2019, presented at IEEE Symposium on Security and Privacy) reverse-engineered the firmware of widely sold Samsung and Crucial SSDs and found catastrophic implementation flaws:

- The Samsung 840 EVO accepted *any password* — including an empty string — and still "decrypted" the drive. The password was used to look up a key in a table; when that lookup failed, the firmware fell through to unlocking with the master key regardless.
- The Samsung 850 EVO had its encryption key derived from a value that was not properly secret — breaking the drive's encryption did not require the user's password at all.

- Crucial MX100 and MX200 drives had similar vulnerabilities where the encryption key was insufficiently protected from the master password.
- BitLocker on Windows, when it detects Opal-capable hardware, silently defers to hardware encryption — meaning millions of Windows users believed they had BitLocker protection while actually having none.

The firmware implementing this encryption is closed-source and cannot be audited. No independent party can verify it is correct. LUKS2 running on the CPU using AES-NI hardware acceleration is fully open-source, independently audited, and performs equivalently. There is no reason to trust hardware encryption when better software alternatives exist.

Verify AES-NI hardware acceleration is available before installation:

```
<code>grep -m1 aes /proc/cpuinfo # The word 'aes' must appear in the flags line # If ab
```

Recommended SSDs

Drive	Notes
Samsung 870 EVO (2.5” SATA)	Highly reliable, excellent Linux compatibility, widely available globally. Disable TCG Opal in Samsung Magician before use.
WD Blue SA510 (2.5” SATA)	Clean public firmware history, good price-to-performance. Straightforward with LUKS.
Crucial MX500 (2.5” SATA)	Solid performer; explicitly disable hardware encryption in Crucial Storage Executive. Do not use Opal mode.
Kingston A400 (2.5” SATA)	Budget option. No known firmware security history. Suitable for this use case.

Recommended Enclosures

Key requirements: UASP support (required for performance), TRIM passthrough (required for drive health and counter-forensics), no firmware interception of commands.

Enclosure	Notes
Sabrent EC-UASP	USB 3.2 Gen 1, UASP, widely available, no known quirks with Linux TRIM passthrough.
Inateck FE2011	Compact, toolless entry, solid build quality.
OWC Express USB-C	Higher build quality; good Linux compatibility record.
ORICO 2139C3 (transparent)	The transparent shell lets you visually inspect the PCB for unexpected hardware without opening the enclosure — useful for anti-tamper checks.

Verify TRIM passthrough is functional after connecting the drive:

```
<code>lsblk --discard /dev/sdX # DISC-GRAN and DISC-MAX must both be non-zero # A zero
```

Chapter 4. Acquiring Hardware Safely

Why Acquisition Matters

Supply chain interdiction — intercepting a specific delivery to insert hardware implants before it reaches the target — is a documented state capability. The NSA ANT catalog (published by Der Spiegel in 2013) showed purpose-built hardware implants for Cisco routers, Dell servers, and other equipment, inserted during delivery to specific targets. This is primarily a risk for extremely high-value targets, but the precautions cost nothing and take minutes.

Safe Purchase Practices

- **Buy in person at a physical retail store.** A retail purchase is indistinguishable from thousands of other daily purchases at that store. A mail-order delivery to your address can be specifically targeted for interception.
- **Pay with cash.** Card purchases create a permanent, legally accessible record linking your identity to a specific device's serial number. Cash purchases, especially from a large retailer, are effectively untraceable.
- **Do not use loyalty cards, apps, or receipts.** All of these tie the purchase to your identity.
- **Buy from a store outside your immediate neighborhood.** Your regular local shop may know your face. A shop in a different district does not.
- **Separate purchases across time and location.** Buying a specific SSD model, USB enclosure, and laptop together in one transaction creates a distinctive purchase fingerprint. Separate these by days or weeks if possible.
- **Never ship hardware to your home address.** If online purchase is unavoidable, use a P.O. box, a mail-forwarding service, or arrange collection from a trusted intermediary not linked to you.

Physical Inspection on Receipt

Before using any hardware, inspect it against reference teardown images (available on iFixit and manufacturer sites):

```
<code># Verify firmware version is in the expected public release range smartctl -a /de
```

Physical indicators of tampering:

- Misaligned, scratched, or stripped screws on the enclosure
- Adhesive residue or marks around case seams where it was opened
- Components on the PCB that do not match reference teardown photos for that model
- The enclosure does not close flush, or clips that were clearly pried open
- Any additional chips or wires not present in the reference photos

Establish a baseline: Photograph the PCB under good light immediately after purchase, before first use. Store these reference photos on a different device. Compare against them before every subsequent use if the device was out of your control.

Part III — Building the Encrypted System

The installation process itself must be conducted securely. A compromised installation environment can capture your passphrase before encryption ever protects anything.

Chapter 5. Preparing a Trusted Installation Environment

Why the Installation Machine Matters

The machine you use to install Arch Linux becomes a trusted participant in your security chain during the installation process. If a keylogger is active on that machine, every passphrase you type — including your LUKS passphrase — is captured in plaintext before LUKS ever has a chance to protect it. The entire encryption scheme fails at this step if the installation environment is compromised.

For high-threat environments, use one of the following:

- **Tails OS (strongly recommended):** Boot from a Tails USB stick. Tails is an amnesiac operating system that leaves no trace on the host machine and routes all traffic through Tor. Tails is designed specifically for this use case by the Tor Project. Any keylogger on the host machine that operates at the OS level will not run under Tails.
- **A machine you physically control and trust:** A machine that has never been in an adversary's custody, has never been connected to an untrusted network, and whose firmware you have not had reason to distrust.
- **Never:** A library computer, internet café machine, hotel business center, employer-provided computer, or any shared or borrowed machine. These are fundamentally untrustworthy environments.

Verifying the Arch Linux Installation Image

A compromised ISO containing a backdoored installer would be invisible during installation but would compromise every passphrase you enter. Always verify before use:

```
<code># Step 1: Download ISO and signature from archlinux.org wget https://archlinux.or
including "Bad signature" - means the ISO is untrustworthy</code>
```

The fingerprint verification step (Step 3) is not optional. A compromised keyserver could serve a malicious key that would produce a "Good signature" from the wrong key. The fingerprint must be verified against the developer list on archlinux.org from a trusted connection — ideally from a different network than the one used to download the ISO.

For maximum assurance, compare the ISO SHA-256 hash from three different sources: the Arch Linux website, the mirror you downloaded from, and a trusted third-party mirror. All three must match.

Writing the Installer USB

```
<code># Identify the correct device - be absolutely certain lsblk # Confirm /dev/sdX is
```

Pre-Installation UEFI Configuration

Before booting the installer, secure the UEFI firmware settings on the host machine:

1. **Set a UEFI administrator password.** This prevents an attacker with brief physical access from changing boot settings without your knowledge. Choose a strong password that is different from all others.
2. **Disable legacy BIOS/CSM mode.** Boot only in UEFI mode. Legacy mode bypasses Secure Boot and creates compatibility issues with modern security features.
3. **Restrict boot device order.** Set the boot order to boot exclusively from your SATA-to-USB drive. Disable network (PXE) boot, internal drive boot, and other USB boot during normal operation.
4. **Disable Wake-on-LAN.** This prevents network-triggered remote wake events.
5. **Enable Secure Boot.** You will configure it with your own keys in Chapter 9. For initial installation, you may need to temporarily disable it — re-enable it and configure custom keys immediately after installation.

Chapter 6. Disk Sanitization

Why Overwrite Before Encrypting

Before creating the encrypted filesystem, overwrite the entire drive with cryptographically random data. This achieves two distinct security goals:

1. Eliminates prior data. Any files that existed on the drive in a previous life are overwritten. File carving of the "unallocated space" before the LUKS partition reveals nothing.

2. Establishes plausible deniability for the encrypted region. After this step, the drive contains a uniform field of high-entropy data. The LUKS partition (once created and filled with encrypted content) looks identical to the pre-wipe random data. A forensic examiner cannot determine where the wipe ends and the LUKS ciphertext begins — unless the LUKS header is present (this is why detached headers, Chapter 16, are so powerful).

The Correct Method: Cryptographic Overwrite

On SSDs, writing through a cryptographic layer produces output that is computationally indistinguishable from true random noise. One pass is sufficient — the AES-XTS output is already maximally random. Multiple passes provide no additional security on SSDs and cause unnecessary wear.

```
<code># Open a temporary plain dm-crypt device with a random key # The key is generated
the random key is discarded and forgotten cryptsetup close sanitize_container</code>
```

Duration: approximately 30–90 minutes for a 500 GB SSD over USB 3.0 (depending on drive speed and USB generation).

Verify the result produced high-entropy output:

```
<code># Sample the first 100 MiB and measure entropy dd if=/dev/sdX bs=1M count=100 2>/
```

ATA Secure Erase (Supplementary)

ATA Secure Erase is a firmware-level command that instructs the drive controller to erase all NAND cells, including the over-provisioned spare area that is invisible to normal writes. This is a useful supplement to the cryptographic overwrite for SSDs where the over-provisioned area may contain artifacts from previous use.

```
<code># Check if the drive security is frozen (common default state) hdparm -I /dev/sdX
45 seconds for most SSDs; potentially hours for HDDs</code>
```

ATA Secure Erase reliability is entirely dependent on the drive firmware. Some drives report success without actually erasing all cells. Use this as a supplement to the cryptographic wipe above — never as a replacement.

Chapter 7. LUKS2 Full-Disk Encryption

This is the most consequential chapter. LUKS2 is the encryption layer that makes your data unreadable to anyone who seizes the drive without your passphrase. Every parameter here has a specific security rationale — read each one.

Partitioning the Drive

Boot from your verified Arch Linux installer USB and identify the target drive:

```
<code>lsblk # Find your SATA-to-USB drive - e.g., /dev/sdb # Triple-check: this must NO
```

Create the partition table with exactly two partitions:

```
<code>gdisk /dev/sdX # Inside gdisk: o → New GPT partition table (answer y to confirm)
```

Partition	Size	Type	Purpose
/dev/sdX1	1 GiB	EF00 — EFI System	Unencrypted; holds bootloader only. Keep it minimal.
/dev/sdX2	Remainder	8309 — Linux LUKS	Encrypted container holding the entire OS and data.

Choosing a Passphrase

This is the single most important decision in this entire guide. The passphrase is what stands between your data and everyone who seizes the drive. It must be simultaneously memorizable under stress and immune to both targeted and systematic attack. There is exactly one method that satisfies both constraints reliably: diceware.

What is diceware? Diceware generates passphrases by rolling physical dice and looking up the results in a standardized wordlist. Because physical dice are used, no software component participates in the generation — there is no random number generator to compromise, no keyboard logger to capture intermediate state, and no possible bias.

```
<code># Download the EFF Large Wordlist (7776 words; one word per 5 dice rolls) wget ht
generate your own with physical dice)</code>
```

Entropy by passphrase length:

Words	Entropy	Guesses needed (average)	Time at 0.5 guess/sec (Argon2id 4 GiB)
4 words	~51 bits	~1.1×10^{15}	~72 million years
5 words	~64 bits	~9.2×10^{18}	~580 billion years
6 words	**~77 bits**	**~7.6×10^{22}**	**Effectively infinite**
7 words	~90 bits	~6.2×10^{26}	Far exceeds age of universe

Six words is the minimum for serious threat models. Use seven or more if you expect sustained, well-resourced attacks over years.

Do not:

- Use names — your own, family members, pets, celebrities, or places
- Use important dates in any format
- Use song lyrics, movie quotes, or famous phrases — targeted wordlists include these
- Substitute numbers for letters (`p@ssw0rd`-style) — hashcat rule sets cover all common substitutions
- Modify the diceware output — even adding punctuation you chose defeats the randomness guarantee
- Generate it with software instead of physical dice

Creating the LUKS2 Volume

```
<code>cryptsetup luksFormat \ --type luks2 \ --cipher aes-xts-plain64 \ --key-size 512
```

Every parameter explained — do not skip this section:

`--cipher aes-xts-plain64`

AES (Advanced Encryption Standard) in XTS (XEX-based tweaked-codebook mode with ciphertext stealing) mode. AES-256 is the global standard for disk encryption, mandated by US NIST and adopted worldwide. XTS was specifically designed for disk encryption (IEEE P1619 standard) to address the problem that the same plaintext at the same position must not produce the same ciphertext — a property called "sector independence." XTS achieves this by incorporating the physical sector address into each encryption operation. No practical attack against AES-256 exists. The best published academic attacks against AES reduce security by only a few bits and require conditions (related keys, chosen plaintexts) that do not apply to disk encryption.

`--key-size 512`

In XTS mode, the total key is split into two halves: 256 bits for the data encryption key and 256 bits for the XTS tweak key. Specifying 512 bits therefore gives you full 256-bit AES — the maximum and most secure configuration.

`--pbkdf argon2id`

Argon2id won the Password Hashing Competition in 2015 after peer review by hundreds of cryptographers. It is a memory-hard function — correctly computing it requires holding a configurable amount of RAM (not just time). This is decisive against GPU-based and ASIC-based attacks. A GPU cluster that can test billions of SHA-256 hashes per second is constrained by available RAM when testing Argon2id — each attempt requires holding the full memory parameter simultaneously. Argon2id specifically combines two Argon2 variants to resist both side-channel attacks (from the "Argon2i" component) and time-memory tradeoff attacks (from the "Argon2d" component).

`--pbkdf-memory 4194304`

4,194,304 KiB = 4 GiB of RAM required per password attempt. This is the primary cost that makes GPU cracking impractical. A GPU with 24 GB VRAM can test only 6 passwords in parallel. Compare this to SHA-256, where the same GPU can test billions in parallel. The 4 GiB parameter also means this unlock will be slow on your own machine (10+ seconds) — this is intentional and appropriate.

`--pbkdf-force-iterations 6` and `--iter-time 10000`

Six passes over the full 4 GiB memory region, plus 10 seconds of time-based computation regardless of hardware speed. Together, a single unlock attempt takes at minimum 10–15 seconds on modern hardware. This further compounds the Argon2id protection.

`--integrity hmac-sha256`

Enables dm-integrity: every 4096-byte sector of encrypted data receives an HMAC-SHA256 authentication tag. If any byte of ciphertext is modified on disk — whether by accident, by a forensic tool probing the cipher, or by a sophisticated active attack — the integrity check fails and the read returns an I/O error rather than silently returning potentially useful modified plaintext. This closes a class of "chosen-ciphertext attacks" where an adversary who can modify the encrypted data and observe the system's behavior can learn information about the plaintext.

`--sector-size 4096`

Modern SSDs use 4096-byte physical sectors internally. Aligning the LUKS sector size to 4096 bytes ensures each encrypted sector maps directly to one physical sector, maximizing performance and eliminating read-modify-write cycles.

Verifying the LUKS Configuration

`<code>cryptsetup luksDump /dev/sdX2</code>`

Confirm the output shows all of the following:

`<code>Version: 2 Cipher: aes-xts-plain64 Cipher key: 512 bits PBKDF: argon2id Memory: 4`

If any parameter is wrong, re-run `luksFormat` with the correct parameters before proceeding. Do not continue with an incorrectly configured volume.

Opening the LUKS Container

`<code>cryptsetup open /dev/sdX2 cryptroot # Prompts for your passphrase # On success: t`

Chapter 8. Installing Arch Linux

Create Filesystems

```
<code># EFI partition - must be FAT32 mkfs.fat -F32 -n EFI /dev/sdX1 # Root filesystem
```

Mount and Install the Base System

```
<code># Mount encrypted root mount /dev/mapper/cryptroot /mnt # Create and mount EFI pa
linux-hardened is the security-patched kernel pacstrap -K /mnt \ base \ linux \ linux-h
```

The `linux-hardened` package differs from the standard `linux` kernel in security-relevant ways:

- **SLAB_FREELIST_RANDOM:** Randomizes the order of free objects in the kernel memory allocator. Makes heap spray attacks (a common exploit technique) far less reliable — the attacker cannot predict where allocated objects will land.
- **SLAB_FREELIST_HARDENED:** Adds integrity metadata to the allocator's freelist. Detecting freelist corruption allows the kernel to kill a process that is attempting a heap overflow instead of continuing with corrupted state.
- **HARDENED_USERCOPY:** Bounds-checks all copies between kernel memory and userspace. A kernel bug that copies too many bytes to userspace is caught and kills the process rather than leaking kernel memory contents.
- **PAGE_TABLE_ISOLATION (KPTI):** Isolates the kernel's page tables from userspace processes. Mitigates the Meltdown speculative execution vulnerability, which allowed userspace processes to read arbitrary kernel memory on affected Intel CPUs.
- **Restricted /proc access:** Non-root users cannot read other users' process information from /proc. This prevents information gathering by unprivileged processes.

Performance cost of linux-hardened: typically under 5% for common workloads. This is an entirely acceptable trade-off for the security gains.

System Configuration

```
<code>genfstab -U /mnt >> /mnt/etc/fstab arch-chroot /mnt # Timezone ln -sf /usr/share/
use something completely generic; avoid anything identifying echo "localhost" > /etc/ho
```

Configure the initramfs for Encryption

The initramfs is the minimal Linux environment that runs before the real root filesystem is mounted. It must contain the LUKS unlocking logic — the `encrypt` hook — or the system will not know how to decrypt the drive at boot.

```
<code>vim /etc/mkinitcpio.conf # Find the HOOKS line and set it to exactly: HOOKS=(base
```

Chapter 9. Bootloader Hardening

Install GRUB

<code>grub-install \ --target=x86_64-efi \ --efi-directory=/boot \ --bootloader-id=GRUB

Configure GRUB for Encrypted Boot

<code># Get the UUID of your encrypted partition blkid /dev/sdX2 # Copy the UUID value

Set GRUB_CMDLINE_LINUX to the following (replace <UUID> with your actual UUID):

<code>GRUB_CMDLINE_LINUX="cryptdevice=UUID=<UUID>:cryptroot root=/dev/mapper/cryptroot

Parameter	What It Prevents
`init_on_alloc=1`	Zeroes memory before allocating it. Prevents "heap spray" attacks that read old memory contents from newly allocated buffers.
`init_on_free=1`	Zeroes memory after freeing it. Prevents use-after-free vulnerabilities from leaking data.
`vsyscall=none`	Disables the legacy vsyscall interface at a fixed kernel address. This address is a common known-location exploit trampoline — attackers jump here during code reuse attacks.
`debugfs=off`	Disables the debugfs virtual filesystem. Without this, debugfs exposes internal kernel data structures that an attacker with any code execution can use to escalate privileges or read kernel memory.
`lockdown=confidentiality`	Activates kernel lockdown mode at the most restrictive level. Prevents userspace from reading or writing kernel memory via /dev/mem, kexec, hibernation, or other privileged interfaces. Even root cannot extract kernel secrets.
`module.sig_enforce=1`	Rejects any kernel module not signed with the kernel's build-time signing key. Prevents an attacker with root access from loading a malicious kernel module.
`iommu=force`	Forces strict IOMMU enforcement. Without IOMMU, any PCIe device (or a device pretending to be one via Thunderbolt DMA) can read and write all physical RAM, including the LUKS decryption key.
`nohibernate`	Disables hibernation. A hibernate image on disk contains an exact snapshot of RAM — including the LUKS decryption key — written to disk unencrypted. This would completely defeat drive encryption.
`mitigations=auto`	Enables all hardware vulnerability mitigations (Spectre, Meltdown, MDS, etc.) appropriate for the detected CPU. These prevent speculative execution attacks that could leak the LUKS key from kernel memory.

`<code>grub-mkconfig -o /boot/grub/grub.cfg</code>`

GRUB Password Protection

Without a GRUB password, anyone who reaches the GRUB menu can press `e` to edit the boot entry, add `init=/bin/bash` to the kernel command line, and boot directly to a root shell. The LUKS encryption would still protect the data on disk, but the attacker could modify the bootloader for a future evil maid attack (Chapter 18) without leaving obvious traces.

```
<code># Generate a PBKDF2-hashed GRUB password grub-mkpasswd-pbkdf2 # Enter and confirm
```

Unified Kernel Images

A standard GRUB configuration has a meaningful security gap: `grub.cfg` is an unsigned, unencrypted text file on the EFI partition. An attacker with a few minutes of physical access can edit it — changing kernel parameters, adding `init=/bin/bash`, or pointing GRUB at a different kernel — without triggering Secure Boot, because GRUB itself is signed but the config file it reads is not.

A Unified Kernel Image (UKI) solves this by embedding the kernel command line, the initramfs, and the kernel itself into a single EFI binary. The entire binary is Secure Boot signed. Any modification — including changing a single character of the kernel command line — invalidates the signature and causes the UEFI firmware to refuse to boot it.

```
<code>pacman -S ukify # Write the kernel command line to a file for embedding cat >/etc
```

Secure Boot with Custom Keys

Default Secure Boot uses Microsoft's signing keys. This means Microsoft can sign software that boots on your machine — and by extension, so can anyone who compromises Microsoft's infrastructure or obtains a code-signing certificate under their CA hierarchy. For your threat model, you want Secure Boot controlled entirely by you: only software you signed can boot.

```
<code>pacman -S sbctl # Check the current Secure Boot enrollment state sbctl status # I
```

Finishing the Installation

```
<code>exit # exit the chroot environment umount -R /mnt # unmount all filesystems clean
```

On reboot, the machine should boot from the SATA-to-USB drive, display the GRUB menu (password-protected), prompt for your LUKS passphrase, and complete the boot. If it does not work, boot back from the installer, open the LUKS container, mount the filesystems, chroot in, and troubleshoot the GRUB configuration and initramfs hooks.

Chapter 10. Post-Installation System Hardening

Kernel Security Parameters (sysctl)

```
<code>cat >/etc/sysctl.d/99-security.conf << 'EOF' # Hide kernel symbol addresses and p
an ASLR bypass kernel.kptr_restrict = 2 kernel.dmesg_restrict = 1 kernel.printk = 3 3 3
IP spoofing protection via reverse path filtering net.ipv4.conf.all.rp_filter = 1 net.i
reject ICMP redirect messages (MITM route manipulation) net.ipv4.conf.all.accept_redire
TCP SYN flood protection net.ipv4.tcp_syncookies = 1 # Network - disable IP source rout
RFC 1337 TIME_WAIT assassination protection net.ipv4.tcp_rfc1337 = 1 # Network -
disable TCP timestamps (leaks system uptime, used in fingerprinting) net.ipv4.tcp_times
```

Firewall (nftables)

```
<code>pacman -S nftables cat >/etc/nftables.conf << 'EOF' #!/usr/bin/nft -f flush rules
no inbound ports are open } chain forward { type filter hook forward priority 0; policy
```

Disable Unnecessary Services

```
<code># Bluetooth - disable entirely if not used systemctl disable --now bluetooth syst
not needed, exposes local network presence systemctl disable --now avahi-daemon systemc
disable unless actively printing systemctl disable --now cups systemctl mask cups # Ena
```

Encrypted Swap or No Swap

Swap space writes RAM contents to disk. Without encryption, the LUKS master key — which lives in RAM while the drive is mounted — could theoretically be paged to swap and then recovered from disk after the machine is powered off. Never use unencrypted swap on this system.

```
<code># Option 1: Encrypt swap with an ephemeral random key (different on every boot) #
```

Disable Shell History

```
<code># Shell history files are forensic goldmines - they record every command ever typ
```

Automatic Screen Lock

```
<code>pacman -S swayidle swaylock # for Wayland/Sway desktop # Alternative for X11: xau
```

Sixty seconds is short but correct for a high-threat environment. A screen left unlocked for even 90 seconds during an unexpected entry is enough for an attacker to dump memory or install a persistence mechanism. The screen locking before sleep is critical — the system must never sleep with an unlocked screen.

Part IV — Counter-Forensics

Counter-forensics is not about hiding crimes. It is about ensuring that a hostile forensic analysis of your seized, decrypted drive reveals only what you chose to leave there.

=

Chapter 11. What Forensic Tools Can and Cannot Do

The Tools State Forensics Teams Actually Use

Cellebrite UFED 4PC — Primarily a mobile forensics tool adapted for PC use. Can perform physical and logical acquisition. Recognized in courts worldwide. The 4PC version specifically targets Windows, macOS, and Linux systems on laptop/desktop hardware. Does not decrypt LUKS.

Magnet AXIOM — Comprehensive platform covering cloud, mobile, and computer forensics. Exceptional at reconstructing user activity from application artifacts: browser history (including incognito sessions via RAM carving on live systems), email databases, messaging applications, recently opened documents, and cloud sync artifacts. Against a decrypted drive, AXIOM builds a detailed behavioral timeline automatically. Cannot decrypt LUKS.

EnCase (OpenText) — The industry standard for corporate and law enforcement forensics globally. Powerful timeline analysis, file carving, and hash-set matching (identifies known files by their SHA-256 hash without opening them). Courts accept EnCase reports as standard evidence. Cannot decrypt LUKS.

FTK (Forensic Toolkit / Exterro) — Similar to EnCase, preferred in some agencies for large-scale multi-drive investigations. Has integrated GPU-accelerated passphrase recovery modules that interface directly with HashCat for LUKS attacks.

Passware Kit Forensic / Elcomsoft Distributed Password Recovery — Dedicated GPU-accelerated passphrase recovery tools. Can attack LUKS passphrases at high speed for simple KDFs. With Argon2id at 4 GiB, attack rates drop to less than 1 attempt per 2 seconds per GPU. Against a 6-word diceware passphrase, this is not an attack — it is theater.

What a Forensic Examiner Does With Your Drive, Step by Step

Phase 1 — Automated identification (minutes):

Tools scan every sector looking for known magic bytes. LUKS2 is identified by `4C 55 4B 53 BA BE` at the start of the partition. Once detected, the full LUKS2 header is parsed automatically:

```
<code># What the forensics tooling extracts from your LUKS header automatically: crypts
```

Phase 2 — Passphrase attack (ongoing, possibly indefinite):

```
<code># They extract the header for offline cracking: cryptsetup luksHeaderBackup /dev/
```

Phase 3 — EFI partition forensics:

The /boot partition is unencrypted and will be fully examined. A careful analyst will look at every file in the EFI partition, including timestamps of file creation and modification, which reveals when the system was last updated. Limit the EFI partition contents to the minimum required: bootloader binary, kernel image, initramfs. Never store any data in /boot beyond what the boot process requires.

Phase 4 — Reporting:

If decryption fails, the forensic report states: "The device contains a LUKS2-encrypted partition using Argon2id key derivation. Passphrase recovery was not successful. The contents of the encrypted partition could not be examined." The drive is returned or held as exhibit.

After Decryption: What a Thorough Examiner Finds

If an examiner does obtain the passphrase (via coercion or a weak passphrase), they will conduct a thorough examination of the decrypted volume. They use automated pipelines that analyze:

- **Application artifacts:** Browser databases (SQLite files that contain full history even in "private" mode), email MBOX/Maildir files, Signal/Telegram desktop databases, document editor recent-files lists
- **System logs:** Systemd journal, auth.log, syslog, kern.log — unless you configured volatile storage
- **Shell history:** .bash_history, .zsh_history, .python_history — unless you disabled them
- **Thumbnail caches:** Desktop environments automatically generate image thumbnails and cache them in ~/.thumbnails or ~/.cache/thumbnails — often containing images that were "deleted" from the disk
- **Recycle bin / Trash:** Files "deleted" using the GUI file manager move to ~/.local/share/Trash, not to immediate deletion
- **Printer spools, recently used documents, window manager session files**
- **Swap contents:** If swap was unencrypted or uses a predictable key

The chapters that follow address each of these categories.

Chapter 12. SSD Forensics and Why Encryption Is Your Only Real Defense

The Fundamental Problem with SSDs

SSDs do not work like hard drives. When you write data to a hard drive, you write directly to the physical location. On an SSD, the firmware (called the Flash Translation Layer, or FTL) manages an abstraction layer between logical block addresses and physical NAND cell locations. This abstraction is what causes the security problem.

Wear leveling: NAND flash cells have a limited number of program/erase cycles (typically 3,000–100,000 P/E cycles depending on flash type). The FTL distributes writes across all physical cells to equalize wear. When you "overwrite" a logical block, the FTL may:

1. Write the new data to a fresh physical cell in a different location
2. Mark the original physical cell as "stale" — it contains old data but is not immediately erased
3. The original cell joins the "garbage collection queue" — it will be erased when the FTL decides to run GC, which may be hours, days, or never before forensic imaging

Over-provisioning: Consumer SSDs typically have 7–28% more physical NAND than their rated capacity. A "500 GB" SSD physically contains 540–640 GB of flash. The extra cells are used as spare area for wear leveling and bad block replacement. This spare area is completely invisible to any operating system command — `dd`, `shred`, and `wipe` cannot reach it.

The consequence: Standard Linux tools designed for HDD data destruction (`shred`, `wipe`, `srm`, `secure-delete`) **do not reliably delete data on SSDs**. They overwrite the logical address, but the physical NAND cell that previously held that data may remain intact and readable.

What Specialized Forensic Tools Can Recover From SSDs

Three approaches exist for SSD forensics that bypass the standard SATA/USB interface:

- **PC-3000 Flash (ACE Lab):** The industry standard for NAND chip-off and in-system forensics. Communicates with the SSD controller using manufacturer-specific diagnostic commands, bypassing the normal FTL entirely. Can access raw NAND data including over-provisioned area.
- **Chip-off:** Physically remove the NAND chips from the PCB, connect them to a specialized reader, and read the raw NAND pages directly. No encryption = full data recovery. With encryption = only ciphertext.

- **JTAG / ISP:** Connect directly to debug ports on the SSD controller chip to extract firmware, configuration, or data through the chip's debug interface.

The solution to all of these techniques is identical: encrypt the entire drive before writing any data to it. If every physical NAND cell — including over-provisioned spare area — contains only AES-256-XTS ciphertext from the moment the drive was first used, chip-off forensics recovers only random-looking noise. This is why Chapter 6 (disk sanitization) and Chapter 7 (LUKS creation) must be done in that order, before the OS is installed.

TRIM on an Encrypted Drive: Trade-offs

TRIM is a command the OS sends to the SSD saying "these blocks are no longer in use; you may erase them." It helps maintain SSD performance over time by allowing the FTL to pre-erase blocks before they need to be rewritten.

Security trade-off: TRIM leaks information to a physical attacker. By analyzing which logical blocks the SSD reports as discarded, a forensic examiner who accesses the NAND directly can determine the approximate size and layout of your filesystem's used space — even without decrypting anything. They cannot read the data, but they can infer patterns.

For most threat models, TRIM provides a net benefit (drive health + some anti-forensic clearing) and should be enabled. For maximum deniability with hidden VeraCrypt volumes (Chapter 15), disable TRIM — a hidden volume must make the outer volume's unallocated space indistinguishable from random data, which TRIM can disrupt.

```
<code># Enable TRIM passthrough in /etc/crypttab (most threat models): cryptroot UUID=
TRIM is off by default</code>
```

Making Specific Files Truly Unrecoverable

```
<code># Option 1 (best): Never write to disk - use RAM-backed tmpfs for sensitive work
data exists only in RAM umount /mnt/sensitive # RAM cleared immediately on unmount # Op
```

Chapter 13. Minimizing Forensic Artifacts During Use

An adversary who decrypts your drive will conduct a thorough forensic examination of its contents. This chapter systematically addresses each category of forensic artifact that would otherwise accumulate during normal use.

System Logs in RAM Only

By default, systemd-journald writes logs to `/var/log/journal/` on disk. These journal files contain a timestamped record of every significant system event: services started, USB devices connected, network connections established, kernel messages, and application errors. This is an intelligence windfall for a forensic examiner who decrypts the drive.

```
<code>cat >/etc/systemd/journald.conf << 'EOF' [Journal] Storage=volatile RuntimeMaxUse
```

`Storage=volatile` means all logs exist only in `/run/log/journal/` (which is tmpfs — RAM-backed). They disappear completely when the machine shuts down. No historical log files ever touch the drive.

Application Cache and Working Data in RAM

```
<code># Add to /etc/fstab - mount these directories as RAM-backed tmpfs: tmpfs /home/us
```

Document and File Metadata Scrubbing

Every file carries metadata that can identify you, your tools, your location, and your working patterns. This metadata persists even after the file is shared or removed from the drive — the copies others have will still contain it.

- **PDF, DOCX, ODT:** Author name, organization name, revision history showing previous authors, software name and version (including exact build number), creation timestamp, last modification timestamp, total editing time, printer name, template origin path
- **JPEG, PNG, HEIC:** GPS coordinates (latitude, longitude, altitude), camera make, model, and serial number, date and time, software version, lens information, thumbnail of original photo (even if the main image was cropped), scene detection tags
- **Audio/Video:** Recording device, GPS, creation software, encoding settings

```
<code>pacman -S mat2 perl-image-exiftool # Scrub ALL metadata from a file (mat2 is the
```

Always scrub metadata from any file before it leaves the encrypted drive — before emailing, sharing, or publishing. Also scrub any file received from outside before saving it to the encrypted drive, as embedded metadata can tie the file back to its source and to you.

Disabling Access Timestamps

Every time a file is read on a default ext4 filesystem, the kernel updates the file's `atime` (access time). Over time, these timestamps build a detailed record of your reading patterns — which files you accessed, in what order, and when. A forensic analyst examining a decrypted drive can reconstruct a day-by-day activity log purely from atime entries.

```
<code># In /etc/fstab, add noatime and nodiratime to the root mount: /dev/mapper/cryptr
```

USB and Device Connection History

```
<code># With Storage=volatile for journald, USB connection logs disappear on reboot # F
allow only your known devices systemctl enable --now usbguard # Any USB device not in t
```

Cleaning Application-Specific Artifacts

```
<code># KeePassXC - clear clipboard after password copy: # Settings → Security → Clear
disable persistent undo and swap files: cat >>~/.vimrc << 'EOF' set noswapfile set noun
always delete files permanently, never trash them: # In file managers: configure perman
```

Chapter 14. Anti-Forensic Filesystem Configuration

Ext4 Journal Forensics

The ext4 journal is a write-ahead log designed for crash recovery. Before committing filesystem changes, ext4 records them in the journal. This means recently deleted file contents may remain readable in the journal for some time after deletion, even after the filesystem marks the space as free.

A forensic analyst who examines a decrypted ext4 volume with standard tools (`debugfs`, Autopsy, Sleuth Kit) can sometimes recover recently-written data directly from the journal without needing to carve unallocated space.

Mitigation options:

```
<code># Option A: data=writeback mode (recommended balance of safety and anti-forensics
crash recovery is less critical when you control shutdowns</code>
```

Btrfs as an Alternative

Btrfs offers capabilities useful for anti-forensics that ext4 does not provide natively:

Subvolumes and snapshots: You can snapshot a known-clean system state. After any sensitive work session, roll back to the clean snapshot — all data written during the session is completely gone, including artifacts, logs, and any traces of which files were accessed or created.

```
<code># Set up a clean snapshot before first sensitive use mount /dev/mapper/cryptroot
do this from the Arch installer environment) # Mount the LUKS device btrfs subvolume de
```

Transparent compression: Btrfs can compress data transparently using ZSTD. This affects ciphertext distribution slightly but primarily helps reduce the amount of physical NAND written, improving drive longevity and producing more uniform storage patterns.

Restricting /proc Visibility

```
<code># Mount /proc with hidepid=2 - users cannot see other users' processes # In /etc/
```

Part V — Drive Seizure Resistance

These chapters address scenarios where the drive is seized and an attempt is made to access it — including the scenario where you are compelled to provide the passphrase.

Chapter 15. Plausible Deniability

This is the most strategically significant chapter for the specific threat of authoritarian state coercion. If you will be compelled to provide your passphrase — through legal order or physical pressure — mathematical encryption strength alone is insufficient. You need the ability to provide a passphrase that satisfies your adversary's demand while not exposing your actual data.

The Core Concept

You maintain two entirely separate passphrases:

- **Outer/decoy passphrase:** Opens a volume containing convincing, harmless personal data — something that explains why you had an encrypted drive but contains nothing sensitive.
- **Inner/real passphrase:** Opens the hidden volume inside the outer volume, containing your actual sensitive data.

When a forensic examiner receives the outer passphrase and examines the outer volume, they find:

- A real, functional environment with authentic-looking personal files
- A coherent personal narrative — photos, documents, music, browsing history
- No cryptographic evidence that a second volume exists
- No indication that the outer passphrase is anything other than the only passphrase

Critically: **there is no mathematical proof that a hidden volume exists**. The space inside the outer volume that is not occupied by files is indistinguishable from random padding — which is a standard VeraCrypt feature. An examiner cannot prove the unallocated space contains a hidden volume without the inner passphrase. "I don't know what that random space is" is a cryptographically defensible statement.

Method 1: VeraCrypt Hidden Volume

VeraCrypt is the most widely documented and peer-reviewed tool for plausible deniability. It was designed from the beginning to support this specific use case, unlike LUKS which added limited deniability features later.

Install VeraCrypt:

```
<code># Always download from https://veracrypt.fr # Always verify the PGP signature bef
```

Creating the outer volume:

```
<code># Create a file-based outer volume (20 GiB in this example - size to your needs)
```

Populate the outer volume with decoy content BEFORE creating the hidden volume:

```
<code># Mount the outer volume veracrypt --text --mount /home/user/data/backup.vc /mnt/
photos, documents, music # This step defines the outer volume's used space boundary # T
```

Create the hidden volume inside the outer volume:

```
<code># The hidden volume size must fit within the outer volume's unallocated space # I
```

Using the volumes:

```
<code># Mount the real (hidden) volume for daily work: veracrypt --text --mount /home/u
```

Building a Convincing Decoy Volume

A forensic examiner who examines the decoy volume will look for signs that it is staged. A professional examiner knows what a real personal archive looks like — and what a fake one looks like. The decoy must pass this scrutiny.

Signs of a staged decoy that will raise suspicion:

- All files have creation/modification timestamps from a single day or short window
- The file collection is sparse — only a few files of each type
- Photos do not have EXIF data consistent with a real camera or phone
- Documents have no revision history, tracked changes, or embedded metadata from real use
- No application configuration files (a real user leaves config in every app they use)
- Shell history (if present) shows only a handful of commands, all from one session

What a convincing decoy should contain:

```
<code># Spread file timestamps realistically over 2-3 years of fake history: touch -d "
600 photos spanning 2-3 years, varying quality (phone + camera mix) # [ ] Music collect
provides motivation for the encryption # [ ] Application configs with realistic setting
```

LUKS Multiple Key Slots as Partial Deniability

LUKS2 supports up to 32 independent key slots. You can add a secondary passphrase to a different slot — one that you would provide under coercion. Unlike VeraCrypt hidden volumes, this does not protect the content (all slots open the same volume), but it provides plausibility that you only have one passphrase.

```
<code># Add a second passphrase to slot 2 cryptsetup luksAddKey --key-slot 2 /dev/sdX2
```

LUKS key slot deniability is weaker than VeraCrypt hidden volumes because a careful examiner can see exactly how many key slots are active and can test each one. VeraCrypt hidden volume deniability is cryptographically stronger — there is genuinely no way to prove the hidden volume exists.

Chapter 16. Detached LUKS Headers

The Problem with a Standard LUKS Header

By default, the LUKS2 header sits at the very beginning of the encrypted partition. This 16 MB structure contains everything needed to identify the drive as LUKS-encrypted and to begin a passphrase attack: the magic signature, cipher parameters, PBKDF configuration, and the anti-forensic split (encrypted master key). Any forensic tool will immediately detect it.

What a Detached Header Achieves

When the LUKS header is stored separately from the encrypted data, the encrypted partition contains *only* AES-XTS ciphertext — completely indistinguishable from random noise. Without the header file:

- The partition shows no LUKS magic signature — no encryption is detected by any tool
- Cipher parameters are not revealed — no one can even begin an attack
- The existence of encryption cannot be proven — "I don't know what that partition is" is a credible statement
- Possessing the drive without the header is useless — even the passphrase alone is insufficient to decrypt

Creating a LUKS Volume with a Detached Header

```
<code># The header will live on a separate device - NOT on the encrypted drive # First
the header goes into the separate file cryptsetup luksFormat \ --type luks2 \ --header
no header, no signature # Open the encrypted partition using the detached header crypts
```

Storing the Detached Header Securely

The header file must be stored completely separately from the encrypted drive. The security model is two-factor physical: you need both the drive and the header to even attempt decryption.

Location	Security Level	Availability
Second USB drive on your person (physically separate from the main drive)	High	Always available while you have both
Printed QR code stored in a known physical location	High	Available if you can reach it
Memorized (encode as a short text string)	Very High	Always, if you survive
Trusted custodian in another jurisdiction (encrypted copy)	High	Available via communication
Encrypted cloud storage (GPG-protected)	Medium	Network-dependent

Encrypt the header before storing it anywhere:

```
<code># GPG symmetric encryption with strong parameters gpg --symmetric \ --cipher-algo
safe to store anywhere # Decrypt when needed (before booting): gpg --decrypt luks-heade
```

Shamir's Secret Sharing: Distributing the Header Key

For the highest-stakes scenarios, split the header encryption passphrase across multiple trusted people using Shamir's Secret Sharing. Any M of N shares reconstruct the secret; M-1 shares reveal nothing about the passphrase.

```
<code>pacman -S ssss # Split passphrase into 5 shares, any 3 reconstruct it echo -n "he
passphrase is reconstructed</code>
```

Practical outcome: an adversary must reach and successfully coerce three different people in three different countries simultaneously. If any one of them cannot be reached or refuses to cooperate, the data is permanently inaccessible — including to you, so ensure the distribution is managed carefully.

Chapter 17. Hardware Security Tokens for LUKS

A hardware security token adds a physical "something you have" factor to LUKS unlocking. The encryption key never leaves the token's secure element — cloning is cryptographically infeasible. Seizing the drive alone is insufficient; the token must also be present.

FIDO2 with systemd-cryptenroll

```
<code>pacman -S systemd libfido2 # List connected FIDO2 tokens systemd-cryptenroll --fi
```

At boot, the system prompts you to insert the token and enter its PIN. The token performs a cryptographic challenge-response using its internal private key — a key that cannot be extracted, copied, or cloned. If the token is lost, destroyed, or retained by an adversary, the FIDO2 key slot is inaccessible. Your passphrase-based key slot (slot 0) remains as the backup.

YubiKey HMAC-SHA1 Challenge-Response

```
<code>pacman -S yubikey-full-disk-encryption yubico-pam # Program the YubiKey's second
```

Deliberate Key Slot Strategy

Slot	Type	Purpose
0	Diceware passphrase (6-word)	Primary — reliable without hardware, resistant to brute force
1	FIDO2 token + PIN	Two-factor convenience; requires physical token
2	Diceware (decoy — different words)	Duress passphrase for coercion scenarios (see Chapter 21)
31	Nuke passphrase	Destroys all key material when entered (see Chapter 21)

Chapter 18. Anti-Evil-Maid: Detecting Tampering

The Evil Maid Attack Explained

An adversary with brief, unobserved physical access to your powered-off machine can execute a devastating attack without breaking your encryption:

1. Boot from their own USB drive (bypassing your boot order if BIOS passwords are not set)
2. Replace your GRUB EFI binary on the EFI partition with a modified version that, when you type your LUKS passphrase, quietly sends it to the attacker before pretending to fail (showing an "incorrect passphrase" error)
3. Alternatively: insert a hardware keylogger between your keyboard and motherboard
4. Return the machine looking exactly as it was — no visible changes
5. You boot normally, type your passphrase, the machine "works" — and the attacker now has your passphrase

This attack is named after the "evil hotel maid" who enters your room while you are out. It is documented, practical, and requires no cryptographic expertise — only brief physical access and a prepared USB drive.

Defense 1: Secure Boot with Custom Keys

With Secure Boot active and your custom keys enrolled (Chapter 9), the attacker's malicious GRUB binary is not signed with your Secure Boot key. The UEFI firmware refuses to execute it. This stops the bootloader-replacement attack entirely — the attacker would need your Secure Boot private key (which is stored offline and physically protected) to sign a malicious bootloader.

Defense 2: UEFI Passwords

Without a UEFI administrator password, the attacker can simply enter the UEFI menu and change the boot order to boot their USB first. With a UEFI administrator password set:

- **BIOS administrator password:** Required to modify any UEFI setting, including boot order

- **Power-on password:** Required before the machine POSTs at all — an additional hurdle even before the bootloader
- **Boot device priority locked:** Set to boot only from your SATA-to-USB drive
- **All other boot paths disabled:** USB boot, internal drive, PXE network boot — all disabled

BIOS passwords can sometimes be reset by removing the CMOS battery or a motherboard jumper. This requires opening the laptop case — which your physical tamper indicators will detect.

Defense 3: Physical Anti-Tamper Indicators

Glitter nail polish canary (highly recommended):
Apply clear nail polish mixed with fine glitter across all case screws and at case seams. Photograph the glitter pattern from multiple angles before leaving the device unattended. The glitter distributes randomly and creates a pattern that is impossible to replicate exactly — even with samples of the same glitter and the same nail polish. Before each use, compare the current pattern against your saved reference photographs. Any discrepancy indicates the case was opened.

This technique is used by real security researchers, costs less than five dollars, and requires no technical skill. It is cheap, reliable, and widely understood to be effective. Joanna Rutkowska documented its use in the Qubes OS security model.

UV-reactive ink: Apply to screws and seams. Inspect with a UV flashlight. Invisible in normal light, obvious under UV.

Tamper-evident security seals: Single-use labels with "VOID" patterns embedded in the adhesive, available from industrial suppliers. Apply to all case screws.

Standard pre-use physical inspection:

- All screws present, properly seated, no scratching or stripping marks
- Case seams closed flush with no signs of prying
- All external ports checked for inserted devices
- Keyboard fully seated and not raised at any corner
- USB ports inspected for small keylogger devices

Defense 4: Hardware Keylogger Detection

```
<code># Establish a baseline of connected hardware when the machine is clean: lsusb >~/
```

Chapter 19. TPM2 and Measured Boot

What a TPM2 Chip Does

A Trusted Platform Module (TPM2) is a dedicated security microcontroller present on most modern laptop and desktop motherboards. It provides three functions relevant to your threat model:

1. **Measured boot:** As the machine starts, each boot component (UEFI firmware → bootloader → kernel → initramfs) is measured — SHA-256 hashed — and the hash is recorded in a set of registers called Platform Configuration Registers (PCRs). These measurements are created by hardware, not software, and cannot be forged by any software running after the measurement is taken.

2. **Key sealing:** The TPM can store a secret (such as the LUKS master key or a key that unlocks a key slot) and bind its release to specific PCR values. If the boot chain has been tampered with, the PCR values will differ from when the key was sealed, and the TPM will refuse to release it.

3. **Physical tamper resistance:** The TPM chip is designed to resist physical extraction of secrets. Keys stored inside cannot be retrieved by probing the chip — the chip self-destructs or becomes permanently locked after a number of failed attempts.

How PCRs Detect Evil Maid Attacks

PCR	What It Measures	What Changes It
PCR0	UEFI firmware	Firmware update or replacement
PCR1	UEFI firmware configuration	BIOS settings change
PCR7	Secure Boot state and keys	Secure Boot enabled/disabled, key changes
PCR4	Boot manager (GRUB EFI binary)	GRUB binary replacement — evil maid attack
PCR8	Boot manager configuration	grub.cfg modification
PCR9	Boot entries, kernel, initramfs	Kernel or initramfs modification

If an evil maid replaces your GRUB binary with a keylogging version: PCR4 changes. The TPM-sealed LUKS key cannot be released. The system falls back to requesting the manual passphrase — and you know immediately that something is wrong, because the system normally auto-unlocks.

Setting Up TPM2 Auto-Unlock with clevis

```
<code>pacman -S tpm2-tools tpm2-tss clevis # Verify the TPM2 chip is present and respon
```

Boot behavior with TPM2 active:

- **Normal boot (no tampering):** PCR values match those recorded during sealing → TPM releases the key → LUKS unlocks automatically → no passphrase required
- **Tampered boot (evil maid attack):** PCR4 (or another PCR) differs → TPM refuses to release the key → system asks for manual passphrase → you know the machine was tampered with

Re-sealing After System Updates

```
<code># After any update that modifies the kernel, initramfs, GRUB, or grub.cfg: clevis
```

If you update the kernel or bootloader without re-sealing, the next boot will fail TPM attestation and request the manual passphrase. This is safe — you simply enter your passphrase manually — but you must then re-seal before the next reboot to restore automatic unlocking.

Part VI — Emergency Procedures

Prepare these procedures in advance. In an actual emergency you will have seconds, not minutes, and you will be under stress. Know exactly what you will do before you need to do it.

Chapter 20. Emergency Destruction

Critical insight: you do not need to destroy the entire SSD. You only need to destroy the LUKS header — a 16 MB structure at the start of the partition. Without the header, the remaining ciphertext is permanently and mathematically unrecoverable regardless of computational power applied. The key slots and master key material exist only in those 16 MB. Destroying them makes the rest of the drive — however many gigabytes — permanently inaccessible.

Method 1: Software Destruction (Fastest — Seconds)

If you have a terminal and root access on a running machine:

```
<code># Option A: cryptsetup erase - most targeted, fastest # Overwrites all key slots
takes 1-2 seconds cryptsetup erase /dev/sdX2 # Option B: dd random data over the header
takes 5-10 seconds dd if=/dev/urandom of=/dev/sdX bs=4K count=8192 conv=fsync # Option
most complete cryptsetup erase /dev/sdX2 && sync && poweroff -f</code>
```

Create a ready-to-execute script and keyboard shortcut now, before you need it:

```
cat > /usr/local/bin/emergency-destroy.sh << 'EOF' #!/bin/bash echo "EMERGENCY DESTROY
3 seconds to abort with Ctrl+C" sleep 3 logger -t emergency "DESTRUCTION INITIATED" cry
add to ~/.config/sway/config: bindsym Ctrl+Alt+End exec /usr/local/bin/emergency-destro
```

Method 2: Physical Destruction

When software destruction is not possible — machine is locked, powered off, or the drive is not currently open — physical destruction must achieve one specific goal: shatter the NAND flash chips.

What works:

- **Bend and snap the PCB:** Remove the SSD from the enclosure. A 2.5” SATA SSD PCB (approximately 10 cm × 7 cm) will snap in half with firm two-handed bending. This shatters the NAND packages. The NAND chips are ceramic-encased silicon — they crack when the board flexes.
- **Heel strike:** Drop the bare PCB on a hard floor and apply your full weight through a heel. The NAND chips crack on impact.
- **Crushing:** Any heavy object applied with force to the flat PCB face will crack the NAND packages.

- **Water immersion:** Submersing a running, powered SSD in water (especially saltwater for conductivity) causes immediate and permanent failure. Note: this only works if the drive is powered on during immersion.

What does NOT work:

- **Magnets:** SSDs use NAND flash, not magnetic storage. No magnet has any effect on NAND data — not even industrial-strength rare earth magnets or degaussers.
- **Fire:** NAND flash chips can survive temperatures that melt plastic enclosures and solder. NAND requires sustained temperatures above 700℃ to destroy data — far above what a typical fire produces. Data can sometimes be recovered from SSDs recovered from burnt buildings.
- **Single hammer blow to the enclosure:** The enclosure may absorb the impact. The PCB inside may be undamaged. Always remove the SSD from the enclosure before attempting destruction.

Method 3: Pre-Staged Destruction

```
# One-button, no-confirmation destruction script - use with caution cat > /usr/local/bi
```

This script can be triggered by the dead man's switch (Chapter 22), by a remote signal, or by a pre-arranged local event (machine closes its lid in a specific location, a particular network becomes reachable, etc.).

Chapter 21. LUKS Nuke Passphrase

The LUKS nuke passphrase is a passphrase that, when entered at any LUKS unlock prompt, destroys all key material instead of unlocking the drive. From the outside, entering it looks identical to entering any other wrong passphrase — the system pauses (while destroying the key slots), then reports that the passphrase was incorrect. The key material is gone. The drive is permanently locked.

This is specifically designed for coercion scenarios: you are forced to type "your passphrase" into the machine under observation. You type the nuke passphrase. The examiner sees what appears to be a failed authentication attempt. The data is destroyed.

Installation and Configuration

```
<code># Install from AUR git clone https://aur.archlinux.org/cryptsetup-nuke.git cd cry
# you must not confuse them under stress # Test the nuke (on a test drive, not your rea
the drive is permanently locked</code>
```

Designing the Nuke Passphrase

- **Distinct from your real passphrase:** Choose words or a structure completely different from your diceware passphrase. Consider a single word, a short phrase in a different language, or a deliberately simple but unique phrase.
- **Memorable under extreme stress:** The nuke passphrase must be recallable in a highly stressful situation (detention, interrogation). Simplicity serves this goal better than security — once you type it, it doesn't need to be secret anymore.
- **Not guessable from context:** It should not be a variant of your name, a word visible in the room, or anything an adversary might guess you would choose for a "backup passphrase."
- **Never written down:** The nuke passphrase must be memorized and never stored anywhere. If it is stored, an adversary who finds the storage learns what it is — they would then avoid triggering it and would know to use coercion for the real passphrase instead.

Key Slot Management Overview

```
<code># List all active key slots cryptsetup luksDump /dev/sdX2 | grep -A3 "Keyslots" #
```

Chapter 22. Dead Man's Switch

A dead man's switch (DMS) automatically destroys the drive's key material if you fail to "check in" within a defined window. This protects against scenarios where you are detained or incapacitated without the opportunity to trigger destruction manually — you simply stop checking in, and the system acts on your behalf.

The Check-In Daemon

```
# Configuration file mkdir -p /etc/deadman cat > /etc/deadman/config << 'EOF' CHECKIN_I
maximum time between check-ins CHECKIN_FILE=/home/user/.deadman_alive NUKE_TARGETS="/de
```

Automatic Check-In During Normal Use

```
<code># Reset the clock on every bash command: # Add to /etc/bash.bashrc or ~/.bashrc:
```

Setting the Right Interval

The interval is a balance between two failure modes:

- **Too short:** You accidentally trigger destruction during normal sleep, travel, or an unexpected absence. 4 hours would trigger for anyone who sleeps normally.
- **Too long:** An adversary who seizes your running machine has too much time to extract data before the switch triggers. 48 hours gives them nearly two days.

12 hours is a reasonable default for most people's work patterns. If you operate in environments where you might not have access to a computer for longer periods, extend to 24 hours. If you need tighter control, use 6 hours with the auto-check-in timer active during work sessions.

Part VII — Operational Security

Technical security controls fail without behavioral discipline. These chapters cover the operational practices that make the technical controls meaningful.

Chapter 23. Network Anonymization

Encrypting your drive protects data at rest. Network activity creates a separate, persistent record that survives even if your drive is destroyed. Your ISP logs every connection. State surveillance systems log DNS queries, connection metadata, and traffic patterns. This chapter covers minimizing that record.

Tor: The Foundation

Tor routes your traffic through three volunteer relays (Guard → Middle → Exit), encrypting each layer independently. Your ISP sees only an encrypted connection to your Guard node — they cannot see your destination, the content, or the timing of individual requests at the application layer. The destination sees only the Exit node's IP address.

```
<code>pacman -S tor torsocks systemctl enable --now tor # Verify Tor is routing traffic
```

Tor configuration hardened for high-threat environments:

```
<code>cat >/etc/tor/torrc << 'EOF' DataDirectory /var/lib/tor Log warn file /var/log/to
```

Pluggable Transports: When Tor Is Blocked

Authoritarian states deploy deep packet inspection (DPI) systems that detect and block standard Tor traffic. Even when content is encrypted, Tor connections have recognizable statistical characteristics (packet sizes, timing patterns, handshake sequences). Pluggable transports disguise Tor traffic as something benign:

Transport	Disguise	Best For
obfs4	Random noise — no identifiable protocol grammar	States that block by known protocol signatures
Snowflake	WebRTC — same as video calling traffic	States that must keep video calling working; very hard to block without collateral
meek-azure / meek-amazon	HTTPS to Microsoft/Amazon CDN	States that cannot block entire cloud providers
WebTunnel	Camouflages as HTTPS website traffic	High-censorship environments; newer protocol

```
<code># Install obfs4 transport client pacman -S obfs4proxy # Get bridge addresses from
```

Tor Browser: Correct Usage

Running a standard browser (Firefox, Chromium) with Tor as a proxy is significantly less secure than Tor Browser. Tor Browser is hardened specifically to prevent browser fingerprinting — techniques that identify individual users based on browser characteristics (installed fonts, screen resolution, plugin list, canvas rendering, WebGL behavior, time zone) even across Tor circuits.

```
# Verify before running gpg --auto-key-locate nodefault,wkd --locate-keys torbrowser@to
```

Rules for using Tor Browser correctly:

- **Never resize the browser window.** Window size is a fingerprint. All Tor Browser users with the default window size appear identical. Resizing makes you unique.
- **Never install extensions.** Every extension changes your fingerprint. Even uBlock Origin makes you distinguishable from other Tor Browser users who don't have it.
- **Set Security Level to Safest.** This disables JavaScript. Many sites will not work. This is the security working correctly for high-threat environments.
- **Never log into personal accounts.** Your Google, Facebook, or email login identity is not anonymous over Tor — it identifies you directly. Creating a Tor-only pseudonymous identity (that is never used from your real IP) is a different matter.
- **Do not open downloaded documents while online.** PDFs, DOCXs, and other documents can contain network calls that bypass Tor. Open them offline, or in a sandboxed environment.

MAC Address Randomization

Your WiFi adapter's MAC address is transmitted to every access point you connect to and is logged by network operators. A consistent MAC address across different locations creates a location tracking record. Randomize it:

```
<code>cat >/etc/NetworkManager/conf.d/mac-randomization.conf << 'EOF' [device] wifi.sca
```

Encrypted DNS

DNS queries reveal every domain you visit to your ISP and network operator. Over Tor, DNS is handled by the exit node and is effectively anonymized. For non-Tor traffic (or as a baseline when Tor is not running):

```
<code>pacman -S dnscrypt-proxy cat >/etc/dnscrypt-proxy/dnscrypt-proxy.toml << 'EOF' li
```

Chapter 24. Day-to-Day Operational Security

Compartmentalization

The most important operational security principle: **separate identities, devices, accounts, and contexts must never cross.** Any connection between your real identity and your protected activities is a vulnerability. Once two compartments are linked, that link is permanent and cannot be undone.

Compartment	Device	Network	Activities
Personal / Real Identity	Personal phone and laptop	Home ISP, mobile data — both linked to your identity	Social media under real name, banking, legal personal activities
Protected / Pseudonymous	Your encrypted SATA-to-USB drive	Tor only — never direct internet from this compartment	All sensitive activities, protected communications
Emergency / One-Time	Tails OS on a separate USB stick	Tor via public WiFi not linked to your identity	Crisis communications, one-time contacts

Rules that must never be broken:

- Never log into personal accounts (Google, Facebook, your ISP portal, your bank) from the protected compartment. Even the timing of your login — "someone logged into Gmail from Tor at 14:32 EST" — can be correlated with other intelligence.
- Never access protected resources from your personal compartment device. Logging into a Tor hidden service from your home IP, even once, links your real identity to that service permanently in your ISP's logs.
- Never discuss protected compartment activities on personal compartment communications. The content does not need to be captured — mentioning it at all creates a connection.
- When in doubt: create a third separate compartment rather than crossing between existing ones.

Physical Security Habits

Shut down, never sleep. When your machine is sleeping or hibernating (despite `nohibernate`), the LUKS decryption key remains in RAM. Cold boot attacks require a

powered-on or recently powered-off machine with RAM contents that have not yet decayed. When you power off completely, RAM loses all contents within seconds to minutes (faster if cooled; slower in freezing temperatures). Always shut down:

```
<code># Configure automatic shutdown on lid close: # In /etc/systemd/logind.conf: Handl
```

Screen lock on every departure. Build the habit of locking the screen every time you step away — even for thirty seconds. An unlocked screen is an opportunity. Configure the lock shortcut:

```
# In ~/.config/sway/config: bindsym $mod+l exec swaylock -f -c 000000 # Lock immediatel
```

Passphrase entry awareness:

- Be aware of cameras (CCTV, phone cameras held by bystanders) before entering your passphrase in a public space
- Reflective surfaces — glass, polished metal, cups of liquid — can reflect your screen to someone behind you
- Shoulder surfing: know who is behind you when working in public spaces. Sit with your back to a wall.
- At checkpoints or border crossings: power down before approaching. A powered-off, encrypted device requires the passphrase to access — which you are not required to provide (with jurisdiction-appropriate caveats; see Chapter 26).

Password Management

```
<code>pacman -S keepassxc # KeePassXC configuration for high-threat environments: # Set
never synced to cloud services # Key file stored separately from the database (on hardw
```

Password reuse is how one data breach cascades into access to all your accounts. A forensic examiner who recovers one password (from an unencrypted device, or from a breached service) should not thereby gain access to anything else.

Travel Security

Border crossings are a specific high-risk scenario. Customs officers in many countries can legally demand device access, and in some jurisdictions (United States, United Kingdom, Australia) refusal can result in detention, fines, or device seizure.

- **Power down before any checkpoint.** A powered-off, encrypted drive requires the passphrase to access. Agents cannot access its contents through UEFI menus or boot manipulation if Secure Boot is configured correctly. Do not sleep, do not hibernate — shut down completely.

- **Consider traveling without sensitive data.** For very high-risk crossings, leave the encrypted drive at your destination and travel with a clean machine or no machine at all. Retrieve the drive after crossing, or use secure remote access to your data from a trusted machine at the destination.
- **Know the specific law for your specific crossing.** The rules differ between countries and even between entry points within the same country. Have a lawyer's number accessible before you cross.
- **Have a plan if the device is seized.** What will you say? Who will you call? Will you provide the outer/decoy passphrase? Will you refuse? Decide in advance — you will not be given time to think at the checkpoint.

Chapter 25. Encrypted Communications

Your encrypted drive protects data at rest. What you transmit creates a separate, persistent record that does not disappear when you destroy the drive. Treat communications as a parallel but distinct threat surface.

Signal

Signal provides end-to-end encrypted messaging and calls. It uses the Double Ratchet algorithm — each message generates a new key, meaning that even if an adversary obtains your current key material, they cannot decrypt past messages (forward secrecy) and future sessions generate entirely new keys (break-in recovery). This is the strongest practical security available in a widely-deployed consumer application.

```
<code># Install via Flatpak for automatic updates: flatpak install flathub org.signal.S
```

Required configuration for high-threat environments:

- **Disappearing messages:** Set for every conversation. 24 hours or less for high-risk contacts. Messages deleted from both devices after the timer expires.
- **Safety numbers verification:** Compare the 60-digit Safety Number with each contact in person or by voice. This proves that Signal is not performing a man-in-the-middle attack — that you are actually communicating with the person you think you are, not an interceptor.
- **Registration lock:** Enable this (Settings → Account → Registration Lock) with a strong PIN. Prevents an adversary who steals your phone number from registering your Signal number on a new device.
- **Note on phone numbers:** Signal requires a phone number to register. That phone number is a link to your real identity. For high-threat scenarios, consider obtaining a SIM card without identity linkage, or using alternatives that require no phone number.

Higher-Anonymity Messaging Alternatives

Tool	Identifier Required	Network	Best For
Briar	None — no registration at all	Tor, Bluetooth, local WiFi — P2P, no server	Activists who need offline capability and no identity linkage
Session	Random session ID (not phone number)	Decentralized onion routing	Pseudonymous group communications
SimpleX	None — no user identity exists at all	SimpleX servers (can self-host)	Maximum metadata protection; no linkable identity
Matrix/Element (self-hosted)	Username on your server	Federation with E2E encryption	Group communications with full infrastructure control

GPG for Encrypted Email

SMTP email is transmitted in cleartext between servers by default. GPG provides end-to-end encryption at the message level — the server never sees the message content, only the encrypted payload.

```
pacman -S gnupg thunderbird # Generate an Ed25519 key pair (modern elliptic curve -
faster and more secure than RSA 2048) gpg --expert --full-generate-key # Choose: (9) EC
```

GPG encrypts only the message body. The email headers — To, From, Subject, Date, and message routing information — are completely unencrypted and visible to your email provider, their ISP, and any surveillance system between. For communications where even the existence of contact is sensitive, email is the wrong channel — use Signal, Briar, or SimpleX instead.

Chapter 26. Legal Rights and What to Say Under Duress

The Legal Landscape

Jurisdiction	Relevant Law	Penalty for Refusal
United States	5th Amendment (self-incrimination); "foregone conclusion" doctrine may override it	Contempt of court — potentially indefinite pre-trial detention
United Kingdom	RIPA 2000, Sections 49 and 53	Up to 2 years (5 years in national security cases)
Australia	TOLA Act 2018; Criminal Code Act 1995 s.3LA	Up to 10 years imprisonment
Canada	No specific compelled decryption statute; Charter rights debated case-by-case	Contempt in some circumstances
France	Article 434-15-2 of the Code Pénal	3 years and €45,000 fine; 5 years if terrorism-related
Germany	Nemo tenetur se ipsum accusare (constitutional right against self-incrimination)	Limited — courts generally respect the right
Authoritarian states (various)	No meaningful legal protection; laws may not be published or consistently applied	Arbitrary; detention without charge possible

What to Say If Your Device Is Seized

The following principles apply regardless of jurisdiction. They are not legal advice — consult a qualified lawyer familiar with your specific country's laws before any high-risk situation:

1. Say as little as possible.

"I am invoking my right to remain silent. I would like to speak with a lawyer before answering any questions."

This is the correct answer to nearly every question you will be asked. Volunteering information — even true, harmless information — creates an inconsistency record that can later be used against you. Every statement you make can be quoted back to you in a way that contradicts another statement. Silence cannot be contradicted.

2. Do not confirm the device is encrypted.

"I cannot comment on the device's configuration without speaking with my lawyer."

You are not required to provide a tutorial on your security setup. The forensics team will determine what they are dealing with independently. You gain nothing by informing them and may give up strategic information.

3. Do not confirm how many passphrases exist or acknowledge the existence of hidden volumes.

If you have set up VeraCrypt plausible deniability, the existence of a decoy passphrase is consistent with having only one passphrase. Do not volunteer that others exist. Do not look uncomfortable when asked. You practiced providing only the decoy passphrase — behave as though it is the only one.

4. Do not interact with the device while observed.

Any interaction with the device — even appearing to try to unlock it or claiming you forgot the passphrase after entering something — provides evidence about your relationship to the device. Do not touch it without your lawyer present.

5. If legally compelled to provide a passphrase, know your decision in advance.

This decision — which passphrase to provide, whether to refuse, whether to activate the nuke — must be made in advance and practiced. Under detention you will be tired, frightened, and pressured. You will not be able to think clearly. Make the decision now, in a calm moment, in consultation with a lawyer who knows your situation.

6. Document everything you can about the encounter.

Agent names or badge numbers, time, location, what was asked, what was said. Write this down immediately after the encounter or when you have access to writing materials. This documentation may be important for legal proceedings.

Before Any High-Risk Situation

- **Have a lawyer's contact on paper before you leave.** A phone number and a brief description of your situation, carried separately from your devices.
- **Brief your support network.** Someone who knows your travel plan, knows you may be at risk, and knows to take action (contact the lawyer, contact a journalist organization, contact an embassy) if you do not check in by a specific time.
- **Set a check-in schedule.** "I will message you at 18:00 every day I am in this location. If you don't hear from me by 18:30, call the lawyer and wait for my call."
- **Prepare a prearranged distress signal.** A specific word or phrase in a normal-sounding message that means "I am detained, take action." Both you and your support contact must memorize this signal and know exactly what action it triggers.

Organizational Resources

These organizations provide free emergency assistance to journalists, human rights defenders, and activists:

- **Access Now Digital Security Helpline** — accessnow.org/help — Free, 24/7, available in many languages
- **Front Line Defenders** — frontlinedefenders.org — Emergency response for human rights defenders
- **Committee to Protect Journalists** — cpj.org/safety — Digital security for journalists
- **EFF Surveillance Self-Defense** — ssd.eff.org — Guides and resources by threat level

Appendix A: Quick Reference Checklist

Use this before and after any significant system change, and review it periodically to ensure drift has not occurred.

Installation & Encryption

- Arch Linux ISO verified with GPG signature; fingerprint confirmed against archlinux.org
- Drive fully overwritten with cryptographic random data before installation
- LUKS2 created: AES-XTS-256, Argon2id, ≥4 GiB memory, ≥6 iterations
- Diceware passphrase: 6+ words, generated with physical dice, memorized
- LUKS header: standard or detached (stored separately from drive)
- `linux-hardened` kernel installed
- `encrypt` hook present in `/etc/mkinitcpio.conf`, in correct position
- initramfs rebuilt after any hook or kernel change

Boot Security

- UEFI administrator password set
- Boot order locked: only your SATA-to-USB drive, all other paths disabled
- Secure Boot enabled with your custom keys (sbctl)
- GRUB password set with PBKDF2 hash
- TPM2 measured boot configured (clevis) if TPM2 chip is present
- pacman hook re-signs EFI binaries and re-seals TPM after updates

System Hardening

- sysctl security parameters applied via `/etc/sysctl.d/99-security.conf`
- nftables active with default-deny inbound policy
- Unnecessary services disabled and masked

- Shell history disabled system-wide
- Swap disabled or encrypted with ephemeral random key
- Screen lock: 60-second timeout, activates before sleep
- Lid close triggers shutdown (not sleep)

Counter-Forensics

- journald: `Storage=volatile` (logs in RAM only, gone on shutdown)
- `~/.cache`, `/tmp`, `/var/tmp` on tmpfs
- `noatime,nodiratime` on all persistent mounts
- mat2 and exiftool installed; metadata scrubbed from all files before sharing
- ext4 journal configured to `data=writeback` or disabled

Drive Seizure Resistance

- Deniability strategy implemented (VeraCrypt hidden volume and/or detached header)
- Decoy volume populated with convincing content spanning realistic timeframe
- Hardware security token enrolled as second factor (if threat model warrants)
- Glitter nail polish canary applied; reference photo taken and stored
- Pre-use physical inspection procedure understood and practiced
- USB baseline documented (`lsusb`, `lspci` outputs saved)

Emergency Procedures

- Emergency destruction script at `/usr/local/bin/emergency-destroy.sh`
- Emergency destruction keyboard shortcut configured and tested on a spare drive
- LUKS nuke passphrase configured and memorized (not written down)
- Dead man's switch daemon enabled and check-in timer working
- Physical destruction procedure understood; practice run completed
- Support network briefed; check-in schedule established; distress signal agreed

Operational Security

- Tor daemon running; pluggable transport (obfs4 or Snowflake) configured
- Tor Browser installed and verified; security level set to Safest
- MAC address randomization active for WiFi and Ethernet
- dnscrypt-proxy running; `/etc/resolv.conf` locked
- Signal installed; disappearing messages enabled on all conversations; safety numbers verified
- KeePassXC database on encrypted drive; key file on hardware token
- Lawyer contact known and documented on paper, separate from devices

Appendix B: Emergency Contacts and Resources

Digital Security Emergency Helplines

Organization	What They Offer	Contact
Access Now Digital Security Helpline	Free 24/7 emergency digital security support for journalists, activists, NGOs. Multiple languages.	accessnow.org/help — help@accessnow.org
Front Line Defenders	Security and protection for human rights defenders; emergency response.	frontlinedefenders.org/emergency
Committee to Protect Journalists	Digital security specifically for journalists.	cpj.org/safety
EFF Surveillance Self-Defense	Guides, threat modeling, tool recommendations by threat level.	ssd.eff.org
Reporters Without Borders	Support for journalists facing digital threats.	rsf.org

Privacy-Respecting Tools Referenced in This Guide

Tool	Purpose	Source
Tor Browser	Anonymous web browsing with anti-fingerprinting	torproject.org
Tails OS	Amnesiac live operating system — no trace on host machine	tails.boum.org
Signal	End-to-end encrypted messaging and calls	signal.org
Briar	P2P encrypted messaging — no server, works offline via Bluetooth	briarproject.org (F-Droid)
SimpleX	Encrypted messaging with no user identifier	simplex.chat
KeePassXC	Local password manager with Argon2id KDF	keepassxc.org
VeraCrypt	Full-disk and container encryption with hidden volumes	veracrypt.fr
Protonmail	Encrypted email with servers in Switzerland	proton.me
Riseup	Email and VPN operated by and for activists	riseup.net (invite required)
mat2	Metadata stripping for documents and images	0xacab.org/jvoisin/mat2

Key Academic References

"Self-Encrypting Deception: Weaknesses in the Encryption of Solid State Drives"
Carlo Meijer and Bernard van Gastel — IEEE Symposium on Security and Privacy, 2019
ru.nl/publish/pages/909282/draft-paper.pdf
The paper that documented catastrophic weaknesses in hardware SSD encryption (Samsung, Crucial), motivating this guide's insistence on LUKS2 software encryption.

"Lest We Remember: Cold Boot Attacks on Encryption Keys"
J. Alex Halderman et al. — USENIX Security Symposium, 2008
usenix.org/…/halderman.pdf
Demonstrated that DRAM retains contents for seconds to minutes after power loss (longer when cooled). Motivates this guide's emphasis on full shutdown over sleep.

"Security Analysis of a Full-Disk-Encryption Protocol"
The Cryptography and Security research community's ongoing work on XTS mode and LUKS security properties.
See: ArchWiki dm-crypt at wiki.archlinux.org/title/Dm-crypt

Security is a continuous practice. Revisit this checklist regularly. Update software. Test your emergency procedures before you need them. No technical control substitutes for awareness, discipline, and a support network.

The Techno Anarchist
Encrypted Arch Linux on SATA-to-USB
A Practical Field Guide for Resisting Drive Seizure, Forensic Analysis, and State Coercion

Legal notice: All techniques described are documented open-source security practices. Encryption, privacy tools, and secure deletion are legal in most jurisdictions and are actively advocated by the Electronic Frontier Foundation, Amnesty International, and Access Now. Understanding your local laws is your responsibility. This guide is published for educational and civil liberties purposes.
Revised July 2026 · 26 Chapters · 7 Parts