Modern Cryptography: A Deep Self-Study Guide — Free Sample

Security Definitions, Protocols, Engineering Failures, Zero-Knowledge Systems, and Post-Quantum Migration

Kyle Wu

2026-07-15

Version, Rights, and Scope of Use

Publisher. Published by Readude, an independent digital publishing brand operated by Kyle Wu. The selected primary domain is `readude.com`. Website, checkout, payment, and fulfillment authorization remain outside this manuscript's editorial status.

Version status. This file is the content-complete v1.0.0-rc.3 Release Candidate: the primary canonical English edition for the formal Beta and proposed paid Early Access bundle. It is not stable v1.0.0, an independently peer-reviewed text, a cryptographic audit, or authorization for public sale. Public sale remains conditional on the documented Dodo/transaction lifecycle, device QA, Beta triage, R2, and final release gates.

AI and translation disclosure. AI systems assisted research organization, drafting, editing, and translation. Kyle Wu remains the author and publisher and is responsible for editorial judgment. English is the primary canonical edition. Traditional Chinese is a full-content-parity companion candidate for Beta and Early Access; Simplified Chinese remains experimental and is excluded from the formal 2026-08-03 Beta and initial paid bundle. Flagged proofs, time-sensitive claims, and localization issues remain in the review register. See "About This Edition" for the fuller method and its limits.

Third-party names and works. Kryptos is a copyrighted artwork by James Sanborn. This book mentions the work and related facts for criticism, history, and education. It is not sponsored, authorized, or endorsed by James Sanborn, the Central Intelligence Agency, Paradigm, or any other referenced organization. The commercial edition does not use photographs of the sculpture, Paradigm imagery, or the complete K4 ciphertext as cover art, decoration, or primary sales material.

Limits of use. This book is educational material. It is not a security audit, legal advice, tax advice, investment advice, or financial advice. Each code example must be interpreted according to its chapter label. Examples that have not undergone an independent security audit must not protect real assets, keys, wallets, identities, or production systems.

Currency and errata. The technical status date is July 15, 2026. Standards, libraries, attacks, and post-quantum migration guidance continue to change. Before implementation, readers should consult the primary specification, current errata, and applicable deployment profile. Version and errata records appear at the end of this edition.

Critical Safety Notice

This book contains many educational implementations. Code labeled toy implementation, derivation only, or not for production exists to expose mathematical structure and failure modes. Real systems should use reviewed libraries, standardized algorithms, disciplined key management, constant-time implementations where required, published test vectors, and formal security review.

In particular:

- Do not design a new cryptographic algorithm or protocol merely because an example appears understandable.
- Do not confuse “decrypts successfully” with “secure.”
- Do not reuse a nonce, initialization vector, one-time signature nonce, or one-time key when the construction requires uniqueness.
- Do not encrypt without authenticating integrity.
- Do not use a general-purpose hash directly as a password hash, MAC, or KDF.
- Do not place private keys, seed phrases, API secrets, HSM/KMS permissions, or live credentials in logs, error messages, test fixtures, or AI conversations.
- Do not use this book’s examples to protect real assets.

Preface: Why Begin with Kryptos?

Kryptos is an encrypted sculpture installed at the headquarters of the United States Central Intelligence Agency. It combines ciphers, physical materials, spatial design, historical clues, and human puzzle-solving into a challenge that has persisted for decades. Its value is not limited to whether a particular ciphertext has been solved. It forces the solver to confront the full cryptanalytic workflow:

1. Define the data. Which characters are ciphertext? Which are formatting? Which may be mistakes or deliberate noise?
2. Propose models. Substitution, transposition, polyalphabetic substitution, matrix operations, or a layered construction?
3. Exploit prior information. Language statistics, known plaintext, the creator's habits, the work's context, physical orientation, and external clues.
4. Construct a scoring function. How do we measure whether one candidate is more language-like than another?
5. Search the hypothesis space. Hand analysis, exhaustive search, heuristics, SAT/SMT, machine learning, or human-guided exploration?
6. Verify an answer. Finding text that appears plausible is not the same as proving that it is the unique, intended solution.
7. Govern a secret. Once the answer is known, how can others test submissions without revealing it?

In June 2026, Paradigm publicly described an architecture for custody and verification of K4. A candidate plaintext is first hashed with SHA-256. A secret key held in Google Cloud KMS then produces an HMAC tag, allowing comparison against a reference tag without publishing an unkeyed hash that would enable unrestricted offline guessing. Paradigm stated that Sanborn entered the reference answer on an isolated device, that the plaintext never left that device, and that the device was subsequently wiped. The case connects classical cryptanalysis to modern hashing, MACs, key management, input normalization, rate limiting, trusted hardware, and governance of high-value secrets.

This book follows the same spirit: cryptography is not a catalog of algorithms. It is a method for connecting intuition, mathematics, adversarial models, proofs, and engineering constraints.

How to Use This Book

Learning Paths

Complete 24-Week Path

Plan for 8–12 hours per week. Read in order, complete every Core exercise, and finish one MVP capstone. Appendix C separates each week into Core, Stretch, and Research extension work.

12-Week Accelerated Path

Prioritize Chapters 0, 3, 5–7, 10–12, 14–20, 22–26, 28–30, and 32. Pair adjacent chapters, complete one experiment per session, and use the Core guided hints only after attempting the exercise.

Engineering-First Path

Read Chapters 0, 7, 10–13, 18–21, and 27–30. This route emphasizes interfaces, state, parsers, key lifecycle, testing, incidents, and migration. Return to Chapters 3–6 whenever a proof or bound is doing real design work.

Blockchain and Zero-Knowledge-First Path

Read Chapters 0, 3–6, 11–17, and 22–27 before Chapter 31. Do not skip finite fields, security games, hashing, signatures, transcripts, or Chapter 24's field-semantics warnings.

Formal-Security-First Path

Read Chapters 3–7, 12–18, and 22–26. Reconstruct the central argument in each of the four Detailed Proof Sketches, then write one definition gap and one concrete bound for each system you study.

Incident and Audit-First Path

Read Chapters 0, 6, 10, 13–14, 17, 19–21, 24, and 28–30. Begin with Chapter 28's case-study template, follow every failure back to its primitive or protocol chapter, and finish with an executable release-evidence matrix.

The Chapter Learning Loop

1. Read the threat model and security objectives first.
2. Derive correctness yourself before looking at an answer.
3. Run the experiment and deliberately violate one assumption.
4. Write one paragraph explaining why the construction is secure.
5. Write a second paragraph explaining the conditions under which it fails.
6. Complete the end-of-chapter exercises, then repeat at least one exercise a week later.

Competency Levels

Level	You should be able to…
L1 — Intuition	Explain what a primitive is for and identify its primary risks.
L2 — Derivation	Work through a small example by hand and prove correctness.
L3 — Security	Define a security game, the adversary's capabilities, and the winning event.
L4 — Engineering	Compose standard library primitives correctly while handling serialization, nonces, errors, and versioning.

Level	You should be able to…
L5 — Audit	Identify replay, downgrade, key-substitution, side-channel, state-machine, and key-lifecycle failures in a protocol description.
L6 — Research	Read papers, reconstruct proof sketches, compare assumptions and concrete security, and formulate testable new questions.

Contents

Chapter 0 — What Does Cryptography Protect?

Cryptography starts with a protected asset and a precisely bounded adversary.

Chapter Map

- Prerequisites: None beyond basic software concepts
- Core path: Assets, adversary, boundary, winning event
- Advanced path: Concrete security and residual risk
- Research path: Translate product requirements into games
- Estimated core time: 45 min
- Estimated full time: 90 min
- Primary chapter type: Foundation

Learning Objectives

After completing this chapter, you should be able to:

- Replace the vague requirement “we need encryption” with testable security objectives.
- Specify assets, trust boundaries, adversarial capabilities, winning events, and failure consequences.
- Distinguish confidentiality, integrity, source authentication, freshness, unlinkability, and availability.
- Explain Kerckhoffs’s principle, the security parameter, and computational security.
- Distinguish encoding, compression, obfuscation, steganography, and cryptography.

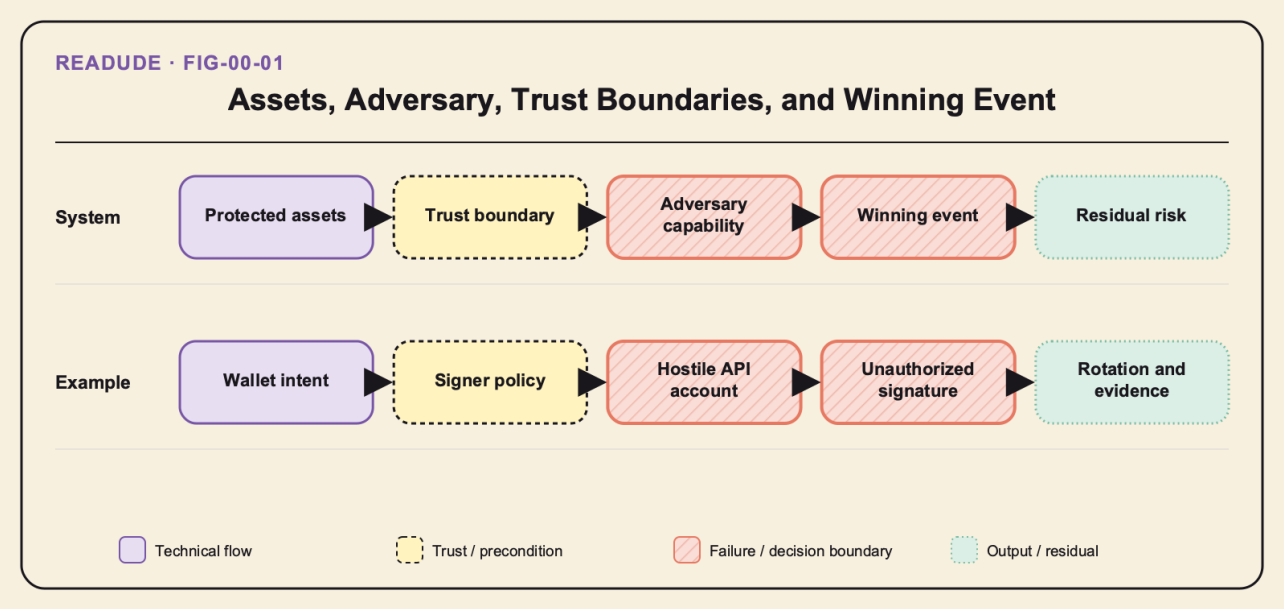

Figure 1: fig-00-01 — A threat-model flow from protected assets through trust boundaries to adversary capabilities and a measurable winning event.

Why This Matters

A design cannot choose the right primitive until it states what failure means. Cloud notes, transaction signers, and authentication services all cross several trust boundaries.

A recurring mistake is to assume that the phrase “we need encryption” is a specification; it is not yet a threat model. This chapter keeps the guarantee, its assumptions, and the engineering boundary in the same view.

Threat Model or Problem Statement

Problem boundary. A design cannot choose the right primitive until it states what failure means.

Model limit. A confidentiality game omits availability, endpoint compromise, metadata, and authorization mistakes.

Core Concepts

0.1 Begin with Assets and Adversaries

The first question in cryptographic design is not "AES or RSA?" It is:

> After the adversary acquires which capabilities must the system still guarantee which properties?

A minimal threat model should include the following fields:

Field	Questions
Assets	Which matter most: plaintext, private keys, identity, transaction authorization, metadata, or availability?
Principals	Who may create, read, modify, delete, or sign data?
Trust boundaries	Which components are trusted: browser, server, KMS, HSM, cloud administrator, or supply chain?
Adversarial capabilities	Passive eavesdropping, active packet modification, replay, chosen plaintexts, chosen ciphertexts, partial-key exposure, memory reads, or clock control?
Winning event	Guess the challenge bit, forge a new message, recover a private key, or cause two endpoints to accept different transcripts?
Cost and time	Online-query count, offline computation, memory, quantum capability, or attack window?
Consequences	One record exposed, all history decrypted, permanent identity impersonation, or irreversible asset transfer?

Example: A Wallet-Signing Service

"Protect the private key" is still too vague. A better specification is:

- The adversary may fully control the API network, replay old requests, and manipulate transaction fields.
- The adversary cannot directly read the private key inside the HSM, but may compromise a low-privilege application account.
- The system must sign only policy-compliant EIP-712 structures containing a unique request ID, an approved chain ID, an approved contract, and an applicable spending limit.
- An authorization must not be replayable on another chain, against another contract, under another protocol version, or after expiration.
- A policy bypass, raw-signature oracle, or unlogged administrator action is a security failure.

Only after writing this down do we have enough information to choose domain separation, serialization, replay protection, authorization, signature formatting, and audit controls.

0.2 Core Security Properties

Confidentiality

An unauthorized party cannot learn the protected content. Modern definitions usually demand more than an inability to decrypt the entire message: a ciphertext should not reveal any efficiently computable information about the plaintext beyond what the model explicitly permits.

Integrity

Unauthorized modification must be detected. Encryption alone generally does not provide integrity. A malleable ciphertext may even let an adversary alter the decrypted result in a controlled way without knowing the plaintext.

Authenticity

The receiver establishes that data originated from a party holding a particular secret or private key. A MAC provides source authentication under a shared secret. A digital signature provides publicly verifiable source authentication.

Freshness

The message belongs to the current protocol execution rather than being an old message replayed into a new context. Typical mechanisms include counters, sequence numbers, nonces, timestamps, challenges, and protocol state machines. A signature by itself does not prevent replay.

Forward Secrecy

If a long-term private key is compromised later, past session keys should remain unrecoverable. This property commonly depends on ephemeral Diffie–Hellman secrets that were securely erased.

Post-Compromise Security

After an adversary temporarily obtains endpoint state and later loses control, the protocol can gradually restore confidentiality and authenticity by mixing in fresh secrets. The Double Ratchet is an important example.

Anonymity, Unlinkability, and Unobservability

Encrypting content does not hide who communicates with whom. Length, timing, IP addresses, reused public keys, transaction graphs, and gas patterns can all leak metadata.

Non-Repudiation

Use this term carefully in engineering. A digital signature can show that a private key produced a signature under specified assumptions. It does not by itself prove that a particular natural person consciously approved the action. That conclusion also depends on key custody, malware resistance, authorization workflow, law, and the evidentiary chain.

Availability

Cryptography rarely guarantees availability on its own. An adversary may still drop packets, delete ciphertexts, lock access to keys, exhaust an HSM quota, or trigger expensive verification paths.

0.3 Correctness and Security Are Different Claims

For an encryption scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, correctness may be written as:

$$\Pr[\mathsf{Dec}_k(\mathsf{Enc}_k(m)) = m] \geq 1 - \varepsilon.$$

Correctness says that the legitimate workflow recovers the plaintext. Security says that every efficient adversary has only a small advantage in a specified game. Many catastrophically insecure schemes are perfectly correct.

0.4 Kerckhoffs's Principle

Assume that the adversary knows the algorithm, protocol, source code, data format, and parameters. Security should depend only on keys that are protected and replaceable. Engineering consequences include:

- Do not treat a proprietary format or unpublished algorithm as the primary defense.
- The protocol should remain secure after public disclosure and scrutiny.
- A compromised key should be replaceable without redesigning the entire system.
- Public review, test vectors, and interoperability are security assets.

Security through obscurity can add friction, but it cannot replace analyzable cryptographic security.

0.5 Security Parameters and Negligible Functions

Let λ denote the security parameter. The adversary is usually modeled as probabilistic polynomial time (PPT), and its success advantage should be negligible:

$$\forall c > 0,\ \exists N,\ \forall \lambda > N : \quad \mathrm{negl}(\lambda) < \lambda^{-c}.$$

Engineering decisions also require concrete security. An asymptotically valid reduction may still be useless if it loses a factor of 2^{40}, requires an unrealistic number of queries, or ignores multi-user amplification.

0.6 Cryptography Is Not the Same as These Techniques

Technique	Primary purpose	Requires a secret key?
Encoding	Representation and transport, such as Base64	No
Compression	Reduce redundancy	No
Obfuscation	Increase the cost of understanding or reverse engineering	Not necessarily
Steganography	Hide the existence of a message	Not necessarily
Encryption	Protect content under a key model	Yes
Hashing	Produce a fixed-length digest; properties depend on use	No
MAC	Authenticate integrity and origin under a shared secret	Yes
Signature	Sign with a private key and verify publicly	Yes

Base64, XOR with a fixed constant, character substitution, and custom compression are not modern encryption.

0.7 Security-Requirements Worksheet

Before designing a new system, complete the following worksheet:

Code `code-00-01` — Pseudocode / trace / format; not runtime-tested.

```
Assets:
Data owners:
Authorized readers/signers:
What the adversary can observe:
What the adversary can modify:
Oracles available to the adversary:
Long-term secrets the adversary may obtain:
Historical/future windows that must remain protected:
```

```
Acceptance conditions:
Rejection conditions:
Definition of replay:
Serialization and domain:
Key creation, storage, rotation, revocation, and destruction:
Acceptable leakage:
Unacceptable failure:
```

Worked Example: A cloud note and a wallet signer

Goal. Turn two vague security requests into testable objectives.

Inputs and assumptions. A note service with sharing and a signer behind an HSM; the network and application account may be hostile.

Steps.

1. List assets and principals.
2. Mark trust boundaries and adversarial capabilities.
3. Write a winning event and the unacceptable blast radius.

Result. The note requires confidentiality, integrity, revocation semantics, and metadata limits; the signer requires typed authorization, freshness, and policy enforcement.

What this establishes. The worksheet separates cryptographic objectives from operational controls.

What this does not establish. It does not select an algorithm or prove that the implementation meets the objectives.

Definition Gap

What the chapter's main definitions cover. Start with assets and winning events.

What remains outside the model. A confidentiality game omits availability, endpoint compromise, metadata, and authorization mistakes.

Why the omitted property matters in deployment. Confidentiality or authenticity can hold while availability, metadata exposure, endpoint compromise, authorization, or recovery still fails; each residual asset needs an explicit owner, control, and test.

Engineering Notes

Store the threat model beside the protocol version. A diagram that lives only in a launch deck will age faster than the system.

End-of-Chapter Exercises

1. [ex-00-c01 · Concept · Core] For a cloud-notes application, write separate security objectives for confidentiality, integrity, revocation of sharing, and metadata protection.
2. [ex-00-d01 · Derivation · Core] Explain why "every API request is signed" is still insufficient to prevent replay.
3. [ex-00-e01 · Engineering · Intermediate] Compare the trust boundaries of server-side encryption and end-to-end encryption.
4. [ex-00-a01 · Attack · Intermediate] For a transaction signer, list at least five engineering failures outside the cryptographic algorithm itself that could still cause asset loss.
5. [ex-00-p01 · Proof · Advanced] Evaluate the claim: Does a public salt violate Kerckhoffs's principle? Why or why not?
6. [ex-00-e02 · Engineering · Intermediate] Define a winning event for metadata privacy in the cloud-note example.

7. [ex-00-a02 · Attack · Advanced] Rewrite the signer's authorization as a typed, replay-resistant acceptance predicate.
8. [ex-00-r01 · Research · Research] Give one example where correctness holds but confidentiality fails.

Key Takeaways

- Start with assets and winning events.
- Correctness and security are different claims.
- Freshness and authorization require protocol state.
- Keys should be replaceable; algorithms should withstand disclosure.
- Residual risk belongs in the design record.

Primary References

- Auguste Kerckhoffs, La Cryptographie Militaire
- Jonathan Katz and Yehuda Lindell, Introduction to Modern Cryptography
- Dan Boneh and Victor Shoup, A Graduate Course in Applied Cryptography
- NIST Cryptographic Standards and Guidelines

Chapter 1 — Kryptos: From Puzzle Solving to a Modern Verifier

> Kryptos is useful because it separates finding an answer from proving, storing, and checking one.

Chapter Map

- Prerequisites: Chapter 0
- Core path: Puzzle workflow and keyed equality verification
- Advanced path: Canonicalization, custody, and online guessing
- Research path: Verifier governance and evidence boundaries
- Estimated core time: 60 min
- Estimated full time: 120 min
- Primary chapter type: Foundation

Learning Objectives

After completing this chapter, you should be able to:

- Decompose an artistic puzzle into data, models, scoring, search, and verification.
- Explain how classical cryptanalysis combines statistics with external context.
- Distinguish an unkeyed hash commitment, an HMAC-based verifier, and a zero-knowledge proof.
- Design an online equality-verification service that does not disclose the reference answer.

Why This Matters

The K4 custody and verification architecture connects classical search with hashing, MACs, KMS policy, and rate limits. A plausible plaintext is not a verified solution, and a verifier creates its own attack surface.

The dangerous shortcut is this: A keyed equality oracle is sometimes called zero knowledge even though it has neither the usual prover knowledge nor simulator guarantee. The remedy is to trace the claim from interface to adversary and then to deployment state.

Threat Model or Problem Statement

Problem boundary. A plausible plaintext is not a verified solution, and a verifier creates its own attack surface.

Model limit. The model does not cover custody operators, logging, canonicalization mismatch, denial of service, or the entropy of the answer space.

Core Concepts

1.1 Establishing the Factual Boundary

Kryptos was created by artist Jim Sanborn and unveiled at CIA headquarters in 1990. The sculpture contains several ciphertext passages and a modified Vigenère tableau. Its first three passages have been solved; K4 is a 97-letter passage.

Its status in 2026 must be stated with attribution. According to reporting by the Associated Press, researchers working with Smithsonian archival material located and reconstructed plaintext-related material in 2025, and Sanborn confirmed the archival finding. That is an archival plaintext discovery: it is not the same as deriving the plaintext by cryptanalysis, and it did not publish a reproducible reconstruction of the encryption method, a decryption procedure, or a key. The AP also reported the minimum relevant auction fact: archival and solution-related materials were sold in November 2025. This book draws no further conclusion about ownership, reproduction rights, or commercial permission from that sale.

In June 2026, Paradigm announced that it had become custodian of the K4 secret and described the passage as still lacking a publicly accepted cryptanalytic solution. The three claims must remain separate: someone may possess archival plaintext; the public may still lack a cryptanalytic derivation; and the encryption method may still lack a public, reproducible reconstruction. This chapter uses K4 only as a historical example of controlled verification and evidence boundaries. See the CIA description of the work, Paradigm's attributed 2026 announcement, and the Associated Press report.

The complete K4 ciphertext is available from official CIA or Paradigm pages. This release candidate does not reproduce the complete ciphertext or any archival plaintext. That editorial choice is not a legal conclusion about the scope of anyone's rights. After obtaining an official transcription for private study, use the following fingerprint to check that it was copied consistently:

Code `code-01-01` — Pseudocode / trace / format; not runtime-tested.

```
Alphabet: A-Z
Length: 97
SHA-256: eea813570c7f1fd3b34674e47b5c3da8948026f5cefee612a0b38ffaa515ceab
```

This book neither supplies nor speculates about the K4 plaintext. K4 serves only as a case study in data forensics, analytical method, and controlled answer verification.

1.2 Decoding Is Not a Single Algorithm

A rigorous cryptanalysis pipeline includes the following stages.

Step 1: Data Forensics

- Preserve the original characters, line breaks, question marks, positions, and source.
- Create a canonical working copy without overwriting the raw data.
- Record every transcription difference, known misspelling, and correction.
- Separate observed facts from solving hypotheses.

Step 2: Build Families of Candidate Models

Examples include:

- monoalphabetic substitution;
- polyalphabetic substitution;
- Vigenère or Beaufort variants;
- columnar transposition;
- route ciphers;
- fractionation;
- matrix or Hill-like transformations;
- layered combinations;
- homophonic substitution; and
- insertion, deletion, or position-dependent rules.

The presence of a tableau on the sculpture does not justify assuming that every section uses standard Vigenère directly.

Step 3: Extract Invariants and Statistics

- letter frequencies and repeated substrings;
- index of coincidence;
- periodic autocorrelation;
- n-gram scores;
- Kasiski distances;
- relative positions of repeated patterns;
- keystream constraints induced by known or guessed plaintext;

- self-mapping positions; and
- letter counts preserved under transposition.

Step 4: Define a Scoring Function

For a candidate English plaintext, one possible composite score is:

$$S(x) = w_1 \log P_{\text{4gram}}(x) + w_2 \log P_{\text{word}}(x) + w_3 S_{\text{crib}}(x) - w_4 S_{\text{violation}}(x).$$

A high language-model score is not a cryptographic proof. Short texts, proper nouns, deliberate misspellings, and mixed languages can all distort the score.

Step 5: Search

- Exhaustive search: the most reliable approach when the space is small enough.
- Hill climbing or simulated annealing: often useful for substitution and transposition.
- Genetic algorithms: capable of exploring mixed discrete structures, but vulnerable to noisy fitness functions.
- SAT/SMT or constraint programming: powerful when explicit constraints are available.
- Bayesian search: makes historical clues and model priors explicit.
- Human-machine collaboration: humans propose structure; machines enumerate and score.

Step 6: Falsification and Verification

A credible candidate should, at minimum:

- explain every character, not merely reveal an attractive fragment;
- use consistent rules, with externally supported exceptions;
- make correct predictions on clues that were not used for fitting;
- be reproducible by a third party from the raw data;
- avoid arbitrary post-hoc parameter adjustment; and
- exclude tied or near-equivalent candidates before claiming uniqueness.

1.3 A Research-Log Format

Record every experiment in a structured form:

Code `code-01-02` — Schema / configuration example; not runtime-tested.

```yaml
experiment_id: K4-2026-001
input_hash: SHA256(...)
source: Paradigm/CIA transcription
normalization: uppercase, A-Z only, no deletion
cipher_family: columnar_transposition + vigenere_variant
parameters:
  key_length: 8..14
  route: [row, spiral]
constraints:
  - plaintext[21:25] == "EAST"
score:
  tetragram_weight: 1.0
  crib_weight: 4.0
random_seed: 20260710
top_candidates:
  - score: ...
    plaintext: ...
result: rejected
reason: fails held-out positional constraint
```

This discipline limits researcher bias caused by preserving only the most visually appealing result.

1.4 The Cryptography of Paradigm's Verifier

Paradigm's 2026 description can be abstracted as follows:

1. Sanborn enters the reference answer m^* on an isolated device.
2. The device computes $h^* = \mathsf{SHA256}(m^*)$.
3. It sends h^* to an HMAC key k protected by a cloud KMS:

$$t^* = \mathsf{HMAC}_k(h^*).$$

4. The input device is wiped.
5. When a user submits a candidate m, the system computes the corresponding tag t and uses a constant-time comparison to test $t = t^*$.

Why Not Publish Only the SHA-256 Hash of the Answer?

If $h^* = \mathsf{SHA256}(m^*)$ is public, anyone can test candidates offline. Preimage resistance does not increase the entropy of a low-entropy answer. When the answer comes from a constrained natural-language space, dictionaries, templates, and contextual clues can reduce the guessing cost dramatically.

A secret HMAC key prevents an external attacker from computing verification tags offline. Candidates must instead pass through a controlled online oracle. Fees, rate limits, and monitoring can further reduce large-scale enumeration.

Why This Is Not a Zero-Knowledge Proof

This construction is an equality oracle backed by a secret reference tag. A zero-knowledge proof ordinarily requires a prover to establish a statement while the verifier learns no information about the witness beyond the truth of the statement, under formal definitions of completeness, soundness, and zero knowledge. "The server does not store readable plaintext but can compare submissions" is not, by itself, a zero-knowledge proof.

Canonicalization Must Be Defined

Should the following strings count as identical?

Code `code-01-03` — Pseudocode / trace / format; not runtime-tested.

```
BETWEEN SUBTLE SHADING...
BETWEENSUBTLESHADING
between subtle shading...
BETWEEN  SUBTLE  SHADING…
```

A secure, reproducible design fixes a specification such as:

Code `code-01-04` — Pseudocode / trace / format; not runtime-tested.

```
version       = "kryptos-k4-v1"
encoding      = UTF-8
normalization = Unicode NFC
case          = uppercase ASCII
whitespace    = remove or exact, explicitly chosen
punctuation   = explicit
length        = exact
```

It then applies domain separation:

$$h = \mathsf{SHA256}\big(\texttt{"KRYPTOS-K4-V1"} \parallel \mathsf{len}(m) \parallel m\big).$$

The length must use an unambiguous fixed-width or length-prefixed encoding to prevent concatenation ambiguity.

1.5 An Educational Verifier

The following code demonstrates the architecture. It is not Paradigm's implementation:

Code `code-01-05` — Library composition / educational; mapped to `book/code/01_ch01_snippet.py`.

```python
from __future__ import annotations

import hashlib
import hmac
import unicodedata

DOMAIN = b"KRYPTOS-K4-V1\x00"

def canonicalize(candidate: str) -> bytes:
    normalized = unicodedata.normalize("NFC", candidate)
    letters = "".join(ch for ch in normalized.upper() if "A" <= ch <= "Z")
    if len(letters) != 97:
        raise ValueError("canonical plaintext must contain exactly 97 A-Z letters")
    return letters.encode("ascii")

def digest_candidate(candidate: str) -> bytes:
    msg = canonicalize(candidate)
    return hashlib.sha256(DOMAIN + len(msg).to_bytes(4, "big") + msg).digest()

def verification_tag(kms_key_for_demo_only: bytes, candidate: str) -> bytes:
    digest = digest_candidate(candidate)
    return hmac.new(kms_key_for_demo_only, digest, hashlib.sha256).digest()

def verify(reference_tag: bytes, key: bytes, candidate: str) -> bool:
    candidate_tag = verification_tag(key, candidate)
    return hmac.compare_digest(reference_tag, candidate_tag)
```

In production, the HMAC key should never enter application memory. The service should invoke a MAC operation inside a KMS or HSM and restrict IAM permissions, request origins, rates, log contents, and administrator actions.

1.6 Residual Risks in the Verifier

- Abuse of KMS administrative privileges.
- Logging of candidate plaintexts or request bodies.
- A mismatch between canonicalization during reference creation and candidate verification.
- Large-scale automated querying of the equality oracle.
- Error messages that reveal how close a candidate is.
- Replacement of the reference tag.
- Supply-chain or deployment modifications that exfiltrate inputs.
- Leakage through cloud metadata, backups, crash dumps, or tracing.
- Observation of the answer by an insider during the initial ceremony.
- Reuse of an undomain-separated SHA-256 input across applications.

The cryptographic primitives solve only part of the problem. The overall assurance depends on the ceremony and operational controls.

Worked Example: Keyed equality versus zero knowledge

Goal. Classify two protocols that both reveal only accept/reject.

Inputs and assumptions. One server stores an answer-derived keyed tag; another verifier checks a proof of knowledge for a relation.

Steps.

1. Write each party's input and secret.
2. Describe the transcript and what an observer learns.
3. Check for a simulator, extractor, and offline-guessing path.

Result. The first system is a rate-limited keyed equality oracle; the second may be a zero-knowledge proof only under its stated relation and model.

What this establishes. Similar user-visible output can hide very different security definitions.

What this does not establish. It does not validate any candidate K4 plaintext or the operational claims of a live verifier.

Definition Gap

What the chapter's main definitions cover. Search, scoring, and verification are separate stages.

What remains outside the model. The model does not cover custody operators, logging, canonicalization mismatch, denial of service, or the entropy of the answer space.

Why the omitted property matters in deployment. A keyed verifier can confirm equality to a stored target without proving the target's historical truth, provenance, ownership, or publication rights; those claims require separate evidence and policy.

Engineering Notes

Never log submitted candidate plaintexts. An equality oracle can become an extremely well-instrumented guessing service if observability is designed carelessly.

Experiment: Build the K4 Analysis Data Layer

1. Obtain the K4 ciphertext from an official CIA or Paradigm page. Store it with the source URL, access date, and the SHA-256 fingerprint above in an immutable raw file.
2. Write a parser that verifies the presence of exactly 97 letters in A–Z.
3. Output letter frequencies, repeated bigrams and trigrams, the index of coincidence, and coincidence scores for every period from 2 through 20.
4. Preserve the parent hash and parameters for every transformation.
5. Prevent the program from overwriting the raw file.

End-of-Chapter Exercises

1. [ex-01-c01 · Concept · Core] Why should the verifier not compute `HMAC(k, normalize(m))` using a key shared with other services?
2. [ex-01-d01 · Derivation · Core] How does the attack surface change if the service returns "how many letters are correct" instead of a single yes/no result?
3. [ex-01-e01 · Engineering · Intermediate] If the KMS receives a SHA-256 digest rather than plaintext, under what prior information could it still infer the plaintext?
4. [ex-01-a01 · Attack · Intermediate] Design a dual-control ceremony in which no single operator can both observe the answer and exercise the KMS key.
5. [ex-01-p01 · Proof · Advanced] Explain the similarities and differences between this verifier and a commitment scheme.
6. [ex-01-e02 · Engineering · Intermediate] Write an ambiguous encoding scheme that would cause a canonicalization collision.

7. [ex-01-a02 · Attack · Advanced] Explain how rate limits change online guessing cost without increasing answer entropy.
8. [ex-01-r01 · Research · Research] Write a canonicalization differential test for two implementations.

Key Takeaways

- Search, scoring, and verification are separate stages.
- A public digest enables offline guessing over a small answer space.
- Canonicalization is cryptographic protocol data.
- KMS custody does not remove application-layer oracles.
- Historical status claims require dated primary evidence.

Primary References

- Paradigm, The History of Kryptos
- Paradigm, Project Kryptos
- Paradigm, K4 Challenge
- CIA, “Kryptos” Sculpture
- Associated Press, report on the 2025 archival discovery and November auction
- RFC 2104, HMAC
- FIPS 180-4, Secure Hash Standard

The linked topic is available in the complete edition.