

Scratchy: Visual-Scratchpad Multimodal Reasoning for Cryptographic Proof Generation in EasyCrypt

Yupeng Ren^{1,2}, Zhaoxuan Li^{1,2}, Rui Zhang^{1,2}

¹State Key Laboratory of Cyberspace Security Defense,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing 100085, China

²School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
renyupeng24@mails.ucas.ac.cn
{lizhaoxuan,zhangrui}@iie.ac.cn

Abstract

Large language models (LLMs) have recently made substantial progress in formal proof generation, yet presenting distinctive challenges in cryptographic area. Computational security arguments posit that a valid proof must coordinate probability, adversarial games, invariants, assumptions and bounds, which can be provided by a machine-checked framework named EasyCrypt. Although all objects may appear in available context, LLMs still struggle because proof-theoretic dependencies are typically implicit in a linear representation and distributed across multiple programs. So, this paper presents **Scratchy**, a visual-scratchpad approach that exposes these dependencies for multimodal generation. Given the natural-language security description, with formal context and target propositions, the proof objects can be normalized into a typed proof-relation graph. Then a structure-preserving visual compiler transforms the graph into the formula-rich visual proof state that guides a multimodal model in generating the EasyCrypt proof. Also, the **Scratchy-eval**, a 114-task dataset derived from reliable official EasyCrypt files, has been introduced. It contains 64 security-form proof generations and 50 multiple-choice knowledge tests. After a series of evaluations, covering semantic grounding, relational invariants, and game reductions, classical LLMs like GPT-5.6-Sol and Claude-Opus-5 have gained a clear advantage from Scratchy’s structured visual proof states. This contrast suggests that explicit proof structure can make the improvement and multimodal proof-state representation as a promising direction for computer-aided cryptography.

1 Introduction

With the rapid development of Large Language Models (LLMs), formal proof generation has become an important task for machine reasoning. Recently, researchers have expanded formal supervision through synthetic natural language, including

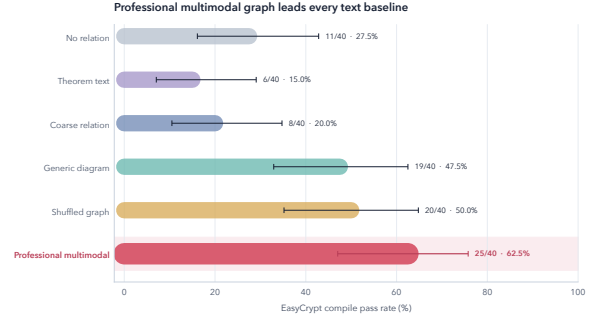


Figure 1: Results of the API-based EasyCrypt proof-generation Observation across six representation conditions. The professional multimodal graph achieves the highest compile pass rate (62.5%, 25/40), outperforming all baselines.

Lean corpora, self-play that alternates conjecturing and proving, and verifier-guided repair of generated proofs (Ying et al., 2024; Dong and Ma, 2025; Ospanov et al., 2025). These advances show that formal libraries and prover feedback can effectively support proof generation. However, the dominant setting remains mathematical theorem proving, and the interface between the model and the prover is still primarily composed of serialized theorem statements, proof code, candidate premises, and diagnostic messages.

Cryptographic proofs have a different structure. A computational security argument must coordinate probabilistic programs, games, adversarial capabilities, invariants, security assumptions, and concrete advantage bounds. Correctness depends not only on whether these objects are available, but also on the direction and order required by the security reduction. Foundational work on machine-checked cryptography developed CertiCrypt for code-based game proofs and subsequently introduced EasyCrypt to elaborate machine-checkable security proofs from game sketches (Barthe et al., 2009, 2011). These systems established that program transformations, relational judgments, and

Table 1: Notation expolited in the Scratchy pipeline and the motivating Observation.

(a) Scratchy proof objects and verification		(b) Observation metrics and routing	
Symbol	Meaning	Symbol	Meaning
$\mathcal{X}, x$	Security setting and task.	$\mathcal{T}, T, t$	Task set, size, and index.
$\mathcal{C}, \Gamma$	Local context and environment.	$\mathcal{M}, M, m$	Model set, size, and index.
ϕ	Target obligation.	$\mathcal{M}_{\text{API}}$	Hosted-model pool.
q	Structured Natural-Language Proof Sketch.	$\mathcal{K}, K, c$	Condition set, size, and index.
$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \lambda)$	Typed relation graph.	$\mathcal{C}_{\text{NR}}, \mathcal{C}_{\text{TH}}, \mathcal{C}_{\text{CR}}$	Text conditions.
$\mathcal{V}, \mathcal{E}$	Graph nodes and edges.	$\mathcal{C}_{\text{GD}}, \mathcal{C}_{\text{SG}}, \mathcal{C}_{\text{PG}}$	Graph conditions.
τ, λ	Type and label maps.	$\mathcal{K}_{\text{text}}$	Text-condition set.
$\mathcal{R}$	Visual compiler.	$\mathcal{K}_{\text{vctl}}$	Visual-control set.
$I_{\mathcal{G}} = \mathcal{R}(\mathcal{G})$	Visual proof state.	y_{mtc}	Compile outcome.
G_i, G_L, G_R	Hybrid and paired games.	$S_{\text{mc}} = \sum_t y_{\text{mtc}}$	Compile count.
v_i^L, v_i^R	Aligned state values.	$\hat{p}_c$	Pooled pass rate.
r_i	Transition cue.	$[L_c, U_c]$	Wilson interval.
Δ_{ij}	Game distance.	$c_m^{\text{text}}, c_m^{\text{vctl}}$	Best text/visual controls.
ϵ_i	Game-hop bound.	$\delta_m^{\text{text}}, \delta_m^{\text{vctl}}$	Professional-condition gains.
$\pi, \hat{\pi}$	Proof fragments.	z_{mt}	Outcome class.
p_{θ}, θ	Generator and parameters.	ℓ	Language-layer index.
$S[\hat{\pi}]$	Reconstructed source.	$H_{\ell}^{(c)}, \Delta H_{\ell}$	Routing entropy/change.
$\mathcal{V}_{\text{EC}}$	EasyCrypt verifier.	$U_{\ell}^{(c)}, \Delta U_{\ell}$	Expert utilization/change.
$\top, \perp$	Verification outcomes.	D_{JS}	Routing divergence.
$\Pr[G_i \Rightarrow 1], \mathcal{I}$	Probability and invariant.	$N_{\text{img}}, A_{\text{vis}}$	Image tokens/activation.

concrete security bounds can be rigorously checked within a formal framework. Researchers further proposed EasyCrypt proof workflow and demonstrated representative applications to cryptographic standards and post-quantum constructions (Barthe et al., 2012; Almeida et al., 2019; Barbosa et al., 2021).

Despite the relational of cryptographic proofs, recent proving pipelines still use textual or symbolic sequences as their primary interface. They construct natural-language with formal corpora to improve training, search, and feedback (Ying et al., 2024; Dong and Ma, 2025; Wang et al., 2025; Ospanov et al., 2025). However, these works do not present the distributed relationships within a proof to the generator as a unified visual state. A linear interface preserves symbolic content, but it does not naturally expose object types. Consequently, the relationships between the direction of a game hop and multiple dependency paths running from local obligations to the security goal cannot be simultaneously represented.

Multimodal images provide a representation channel that remains underexplored. Researchers have begun to treat images as intermediate reasoning states. Visual Sketchpad allows a model to draw visual marks, GraphVis preserves graph structure through the visual modality, and MIRA evaluates problems that require intermediate visual cues (Hu et al., 2024; Deng et al., 2024; Zhou et al., 2026). In an adjacent formalization setting, LeanEuclid studies diagram-dependent Euclidean geometry with GPT-4V, but targets mathematical aut-

oformalization rather than machine-checked cryptographic security proofs (Murphy et al., 2024). Thus, whether task-specific images can improve machine-checkable security-proof generation, and which visual proof state is most effective, are still worth diving into.

A preliminary observation indicates that task-specific, structured proof diagrams can improve EasyCrypt proof generation across multimodal models, whereas generic or structurally corrupted diagrams provide weaker benefits. This finding suggests that the gain depends not only on the visual modality itself but also on the faithful organization of proof objects and their relations, motivating the design of Scratchy.

To sum up, this paper proposes **Scratchy**. While the security attributes are received, within local formal context, target obligation, and proof guidance, the distributed objects can be normalized into a typed proof-relation graph. Its nodes distinguish threat models, program facts, proof obligations, rules, and security goals. At the same time, the constraint, transformation, and discharge relations are supported by directed edges encoding. The graph specifies the semantic structure that a visual representation must preserve, turning the search for an effective image into a constrained proof-state design problem.

As shown in Section. 4, a structure-preserving visual compiler renders the relation graph as a formula-rich visual scratchpad. Guided by different states, the multimodal model performs program unfolding, state alignment, rule application, and goal

discharge to generate an EasyCrypt proof fragment. The fragment then enters an EasyCrypt verification loop for parsing and type checking, obligation discharge, and target closure.

For systematic evaluation, this paper further introduces **Scratchy-eval**. The benchmark derives 114 tasks from 26 source groups and 37 pinned official EasyCrypt files. It contains 64 free-form proof generation tasks and 50 multiple-choice tasks on semantic recognition. The tasks are evaluated under six paired representation conditions. Compilation under EasyCrypt is the primary criterion for generation.

The main contributions of this work are as follows:

Empirical Finding

This paper provides empirical evidence that structured visual proof states can improve EasyCrypt security-proof generation and that relation fidelity matters beyond image presence alone.

Scratchy

This paper introduces Scratchy, which treats visual-representation selection as a constrained proof-state design problem and realizes it through typed proof-relation graph, structure-preserving visual compiler, and an EasyCrypt verification loop.

Scratchy-eval

This paper contributes Scratchy-eval, a benchmark with 114 tasks and six paired conditions for jointly evaluating proof generation, static validity, and semantic recognition.

2 Related work

2.1 Machine-Checked Cryptographic Proofs

Machine-checked cryptography regards security arguments as formally representable and verifiable objects, which has led to the development of CertiCrypt and EasyCrypt (Barthe et al., 2009, 2011). Subsequently, studies have further developed dedicated cryptographic-proof workflows and extensions such as EasyPQC (Barthe et al., 2012; Almeida et al., 2019; Barbosa et al., 2021). With program transformations, relational judgments, and probabilistic bounds, these tools provide strong support for formal cryptographic proofs and offer a correctness foundation for the generating results.

2.2 Language Models for Cryptographic Formalization

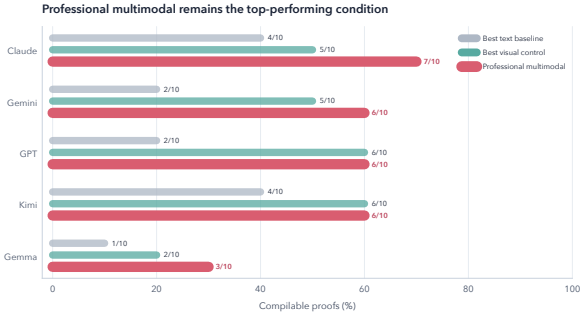
CrypFormBench evaluates language models on interpretation, generation, completion, transformation, and correction across cryptographic formal-verifier languages (Li et al., 2026). ShannonProver more directly targets EasyCrypt, constructing proof scripts from a supplied security model and lemma-level decomposition (Ma et al., 2026). Differently, this paper pays attention to another complementary bottleneck, which is compiling distributed formal context into a structured visual state and studies whether image presence and relation fidelity affect machine-checkable proof generation.

2.3 Language Models for Formal Proving

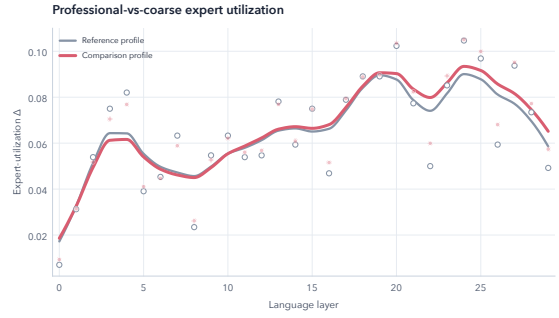
Recently, researcher expands supervision and evaluation through synthetic Lean corpora and difficult multilingual theorem benchmarks (Ying et al., 2024; Tsoukalas et al., 2024). Proof search has been improved through conjecturer-prover self-play (Dong and Ma, 2025), proof-assistant-guided reinforcement learning and tree search (Xin et al., 2025), interleaved informal reasoning and formal tactics (Lin et al., 2025), scaffolded synthesis with self-correction (Lin et al., 2026), and structure-aware premise selection (Wang et al., 2025). Verifier-integrated systems further repair Lean proofs from compiler feedback with Isabelle theorem proving (Ospanov et al., 2025; Lin et al., 2024). Although these methods improve data, search, and feedback, their model-facing states remain predominantly textual or symbolic and mainly target mathematical theorems or general program verification.

2.4 Visual Intermediate Reasoning

Multimodal reasoning increasingly treats images as intermediate computational states. Visualization-of-Thought, Vision-Augmented Prompting, Visual Sketchpad, and CogCoM respectively render reasoning traces, synthesize and refine visual cues, reuse drawn marks, or expose traceable visual manipulations (Wu et al., 2024; Xiao et al., 2024; Hu et al., 2024; Qi et al., 2025). GraphVis and GITA further show that visual encodings can preserve graph connectivity and that graph layout affects reasoning (Deng et al., 2024; Wei et al., 2024). Visual CoT and MIRA evaluate localized or indispensable intermediate visual evidence, while LeanEuclid studies diagram-dependent mathematical



(a) Model-wise comparison



(b) Expert-utilization profiles (Gemma-4-26B-A4B)

Figure 2: Complementary views of the Observation. The gray curve is the observed reference profile, and the rose curve is a comparison profile anchored to it. Models involve Claude-opus-5, Gemini-3.7-flash, Kimi-k3, GPT-5.6-sol and Gemma-4-26B-A4B.

autoformalization (Shao et al., 2024; Zhou et al., 2026; Murphy et al., 2024). However, it is still a challenge for these approaches compiles to type cryptographic proof relations into a formula-rich image whose generated proof is judged by EasyCrypt. Continuously, **Scratchy** treats relation fidelity as the central visual-design variable, which provides a good feasible solution.

3 Observation

Instructively, this paper examines a narrower question, **whether the representation of available proof relations can affect EasyCrypt proof generation with LLMs**.

This observation uses ten paired EasyCrypt obligations (Table 2). Four hosted multimodal models form the behavioral summary, while a frozen local Gemma-4-26B-A4B supplies an internal diagnostic. The six conditions include no-relation, theorem-only, and coarse-relation text; a generic diagram and a shuffled-edge graph as visual controls; and a complete professional graph (Scratchy). Within a paired run, reference fragments are hidden from the model, and a generated fragment is counted as successful only when it compiles under the pinned EasyCrypt environment.

The aggregate API summary gives the professional graph 25/40 compilable fragments (62.5%), compared with 20/40 (50.0%) for the strongest visual control and 11/40 (27.5%) for the strongest text condition (Fig. 1). The model-wise view shows that professional graph is consistently above text condition, whereas its additional margin over the best visual control is smaller and varies by model (Fig. 2(a)). The contrast suggests that visual organization may provide a broad benefit, while faithful encoding of proof relations can supply a narrower,

Table 2: Sources of the ten paired Observation samples. All files are drawn from the pinned official EasyCrypt repository snapshot (EasyCrypt Development Team, 2026).

ID	Focus	Proof object	Official source
01	Semantic	Well-foundedness	WF-examp.ec
02	Semantic	Dice point mass	Dice4_6.ec
03	Semantic	While termination	WhileSampling.ec
04	Relational	Async invariant	async-while.ec
05	Relational	Bad-event split	FundamentalLemma.ec
06	Reduction	Pedersen hiding	Pedersen.ec
07	Reduction	Schnorr soundness	SchnorrPK.ec
08	Reduction	ElGamal-DDH hop	elgamal.ec
09	Relational	PIR coupling	PIR.ec
10	Reduction	Stateful PRG bound	PRG.ec

dependent contribution.

The local model provides a complementary signal about whether the visual input is computationally engaged. This paper examines the layerwise difference $\Delta U_\ell = U_\ell^{(c_{PG})} - U_\ell^{(c_{CR})}$ between the professional graph and coarse-relation text. The recorded reference profile and the compared profile remain close and positive across all 30 language layers (Fig. 2(b)). The trajectory changes across depth rather than appearing as a single constant offset, indicating that the representation condition is associated with a structured change in expert usage.

Therefore, the Observation shows that the representation of proof relations is itself an important part of the generation process. The performance differences across conditions indicate that structured visual states can help coordinate distributed proof objects, while the layerwise changes in expert utilization show that the model responds to this organization across network depth. These findings motivate moving beyond manually selected

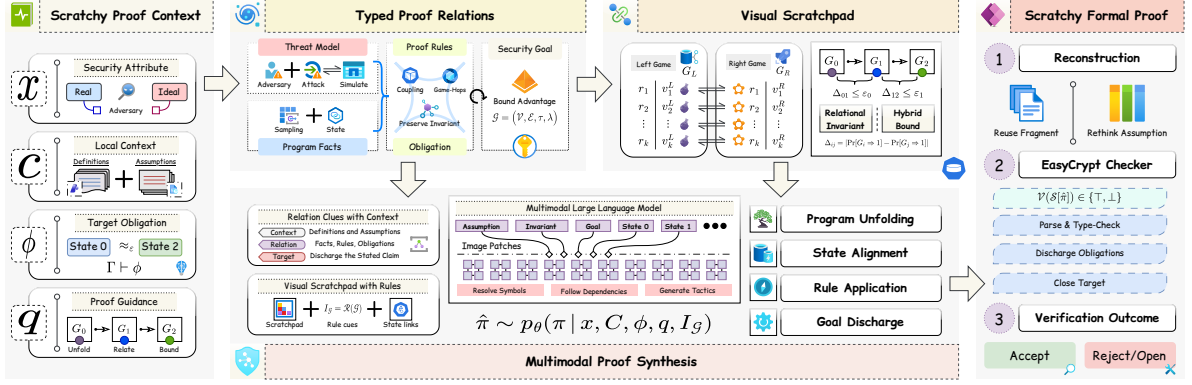


Figure 3: The Scratchy pipeline. Structured proof context and an SNPS are deterministically compiled into a typed proof-relation graph and a formula-rich visual scratchpad. A multimodal model follows the explicit dependencies to synthesize an EasyCrypt proof fragment, which is reconstructed and checked for acceptance or an open outcome.

diagrams toward a systematic mechanism for constructing visual proof states. Accordingly, this paper introduces **Scratchy** to address these challenges, which organizes security assumptions, program states, proof obligations, and reduction goals into a structured visual interface for multimodal EasyCrypt proof generation.

4 Methodology

In **Scratchy**, a **Structured Natural-Language Proof Sketch (SNPS)** is taken as the input to visual composition. The SNPS expresses proof objects and directed relations through fixed fields, allowing relation extraction and image construction to proceed deterministically. And LLMs enter after the visual proof state has been constructed, where it generates the EasyCrypt fragment. Fig. 3 presents the full pipeline, and Table 1 defines the notation.

4.1 Unified Input and Output

The unified input consists of the security setting $\mathcal{X}$, task description x , local EasyCrypt corpus $\mathcal{C}$, formal environment Γ , target obligation ϕ , and SNPS q . The SNPS contains six fields, including **THREAT MODEL**, **PROGRAM FACTS**, **PROOF OBLIGATIONS**, **PROOF RULES**, **SECURITY GOAL**, and **RELATIONS**. The first five fields declare proof objects, **RELATIONS** declares the source, target, and meaning of directed dependency. Identifiers and formulas in q are written exactly as they occur in $\mathcal{C}$, Γ , or ϕ .

The same interface is used throughout the pipeline:

$$(\mathcal{X}, x, \mathcal{C}, \Gamma, \phi, q) \longrightarrow \mathcal{G} \longrightarrow I_{\mathcal{G}}, \quad (1)$$

$$I_{\mathcal{G}} \longrightarrow \hat{\pi} \longrightarrow \mathcal{V}_{\text{EC}}(\mathcal{S}[\hat{\pi}]). \quad (2)$$

Algorithm 1 Deterministic SNPS-to-Graph Compilation

Require: q grounded in $(\mathcal{X}, x, \mathcal{C}, \Gamma, \phi)$
Ensure: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \lambda)$
1: Parse the six ordered fields of q
2: **for all** proof objects declared in q **do**
3: Add a node to $\mathcal{V}$; assign τ and λ
4: **end for**
5: **for all** source-to-target relations in q **do**
6: Add the corresponding directed edge to $\mathcal{E}$
7: **end for**
8: Order dependency paths toward ϕ
9: **return** $\mathcal{G}$

Each output is therefore the direct input to the next operation, preserving a traceable route from the SNPS to the verification outcome.

4.2 SNPS-to-Graph Compilation

The first key step compiles the SNPS into the typed proof-relation graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \lambda). \quad (3)$$

Each SNPS item becomes a node in $\mathcal{V}$. Its field uniquely determines the node type recorded by τ , while λ retains its natural-language description, EasyCrypt identifier, and formula. Each entry in **RELATIONS** becomes a directed edge in $\mathcal{E}$. The compiler consequently performs parsing, matching, and normalization without inferring relations absent from the SNPS.

The two deterministic procedures are given in Algorithms 1 and 2.

There are distinct roles for five node types. A threat-model node states the security consequence addressed by the proof; program-fact nodes provide available evidence; proof-obligation nodes state intermediate claims; proof-rule nodes describe permitted transformations; and the security-goal node

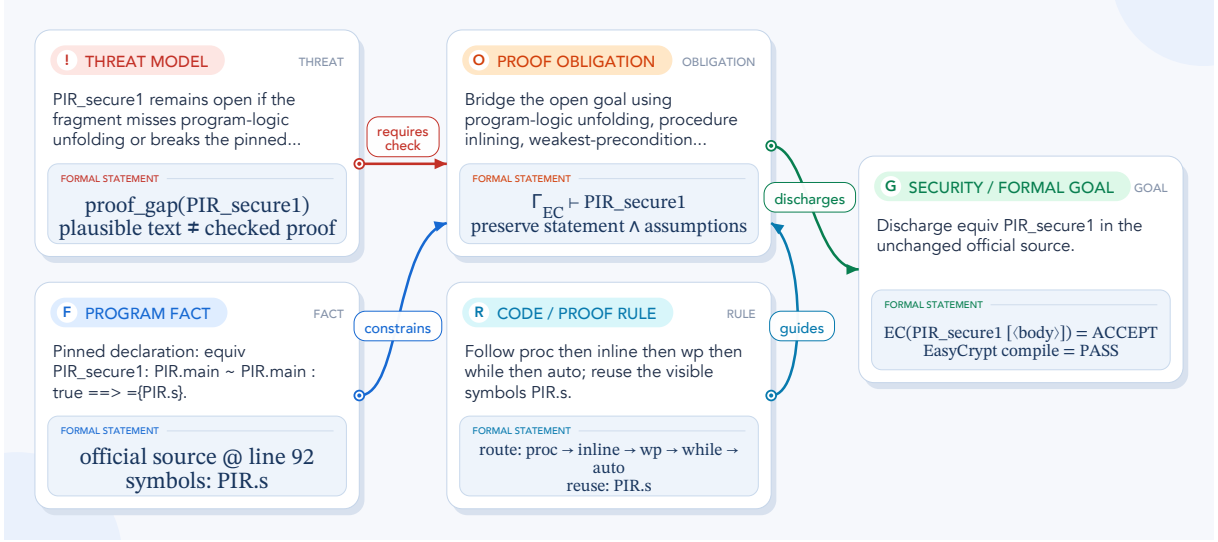


Figure 4: Professional visual proof state for a PIR relational obligation. The graph organizes source facts, the target, a proof-rule route, and the machine-checkable goal into a directed path.

Algorithm 2 Structure-Preserving Visual Compilation

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \lambda)$

Ensure: $I_{\mathcal{G}}$

- 1: Arrange nodes by the roles recorded in τ
- 2: Render the text, formulas, and code recorded in λ
- 3: **for all** directed edges in $\mathcal{E}$ **do**
- 4: Draw its tail, arrowhead, and relation label
- 5: **end for**
- 6: Highlight the dependency paths ending at ϕ
- 7: **return** $I_{\mathcal{G}} = \mathcal{R}(\mathcal{G})$

corresponds to ϕ . Edges point from a justification to the claim it supports. And both proof order and dependency are recorded by the complete path.

4.3 Graph-to-Visual Compilation

The second key step applies visual compiler $\mathcal{R}$:

$$I_{\mathcal{G}} = \mathcal{R}(\mathcal{G}). \quad (4)$$

Structure preservation means that node roles, semantic labels, edge directions, and the path to ϕ remain directly readable in the output. Role-specific colors and icons encode τ ; formula and code cards display λ ; and tail markers, arrowheads, and relation labels encode $\mathcal{E}$. a stable, robust, and professional layout is constructed.

For relational proofs, $I_{\mathcal{G}}$ places G_L and G_R in parallel and connects v_i^L , r_i , and v_i^R at corresponding program points. For game-based proofs, it orders G_i along the reduction and displays next to the relevant hop.

$$\Delta_{ij} = |\Pr[G_i \Rightarrow 1] - \Pr[G_j \Rightarrow 1]|, \quad \Delta_{ij} \leq \epsilon_i \quad (5)$$

The relational invariant $\mathcal{I}$ is placed beside the state correspondence it constrains.

```

module PIR = {
  var s, s' : int list
  proc main(i : int) = {
    var j <- 0;
    (s, s') <- ([], []);
    while (j < N) {
      (* sample and update s, s' *)
      j <- j + 1;
    }
  }
}

equiv PIR_secure1:
  PIR.main ~ PIR.main : true ==> ={PIR.s}.
proof.
  proc; inline *; wp.
  while (= {j, PIR.s}); auto.
qed.

```

Listing 1: Abridged EasyCrypt source for the PIR example. The relational target and proof fragment instantiate the SNPS route encoded by the professional graph.

For example, Fig. 4 illustrates the resulting visual proof state with a representative PIR obligation. The corresponding EasyCrypt excerpt is shown separately in Listing 1.

4.4 Proof Synthesis and Verification

After $I_{\mathcal{G}}$ has been constructed, the multimodal generator reads both the textual context and the image:

$$\hat{\pi} \sim p_{\theta}(\pi \mid x, \mathcal{C}, \phi, q, I_{\mathcal{G}}). \quad (6)$$

The graph route organizes generation into program unfolding, state alignment, rule application, and goal discharge. Text preserves exact EasyCrypt syntax, while $I_{\mathcal{G}}$ preserves the relation structure needed to produce $\hat{\pi}$.

Finally, Scratchy inserts $\hat{\pi}$ into its original location to form $\mathcal{S}[\hat{\pi}]$. EasyCrypt parses and type-checks the reconstructed source, discharges its obligations, and closes the target:

$$V_{\text{EC}}(\mathcal{S}[\hat{\pi}]) \in \{\top, \perp\}. \quad (7)$$

An outcome of $\top$ accepts the fragment, while $\perp$ returns the open proof state to reconstruction. Algorithms 1 and 2 remain fully deterministic. The LLM is used only for the transition from I_G to $\hat{\pi}$.

5 Experiments

5.1 Scratchy-eval

Scratchy-eval is a provenance-preserving benchmark constructed from the EasyCrypt corpus (EasyCrypt Development Team, 2026). It contains 114 tasks drawn from 26 protocol or formal-program sources and 37 official files (13,761 lines). 64 tasks require free-form EasyCrypt proof generation, while 50 multiple-choice tasks cover static validity and semantic recognition.

The generation set comprises 55 lemmas, seven equivalence judgments, and two expectation-Hoare judgments. Each generation fixture preserves the imports and dependency prefix of an official target but replaces its proof. Static-validity items pair an official statement with syntactically defective alternatives, and semantic items test the formal role of local constructs. Appendix A provides the Data Card and summarizes source coverage and construction.

5.2 Experimental Setup

This paper evaluates Claude Opus 5, Gemini 3.7 Flash, GPT-5.6-Sol, and Kimi K3. Each model produces one response for every task under six paired representation conditions. The complete condition receives the local source, task-specific professional graph, and EasyCrypt decoding card; the other conditions remove source details, proof paths, identifiers, or edge structure to test which parts of the visual proof state contribute to generation.

Each model run contains 684 units, giving 2,736 model task condition units in total. The task, response contract, and output budget remain fixed within each six-way comparison, while condition order is deterministically rotated by task. Before any model call, 164 fixtures are dynamically checked: the 64 hidden reference proofs and four isolated options for each of the 25 static-validity tasks.



Figure 5: Generation pass rates across seven source families, pooled over four models. Each vertical group compares CR (coarse proof-relation text), SG (shuffled-edge graph), and PG (Scratchy professional graph); bar height and top labels report the percentage of fragments accepted by EasyCrypt. PG leads on the four cryptography and reduction-oriented families, SG leads on foundations and program analysis, and the two graph conditions tie on program logics.

The primary generation metric is *compile pass rate*: a fragment must parse, type-check, discharge its obligations, and close the target in the pinned EasyCrypt environment. Exact agreement with the official fragment is only diagnostic because several valid proofs may establish the same target. Multiple-choice tasks use accuracy, with static-validity selections independently confirmed by compilation. Paired confidence intervals are obtained by bootstrap resampling the 26 protocol or formal-program sources, preserving dependencies among tasks derived from the same source.

5.3 Main Results

Table 3 shows a consistent model-level result: PG is the strongest generation condition for all four models. Pooled over models, it compiles 54 of 256 fragments (21.1%), compared with 24 (9.4%) under SG and 11 (4.3%) under CR. The complete representation therefore reaches $2.25\times$ the SG rate and $4.91\times$ the CR rate. PG ranges from 14.1% on Claude to 28.1% on Gemini, showing that the ordering is stable even though absolute proof ability differs across models.

The multiple-choice tasks remain (97.0–99.5%), making free-form generation the principal source of discrimination. Among the 54 compiling PG outputs, only 10 exactly match the official fragment.

Table 3: Scratchy-eval results under six representation conditions. Each numeric entry is passed/total (%). NR: no relation knowledge; TH: theorem-only text; CR: coarse proof-relation text; GD: generic diagram; SG: shuffled-edge graph; PG: Scratchy professional graph.

Model or task set	NR	TH	CR	GD	SG	PG
☀ Claude Opus 5	0/64 (0.0)	3/64 (4.7)	5/64 (7.8)	1/64 (1.6)	1/64 (1.6)	9/64 (14.1)
🔹 Gemini 3.7 Flash	2/64 (3.1)	2/64 (3.1)	3/64 (4.7)	9/64 (14.1)	9/64 (14.1)	18/64 (28.1)
🌀 GPT-5.6-Sol	2/64 (3.1)	3/64 (4.7)	2/64 (3.1)	4/64 (6.2)	6/64 (9.4)	12/64 (18.8)
📄 Kimi K3	1/64 (1.6)	1/64 (1.6)	1/64 (1.6)	5/64 (7.8)	8/64 (12.5)	15/64 (23.4)
Pooled generation	5/256 (2.0)	9/256 (3.5)	11/256 (4.3)	19/256 (7.4)	24/256 (9.4)	54/256 (21.1)
Pooled MCQ	194/200 (97.0)	199/200 (99.5)	198/200 (99.0)	198/200 (99.0)	199/200 (99.5)	199/200 (99.5)
Overall	199/456 (43.6)	208/456 (45.6)	209/456 (45.8)	217/456 (47.6)	223/456 (48.9)	253/456 (55.5)

Table 4: Paired generation comparison over 256 model-task pairs. PG: Scratchy professional graph; CR: coarse proof-relation text; SG: shuffled-edge graph; W/L/T: PG wins/losses/ties; pp: percentage points; CI: source-group bootstrap 95% confidence interval.

Compared with	W/L/T	Gain (pp)	95% CI
CR	53/10/193	+16.8	[8.8, 24.6]
SG	39/9/208	+11.7	[3.5, 19.8]

The remaining 44 are textually different proofs accepted by EasyCrypt, which would be missed by reference-only scoring.

5.4 Protocol and Proof-Structure Analysis

Figure 5 locates the gain precisely. PG reaches 33.3% on complex cryptography and 21.4% on reduction paths. Generic visual organization can be sufficient for more local program facts, whereas the largest professional-graph advantage appears when a target depends on a protocol-specific reduction path.

At the individual-source level, PG compiles 9/20 model-target pairs for two-server PIR, 7/16 for UC Diffie-Hellman encryption, 5/16 for syntactic up-to reductions, and 5/16 for the PRG tutorial. It also obtains 3/12 on Pedersen commitment where both CR and SG obtain zero. The broader source coverage behind these counts is summarized in Table 5.

The same pattern appears across formal judgment types. PG compiles 49/220 lemma pairs (22.3%), compared with 23/220 for SG (10.5%) and 10/220 for CR (4.5%). For relational equiv targets, PG reaches 5/28 (17.9%), whereas both SG and CR reach 1/28 (3.6%). None of the eight model-task pairs formed by the two ehoare targets compiles under any condition.

The paired comparison in Table 4 controls for both model and target. Against CR, PG turns 53

failures into passes while losing 10 previously passing pairs; against SG, the corresponding counts are 39 and nine. Both source-group intervals remain above zero. Moreover, 36 of the 54 PG successes are not reproduced by any of the five ablations for the same model and target. Fourteen are shared with a visual control, two with a text condition, and two with both families. The aggregate improvement is thus composed mainly of new machine-accepted proofs, rather than repeated successes on tasks already handled by simpler representations.

5.5 Failure and Efficiency Analysis

Among the unsuccessful PG generation units, the largest recurring groups are parse errors (30), in-applicable rewrites reported as **nothing to rewrite** (30), incomplete proofs (19), and by-blocks that cannot close their goals (11). Together they account for 90 failures (44.6%). The remaining bottleneck is therefore often the local instantiation of the proof route in exact EasyCrypt syntax.

6 Discussion

6.1 A Machine-Checked Proof Contrast

Task 064 provides a concrete view of what changes between a successful and an unsuccessful generation. The target is the relational judgment shown in Figure 4. Listing 2 compares the official fragment with two outputs from GPT-5.6-Sol for this same target. The official fragment and the professional-graph output are accepted by EasyCrypt.

All three fragments begin with the same local route, including expose the paired procedures with proc, unfold calls with inline, and propagate postconditions with wp. The decisive step is the loop. The rejected output invokes autoimmediately and never states what must remain synchronized across iterations.

In contrast, the professional-graph output aligns

```

(* Official reference: accepted *)
proc; inline *; wp.
while (={j,PIR.s}); auto.

(* GPT + PG: accepted *)
proc; inline *; wp;
while (={j} /\ ={PIR.s}).
  wp; rnd; skip; auto.
auto.

(* GPT + CR: rejected *)
by
  proc;
  inline *;
  wp;
  auto.

```

Listing 2: Machine-checked contrast for the PIR target. PG denotes the professional graph and CR denotes coarse relation text. The official and PG fragments compile, whereas CR omits the loop invariant and leaves an open goal.

the random assignment with `rnd`, and discharges the loop body before closing the residual goal. Unlike the official proof, GPT produces the equivalent conjunction and an explicit body proof. Machine checking therefore identifies a valid alternative that textual exact match would mark as different.

6.2 Implications for Scratchy

This example reflects the aggregate pattern. The professional graph is strongest for all four models and yields its clearest gains on protocol-specific reductions and relational judgments. For Task 064, that route makes the preserved state and the loop rule visible at the point where a linear hint remains underspecified. The result supports Scratchy’s central design which compiling an SNPS into a relation-faithful visual state before generation.

Many remaining failures occur when a plausible route must be instantiated as exact syntax, especially around rewrites, side goals, and program logic tactics. This suggests that visual compiler should continue to expose proof structure, while the decoder and verification loop should become more precise at converting that structure into locally valid EasyCrypt steps. Extending verifier-guided revision at this final boundary is likely to improve proof completion without changing the deterministic visual construction.

7 Conclusion

This paper presents Scratchy, a visual-scratchpad framework for machine-checkable cryptographic proof generation. Scratchy deterministically compiles a Structured Natural-Language Proof Sketch

into a typed proof-relation graph and a formula-rich visual state, then guides a multimodal model to produce an EasyCrypt fragment whose correctness is decided by the formal verifier. Across Scratchy-eval, the professional graph consistently improves generation over textual and visual-control conditions, with particularly useful gains for relational and reduction-oriented proofs.

Future work will extend Scratchy-eval to additional protocols and proof systems, refine SNPS authoring and visual compilation, and use verifier feedback for targeted repair of syntax and residual obligations. A further direction is to study which visual relations are most useful for different proof families, enabling proof states that remain concise while adapting their structure to the target judgment.

References

- José Bacelar Almeida, Cécile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. 2019. [Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3](#). In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1607–1622. ACM.
- Manuel Barbosa, Gilles Barthe, Xiong Fan, Benjamin Grégoire, Shih-Han Hung, Jonathan Katz, Pierre-Yves Strub, Xiaodi Wu, and Li Zhou. 2021. [EasyPQC: Verifying post-quantum cryptography](#). In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2564–2586. ACM.
- Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin. 2012. [Computer-aided cryptographic proofs](#). In *Interactive Theorem Proving*, volume 7406 of *Lecture Notes in Computer Science*, pages 11–27. Springer.
- Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. 2011. [Computer-aided security proofs for the working cryptographer](#). In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer.
- Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. [Formal certification of code-based cryptographic proofs](#). In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 90–101. ACM.
- Yihe Deng, Chenchen Ye, Zijie Huang, Mingyu Derek Ma, Yiwen Kou, and Wei Wang. 2024. [GraphVis:](#)

- Boosting LLMs with visual knowledge graph integration. In *Advances in Neural Information Processing Systems*, volume 37, pages 67511–67534. Curran Associates, Inc.
- Kefan Dong and Tengyu Ma. 2025. *STP: Self-play LLM theorem provers with iterative conjecturing and proving*. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 14114–14136. PMLR.
- EasyCrypt Development Team. 2026. EasyCrypt: Computer-aided cryptographic proofs. GitHub repository.
- Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Ranjay Krishna. 2024. *Visual sketchpad: Sketching as a visual chain of thought for multimodal language models*. In *Advances in Neural Information Processing Systems*, volume 37, pages 139348–139379. Curran Associates, Inc.
- Zhaoxuan Li, Qionglu Zhang, Hengyuan Liu, Xiaoyan Gu, Xianhui Lu, Hongbo Liu, Bingzheng Wang, Haihui Fan, Ziming Zhao, Rui Zhang, and Li Zhou. 2026. *CrypFormBench: Benchmarking formal analysis capability of large language models for cryptographic schemes*. *Proceedings of the ACM on Software Engineering*, 3(FSE).
- Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. 2025. *Lean-STaR: Learning to interleave thinking and proving*. In *International Conference on Learning Representations*, pages 66041–66062.
- Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. 2024. *FVEL: Interactive formal verification environment with large language models via theorem proving*. In *Advances in Neural Information Processing Systems*, volume 37, pages 54932–54946. Curran Associates, Inc.
- Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. 2026. *Goedel-Prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction*. In *International Conference on Learning Representations*, pages 11793–11818.
- Yiping Ma, Yu-Lin Tsai, Mayank Rathee, Deevashwer Rathee, François Dupressoir, Pierre-Yves Strub, and Raluca Ada Popa. 2026. *ShannonProver: Towards automating formal cryptographic proofs*. Preprint, arXiv:2607.02847. ArXiv preprint.
- Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. 2024. *Autoformalizing euclidean geometry*. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 36847–36893. PMLR.
- Azim Ospanov, Farzan Farnia, and Roozbeh Mohit. 2025. *APOLLO: Automated LLM and lean collaboration for advanced formal reasoning*. In *Advances in Neural Information Processing Systems*, volume 38, pages 41599–41633. Curran Associates, Inc.
- Ji Qi, Ming Ding, Weihan Wang, Yushi Bai, Qingsong Lv, Wenyi Hong, Xu Bin, Lei Hou, Juanzi Li, Yuxiao Dong, and Jie Tang. 2025. *CogCoM: A visual language model with chain-of-manipulations reasoning*. In *International Conference on Learning Representations*, pages 11090–11110.
- Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. 2024. *Visual CoT: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning*. In *Advances in Neural Information Processing Systems*, volume 37, pages 8612–8642. Curran Associates, Inc.
- George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Amitayush Thakur, and Swarat Chaudhuri. 2024. *PutnamBench: Evaluating neural theorem-provers on the putnam mathematical competition*. In *Advances in Neural Information Processing Systems*, volume 37, pages 11545–11569. Curran Associates, Inc.
- Zichen Wang, Anjie Dong, and Zaiwen Wen. 2025. *Tree-based premise selection for Lean4*. In *Advances in Neural Information Processing Systems*, volume 38, pages 32621–32655. Curran Associates, Inc.
- Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James T. Kwok, and Yu Zhang. 2024. *GITA: Graph to visual and textual integration for vision-language graph reasoning*. In *Advances in Neural Information Processing Systems*, volume 37, pages 44–72. Curran Associates, Inc.
- Wenshan Wu, Shaoguang Mao, Yadong Zhang, Yan Xia, Li Dong, Lei Cui, and Furu Wei. 2024. *Mind’s eye of LLMs: Visualization-of-thought elicits spatial reasoning in large language models*. In *Advances in Neural Information Processing Systems*, volume 37, pages 90277–90317. Curran Associates, Inc.
- Ziyang Xiao, Dongxiang Zhang, Xiongwei Han, Xiaojin Fu, Wing Yin Yu, Tao Zhong, Sai Wu, Yuan Wang, Jianwei Yin, and Gang Chen. 2024. *Enhancing LLM reasoning via vision-augmented prompting*. In *Advances in Neural Information Processing Systems*, volume 37, pages 28772–28797. Curran Associates, Inc.
- Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanbiao Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. 2025. *DeepSeek-Prover-V1.5*.

Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*, pages 72274–72303.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. 2024. [Lean workbook: A large-scale Lean problem set formalized from natural language math problems](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 105848–105863. Curran Associates, Inc.

Yiyang Zhou, Haoqin Tu, Zijun Wang, Zeyu Wang, Niklas Muennighoff, Fan Nie, Chaorui Deng, Shen Yan, Haoqi Fan, Yejin Choi, James Zou, Cihang Xie, Huaxiu Yao, and Qinghao Ye. 2026. [When visualizing is the first step to reasoning: MIRA, a benchmark for visual chain-of-thought](#). In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26154–26164.

A Scratchy-eval Data Card

Scratchy-eval is designed to compare how alternative representations of proof relations affect machine-checkable EasyCrypt generation. It also includes compact diagnostic tasks for static validity and semantic recognition. The intended unit of comparison is a fixed model, task pair evaluated under the same response contract and verifier.

This dataset contains 114 tasks, including 64 free-form proof-generation targets and 50 multiple-choice diagnostics. The source material spans 26 protocol or formal-program groups. Table 5 reports category-level coverage; task-level source pointers are retained in the accompanying metadata.

A.1 Representative datum

Task 003 is drawn from the official while-sampling example in the foundations family. Its source is sufficiently compact to reproduce in full. Listing 3 includes the complete pinned file: imports, declarations, probabilistic program, target, and accepted proof. To construct the generation task, only the body of `Sample_lossless` is replaced by a proof hole.

The generation contract asks for the missing proof body rather than the surrounding declaration. Listing 4 gives the complete canonical output for Task 003. This fragment reconstructs the official source above and is accepted by EasyCrypt; a textually different response is also correct when it closes the same target under the verifier.

For generation, a declaration is selected from its source and the required imports, definitions, and dependency prefix are preserved while the official proof is replaced. The hidden fragment is used

```

require import Real Distr.

type t.

op sample: t distr.
axiom sample_ll: is_lossless sample.

op test: t -> bool.
axiom pr_ntest: 0%r < mu sample (predC test).

module Sample = {
  proc sample () : t = {
    var r : t;

    r <$ sample;
    while (test r) {
      r <$ sample;
    }
    return r;
  }
}.

lemma Sample_lossless: islossless Sample.sample.
proof.
proc; seq 1: true=> //.
+ by auto=> />; exact/sample_ll.
while true (if test r then 1 else 0) 1 (mu sample (predC test)
) => //.
+ by move=> _ r; case: (test r).
+ move=> ih; seq 1: true=> //.
  by auto; rewrite sample_ll.
+ by auto; rewrite sample_ll.
rewrite pr_ntest=> /= z; conseq (: true ==> !test r).
+ smt().
by rnd; auto=> />.
qed.

```

Listing 3: Complete official EasyCrypt source underlying Scratchy-eval Task 003. The task removes the proof body of `Sample_lossless` while preserving the surrounding file as its compilation context.

for fixture preflight and analysis, but is not shown to the model. The multiple-choice portion asks either whether a local EasyCrypt form is valid or which formal role a construct serves. All answers are checked against the pinned source or compiler behavior.

A.2 Representations and scoring

Each task supports three textual conditions and three graph conditions. The professional graph presents typed proof objects, local formulas, directed dependencies, and a route to the target; the remaining conditions remove relation detail or alter visual structure. Generation is scored by EasyCrypt acceptance after source reconstruction, while the diagnostic tasks use answer accuracy. This separates executable proof completion from recognition of formal knowledge.

The release record retains source provenance, task type, target information, graph metadata, supported conditions, and verification artifacts. Scratchy-eval is intended for controlled representation studies over the included EasyCrypt material. Its category-level coverage is broader than the mo-

Table 5: Category-level coverage of Scratchy-eval. Representative material and proof emphasis are summarized without enumerating task IDs or hidden fragments. The generation counts sum to 64; static-validity and semantic-recognition diagnostics accompany all seven families.

Source family	Representative material	Proof emphasis	Judgments	Gen.
Foundations	Recursion; distributions; sampling	Termination and probability	lemma/equiv	10
Program logics	Relational while programs	Coupling and invariants	lemma	2
Simple crypto	Pedersen; Schnorr; ElGamal	Correctness and basic security	lemma	7
Complex crypto	PIR; stateful PRG	Relational privacy and state	lemma/equiv	9
Reduction paths	Up-to; plug-and-pray	Game transitions and bounds	lemma	7
Extended crypto	AE; UC/CCA; hashed ElGamal	Composed and oracle reductions	lemma/equiv	23
Program analysis	Adversary bounds; Quickselect	Expectations and data invariants	lemma/ehoare	6

```

proc; seq 1: true=> //.
+ by auto=> />; exact/sample_ll.
while true (if test r then 1 else 0) 1 (mu sample (predC test)
=> //.
+ by move=> _ r; case: (test r).
+ move=> ih; seq 1: true=> //.
  by auto; rewrite sample_ll.
+ by auto; rewrite sample_ll.
rewrite pr_ntest=> /= z; conseq (: true ==> !test r).
+ smt().
by rnd; auto=> />.

```

Listing 4: Complete accepted output for Scratchy-eval Task 003. This canonical fragment is taken from the official answer record and contains exactly the code required at the proof hole.

tivating Observation, while remaining a focused benchmark rather than a census of cryptographic formalization.