DKG: A Dual-Layer Knowledge Graph and LLM Co-Driven Framework for Fully Automated Side-Channel Analysis of Cryptosystems

Wenquan Zhou, An Wang, Yaoling Ding, Jiakun Li, Jingqi Zhang, and Liehuang Zhu, *Senior Member, IEEE*

***Abstract*—Cryptographic algorithms underpin billions of devices, from payment smartcards to smartphone secure enclaves and industrial controllers. Yet their physical implementations risk leaking secret keys. Non-profiled side-channel analysis (SCA) evaluates whether such leakage enables key recovery, but executing it end-to-end remains expert-intensive. Large language model (LLM) agents now automate complex workflows across many domains, yet reliable key recovery requires specialized SCA knowledge that must be extracted and organized from scattered technical literature. In this paper, we propose DKG, a dual-layer knowledge graph and LLM co-driven key-recovery agent framework. We introduce the first graph schema designed specifically for SCA: an upper layer of configuration components and a lower layer of diagnostic strategies, joined by cross-layer relationships. An LLM-driven three-step extraction pipeline (*Locate & Rewrite*, *Triple Extraction*, and *Knowledge Fusion*) builds the graph from raw technical documents without manual annotation, and at runtime the retrieved knowledge drives a closed-loop autonomous attack workflow. Built from a corpus of 166 technical documents, the graph contains 1,334 entities and 1,813 relationships. On eight datasets covering AES, DES, and SM4 across smartcards, FPGAs, and MCUs, DKG with DeepSeek-V3.2 achieves a 97.5% end-to-end key-recovery success rate. By comparison, ReAct-RAG, a text-retrieval-augmented agent, achieves only 68.8%. Across the open-weights backbones DeepSeek-V3.2, GLM-5, and Qwen3.5, DKG achieves an average success rate of 93.3%. To the best of our knowledge, DKG is the first framework to achieve consistent end-to-end automated key recovery across diverse implementations. Its automated pipeline lets the graph grow with new literature, and its dual-layer organization provides a basis for extending DKG to additional SCA techniques.**

***Index Terms*—Automated key recovery, autonomous agent, correlation power analysis, knowledge graph, large language model, non-profiled attack, side-channel analysis.**

I. Introduction

Cryptographic algorithms protect devices in daily life, from payment smartcards and automotive key fobs to smartphone secure enclaves and industrial controllers [1]. The security of these systems rests on cryptographic-key secrecy, which depends on both the algorithms and their physical implementations. During cryptographic operations, power consumption and electromagnetic (EM) emanations can reveal key-dependent information, enabling side-channel analysis (SCA) to recover secret keys [1], [2]. Side-channel evaluation examines this exposure through complementary approaches: leakage assessment identifies statistical evidence of information leakage [3], whereas key-recovery tests establish whether an attack can exploit measured leakage to reconstruct the key [1], [4].

Attack-oriented evaluation is constrained by an expertise bottleneck. In non-profiled SCA, analysts work directly with measurements from the target implementation and must navigate a compositional decision space throughout each attack campaign. Selecting attack configurations and diagnosing unsuccessful attempts draw on accumulated SCA expertise, making evaluation throughput dependent on the specialist effort available to evaluation laboratories, certification bodies, and vendors' in-house security teams.

Among non-profiled techniques, correlation power analysis (CPA) [4] illustrates this complexity. An attack campaign requires interlocking decisions on attack point, leakage model, and preprocessing whose optimal settings depend on each other [1]. Diagnosing an unsuccessful attempt then requires interpreting the observed results and choosing the next configuration change or recovery action. In the traditional workflow, tools perform the underlying computations, while analysts supply the knowledge needed to configure these operations and interpret their results. Fig. 1 contrasts this workflow with the DKG-driven agent we propose.

Large language model (LLM)-based agents have achieved autonomous multi-step reasoning and tool invocation across many domains [5]–[8]. Applying this capability to CPA requires specialized domain knowledge spanning component selection, preprocessing strategies, and failure diagnosis. This knowledge, however, remains scattered across academic papers, evaluation standards, vendor manuals, and tool documentation.

Retrieval-augmented generation (RAG) supplies LLM agents with relevant information from external documents [9]. Combined with tool use, document retrieval supports domain workflows such as chemical experimentation in Coscientist and penetration testing in PentestAgent [10], [11]. In CPA, the agent must coordinate attack components and interpret unsuccessful attempts to choose its next action. These decisions

This work was supported in part by the National Natural Science Foundation of China under Grant 62272047 and Grant 62502035, in part by the Beijing Natural Science Foundation under Grant L244044 and Grant L251068, and in part by the State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, under Grant KFZD2503. *(Wenquan Zhou and An Wang contributed equally to this work.) (Corresponding author: Yaoling Ding.)*

Wenquan Zhou, An Wang, Yaoling Ding, Jiakun Li, Jingqi Zhang, and Liehuang Zhu are with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: {zhouwenquan, wanganl, dyl19, li-jk, zhangjq, liehuangz}@bit.edu.cn).

Wenquan Zhou and An Wang are also with the State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao 266237, China.

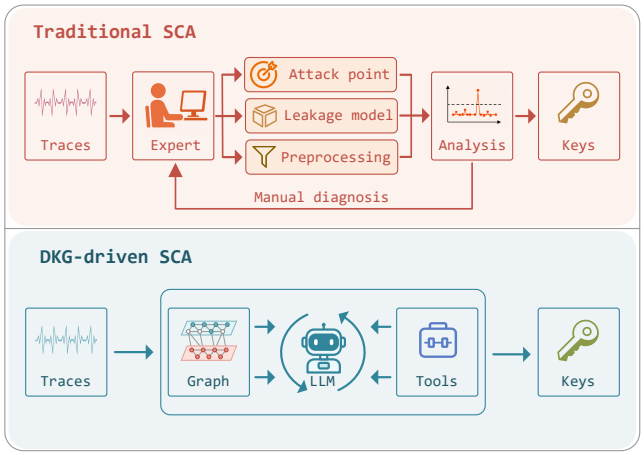


Fig. 1. Traditional versus DKG-driven SCA workflows.

motivate a knowledge representation that explicitly connects component dependencies with diagnostic guidance.

Retrieved text passages provide descriptions from which the agent interprets component dependencies and the ordering of actions. A structured graph makes these relationships explicit, connecting configuration choices with diagnostic guidance. Constructing such a graph requires consolidating heterogeneous documents into a consistent representation of expert decision logic, including the components, conditions, and actions described across sources. Existing graph designs emphasize event narration, diagnostic reasoning, or retrieval augmentation (Section II-B). For non-profiled SCA, the schema must connect compositional configuration with failure diagnosis so that diagnostic guidance can lead to concrete configuration changes.

Existing research has begun to turn dispersed security knowledge into resources for automated analysis (Table I). AttacKG and its successor AttacKG+ construct knowledge graphs (KGs) from cyber-threat intelligence, organizing attack entities and their relationships into explicit knowledge bases (KBs) [12], [13]. At the execution level, PentestAgent combines domain knowledge retrieval with multi-agent collaboration to support penetration-test planning and execution [11]. Within SCA, SPA-Prompt embeds expert strategies in prompts to guide simple power analysis and key recovery [14], while SCA-GPT supplies retrieved domain knowledge to an agent for standards-oriented leakage evaluation [15]. However, existing approaches offer limited support for the interdependent attack configuration and failure diagnosis required by CPA, leaving end-to-end key recovery reliant on expert judgment.

To address these challenges, we propose **DKG**, a framework in which LLMs build a domain-specific KG offline and the KG guides an LLM agent at runtime for automated non-profiled SCA. Its graph schema adopts a dual-layer architecture. The upper layer models what components can be combined and in what order across the SCA decision space. The lower layer encodes expert diagnostic reasoning as structured cause-effect chains that connect symptoms to root causes and then to concrete remedial actions. To construct this graph at scale, we design an LLM-driven three-step extraction pipeline that automatically builds the graph from raw technical documents without manual annotation. At runtime, the agent retrieves related components, constraints, and diagnostic guidance from the graph through multi-hop traversal over its typed entities and relationships. This knowledge supports attack planning, failure diagnosis, and autonomous configuration adjustment during key recovery.

TABLE I
COMPARISON WITH RELATED KNOWLEDGE-GRAPH AND AGENT FRAMEWORKS.

Method	SCA	KB	KG	Key recovery
AttacKG [12]	×	✓	✓	×
PentestAgent [11]	×	✓	×	×
SCA-GPT [15]	✓	✓	×	×
SPA-Prompt [14]	✓	×	×	✓
DKG (ours)	✓	✓	✓	✓

Our contributions are as follows:

1) We introduce **a dual-layer graph schema** for non-profiled SCA that separates compositional configuration from causal diagnosis: an upper configuration space (12 entity types modeling how attack components compose) and a lower diagnostic strategy space (four entity types mapping failure symptoms to remedial actions), connected by 11 relationship types. To our knowledge, this is the first graph schema designed for SCA automation.
2) We develop a **three-step LLM-driven extraction pipeline** (Locate & Rewrite, Triple Extraction, and Knowledge Fusion) that automatically constructs the graph from 166 raw technical documents without manual annotation. The resulting graph contains 1,334 entities and 1,813 relationships. Annotation of sampled graph items yields 95.0% precision, and matching against an annotated ground truth yields 88.5% recall, for an F_1 of 91.6%. The schema covers 91.0% of the in-scope knowledge.
3) We implement a **KG-augmented LLM agent framework** that integrates graph retrieval with SCA tools, and evaluate it on eight datasets covering AES, DES, and SM4 across smartcards, FPGAs, and MCUs. With DeepSeek-V3.2, DKG achieves a 97.5% end-to-end key-recovery success rate (SR), compared with 68.8% for ReAct-RAG, a text-retrieval-augmented agent. A four-configuration ablation examines the individual and combined roles of specialized tools and graph access. Across DeepSeek-V3.2, GLM-5, and Qwen3.5, DKG achieves an average SR of 93.3%.[1]

II. Background

A. Non-Profiled Side-Channel Analysis

SCA techniques are broadly divided into two categories: profiled and non-profiled. Profiled attacks, such as template attacks [16] and their deep-learning-based successors [17]–[19], first build a statistical or learned model of the target

[1]Code and results are available at https://github.com/wenquan222/DKG.

device's leakage using a profiling device with a known key, then apply this model to recover the secret key from a separate target device. While powerful, profiled attacks require access to an identical or near-identical profiling device and knowledge of the key during the profiling phase, which limits their applicability in practice.

Non-profiled attacks impose no such requirements: the attacker needs only the measured traces from the target device and publicly known information such as plaintexts or ciphertexts. Among these techniques, CPA [4] evaluates candidate keys by correlating predicted leakage from intermediate computations with measured traces. Its configuration includes the target intermediate value, the leakage model, and trace preprocessing.

B. Knowledge Graphs and Extraction Quality Metrics

KGs represent structured knowledge as directed graphs of typed entities connected by typed relationships [20]. In domain expert systems, graphs serve as machine-readable encodings of expert decision logic. Across application domains, existing graphs fall into three principal modes by cognitive function: *event-narrative graphs* that describe what happened, *diagnostic-reasoning graphs* that map symptoms to causes and remedial actions, and *retrieval-augmentation graphs* that supply relevant context to downstream tasks.

Diagnostic-reasoning graphs, developed in industrial and clinical settings, model chains that link a fault phenomenon to its root cause and remediation action to guide troubleshooting [21] or represent disease–symptom associations learned from electronic medical records [22]. *Event-narrative graphs* extract attack technique structure from cyber-threat intelligence reports, with AttacKG [12] using traditional natural language processing (NLP) pipelines and AttacKG+ [13] upgrading to LLM-based extraction. *Retrieval-augmentation graphs* such as GraphRAG [23] organize entities into communities and generate community summaries to support question answering over an entire document collection.

KGs in these modes are built by either manual curation or automated extraction from unstructured text. Recent work has shifted from rule-based and traditional NLP pipelines to LLM-driven extraction [20], [24], motivated by its handling of paraphrasing and schema-specific terminology across heterogeneous documents.

When a graph is constructed, its quality must be evaluated on two fronts: whether the extracted content is correct, and whether the underlying schema is expressive enough to absorb the target domain knowledge. We use precision, recall, and F_1 to evaluate extraction quality [25]–[27], and define schema coverage to assess how much in-scope knowledge the ontology can represent.

Precision (P) estimates, from a random sample of graph items, what fraction of the extracted triples are correct:

$$P = \frac{|T_{\text{correct}}| + 0.5\,|T_{\text{partial}}|}{|T_{\text{sampled}}|}, \tag{1}$$

where T_{sampled} is the sampled set, T_{correct} contains the sampled items judged fully correct, and T_{partial} contains those judged correct in substance but imprecise in entity typing or source support, which receive half credit. Intuitively, a high P means the extraction pipeline rarely produces spurious or incorrectly typed triples. A low P indicates that the graph is polluted with noise.

Recall (R) measures what fraction of the triples that should have been extracted were actually captured:

$$R = \frac{|T_{\text{extracted}} \cap T_{\text{GT}}|}{|T_{\text{GT}}|}, \tag{2}$$

where $T_{\text{extracted}}$ denotes the set of triples produced by the extraction pipeline, and T_{GT} is a manually annotated ground-truth set of triples representing the ideal extraction from an annotated document subset. A high R means the pipeline misses little; a low R indicates that valuable knowledge present in the source documents is lost.

F_1 score is the harmonic mean of P and R, balancing the trade-off between the two:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}. \tag{3}$$

P and R capture complementary failure modes: low P corresponds to over-extraction, while low R corresponds to under-extraction. Reporting both alongside F_1 gives a complete picture of extraction quality.

Schema coverage (Cov_{in}) goes beyond triple-level correctness to assess whether the ontology design itself is expressive enough to absorb the domain knowledge present in the source corpus. For domain-scoped graph construction, where the corpus includes both in-scope and out-of-scope material, we define a coverage metric that excludes out-of-scope items from the denominator:

$$\text{Cov}_{\text{in}} = 1 - \frac{|T_{\text{truly_unknown}}|}{|T_{\text{total}}| - |T_{\text{out_of_scope}}|}. \tag{4}$$

Here T_{total} is the full set of candidate extraction items from the source corpus, $T_{\text{truly_unknown}}$ denotes items that fall within the declared scope but cannot be represented by the current schema, and $T_{\text{out_of_scope}}$ denotes items correctly rejected as outside the scope. A high Cov_{in} indicates that the schema covers the target domain well; the residual gap quantifies how much knowledge the schema cannot yet express.

C. LLM-Based Autonomous Agents

LLMs are now available in both closed-source frontier offerings and open-weights releases, with the latter increasingly approaching frontier capability.

Originally limited to language input and output, LLMs gained the ability to operate in diverse domains through the *agent* paradigm: an LLM is wrapped with callable tools and a context-management loop in which it reasons about the next step, invokes a tool, observes the result, and iterates [5]. Agents built on this loop now drive autonomous workflows across software engineering, open-ended exploration, and offensive security.

In specialized domains, agents are typically grounded through two complementary mechanisms. *Tool augmentation* equips the agent with callable domain-specific operations, illustrated by general-purpose frameworks such as

MetaGPT [28], Voyager [29], and ToolLLM [30] and by security-specific systems including PentestGPT [7] and AutoAttacker [8]. *Knowledge augmentation* supplies decision-relevant context at query time, ranging from prompt engineering that injects a fixed playbook to RAG [9] that dynamically retrieves text passages and GraphRAG [23] that augments retrieval with graph-structured summaries. Tool use and knowledge retrieval can operate within the same reasoning-and-action loop [5], [11]. Retrieved information informs the next action, while execution feedback guides subsequent reasoning and retrieval.

D. Threat Model

We adopt the standard non-profiled SCA setting against unprotected symmetric block-cipher implementations. The attacker records power or EM traces from the target device during cryptographic operations, together with the corresponding plaintexts or ciphertexts and the algorithm identity; the attacker knows neither the secret key nor a leakage model trained on a profiling device. The objective is to recover a key that passes encrypt-decrypt verification. The entire attack pipeline runs autonomously through an LLM-driven agent with no human expert in the loop.

III. The DKG Framework

A. Overview and Motivation

DKG converts scattered SCA expertise into a KG that an LLM agent traverses to recover keys autonomously. Realizing this poses two challenges that organize our design.

Challenge 1: The schema of the SCA knowledge graph. SCA expert knowledge separates into two categorically different forms. (i) *Compositional configuration*: multiple components that must be compatible and ordered. (ii) *Causal diagnosis*: failure symptoms, their probable causes, the corresponding remedial actions, and the configuration components those actions modify. Existing graph designs emphasize event narration, diagnostic reasoning, or retrieval augmentation (Section II-B). Supporting non-profiled SCA requires a schema that connects compositional configuration with causal diagnosis.

Challenge 2: Constructing the SCA knowledge graph at scale. The expert knowledge needed to populate this schema is dispersed across hundreds of academic papers, evaluation standards, vendor manuals, and tool documentation, each written for human readers in different styles. Constructing a consistent graph from these sources requires locating relevant content, aligning it with the target schema, and resolving cross-document duplication. LLM-driven graph construction [20], [24] must therefore accommodate SCA-specific terminology and the two forms of knowledge defined above.

Fig. 2 sketches how DKG addresses the two challenges. Its offline phase (left, Sections III-B–III-C) constructs the dual-layer graph from heterogeneous documents through a three-step LLM-driven pipeline. The online phase (right, Section III-D) integrates an LLM agent with SCA tools and graph retrieval and runs a closed loop against real traces.

B. Dual-Layer Knowledge Graph Schema

Expert SCA knowledge falls into two categorically different forms. The first answers "how should an attack be configured": what algorithm, device, attack point, leakage model, and preprocessing to use. We refer to this as the *configuration space*, and it is intrinsically compositional, with compatibility, substitutability, and sequential structure. The second answers "what to do when something goes wrong": identifying the probable cause of an observed symptom and the corrective action. We refer to this as the *diagnostic space*, and it is causal, prioritized, and conditionally triggered.

We organize the two forms into separate layers so that configuration and diagnostic queries follow different relationship types. A configuration query such as "hardware AES preprocessing" follows compatibility, substitution, and ordering relationships in the upper layer. A diagnostic query such as "hardware AES failure diagnosis" follows observation, cause, and action relationships in the lower layer. Cross-layer relationships connect diagnostic Actions and Strategies to the upper-layer components they affect, translating diagnostic conclusions into concrete configuration adjustments.

Upper layer: configuration space graph. Because the configuration space is compositional, the upper layer uses entity types spanning the decision axes an analyst traverses and relationship types capturing the structural properties of those axes (compatibility, substitutability, and sequence).

TABLE II
Upper-layer entity types.

Entity Type	Description
Algorithm	Target cipher (AES, DES, SM4, ...)
DeviceClass	Target device (smartcard, FPGA, MCU, ...)
TraceType	Side-channel measurement (power, EM)
Countermeasure	Protection mechanism (masking, hiding, ...)
AttackPoint	Intermediate value position (S-box output/input)
LeakageModel	Statistical leakage assumption (Hamming weight (HW), Hamming distance (HD), ...)
Preprocessing	Trace preprocessing (alignment, filtering, ...)
StatMethod	Statistical distinguisher (CPA, differential power analysis, mutual information analysis, ...)
Pipeline	Named end-to-end methodology
DiagnosticMethod	Failure symptom diagnosis
ValidationMethod	Post-attack verification
RecoveryMethod	Partial-key search

TABLE III
Upper-layer relationship types.

Relationship	Direction and Semantics
has_attack_point	Algorithm → AttackPoint
acquires_via	DeviceClass → TraceType
weakens	Countermeasure → StatMethod
works_with	Any ↔ Any (general compatibility)
alternative_to	Same-type ↔ Same-type
precedes	Preprocessing → Preprocessing/StatMethod
consists_of	Pipeline → Preprocessing/StatMethod

We organize the decision axes of the classical SCA workflow [1] into 12 entity types (Table II): the target (Algorithm,

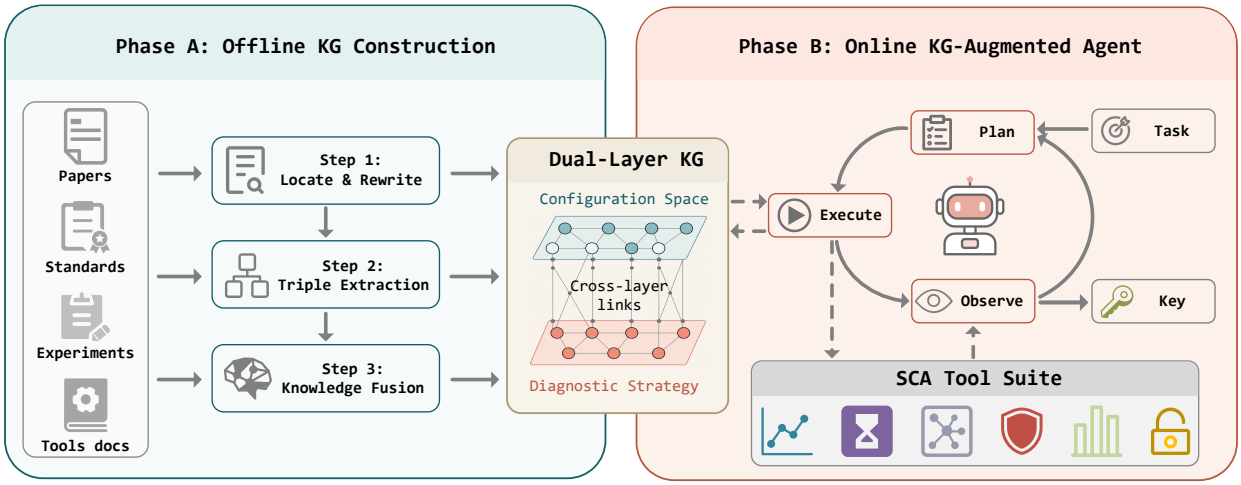


Fig. 2. The DKG framework.

DeviceClass, TraceType, Countermeasure), the attack design (AttackPoint, LeakageModel, Preprocessing, StatMethod), the methodology aggregation (Pipeline), and post-attack handling (DiagnosticMethod, ValidationMethod, RecoveryMethod).

The seven relationship types (Table III) serve three functional roles. *Anchoring* relationships (`has_attack_point`, `acquires_via`) bind a configuration to a concrete cipher and acquisition setup. *Compositional* relationships (`works_with`, `alternative_to`, `precedes`, `consists_of`) encode the compatibility, substitutability, ordering, and aggregation structure of the configuration space. The *adversarial* relationship (`weakens`) captures how countermeasures degrade statistical attacks.

Lower layer: diagnostic strategy graph. Because the diagnostic space is causal, prioritized, and conditionally triggered, the lower layer uses entity types that represent the causal chain from symptom to remedy (Table IV). A higher-level Strategy type holds knowledge whose internal structure resists triple decomposition. Intra-layer and cross-layer relationships connect these entities (Table V).

TABLE IV
LOWER-LAYER ENTITY TYPES.

Entity Type	Description
Observation	Measurable symptom encountered during an attack
Cause	Candidate root cause of an observation
Action	Remedial step that resolves a cause
Strategy	Structured expert knowledge (checklists, anti-patterns, parameter bundles, generalizable principles)

TABLE V
LOWER-LAYER RELATIONSHIP TYPES.

Relationship	Direction and Semantics
indicates	Observation → Cause (intra-layer)
resolves	Action → Cause (intra-layer)
relates_to	Action → upper-layer entity (cross-layer)
strategy_applies_to	Strategy → upper-layer entity (cross-layer)

The first three entity types form the classical diagnostic chain. *Observation* entities describe measurable symptoms during an attack (e.g., low correlation, high alignment discard rate). *Cause* entities describe candidate root causes (e.g., wrong leakage model, noisy alignment reference). *Action* entities describe remedial steps (e.g., switching attack point, adjusting alignment parameter). A fourth entity, *Strategy*, captures knowledge whose internal ordering, bundling, or negation semantics would be lost under triple decomposition: troubleshooting checklists, anti-pattern sets, parameter preset bundles, and generalizable principles.

The four relationship types fall into two groups. Intra-layer, `indicates` and `resolves` chain together to form a complete diagnostic trajectory (Observation → Cause → Action). Cross-layer, `relates_to` and `strategy_applies_to` link Actions and Strategies back to the upper-layer components they modify, closing the loop between diagnosis and configuration.

C. LLM-Driven Knowledge Extraction Pipeline

The pipeline automatically constructs the dual-layer graph defined in Section III-B from a corpus of raw technical documents. Manual curation across hundreds of heterogeneous documents is impractical, and the corpus's non-canonical phrasing, paragraph-level context dependencies, and schema-specific terminology resist traditional rule-based extractors; we therefore use an LLM to perform the extraction.

Step 1: Locate and rewrite. Raw technical documents are not written for graph construction: relevant SCA knowledge is scattered across paragraphs, embedded in informal prose, and intermixed with out-of-scope content. This step uses an LLM to locate the relevant content and rewrite it into a structured intermediate representation aligned with the schema's two layers. The upper-layer extraction captures attack records (cipher, device, attack point, leakage model, preprocessing, statistical method) and named methodologies (Pipeline candidates). The lower-layer extraction captures symptom-cause-remedy chains and strategy texts (checklists, anti-patterns, parameter bundles, decision principles).

Step 2: Triple extraction. This step converts the intermediate representation into the typed entities and relationships required by the schema. An LLM assigns each piece of content to a schema-defined entity type and establishes the relationships among them. Upper-layer attack records become typed entities and relationships, and named methodologies become Pipeline entities with ordered `consists_of` steps. Lower-layer diagnostic chains become Observation, Cause, and Action entities linked by `indicates` and `resolves` relationships. Strategy texts stay intact as Strategy entities.

Step 3: Knowledge fusion. This step merges per-document fragments into a unified graph through four operations. *Entity unification* merges entities referring to the same concept under a canonical name (e.g., "AES-128" and "128-bit AES"). *Diagnostic refinement* reviews merged Observation, Cause, and Action entities to remove generic entries, correct mismatched operation types, and merge duplicate chains. *Cross-document inference* discovers compatibility and substitutability patterns that emerge across multiple sources but appear in no single document. *Cross-layer linking* connects lower-layer Action and Strategy entities to the upper-layer components they target.

D. Closed-Loop KG-Augmented Agent

With the dual-layer graph constructed by the extraction pipeline, the final component of DKG is the agent that uses this knowledge to perform automated key recovery on real traces. The agent operates as a closed-loop system in which an LLM drives high-level decisions, retrieves relevant knowledge from the graph, and executes its decisions through a suite of specialized tools.

Closed-loop workflow. Given a user-specified attack task (e.g., target cipher, device, available traces), the agent first performs a one-time graph retrieval that injects relevant configuration knowledge and applicable strategies into its context. It then enters a closed loop of three steps until the key passes verification or the agent ends the run:

1) **Plan**: decide the next action based on the current understanding and accumulated results.
2) **Execute**: invoke any tool from the suite, including signal-processing tools, statistical attack tools, key search, verification, and the graph retrieval tool (`kg_search`) when additional domain knowledge is needed.
3) **Observe**: examine the tool output (statistical result, verification status, or retrieved knowledge).

Each Observe step feeds back into the next Plan: the agent incrementally adjusts the next action based on what was just observed rather than redesigning from scratch. Which tools to invoke, including whether to call graph retrieval mid-loop, is decided autonomously by the agent. The two retrieval moments and the full tool suite are described below.

Knowledge retrieval. Knowledge from the graph enters the agent at two moments. The first is initial retrieval before the loop begins: given a user-specified attack target, the system retrieves matching configuration entities, related components, and applicable strategies from the graph. The retrieved knowledge provides configuration guidance for the first Plan step.

The second is on-demand retrieval within the loop: the agent invokes the graph retrieval tool when additional guidance is needed. Diagnostic retrieval connects observed symptoms with candidate causes and corrective actions, while Strategy entities provide applicable troubleshooting guidance. Cross-layer links identify the upper-layer components associated with these actions and strategies. Both retrieval moments match relevant graph entities and expand along typed relationships, providing connected knowledge for attack planning and configuration adjustment.

TABLE VI
THE AGENT'S TOOL SUITE.

Group	Tool	Purpose
Statistical attack	`aes_cpa_attack`	CPA on AES traces
	`des_cpa_attack`	CPA on DES traces
	`sm4_cpa_attack`	CPA on SM4 traces
Preprocessing	`moving_average`	Denoising filter
	`static_aligner`	Cross-correlation alignment
	`vertical_aligner`	Baseline correction
Key search	`key_search`	Search candidates for unreliable bytes
Knowledge access	`kg_search`	Runtime graph retrieval
Verification and auxiliary	`key_verify`	Validate a candidate key
	`pt_ct_corr`	Plaintext/ciphertext correlation analysis
	`python_jupyter`	General code execution

Tool suite. The agent operates through 11 tools (Table VI) in two categories. Four attack-related groups mirror upper-schema entity families, so that graph-guided decisions translate directly into tool invocations: *statistical attack* (configurable attack points and leakage models), *preprocessing* (denoising and alignment), *key search*, and *verification and auxiliary*. The remaining *knowledge-access* tool bridges agent and graph, exposing runtime retrieval as a callable in-loop interface.

IV. Experimental Evaluation

We structure the evaluation around four questions: (i) what is the quality of the constructed KG, and what does its construction cost? (ii) how does DKG compare with ReAct-RAG in end-to-end key recovery and execution cost? (iii) how do specialized tools and graph access contribute individually and in combination? (iv) how does DKG perform across different LLM backbones? After introducing the experimental setup, Sections IV-B–IV-E address these four questions in turn.

A. Experimental Setup

Datasets. We evaluate DKG on eight real-world side-channel datasets covering three symmetric ciphers, three device categories, and two trace types (Table VII). Each dataset forms one evaluation case, and case names encode the cipher and platform: 7816 and 14443 denote contact (ISO/IEC 7816) and contactless (ISO/IEC 14443) smartcards, FPGA a Sakura-G board, and STM32F103/F429 MCUs. The datasets span a

representative difficulty range, including software implementations on smartcards and MCUs, and hardware implementations on FPGA and contactless smartcards. We evaluate DKG with CPA [4], the distinguisher used across all eight cases. In the leakage-model notation, prev. ct denotes the ciphertext from the preceding encryption.

TABLE VII
EVALUATION DATASETS.

Case	Type	Traces	Points	Leakage
AES-7816	Power	100	7400	HW (S-box out)
AES-14443	EM	5000	1900	HW (S-box out)
AES-FPGA	Power	2000	300	HD (S-box in $\oplus$ prev. ct)
AES-STM32F103	Power	250	1500	HW (S-box out)
AES-STM32F429	EM	1000	1000	HW (S-box out)
DES-7816	Power	100	2000	HW (S-box out)
DES-14443	EM	10000	300	HD (round in $\oplus$ out)
SM4-7816	Power	100	6500	HW (S-box out)

Implementation platform. The agent is implemented on top of the LangGraph framework, which provides the closed-loop reasoning and tool invocation infrastructure. The graph is hosted in a Neo4j graph database, queried by the agent through the graph retrieval tool described in Section III-D.

All LLM calls are made through the SiliconFlow API service, which provides unified access to the three backbones evaluated in this work: DeepSeek-V3.2, GLM-5, and Qwen3.5-397B-A17B. Temperature is set to 0.1.

All experiments run on a workstation with a 13th-generation Intel Core i5-13600K CPU at 3.50 GHz, 64 GB of RAM, an NVIDIA RTX 3080 GPU with 12 GB of VRAM, and the Windows 11 Professional operating system.

Evaluation metrics. Our evaluation covers three aspects: end-to-end key recovery, agent efficiency, and graph extraction quality.

For end-to-end key recovery, we report SR, the fraction of runs in which the complete correct key is recovered, verified by encrypt-decrypt comparison. For DES smartcard cases, software encryption normalizes parity bits, so SR is computed on the effective encryption key: parity-equivalent byte values are accepted if they produce identical DES encryption results. For agent efficiency, we report LLM rounds, tokens, and runtime, defined respectively as the number of LLM calls, the total tokens consumed, and the wall-clock duration in seconds, each averaged per run. In the result tables, Rnd, Tok, and T denote LLM rounds, token consumption, and runtime, respectively. For graph extraction quality, we use the standard precision P, recall R, and F_1 together with the schema coverage $\mathrm{Cov}_{\mathrm{in}}$ defined in Section II-B. For each experimental configuration, every dataset is run 10 times to assess execution variability, and the metrics are averaged across runs.

B. Knowledge Graph Construction and Quality

Construction results. The corpus consists of 166 source documents: 80 academic papers drawn from venues such as ASIACRYPT, EUROCRYPT, TCHES, and ePrint, and 86 engineering materials covering evaluation standards, vendor manuals, open-source tool documentation, evaluation guides, hardware datasheets, tutorial slides, and practitioner notes. The corpus is scoped to non-profiled side-channel analysis of symmetric block ciphers.

Processing the corpus through the three-step extraction pipeline (Section III-C) yields a KG with 1,334 entities and 1,813 relationships, summarized in Fig. 3 and Fig. 4. The upper layer contains 462 entities and 953 relationships covering the configuration space; the lower layer contains 872 entities and 860 relationships (558 intra-layer plus 302 cross-layer to the upper layer). Additional technical documents can be processed through the same extraction and fusion pipeline to enrich the graph's configuration and diagnostic knowledge.

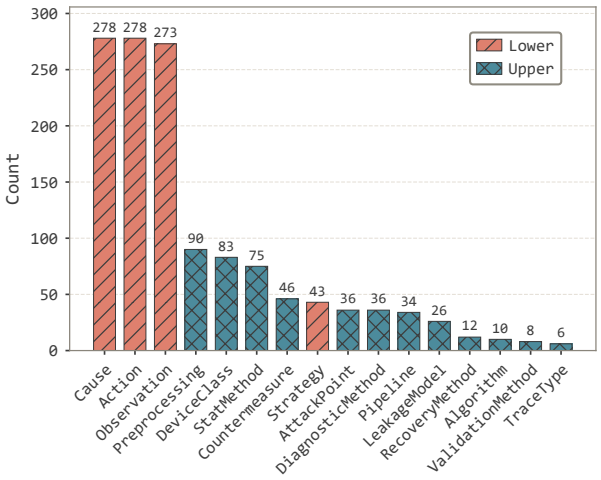


Fig. 3. Entity counts by layer.

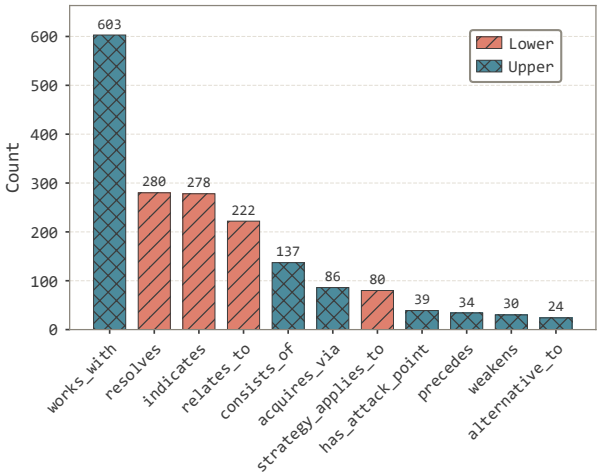


Fig. 4. Relationship counts by layer.

The LLM-based graph construction incurs a one-time cost, and the resulting graph is reused across attack runs. Table VIII breaks down its cost across the three extraction steps. The full LLM phase consumes 10,898 s of runtime and 7.9M tokens across 552 LLM calls, with Step 1 (Locate & Rewrite) being the dominant cost driver at 5.1M tokens because every document is processed in full. Step 3 is by far the cheapest stage in token count because it operates on the merged graph rather than on raw documents.

TABLE VIII
CONSTRUCTION COST.

Stage	Runtime (s)	LLM calls	Tokens
Step 1 (Locate & Rewrite)	5,422	332	5.1M
Step 2 (Triple Extraction)	3,639	214	2.4M
Step 3 (Knowledge Fusion)	1,837	6	0.4M
Total (LLM phase)	10,898	552	7.9M

Quality evaluation. To assess the quality of the constructed graph, we draw a stratified random sample of 250 items from the final fused graph: 100 upper-layer relationships, 100 lower-layer items, and 50 cross-layer relationships. Stratification is proportional to the relation-type frequency within each layer. Three annotators independently label each item against three criteria: (1) entity types match the schema, (2) relationship type is semantically appropriate, and (3) the item accurately reflects the source document. Disagreements are resolved by discussion.

To estimate recall, we exhaustively annotate five corpus documents, two academic papers and three engineering materials, and obtain 113 in-scope ground-truth items after scope pruning. Drawing on prior work on LLM-based evaluation [31], [32], we use an LLM to judge whether each ground-truth item is semantically represented in the graph. Fig. 5 reports the per-layer precision, recall, and F_1.

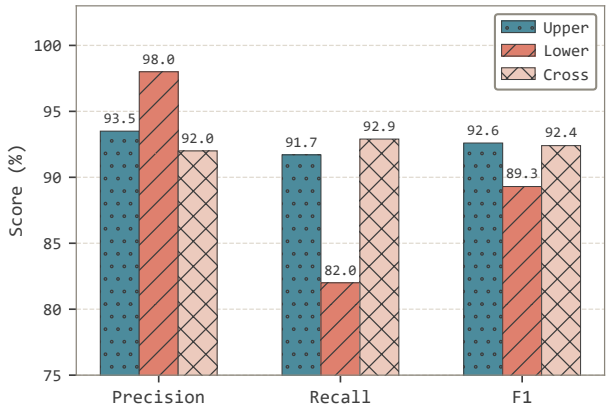


Fig. 5. Extraction quality by layer.

Annotation of the 250 sampled graph items yields an overall precision of 95.0%, and matching the graph against the 113 ground-truth items yields a recall of 88.5%, for an overall F_1 of 91.6%. Per-layer F_1 ranges from 89.3% (lower) to 92.6% (upper). Schema coverage of 91.0% means only 9.0% of in-scope candidates fall outside the current ontology. Cohen's $\kappa \approx 0.80$ confirms substantial inter-annotator agreement.

C. Comparison with ReAct-RAG

We compare DKG with a retrieval-augmented agent that combines ReAct-style reasoning and tool use with text retrieval, denoted *ReAct-RAG* [5], [9]. Both agents use DeepSeek-V3.2, the same SCA execution tools, and the same eight CPA evaluation cases. ReAct-RAG retrieves passages from the 166-document corpus at task initialization and as needed during execution. These passages provide domain knowledge for attack planning and subsequent decisions. DKG retrieves configuration and diagnostic knowledge from the dual-layer graph. We compare their end-to-end key-recovery success and execution cost.

TABLE IX
DKG VS. REACT-RAG ON THE EIGHT CASES.

	ReAct-RAG				DKG (ours)			
Case	Rnd	Tok	T (s)	SR	Rnd	Tok	T (s)	SR
AES-7816	30.8	824K	1128	0.0%	11.7	271K	269	100.0%
AES-14443	18.5	454K	1013	80.0%	13.5	320K	483	100.0%
AES-FPGA	13.8	234K	335	100.0%	11.5	256K	277	100.0%
AES-STM32F103	22.9	513K	462	80.0%	14.5	366K	309	100.0%
AES-STM32F429	12.7	198K	383	100.0%	10.6	250K	301	100.0%
DES-7816	44.6	1.3M	1171	0.0%	15.1	365K	309	100.0%
DES-14443	11.3	162K	458	100.0%	16.5	500K	387	80.0%
SM4-7816	25.7	551K	600	90.0%	17.8	417K	365	100.0%
Mean	22.5	535K	694	68.8%	13.9	343K	337	97.5%

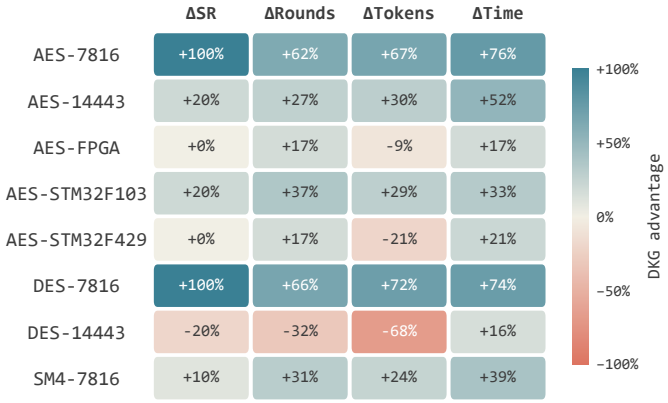
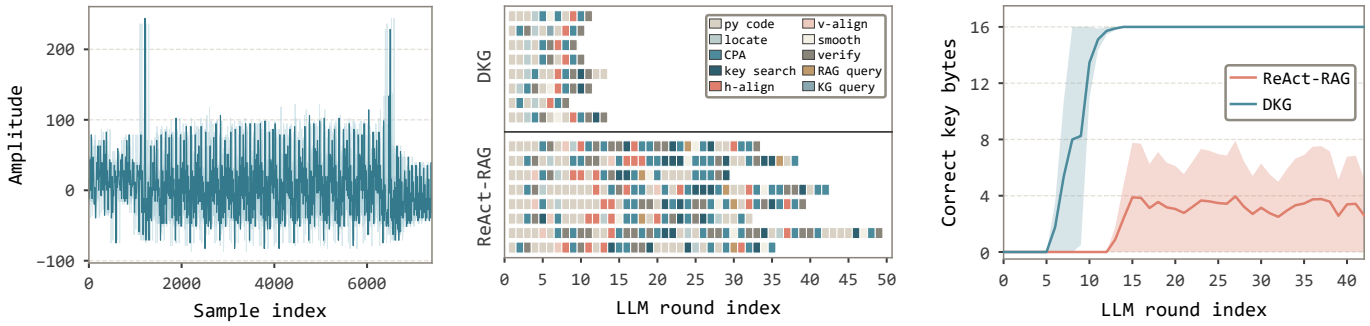


Fig. 6. Per-case DKG advantage over ReAct-RAG across four metrics.

With the same tools, corpus, backbone, and task setup, ReAct-RAG reaches 68.8% while DKG reaches 97.5%, a difference of 28.7 percentage points (Table IX and Fig. 6). Because the two agents share all other components, this gap reflects how the knowledge is organized. DKG represents configuration and diagnostic knowledge as entities connected by explicit relationships. These connections let the agent retrieve candidate causes, corrective actions, and relevant configuration components associated with observed symptoms, providing coherent guidance for subsequent decisions.

The difference is most pronounced on AES-7816 and DES-7816, both 100-trace cases, where DKG recovers the key in all 10 runs and ReAct-RAG in none. The gap is smaller on SM4-7816, which also uses 100 traces. These results indicate that knowledge requirements and attack decisions differ across cases even with the same trace count. For AES-7816 and DES-7816, the main decision is how to interpret weak initial CPA results and select subsequent actions. The graph connects symptoms, candidate causes, corrective actions, and applicable strategies, providing guidance for choosing between key search and configuration adjustment.



(a) Raw trace. (b) Tool-call timeline. (c) Correct key bytes per round.

Fig. 7. AES-7816 case study: DKG recovers the full key within 13 LLM rounds while ReAct-RAG does not converge.

On efficiency, DKG reduces the average number of LLM rounds from ReAct-RAG's 22.5 to 13.9, average runtime from 694 s to 337 s, and average token consumption from 535K to 343K. These measurements show that DKG achieves higher key-recovery success while reducing average execution cost.

To understand these performance differences, we examine how the two agents use retrieved knowledge during execution on AES-7816 and DES-7816. The case studies connect observed tool outputs with subsequent configuration changes, recovery actions, and verification outcomes.

AES-7816. Both agents launch with the same initial configuration, HW leakage on the S-box output with no preprocessing, and observe the same weak signature on the first CPA call (Fig. 7). Averaged over the 16 bytes, the correlation gap between the top two key candidates collapses to $\sim$0.02, and no byte is correct. From this identical observation the two agents diverge sharply. ReAct-RAG misreads the weak gap as a model selection error and spends up to 50 LLM rounds cycling through the HW and identity leakage models on three attack points: S-box out, S-box in, and S-box in $\oplus$ prev. ct. It also invokes the key-search tool up to six times, with budgets escalating to 5 million combinations, and resolves no byte, as the bottom rows of Fig. 7b show. DKG instead follows two lower-layer Strategy entities. The first states that low first-pass correlation on software implementations more often stems from region selection or misalignment than from the attack point. The second anchors static alignment on the variance-peak region. DKG aligns the traces, re-runs CPA, and recovers all 16 bytes within 8–13 LLM rounds. All 10 DKG runs pass encrypt-decrypt verification, and none of the 10 ReAct-RAG runs do, as Fig. 7c reports.

DES-7816. The same diagnostic pattern unfolds on DES (Fig. 8). Both agents first observe a weak average correlation ($\sim$0.40) with the standard HW model on the S-box output. ReAct-RAG then spends 44.6 LLM rounds on average chasing alternative attack points (round in $\oplus$ out, S-box in $\oplus$ out) and inflating key-search budgets without convergence. DKG follows two Strategy entities in priority order. The first anchors static alignment on the variance-peak region, which raises the average correlation to $\sim$0.74. When verification still fails on a subset of subkeys, the second directs the agent to widen the key search over the low-confidence subkeys before changing any other setting, and this search recovers the 8-byte master key. All 10 DKG runs pass encrypt-decrypt verification within 10–18 LLM rounds; all 10 ReAct-RAG runs fail (Fig. 8c).

These two cases illustrate how the graph connects failure diagnosis with subsequent actions. Following weak initial CPA results, ReAct-RAG tries alternative leakage models and attack points and increases key-search budgets. DKG follows Strategy entities that point to misalignment, performs static alignment, and repeats CPA. The `strategy_applies_to` links of these entities identify static alignment as the preprocessing component to change. On DES-7816, verification still fails for some subkeys after alignment, and an applicable Strategy guides the agent to perform a constrained key search before changing configuration. These trajectories show how configuration knowledge, diagnostic relationships, and strategies jointly guide the agent in translating tool feedback into subsequent configuration adjustments and key-recovery actions.

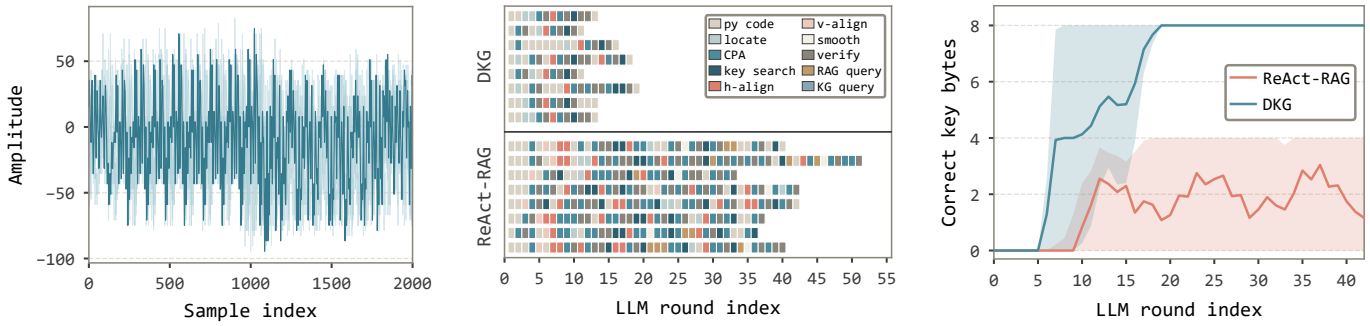
Inside an AES-FPGA run. The head-to-head results above compare *outcomes*, whereas here we zoom into a single run to show *how* DKG reasons (Table X). After a weak correlation with the standard HW model, the agent uses graph guidance to switch to an HD model and try alternative attack points. Verification rejects the initial key candidate and accepts its complement, completing key recovery.

TABLE X
STEP-BY-STEP TRACE OF A DKG RUN ON AES-FPGA.

Step	DKG's logic
1	Read in and inspect the raw power trace.
2	A vertical offset shows up, so align the traces vertically.
3	Run a baseline CPA with the HW model (S-box out).
4	Weak peak, so switch to the HD model (round in $\oplus$ out).
5	Still too weak, so try HD (S-box in $\oplus$ prev. ct).
6	Correlation finally peaks, but encrypt-decrypt still fails.
7	By HD symmetry, take the complement key; it then matches.

D. Ablation Study

This experiment isolates the contribution of each DKG component using ablation across two axes: presence of specialized



(a) Raw trace. (b) Tool-call timeline. (c) Correct key bytes per round.

Fig. 8. DES-7816 case study: DKG recovers the 8-byte master key within 18 LLM rounds while ReAct-RAG never closes the gap.

TABLE XI
ABLATION STUDY OVER TOOLS AND KNOWLEDGE.

	Base				Tools				Knowledge				DKG			
Case	**Rnd**	**Tok**	**T (s)**	**SR**	**Rnd**	**Tok**	**T (s)**	**SR**	**Rnd**	**Tok**	**T (s)**	**SR**	**Rnd**	**Tok**	**T (s)**	**SR**
AES-7816	14.9	155K	2160	40.0%	37.1	816K	800	0.0%	19.1	765K	2179	10.0%	11.7	271K	269	100.0%
AES-14443	14.5	208K	2170	0.0%	28.6	539K	2197	70.0%	20.5	859K	2717	0.0%	13.5	320K	483	100.0%
AES-FPGA	12.3	135K	873	0.0%	18.6	299K	475	80.0%	20.3	760K	1228	0.0%	11.5	256K	277	100.0%
AES-STM32F103	13.1	138K	1477	30.0%	28.7	603K	635	40.0%	19.0	701K	1500	10.0%	14.5	366K	309	100.0%
AES-STM32F429	16.7	194K	1994	70.0%	17.3	244K	443	100.0%	16.5	560K	1302	60.0%	10.6	250K	301	100.0%
DES-7816	14.6	195K	1249	0.0%	32.9	638K	600	10.0%	18.8	729K	1168	0.0%	15.1	365K	309	100.0%
DES-14443	17.1	256K	1284	0.0%	13.2	151K	317	100.0%	20.6	825K	1665	0.0%	16.5	500K	387	80.0%
SM4-7816	20.3	335K	3197	0.0%	28.1	559K	577	50.0%	18.1	642K	1426	0.0%	17.8	417K	365	100.0%
Mean	15.4	202K	1801	17.5%	25.6	481K	756	56.2%	19.1	730K	1648	10.0%	13.9	343K	337	97.5%

SCA tools, and presence of our dual-layer KG. All four configurations share the same LLM backbone (DeepSeek-V3.2); they differ only in which DKG components the agent has access to. *Base* provides only a Python execution environment; *Tools* augments the Python execution environment with specialized SCA tools; *Knowledge* provides a Python execution environment and graph retrieval; *DKG* is the full framework with both tools and structured graph. Table XI reports the per-case results across the eight datasets; Fig. 9 visualizes the SR distribution and Fig. 10 the per-case efficiency distribution.

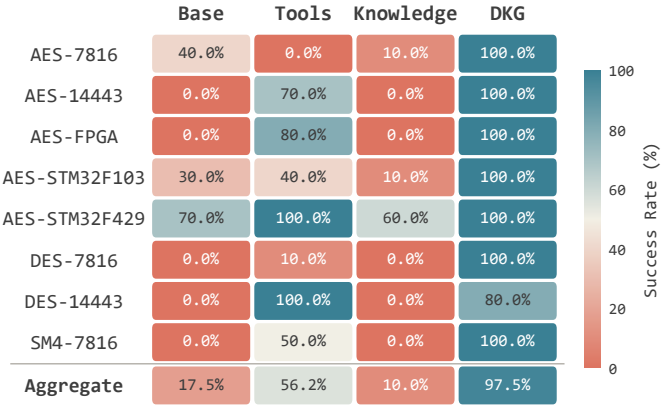


Fig. 9. End-to-end SR by case and configuration.

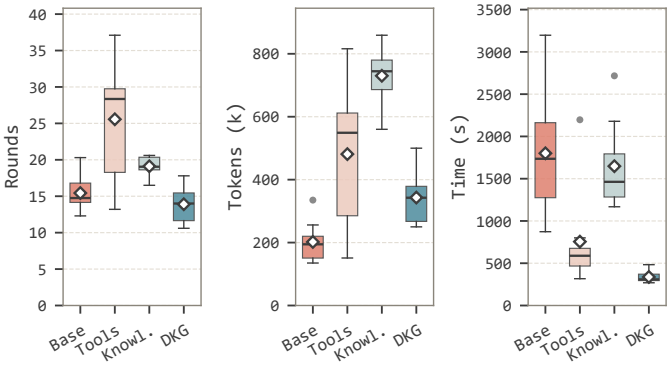


Fig. 10. Per-case efficiency distribution by configuration.

The ablation isolates four design points across the plane. *Base* (Python execution only) reaches 17.5% SR: without specialized SCA tools, the agent implements CPA in Python from scratch and recovers keys mainly on the software AES cases. *Tools* (specialized SCA tools, no graph access) reaches 56.2%. These tools encapsulate attack computations, preprocessing, and key-recovery operations, allowing the agent to focus on selecting, configuring, and coordinating these operations. *Knowledge* (Python execution and graph access) reaches 10.0%. The agent retrieves configuration and diagnostic guidance from the graph but must implement the attack computations in

Python. Recovery succeeds most often on AES-STM32F429 (6/10) and in some runs on AES-STM32F103 and AES-7816, with all remaining cases unsuccessful. Knowledge falls below Base because the graph prescribes a longer expert workflow, including alignment, diagnosis, and key search, which the agent must implement in Python without reliable tools; errors in these steps, such as distorted alignment, offset the guidance. *DKG* (full framework, both tools and structured graph) reaches 97.5%, demonstrating that the two components are complementary: tools provide the procedural foundation, and the dual-layer KG supplies the configuration and diagnostic guidance the agent needs to consistently succeed.

The gap is most pronounced on the three trace-budget-limited cases (AES-7816, DES-7816, SM4-7816, each with only 100 traces): Base reaches 13.3%, Tools 20.0%, and Knowledge 3.3%, while DKG reaches 100.0%. These results highlight the complementary roles of specialized SCA tools and graph-based configuration and diagnostic guidance in recovering keys from the 100-trace cases.

On efficiency, DKG reduces LLM rounds by 45.7% (13.9 vs. 25.6 for Tools), token consumption by 28.7% (343K vs. 481K), and runtime by 55.4% (337 s vs. 756 s), because the agent reaches the correct configuration faster and replans more efficiently with structured guidance than with no knowledge. Base's LLM rounds and tokens (15.4 and 202K) are lower than those of Tools, but its runtime is 2.4× higher (1801 s vs. 756 s). This gap arises because Base runs its own analysis code in the Python execution environment at each round, and this repeated in-loop execution dominates the runtime.

E. Multi-Model Evaluation

To assess DKG's performance across different backbones, we run the full framework with DeepSeek-V3.2, GLM-5, and Qwen3.5-397B-A17B. Each backbone runs the same eight cases with 10 runs per case. Fig. 11 presents SR, LLM rounds, token consumption, and runtime for each backbone.

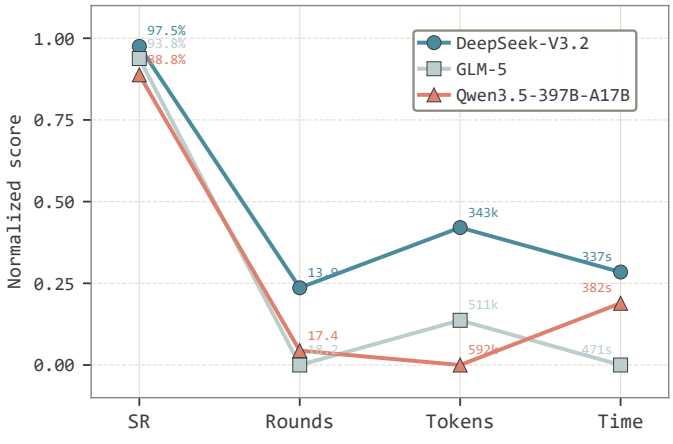


Fig. 11. Multi-model comparison across four metrics.

Across the three backbones, SRs range from 88.8% to 97.5%, with an average of 93.3% over 240 runs. Each attack averages 16.5 LLM rounds, 482K tokens, and 397 s. These results show that DKG maintains high key-recovery SRs across the three backbones. DeepSeek-V3.2 also achieves higher execution efficiency. It averages 13.9 LLM rounds compared with 17.4–18.2 for the other backbones, consumes 343K tokens compared with 511–592K, and takes 337 s compared with 382–471 s. Backbone choice therefore also affects DKG's execution efficiency.

V. Discussion

In this section, we discuss the limitations of DKG and outline several directions for future research.

Limitations. (1) Corpus-driven diagnostic coverage. The diagnostic layer draws on public literature and on practitioner notes in the corpus, and it covers the devices and failure scenarios these sources describe. Issues that rarely appear in either source, such as mains-frequency pickup on low-cost EM probes or thermal drift over multi-hour captures, enter the graph once practitioners document them, because the extraction pipeline processes new notes in the same way as published documents.

(2) Role of the tool suite. DKG runs as a knowledge layer over an expert-engineered tool suite. ReAct-RAG and the Tools configuration share these tools, so their differences from DKG measure the effect of graph-guided decisions. Extending DKG to a new attack setting includes implementing and validating the corresponding tools.

Future work. Future extensions will retain the dual-layer organization of configuration and diagnostic knowledge while incorporating the schema entries, source materials, and execution tools required by each attack setting. Concrete directions include extending DKG to (1) profiled attacks (template, stochastic, deep-learning-based SCA) with associated training tools, and (2) fault-injection and combined fault-SCA scenarios along an orthogonal attack-modality dimension.

VI. Conclusion

We presented DKG, a dual-layer KG and LLM co-driven framework for automated non-profiled side-channel analysis. DKG separates expert knowledge into a configuration space graph and a diagnostic strategy graph, and connects them through cross-layer relationships that translate diagnostic conclusions into concrete configuration adjustments. An LLM-driven three-step extraction pipeline automatically constructs the graph from 166 technical documents, and the constructed graph drives an LLM agent's closed-loop attack workflow through a suite of specialized SCA tools.

On eight real-world datasets covering AES, DES, and SM4 across smartcards, FPGAs, and MCUs, DKG with DeepSeek-V3.2 achieves a 97.5% end-to-end key-recovery SR. ReAct-RAG achieves 68.8%, while the Tools configuration achieves 56.2%. Across DeepSeek-V3.2, GLM-5, and Qwen3.5, the full framework achieves an average SR of 93.3%.

To our knowledge, DKG is the first framework to achieve consistent end-to-end automated key recovery across diverse implementations in the non-profiled SCA setting. More broadly, DKG demonstrates that security-evaluation expertise can be externalized into an inspectable, reusable knowledge

layer, allowing LLM agents to inherit domain-specific operational discipline without embedding that discipline directly into any single model. Future work will extend the same framework to profiled SCA and fault-injection scenarios.

REFERENCES

[1] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[2] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO '99*, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[3] T. Schneider and A. Moradi, "Leakage assessment methodology — a clear roadmap for side-channel evaluations," in *Cryptographic Hardware and Embedded Systems — CHES 2015*, ser. LNCS, vol. 9293. Springer, 2015, pp. 495–513.

[4] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *Cryptographic Hardware and Embedded Systems — CHES 2004*, ser. LNCS, vol. 3156. Springer, 2004, pp. 16–29.

[5] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[6] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 68 539–68 551.

[7] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "PentestGPT: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium*. USENIX Association, 2024, pp. 847–864.

[8] J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li, "AutoAttacker: A large language model guided system to implement automatic cyber-attacks," *arXiv preprint arXiv:2403.01038*, 2024.

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[10] D. A. Boiko, R. MacKnight, B. Kline, and G. Gomes, "Autonomous chemical research with large language models," *Nature*, vol. 624, no. 7992, pp. 570–578, 2023.

[11] X. Shen, L. Wang, Z. Li, Y. Chen, W. Zhao, D. Sun, J. Wang, and W. Ruan, "PentestAgent: Incorporating LLM agents to automated penetration testing," in *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*. ACM, 2025, pp. 375–391, doi: 10.1145/3708821.3733882.

[12] Z. Li, J. Zeng, Y. Chen, and Z. Liang, "AttacKG: Constructing technique knowledge graph from cyber threat intelligence reports," in *Computer Security — ESORICS 2022*, ser. LNCS, vol. 13554. Springer, 2022, pp. 589–609.

[13] Y. Zhang, T. Du, Y. Ma, X. Wang, Y. Xie, G. Yang, Y. Lu, and E.-C. Chang, "AttacKG+: Boosting attack graph construction with large language models," *Computers & Security*, vol. 150, 2025, Art. no. 104220, doi: 10.1016/j.cose.2024.104220.

[14] W. Zhou, A. Wang, Y. Ding, C. Wei, J. Zhang, J. Li, and L. Zhu, "A prompt framework for LLM-based fully automated simple power analysis on cryptosystems," Cryptology ePrint Archive, Paper 2024/2069, 2024.

[15] W. Zhou, A. Wang, Y. Ding, A. Liu, J. Zhang, J. Li, and L. Zhu, "SCA-GPT: A generation-planning-tool assisted LLM agent for fully automated side-channel analysis on cryptosystems," Cryptology ePrint Archive, Paper 2025/1643, 2025.

[16] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems — CHES 2002*, ser. LNCS, vol. 2523. Springer, 2003, pp. 13–28, doi: 10.1007/3-540-36400-5_3.

[17] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering — SPACE 2016*, ser. LNCS, vol. 10076. Springer, 2016, pp. 3–26.

[18] G. Zaid, L. Bossuet, A. Habrard, and A. Venelli, "Methodology for efficient CNN architectures in profiling attacks," *IACR Trans. Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 1, pp. 1–36, 2019.

[19] S. Picek, G. Perin, L. Mariot, L. Wu, and L. Batina, "SoK: Deep learning-based physical side-channel analysis," *ACM Computing Surveys*, vol. 55, no. 11, 2023, Art. no. 227, doi: 10.1145/3569577.

[20] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 7, pp. 3580–3599, 2024.

[21] P. Klein, L. Malburg, and R. Bergmann, "Combining informed data-driven anomaly detection with knowledge graphs for root cause analysis in predictive maintenance," *Engineering Applications of Artificial Intelligence*, vol. 145, 2025, art. no. 110152.

[22] M. Rotmensch, Y. Halpern, A. Tlimat, S. Horng, and D. Sontag, "Learning a health knowledge graph from electronic medical records," *Scientific Reports*, vol. 7, no. 1, 2017, Art. no. 5994, doi: 10.1038/s41598-017-05778-z.

[23] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From local to global: A graph RAG approach to query-focused summarization," *arXiv preprint arXiv:2404.16130*, 2024.

[24] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen, and N. Zhang, "LLMs for knowledge graph construction and reasoning: Recent capabilities and future opportunities," *World Wide Web*, vol. 27, no. 5, 2024, Art. no. 58, doi: 10.1007/s11280-024-01297-w.

[25] C. W. Cleverdon, "The Cranfield tests on index language devices," *Aslib Proceedings*, vol. 19, no. 6, pp. 173–194, 1967.

[26] N. Chinchor, "MUC-7 named entity task definition," MUC-7 task definition, 1997, version 3.5, Sep. 17, 1997. Accessed: Nov. 3, 2026. [Online]. Available: https://www-nlpir.nist.gov/related_projects/muc/proceedings/ne_task.html

[27] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic Web*, vol. 8, no. 3, pp. 489–508, 2017, doi: 10.3233/SW-160218.

[28] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta programming for a multi-agent collaborative framework," in *International Conference on Learning Representations (ICLR)*, 2024.

[29] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, "Voyager: An open-ended embodied agent with large language models," *Transactions on Machine Learning Research*, 2024.

[30] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun, "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," in *International Conference on Learning Representations (ICLR)*, 2024.

[31] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 46 595–46 623.

[32] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, "G-Eval: NLG evaluation using GPT-4 with better human alignment," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, pp. 2511–2522, doi: 10.18653/v1/2023.emnlp-main.153.



Wenquan Zhou received the B.Sc. degree in computer science and technology from Beijing Institute of Technology, Beijing, China, in 2022, where he is currently pursuing the D.Eng. degree with the School of Cyberspace Science and Technology. His research interests include cryptography, side-channel analysis, and large language models.



An Wang is currently an Associate Professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. His main research interests include side-channel analysis, cryptographic engineering, and implementation.



Yaoling Ding was born in 1987. She received her Ph.D. degree from Tsinghua University in 2019. From 2019 to 2021, she worked as a post doctor in Beijing Institute of Technology. She currently works as an Assistant Professor in Beijing Institute of Technology. Her main research interests include side-channel analysis, cryptographic engineering and implementation.



Jiakun Li received the Ph.D. degree from the Institute of Microelectronics, Beijing Institute of Technology, Beijing, China, in 2021. She is currently an Associate Professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology. Her research interests include digital circuit design, VLSI implementation of cryptographic algorithms, and hardware design of cryptographic primitives.



Jingqi Zhang received the B.Eng., M.Sc., and Ph.D. degrees in electronic engineering from Beijing Institute of Technology, Beijing, China, in 2017, 2020, and 2024, respectively, where he is currently a Postdoctoral Fellow with the School of Cyberspace Science and Technology. His research interests include digital circuit design, hardware security, and VLSI implementation of cryptosystems.



Liehuang Zhu (Senior Member, IEEE) is currently a Professor with the School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, China. He was selected into the Program for New Century Excellent Talents in University of the Ministry of Education, China. His research interests include the Internet of Things, cloud computing security, and Internet and mobile security.