Gopher: High-Precision and Deep-Dive Detection of Cryptographic API Misuse in the Go Ecosystem

Yuexi Zhang、Bingyu Li*、Jingqiang Lin、Linghui Li、Jiaju Bai、Shijie Jia、Qianhong Wu

CCS 2024

目录

01

研究背景及现状

研究背景

密码学API误用普遍存在:

- 开发者经验不足
- 文档不清晰
- 生成式AI误导

Go语言在安全关键场景广泛应用，但误用威胁数据安全:

- 应用场景广泛
- 误用风险加剧
- 生态亟需革新

研究现状

现有的相关工作:

- 静态分析：CryptoLint、CogniCryptoSAST、CryptoGuard、LICAM、TAINTCRYPT、CryptoGo、CryptoREX
- 动态分析：SMV-Hunter、AndroSSL、K-Hunt、Crylogger
- 机器学习和评估工作

现有工具失效:

- 规则局限：依赖黑白名单，规则需手动编写
- 静态分析局限：CryptoGo仅支持官方库，无法追踪跨库数据流
- 高误报/漏检：CryptoGo误报率17.3%，漏检64.1%的跨库误用（如封装函数的盐值长度不足）

研究背景

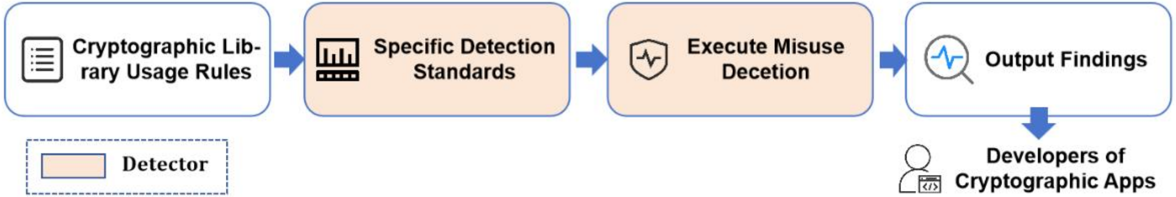

Figure 2: The Framework of Typical Cryptographic API Misuse Detection Tools.

Table 7: Tools for Detecting Cryptographic Misuse in Source Code Across Various Programming Languages.

Name	Method	Lang.	$\#_{Test}$	$\#_{Eco}$	$\#_{Rule}$	Rule Source	Scalable Range †	Cross-Library ‡	Context-, Flow-Sensitive ≀
Crylogger [34]	Dynamic	Java	150	1,780	26	Hard-coded	✗	✓	✗
Xu et al.'s tool [48]	HMM	Java	(3,953)	-	-	Training data	✓	✗	✗
CogniCrypt$_{SAST}$ [27]	Static	Java	50	8,422	7	Manual	✓	✗	✓
CryptoGuard [37]	Static	Java	46	6,181	16	Hard-coded	✗	✗	✓
LICMA [45]	Static	Python	-	946	5	Hard-coded	✗	✗	✗
TAINTCRYPT [36]	Static	C/C++	5	-	(15)	Manual	✓	✗	✓
CryptoGo [29]	Static	Go	120	-	12	Hard-coded	✗	✗	✗
Gopher	Static	Go	145	19,313	19	Manual+Derived	✓	✓	✓

$\#_{Test}$: the number of test projects evaluated in-depth for misuse, where *Xu et al.'s tool* uses the analysis results from [27] that have not been fully verified as the benchmark.
$\#_{Eco}$: the number of test projects for ecosystem assessment or large-scale measurement.
$\#_{Rule}$: the number of misuse rules or types. *TAINTCRYPT* provided detection methods for 15 rules, but due to the diversity of APIs, they only evaluated 5 rules in the experiment.
†: the cryptographic API detection scope can expand without re-development; ‡: supports cross-library detection; ≀: the detection field-/flow-sensitive.

02

Gopher工具

Gopher框架

两大核心组件:

CryDict（约束描述与动态生成）：将自然语言规则转化为标准化约束，通过数据流分析自动推导新约束

Detector（静态误用检测）：基于SSA的程序切片、数据流分析（字段/流/上下文敏感），将CryDict约束映射到具体检测逻辑

运行流程:

规则总结->约束构建->执行检测->约束推导->动态更新

Gopher框架

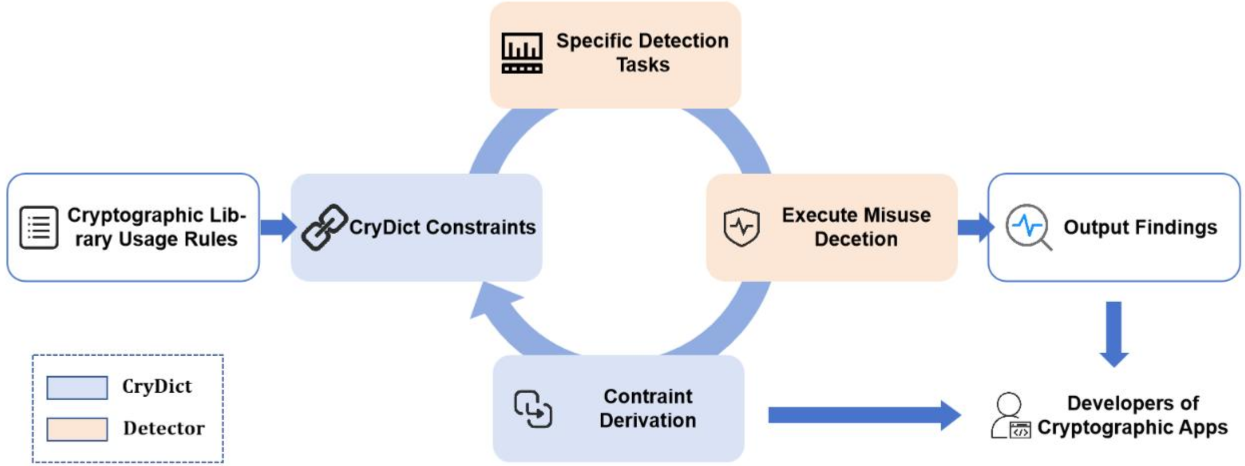

Figure 3: The Framework of Gopher.

CtyDict

检测规则:

- 学术研究
- 权威指南

谓词系统:

- 一元谓词（Unary）：单个参数或字段
- 二元谓词（Binary）：比较两个值

Table 1: Misuse Scenario of the Go Cryptographic Library.

ID	Misuse Scenario	Severity	Predicate
R01	Low salt length in password hash	L	BYTE_LENGTH
R02	Too short key (e.g., 64-bit key in HMAC)	L	BYTE_LENGTH
R03	Parameter with too small scale (e.g., RSA-512)	L	GEQ
R04	Low iteration parameter in password hash	L	GEQ
R05	Dangerous algorithm (e.g., DES)	H	Function Detector
R06	Warning algorithm (e.g., 3DES)	L	Function Detector SECURE_HMAC_HASH
R07	Use predictable IO to generate parameters	M	RANDOM_IO
R08	Predictable/constant salt in password hash	L	RANDOM_BYTES
R09	Key is predictable/constant/used	H	RANDOM_BYTES
R10	IV in CBC/CFB is predictable/constant	M	RANDOM_BYTES
R11	IV in OFB/CTR/GCM is constant	M	NOT_CONST
R12	Use HTTP protocol	H	HTTPS
R13	Skip the certificate verification in TLS	H	EQ_FALSE
R14	Not verify the client in SSH	H	EQ_FALSE
R15	Using deprecated TLS suites (e.g., TLS_RSA_WITH_AES_128_CBC_SHA256)	H	SECURE_TLS_SUITE
R16	Outdated signature algorithm in TLS (e.g., PKCS1WithSHA1)	H	SECURE_TLS_SUITE
R17	Outdated TLS version (< tls1.2)	H	SECURE_TLS_SUITE GEQ
R18	Insecure suite in SSH (e.g., diffie-hellman-group1-sha1)	H	SECURE_SSH_SUITE
R19	Deprecated functions in go std library	L	Function Detector

Table 8: The Predicates Used in CryDict.

Predicate Name	Type	Meaning	Misuse Scenario
BYTE_LENGTH	Binary	Defines the minimum length of a byte slice.	Incorrect parameter configuration
EQ_FALSE	Unary	Specifies that the value must be false.	Skipping verification in SSH/TLS
GEQ	Binary	Sets a minimum value for a number parameter.	Small parameter values
HTTPS	Unary	Requires the use of HTTPS connections.	HTTP connections
RANDOM_BYTES	Unary	Cryptographically secure random byte slices.	Predictable key/IV
RANDOM_IO	Unary	Cryptographically secure random source.	Use of insecure PRNG
NOT_CONST	Unary	Prohibits constant values for slices.	Use of constant IV in OFB/CTR/GCM
SECURE_HMAC_HASH	Unary	Indicates secure hash for HMAC.	Weak algorithms, e.g., HMAC-MD5
SECURE_TLS_SUITE	Unary	Specifies the use of secure TLS suites.	Use of insecure TLS suites
SECURE_SSH_SUITE	Unary	Specifies the use of secure SSH suites.	Use of insecure SSH suites

CtyDict

设计原则：

- 黑名单机制
- 参数独立性
- 并发条件满足（AND逻辑）
- 工具解耦

约束规范：

- 函数级约束
- 参数级约束
- 字段级约束

约束推导：

- 符号执行与数据流传播：符号化参数、数据流追踪、约束生成
- 跨库检测优化：加载被调用库的CryDict约束，避免重复分析，仅验证调用是否符合约束

CtyDict

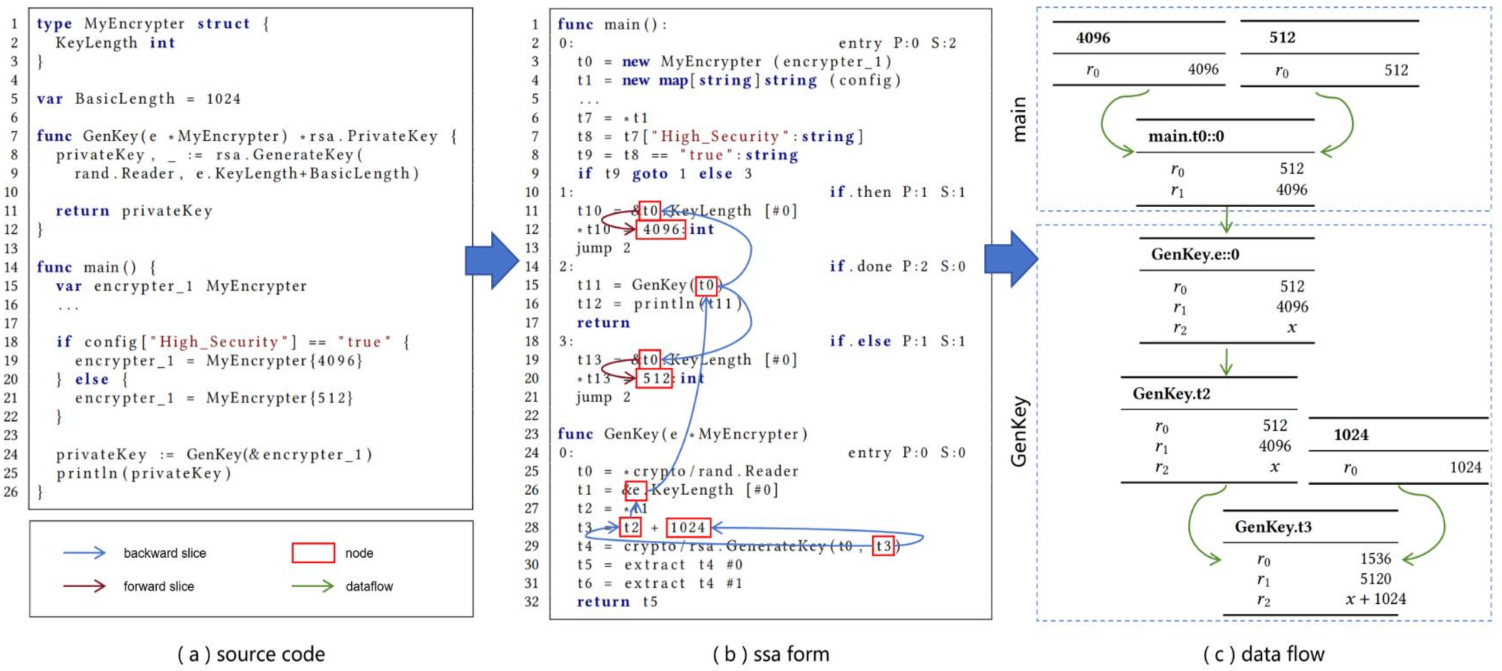

Figure 6: An Example of Detecting the Parameter Scale of RSA Algorithm.

分析demo1.GenKey内部调用rsa.GenerateKey，发现bits参数由KeyLength + 1024计算；符号化KeyLength为变量x，推导$x + 1024 \geqslant 3072 \rightarrow x \geqslant 2048$

e.keylength约束更新：3072->2048

跨库验证：其他项目调用demo1.GenKey时，直接检查参数是否满足GEQ(2048)

CtyDict

参数独立性原则的局限性：

问题：CryDict无法表达“参数间相互补偿”的规则

实际：存在理论局限，但CryDict的简化设计在实际测试中表现优异，在基准中未发现因参数独立性导致的误报

符号执行与循环路径的限制：

问题：循环中的变量可能产生无限多路径

实际：限制每个节点的数据传播次数；忽略复杂循环路径，仅推导“明显安全”的约束

检测器

设计目标：高精度、跨库支持、动态适配

静态分析框架：

- 中间表示（IR）转换
- 程序切片（Program Slicing）

数据流分析：

- 稀疏数据流传播（Sparse Data-flow Propagation）：仅分析影响切片标准的节点
- 符号执行（Symbolic Execution）：参数符号化，推导约束条件

检测器

跨库调用处理：

- 类层次分析（Class Hierarchy Analysis, CHA）：构建函数调用图，识别接口实现
- 约束预加载：调用外部库时，加载其CryDict约束文档，避免重复分析库内代码

误报优化策略：

Go语言特性利用（OP1）：
错误处理机制过滤：若变量可能为nil或空值（如salt = ""），检查是否被if err != nil拦截

03

实验评估

数据集和对比试验

来源：Go官方包仓库，90326->19313（2023年后、go.mod文件、可编译）

小规模测试：145个，高星（＞20k）、高引用（750+）、普通（随机选择）

对比实验（Gopher vs. CryptoGo）

准确性对比：规则、分解、四个优化策略

性能对比：51.5vs47.1 秒/项，牺牲9%时间换取更高精度

约束推导：42个误用（占总数4.3%）通过约束推导发现

Table 2: Detection Results on 145 Small-scale Test Projects.

	Tools	Shared Rules		R19		New Rules		Total		
		$\#_{TP}$	$\#_{FP}$	$\#_{TP}$	$\#_{FP}$	$\#_{TP}$	$\#_{FP}$	$\#_{TP}$	$\#_{FP}$	$\#_{Dr}$
S_i	Gopher	298 (98%)	5	27	0	25	0	350	5	26
	CryptoGo	207 (78%)	58	8	0	0	0	215	58	–
S_s	Gopher	448 (99%)	2	28	0	42	2	518	4	20
	CryptoGo	321 (76%)	102	2	0	0	0	323	102	–
S_o	Gopher	123 (98%)	2	17	0	24	0	164	2	3
	CryptoGo	85 (76%)	31	6	0	0	0	91	31	–

S_i, S_s, and S_o: the 145 test projects selected from Go projects that are highly-imported, highly-starred, and of ordinary categories.
Shared Rules: the rules defined by both tools (13 rules: R03, R05-R13, R15-R17).
New Rules: the new rules introduced by *Gopher* (5 rules: R01, R02, R04, R14, R18).
$\#_{TP}$, $\#_{FP}$: the number of true positives, false positives in a tool's output.
$\#_{Dr}$: the number of extra misuses discovered by the constraint derivation feature.

Table 3: Breakdown of Accuracy on Small-scale Test Projects.

Rules	# of Projects	# of Alerts	$\#_{TP}$	$\#_{Dr}$
R01	4 (2.76%)	9	9 (100%)	1 (11.11%)
R02	5 (3.45%)	6	6 (100%)	0 (0.00%)
R03	30 (20.69%)	79	77 (97.47%)	10 (12.66%)
R04	31 (21.38%)	57	55 (96.5%)	2 (3.51%)
R05*	84 (57.93%)	415	415 (100%)	N/A
R06*	19 (13.10%)	40	40 (100%)	N/A
R07	1 (0.69%)	1	1 (100%)	1 (100.00%)
R08	8 (5.52%)	8	7 (87.5%)	1 (12.50%)
R09	12 (8.28%)	19	17 (89.4%)	1 (5.88%)
R10	3 (2.07%)	5	5 (100%)	2 (40.00%)
R11	1 (0.69%)	2	2 (100%)	2 (100.00%)
R12	22 (15.17%)	82	82 (100%)	12 (14.63%)
R13	52 (35.86%)	148	144 (97.3%)	10 (6.76%)
R14	8 (5.52%)	10	10 (100%)	0 (0.00%)
R15	2 (1.38%)	21	21 (100%)	0 (0.00%)
R16	0	0	N/A	N/A
R17	7 (4.83%)	20	20 (100%)	0 (0.00%)
R18	1 (0.69%)	5	5 (100%)	0 (0.00%)
R19*	26 (17.93%)	70	70 (100%)	N/A
Total	**N/A**	**997**	**986 (98.9%)**	**42**

$\#_{TP}$: the number of true positives *Gopher*'s output.
$\#_{Dr}$: the number of misuses discovered by the constraint rule derivation feature.
*: the constraint derivation is disabled for these rules to avoid unnecessary alerts.

Table 4: Breakdown of the Reduction of False Positives in 145 Small-scale Test Projects via Four Optimization Strategies.

Method	Num	Percentage	Method	Num	Percentage
OP1	162	36.40%	OP3	25	5.62%
OP2	226	50.79%	OP4	32	7.19%
Total	445				

实验结果

整合CryDict与CryptoGo（兼容性，可移植性，领域泛化）：

规则R03（RSA密钥长度）的检测实例从49例增至77例（+57.1%）

CryptoGo首次具备检测R04（密码哈希迭代次数）规则的能力，检出50例

误用总体分布

普遍性：67.31%项目存在至少1次误用，平均2.8误用/项目

高发问题：R05（危险算法）、R19（废弃函数）、R13（跳过TLS验证）

跨库误用特征

1,121次跨库调用中，9.1%存在误用

约束推导覆盖率：约束推导发现1,745例（6.45%）误用，R08、R09、R11影响最深刻（R07？）

Table 5: Distribution of Misuse in Go Ecosystem.

Rules	# of Alerts		$\#_{Dr}$	Rules	# of Alerts		$\#_{Dr}$
	Total	H-Star			Total	H-Star	
R01	376	36	24 (6.38%)	R11	335	23	71 (21.19%)
R02	987	39	48 (4.87%)	R12	3,639	653	360 (9.89%)
R03	2,775	295	158 (5.70%)	R13	5,522	790	112 (2.03%)
R04	5,929	388	206 (3.47%)	R14	1,278	110	0 (0.00%)
R05*	18,668	1,705	N/A	R15	725	107	0 (0.00%)
R06*	2,207	145	N/A	R16	8	0	0 (0.00%)
R07	52	6	11 (21.15%)	R17	574	84	15 (2.61%)
R08	1,011	53	205 (20.28%)	R18	196	18	0 (0.00%)
R09	3,136	167	499 (15.92%)	R19*	6,236	402	N/A
R10	503	9	36 (7.16%)	**Total**	**54,157**	5,027	**1,745**

H-Star: highly-stared projects (with over 1,000 GitHub stars).
*: the constraint derivation is disabled for these rules to avoid unnecessary alerts.

实验结果

Table 6: Partial Security Findings with Acknowledged, Fixed and Assigned CVE IDs.

CVE-ID	CVSS Score	Misuse Rules	Project Type	Project Popularity (#*Github Stars*)
CVE-2024-40464 CVE-2024-40465	8.8 8.8	R13 R05	Web Framework	31,324
CVE-2024-41253	7.1	R13	Web Framework	11,260
CVE-2024-41259	9.1	R05	Music Server	11,020
CVE-2024-41260	7.1	R10	SSO, MFA	10,271

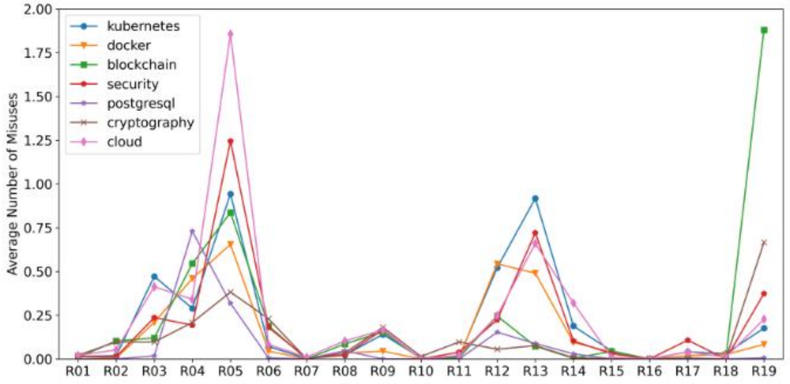

Figure 10: Characteristics of Misuse Distribution Across Application Types.

04

局限性和未来工作

局限性

假阳性（False Positives）

- CryDict的表达能力限制

问题：CryDict为简化约束描述，牺牲了部分表达能力，导致极少数场景下误报
示例：scrypt函数参数组合（如N=2^16, r=16, p=2）可能被误判为不安全
影响：在145个项目测试中，未发现因此类问题导致的误报，风险极低

- 路径不敏感（Path Insensitivity）

问题：静态分析未完全覆盖条件分支，导致不可达路径误报
示例：循环中初始化密钥数组前的空值误判（3例）、RSA密钥长度被误设为0的假设路径（7例）
缓解：通过优化策略（如OP1）过滤部分误报，但未完全消除

- 调用图构建（CHA的局限性）

问题：基于类层次分析（CHA）的动态分派可能误判函数调用关系
示例：Ed25519与RSA共享接口时，密钥生成函数映射错误（1例）
影响：假阳性的发生率最小，与使用更复杂的图表程序呼叫图相关的大量时间成本相比，此权衡是合理的

局限性

假阴性（False Negatives）

- 非Go代码分析缺失：工具仅分析Go源码，忽略CGO等外部代码的密码学调用
- 约束推导范围限制：仅支持扩展官方库的封装API，无法处理完全自研的密码库

约束推导的适用性

- 当前限制：仅针对参数配置类规则（如R03、R08），不覆盖危险函数调用（如R05）
- 设计权衡：避免对危险函数封装产生冗余告警（如func1()调用rc4时仅标记底层函数）

局限性和未来工作

假阴性（False Negatives)

- 非Go代码分析缺失：工具仅分析Go源码，忽略CGO等外部代码的密码学调用
- 约束推导范围限制：仅支持扩展官方库的封装API，无法处理完全自研的密码库

约束推导的适用性

- 当前限制：仅针对参数配置类规则（如R03、R08），不覆盖危险函数调用（如R05)
- 设计权衡：避免对危险函数封装产生冗余告警（如func1()调用rc4时仅标记底层函数)

未来工作

- 路径敏感分析：引入符号执行提升条件分支覆盖
- 混合分析：结合动态验证（如Crylogger）减少误报
- 自研库支持：探索基于LLM的约束生成（如从文档自动提取规则)

演讲完毕 感谢聆听