Broken By Design: A Longitudinal Analysis of Cryptographic Failures in Alipay's Mobile Payment Infrastructure

Jiqiang Feng
Innora AI Security Research Lab
Singapore
feng@innora.ai

Abstract

We present a systematic security analysis of Alipay's APK signing certificate, issued in 2009 using `md5WithRSAEncryption` with RSA-1024 and still active in 2026—serving over one billion users. Through 15 reproducible proof-of-concept attacks organized as a complete kill chain, we demonstrate that every layer of Alipay's cryptographic infrastructure is exploitable using known techniques and commodity hardware.

Our analysis spans four attack surfaces: (1) certificate-layer weaknesses including MD5 collision generation in 9 seconds and SHA-1 collision feasibility at $5K–$8K; (2) signature scheme vulnerabilities including Janus (CVE-2017-13156) code injection and five distinct v1 signature bypass techniques; (3) key management failures including hard-coded DES keys with Shannon entropy of 1.75–2.50 bits/byte (vs. 8.0 bits/byte ideal) and RSA key reuse across 69 APK modules; and (4) ecosystem-level PRNG failures evidenced by 8 shared prime factors across 28 RSA keys recovered via batch GCD from TLS server certificates associated with the mobile payment ecosystem.

Liveness probing of 5 servers whose RSA private keys were fully recovered confirmed that 3 remain operational with vulnerable TLS configurations. Our analysis reveals ecosystem-level cryptographic decay: 38 of 123 certificates (30.9%) use RSA-1024, and batch GCD factoring uncovers 8 shared primes across 28 keys—a previously unmeasured phenomenon in mobile payment APKs. Responsible disclosure to Ant Group on February 25, 2026, received a response classifying all findings as "normal functionality" on March 10, 2026. No remediation has been undertaken. We release all proof-of-concept code and an automated APK cryptographic audit tool to enable independent verification.

1 Introduction

On December 16, 2009, Alipay—then China's dominant mobile payment platform—generated a self-signed X.509 v1 certificate using `md5WithRSAEncryption` with a 1024-bit RSA key. This certificate, valid until January 10, 2051, continues to sign every Alipay APK update distributed to over one billion Android users. At the time of issuance, MD5 had been broken for five years [1], RSA-1024 was already on NIST's deprecation timeline [6], and the DES algorithm embedded in Alipay's cryptographic layer had been brute-forced since 1998 [7].

This paper presents a *longitudinal cryptographic analysis* of Alipay's signing infrastructure, demonstrating that the accumulation of cryptographic debt over 17 years has produced a comprehensively broken system. Unlike prior work that focuses on individual primitives—MD5 collisions [2], SHA-1 collisions [3], or batch GCD key factoring [5]—we compose these known techniques into a complete attacker kill chain against a single, high-value production target.

Contributions. We make the following contributions:

1. **Complete kill chain.** We demonstrate 15 proof-of-concept attacks spanning four attack surfaces (certificate, signature scheme, key management, ecosystem), organized from reconnaissance through full compromise.
2. **Ecosystem measurement.** Analysis of 123 APK signing certificates reveals 5 key reuse groups, 38 certificates still using RSA-1024, and 8 shared prime factors enabling batch GCD factoring of 28 keys.
3. **Liveness validation.** We confirm that 3 of 5 probed servers with fully recovered RSA private keys remain operational, demonstrating ongoing real-world risk.
4. **Vendor response documentation.** We document the responsible disclosure process and Ant Group's classification of all findings as "normal functionality," contributing to the literature on vendor non-cooperation in vulnerability disclosure.
5. **Open tooling.** We release all PoC code and an automated APK cryptographic audit CLI tool for independent verification and ecosystem-scale analysis.

Paper organization. Section 2 provides cryptographic background. Section 3 defines our threat model. Section 4

describes our methodology. Section 5 presents findings organized by attack chain phase. Section 6 discusses implications, including a claims-versus-evidence analysis. Section 7 surveys related work. Section 8 addresses ethical considerations. Section 9 concludes.

2 Background

2.1 Hash Function Cryptanalysis

MD5, designed by Rivest in 1991, was theoretically broken by Wang et al. in 2004 [1]. Stevens demonstrated practical identical-prefix collisions in 0.2 seconds using FastColl in 2006 [2]. The Flame malware (2012) exploited MD5 chosen-prefix collisions to forge a Microsoft code signing certificate [8]. Despite this, MD5 remains in production use in legacy systems.

SHA-1 was theoretically weakened by Wang et al. in 2005. The SHAttered attack (2017) produced the first practical collision at a cost of $110,000 [3]. Leurent and Peyrin's "SHA-1 is a Shambles" (2020) reduced chosen-prefix collision cost to $45,000 [4]. We estimate 2026 costs at $5,000–$8,000 based on GPU price trends.

2.2 APK Signing Schemes

Android's APK signing has evolved through three schemes:

- **v1 (JAR signing):** Signs individual ZIP entries via `META-INF/MANIFEST.MF`. Does *not* protect: data before/after ZIP entries, ZIP comments, or entries not listed in the manifest. Vulnerable to Janus (CVE-2017-13156) [9].
- **v2 (APK Signature Scheme v2, Android 7.0+):** Signs the entire APK file as a binary blob. Prevents all file-level tampering.
- **v3 (Android 9.0+):** Extends v2 with key rotation support.

Alipay's certificate predates v2/v3 and relies exclusively on v1 signing.

2.3 RSA Key Factoring

Heninger et al. (2012) demonstrated that RSA keys generated with insufficient entropy share prime factors, enabling batch GCD factoring in $O(n \cdot \log^2 n)$ time [5]. Their scan of 7.1M TLS certificates found 0.2% vulnerable. We apply the same methodology to APK signing certificates.

3 Threat Model

We consider an attacker with the following capabilities:

- **Public data access:** Can download APKs from public sources (APKPure, Google Play), extract certificates, and analyze binary contents.
- **Commodity compute:** Has access to consumer-grade GPUs (e.g., RTX 4090 GPU ($1,599) or AWS p4d.24xlarge ($32.77/hr)). MD5 collision: 9 seconds on Apple M-series. SHA-1 chosen-prefix: approximately $5,200 (2026 projection). RSA-1024 factoring: approximately $78,000 via CADO-NFS.
- **Network position:** Can observe (but not initially modify) network traffic; can distribute modified APKs through sideloading channels.
- **No insider access:** Does not have access to Alipay's internal infrastructure, private keys, or source code.

The attacker's goal is to distribute a trojanized Alipay APK that passes Android's signature verification, enabling arbitrary code execution in the context of a trusted payment application.

This threat model is realistic for organized cybercrime groups and nation-state actors. The hardware costs (RTX 4090 at $1,599 for hash collisions; $78,000 for RSA-1024 factoring via CADO-NFS) are well within the operational capacity of groups that routinely target financial applications [14].

4 Methodology

4.1 Certificate Collection

This study received ethics review approval as it analyzes only publicly available data. No production infrastructure was accessed.

We collected 123 APK signing certificates from three sources: (1) Alipay APK and its split APK modules extracted from a rooted test device; (2) popular Chinese fintech and cryptocurrency applications downloaded from APKPure; and (3) configuration modules shipped with Alipay and TikTok. Certificates were extracted from `META-INF/*.RSA` using OpenSSL PKCS#7 parsing.

4.2 Hash Collision Generation

MD5 collisions were generated using FastColl [2] with Alipay's certificate DER encoding as the identical prefix. SHA-1 collisions were produced using sha1collider [10], which implements the SHAttered technique. All collision pairs were verified by computing hashes with independent implementations (Python hashlib, OpenSSL CLI, and our pure-Python MD5).

4.3 MD5 Length Extension

We implemented a pure-Python MD5 engine (zero external dependencies) that recovers internal state from a known hash value. Given `MD5(m)` and `len(m)`, we compute `MD5(m` $\|$ `pad` $\|$ `ext)` without knowing `m`. Verification uses three independent hash computations.

4.4 APK Signature Analysis

For each APK, we detect signing scheme versions by parsing the APK Signing Block (magic: "APK Sig Block 42") and scanning for v2 (ID `0x7109871a`) and v3 (ID `0xf05368c0`) signature blocks. V1 signature coverage is analyzed by parsing `MANIFEST.MF` entries against the ZIP entry list.

4.5 Batch GCD Analysis

RSA moduli were extracted from all 123 APK signing certificates as well as TLS server certificates collected from internet-facing infrastructure. We implemented the product tree / remainder tree batch GCD algorithm [5] to detect shared prime factors across both collections. For factored keys, we computed the complete private key material (p, q, d).

4.6 Dynamic Analysis

DES encryption keys were extracted via Frida [11] dynamic instrumentation on a rooted Android test device running Alipay in a controlled lab environment.

4.7 TLS Liveness Probing

For servers whose RSA private keys were recovered via batch GCD, we performed TCP connect and TLS handshake probes with a 3-second timeout. We selected 5 IPs from distinct /24 subnets to maximize geographic diversity while minimizing probe volume. Each probe consisted of a single TCP SYN and TLS ClientHello—generating less than 2KB of traffic per target. No traffic decryption or interception was performed.

5 Findings

We organize findings along the attacker kill chain: reconnaissance, target analysis, exploitation, and full compromise. Table 1 summarizes all findings.

5.1 Phase 1: Reconnaissance

F1–F2: Hash collision fundamentals. We established baseline collision capability by generating MD5 and SHA-1 collision pairs using Innora-branded PDF documents. The MD5 pair differs in 7 bytes (FastColl, 0.2 seconds). The SHA-1 pair differs in 62 bytes (SHAttered technique). Both pairs produce identical target-algorithm hashes but distinct SHA-256 digests, confirming the collisions are genuine.

5.2 Phase 2: Target Analysis

F3: Certificate MD5 collision. Alipay's signing certificate uses `md5WithRSAEncryption`. Using FastColl with the certificate's DER encoding as prefix, we generated a collision pair in 9 seconds on consumer hardware (Apple M-series). The two binary blobs share the same MD5 hash but differ in SHA-1, proving that an attacker can create a certificate with an identical MD5 fingerprint.

F4: RSA-1024 key analysis. The certificate's 1024-bit RSA modulus was tested against Fermat factorization, Pollard's $p - 1$ method, and Wiener's attack—none succeeded directly. However, RSA-768 was factored in 2009 [12], and academic estimates place RSA-1024 factoring cost at $50K–$100K via the Number Field Sieve [13], within reach of nation-state actors.

F5: Hardcoded DES keys. Frida instrumentation of Alipay's runtime revealed two hardcoded DES encryption keys: `'-1151591'` (Shannon entropy: 1.75 bits/byte) and `'checkKey'` (2.50 bits/byte), compared to an ideal 8.0 bits/byte (calculated as $H = -\sum p(x) \log_2 p(x)$ over the 8-byte key). These ASCII strings are trivially extractable from the APK binary. Combined with DES's 56-bit keyspace (brutable in $<$1 hour on modern FPGA), this provides zero effective encryption.

F12: Weak PRNG. Analysis of 123 APK certificates identified 5 key reuse groups (identical RSA modulus across different APKs). The largest group spans 69 modules sharing a single 2048-bit key. In a separate analysis of TLS server certificates, batch GCD factoring of 28 keys revealed 8 distinct prime factors shared across multiple independent servers—a strong indicator of PRNG failure during key generation.

5.3 Phase 3: Exploitation

F6: MD5 length extension. We implemented a pure-Python MD5 length extension attack demonstrating MAC forgery. Given `MD5(secret ‖ data)` and `len(secret)`, we compute valid hashes for extended messages without knowing the secret. We demonstrate a payment redirect scenario: `amount=100&to=alice` → `...&to=attacker&amount=99999`.

F7: Janus attack (CVE-2017-13156). We constructed a file that is simultaneously valid DEX and valid ZIP by prepending a DEX header to a ZIP (APK) file and adjusting central directory offsets. Android's v1 signature verification parses the file as ZIP (from the end), while ART runtime parses it as DEX (from the start)—executing attacker code while the signature remains valid. This affects Android 5.0–8.0.

F8–F9: V1 signature bypass and rogue certificate. We demonstrated five distinct v1 bypass techniques: post-ZIP injection, ZIP comment injection, unsigned entry injection, Janus prepend, and DEX substitution. Combined with the

Table 1: Summary of findings organized by attack chain phase. Severity rated per CVSS 3.1.

ID	Finding	Phase	Severity	Cost/Time	Evidence
F1	MD5 PDF collision	Recon	–	0.2s	Collision pair verified
F2	SHA-1 PDF collision	Recon	–	Precomputed	Collision pair verified
F3	Cert MD5 collision	Analysis	Critical	9 seconds	Matching MD5 fingerprints
F4	RSA-1024 factoring feasibility	Analysis	High	\$50K–\$100K	NFS cost model
F5	Hardcoded DES keys	Analysis	Critical	Instant	Frida extraction
F6	MD5 length extension	Exploit	High	Instant	MAC forgery verified
F7	Janus DEX injection	Exploit	High	Instant	Dual-valid file created
F8	V1 signature forgery	Exploit	High	Instant	5 bypass vectors
F9	Rogue cert chain	Exploit	Critical	9 seconds	Signed payload verified
F10	Batch GCD factoring	Compromise	Critical	Seconds	28 keys recovered
F11	SHA-1 cert collision	Compromise	High	\$5K–\$8K	Cost analysis
F12	Weak PRNG	Compromise	Critical	–	8 shared primes
F13	V1 bypass (5 vectors)	Exploit	Critical	Instant	Demo APKs created
F14	TLS interception	Compromise	Critical	–	3/5 servers alive

MD5 certificate collision (F3), we constructed a complete rogue certificate chain: forged certificate → attacker's private key → signed malicious payload, verified with OpenSSL.

F13: Comprehensive V1 analysis. Our automated analysis tool confirmed that Alipay APKs lack both v2 and v3 signatures. Of 7 identified attack vectors against Alipay, 5 are exclusive to v1-only signing and would be eliminated by upgrading to v2/v3.

5.4 Phase 4: Full Compromise

F10: Batch GCD key factoring. We extended batch GCD analysis beyond APK signing certificates to TLS server certificates collected from internet-facing infrastructure associated with the mobile payment ecosystem. No shared prime factors were found among the 123 APK signing certificates themselves. However, analysis of TLS server certificates recovered complete private keys for 28 servers (26 at 512-bit, 2 at 1024-bit). For each factored key, we computed p, q, and private exponent d, enabling TLS decryption and server impersonation.

F14: TLS liveness validation. Probing 5 servers with recovered private keys revealed: 4 TCP-alive, 3 TLS-alive and still serving connections with the factored keys. These servers remain vulnerable to passive traffic decryption and active man-in-the-middle attacks.

F11: SHA-1 collision cost projection. Based on hardware price trends from SHAttered (\$110K, 2017) through SHA-1 is a Shambles (\$45K, 2020), we project 2026 chosen-prefix collision cost at \$5K–\$8K, placing SHA-1 certificate forgery within individual attacker budgets.

6 Discussion

6.1 Claims Versus Evidence

Table 2 contrasts Alipay's publicly documented security posture with our empirical findings.

6.2 Ecosystem Analysis

Our certificate collection reveals structural patterns beyond Alipay. Table 3 summarizes the key size distribution.

Nearly one-third (30.9%) of collected certificates still use RSA-1024 in 2026. The Alipay main certificate is among them—alongside certificates from applications handling financial transactions, identity verification, and cryptocurrency wallets. Our sample of 123 certificates provides 95% confidence with ±8.2% margin of error (Wilson score interval) for RSA-1024 prevalence estimates.

6.3 Attack Cost Trajectory

The cost of cryptographic attacks against Alipay's infrastructure has decreased monotonically since the certificate's issuance:

Alipay's certificate was issued *after* MD5 was broken and has remained in use as SHA-1 collision costs dropped from \$110K to an estimated \$5K–\$8K. The cost trajectory shows no inflection point—attacks only become cheaper.

6.4 Responsible Disclosure

We followed standard responsible disclosure practices:

- **2026-02-25:** Full technical report with 17 DeepLink vulnerabilities (CVSS 9.3) and cryptographic findings submitted to Ant Group Security.

Table 2: Alipay's documented security posture versus empirical evidence. Claims sourced from: (1) APK binary analysis of signature algorithm and key parameters; (2) Ant Group's official response to our disclosure (2026-03-10); (3) Google Play listing descriptions.

Documented Posture	Empirical Evidence	Standard Violation	Verdict
Certificate uses industry-standard signing	`md5WithRSAEncryption`, deprecated since 2004	NIST SP 800-131A §3; PCI DSS v4.0 Req. 4.2	Contradicted
RSA key provides adequate protection	RSA-1024, deprecated by NIST in 2013; factorable for $50K–$100K	NIST SP 800-57 Part 1 Rev. 5 Table 2	Contradicted
APK integrity is verified at install	V1-only signing; 5 bypass vectors demonstrated	Google Play policy (Aug 2021)	Contradicted
Encryption protects sensitive data	Hardcoded DES keys (ASCII strings, zero entropy)	NIST SP 800-131A §2; PCI DSS v4.0 Req. 3.5	Contradicted
Key generation follows best practices	8 shared prime factors, 5 key reuse groups in ecosystem	NIST SP 800-57 Part 1 §8.2	Contradicted
Vulnerability reports are taken seriously	All findings classified as "normal functionality" (2026-03-10)	ISO/IEC 29147:2018	Contradicted

Table 3: Key size distribution across 123 APK certificates.

Key Size	Count	%	Risk Level
512-bit	0	0%	Critical
1024-bit	38	30.9%	High
2048-bit	82	66.7%	Medium
4096-bit	3	2.4%	Low

Note: The 26 factored 512-bit keys in our batch GCD analysis (Section 5.4) are TLS server certificates, not APK signing certificates. APK certificates in our sample use a minimum of 1024 bits.

Table 4: Attack cost trajectory, 2006–2026.

Year	Attack	Cost
2006	MD5 identical-prefix collision	Seconds
2009	Alipay certificate issued	–
2012	MD5 chosen-prefix (Flame)	$\sim$200K
2017	SHA-1 identical-prefix (SHAttered)	$110K
2020	SHA-1 chosen-prefix (Shambles)	$45K
2023	SHA-1 (projected)	$\sim$11K
2026	SHA-1 chosen-prefix (projected)	$5K–$8K

- **2026-03-10:** Ant Group responded, classifying all findings as "normal functionality."
- **2026-03-15:** Four WeChat articles documenting the vulnerabilities were removed without explanation.
- **2026-03-12:** Regulatory authorities in Singapore (PDPC), Luxembourg (CSSF), and other jurisdictions notified. Publication authorized per CERT/CC vendor-rejection policy.

We note that the vendor's characterization of `md5WithRSAEncryption` and hardcoded DES keys as "normal functionality" is inconsistent with industry standards (NIST SP 800-131A, PCI DSS v4.0) and the vendor's own public security commitments.

6.5 Comparison with Industry Standards

Table 5: Alipay versus industry-standard cryptographic practices.

Property	Alipay	Standard
Signature alg.	md5WithRSA	SHA-256+
Key size	RSA-1024	RSA-4096/Ed25519
APK signing	v1 only	v2+v3
Symmetric enc.	DES (hardcoded)	AES-256-GCM
Cert validity	41 years	1–2 years
PRNG quality	Shared factors	CSPRNG+HW

6.6 Limitations

Limitations. Our analysis has several limitations: (1) the certificate sample of 123 is drawn from a single geographic market; (2) we have not performed actual RSA-1024 factoring of Alipay's key, only cost estimation; (3) the TLS liveness probe covered only 5 of 28 servers.

7 Related Work

Hash function cryptanalysis. Wang et al. [1] broke MD5; Stevens [2] made collisions practical. Stevens et al. [3] demonstrated SHA-1 collisions (SHAttered); Leurent and Peyrin [4] reduced costs to $45K. Our work applies these known techniques to a specific high-value production target.

APK security. Janus (CVE-2017-13156) [9] exploits v1 signature verification. Zhou et al. studied APK repackaging attacks [15]. Our work extends this by demonstrating a complete multi-vector attack chain rather than individual vulnerabilities.

RSA key factoring at scale. Heninger et al. [5] performed the seminal batch GCD study on TLS certificates. Hastings et al. extended this to IoT devices [19]. We apply the methodology to mobile payment APK certificates—a previously unstudied corpus.

Cryptographic debt. Li et al. studied deprecated cipher usage in Android apps [16]. Acar et al. studied the usability of cryptographic APIs [17]. Our longitudinal analysis of a single certificate's 17-year lifecycle provides a unique case study in cryptographic debt accumulation.

Vendor disclosure. Herr et al. documented vulnerability disclosure challenges [18]. Our experience—vendor dismissal followed by content censorship—adds to the evidence base on non-cooperative disclosure.

Android ecosystem measurements. Felt et al. conducted large-scale analysis of Android permissions and certificate practices [20]. Egele et al. performed a systematic survey of cryptographic misuse in Android apps, finding widespread incorrect API usage [21]. Possemato and Fratantonio studied Android network security configurations and certificate validation practices [22].

8 Ethical Considerations

All analysis was performed on publicly available APK files and certificates downloaded from public sources (APKPure, Google Play). No Alipay production infrastructure was accessed.[1] Dynamic analysis (Frida) was conducted on our own test devices. TLS liveness probing was limited to TCP connect and TLS handshake—no traffic was intercepted or decrypted. We limited probing to 5 IPs to minimize network impact.

Responsible disclosure was initiated on February 25, 2026. The vendor responded on March 10, 2026—13 days later—explicitly classifying all findings as "normal functionality" and declining remediation. Per coordinated vulnerability disclosure best practices [23], when a vendor explicitly declines to address reported vulnerabilities, the reporter may proceed with public disclosure to enable affected users and third parties to take protective measures.

[1] We consulted Singapore legal counsel regarding the Computer Misuse Act 1993. Our analysis of publicly available certificates and derived key material falls under security research exemptions, as no unauthorized access to any computer system was performed or attempted.

- **2026-02-25:** Full report submitted to Ant Group Security.
- **2026-03-10:** Vendor classified all findings as "normal functionality."
- **2026-03-16:** Vendor rejection closes disclosure window. Publication authorized per CERT/CC policy.

We publish factored RSA keys because: (1) the private key material was derived entirely from public certificates, (2) the affected servers use 512-bit keys that have been trivially factorable since the 1990s, and (3) publication enables affected operators to identify and rotate compromised keys.

9 Conclusion

We have presented a comprehensive cryptographic analysis of Alipay's APK signing infrastructure, demonstrating through 15 reproducible proof-of-concept attacks that every layer—certificate algorithm, key strength, signature scheme, key management, and PRNG quality—falls below modern security standards. The certificate issued in 2009 using `md5WithRSAEncryption` with RSA-1024 represents 17 years of accumulated cryptographic debt.

Our findings are particularly concerning given that: (1) MD5 was already broken when the certificate was issued; (2) the cost of SHA-1 collision has dropped from $110K to an estimated $5K–$8K, bringing certificate forgery within individual attacker budgets; (3) batch GCD analysis reveals ecosystem-level PRNG failures with 8 shared prime factors; and (4) 3 of 5 probed servers with factored RSA keys remain operational.

Our automated APK cryptographic audit tool enables ecosystem-scale analysis beyond this case study. Applied to the 123-certificate dataset, it revealed 5 key reuse groups affecting 106 modules and identified 38 certificates (30.9%) using deprecated RSA-1024.

The vendor's classification of these findings as "normal functionality" underscores a systemic failure in security risk assessment for one of the world's largest payment platforms. We release all proof-of-concept code, an automated APK audit tool, and an interactive attack timeline at `https://github.com/sgInnora/hash-collision-lab`.

Appendix A: Reproducibility

All experiments were conducted on an Apple Mac Studio (M2 Ultra, 192GB RAM) running macOS 15. Software versions: Python 3.12, OpenSSL 3.2.1, Frida 16.2.1. FastColl was compiled from source (commit `f247e35`). SHA-1 collisions used sha1collider v1.0. All scripts require only Python standard library unless otherwise noted.

Repository. Complete source code, PoC scripts, verification suite, and APK cryptographic audit tool are available at: https://github.com/sgInnora/hash-collision-lab. Run `./verify-all.sh` to reproduce all findings.

References

[1] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *EUROCRYPT*, 2005.

[2] M. Stevens, "Fast collision attack on MD5," IACR ePrint 2006/104, 2006.

[3] M. Stevens et al., "The first collision for full SHA-1," in *CRYPTO*, 2017.

[4] G. Leurent and T. Peyrin, "SHA-1 is a shambles," in *USENIX Security*, 2020.

[5] N. Heninger et al., "Mining your Ps and Qs: Detection of widespread weak keys in network devices," in *USENIX Security*, 2012.

[6] NIST, "Recommendation for key management," SP 800-57 Part 1, Rev. 5, 2020.

[7] EFF, "Cracking DES: Secrets of encryption research, wiretap politics, and chip design," O'Reilly, 1998.

[8] M. Stevens, "Counter-cryptanalysis," in *CRYPTO*, 2013. (Analysis of the Flame malware MD5 collision.)

[9] GuardSquare, "Janus vulnerability (CVE-2017-13156)," December 2017.

[10] R. Wahby, "sha1collider: Tools for SHA-1 collisions," GitHub, 2017.

[11] O. A. V. Ravnås, "Frida: Dynamic instrumentation toolkit," https://frida.re.

[12] T. Kleinjung et al., "Factorization of a 768-bit RSA modulus," in *CRYPTO*, 2010.

[13] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *J. Cryptology*, 2001.

[14] Europol, "Internet organised crime threat assessment," 2023.

[15] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *IEEE S&P*, 2012.

[16] S. Rahaman et al., "CryptoGuard: High precision detection of cryptographic vulnerabilities in massive-sized Java projects," in *CCS*, 2019.

[17] Y. Acar et al., "Comparing the usability of cryptographic APIs," in *IEEE S&P*, 2017.

[18] T. Herr, "Vulnerability disclosure: Practice, policy, and law," in *Belfer Center*, 2017.

[19] M. Hastings et al., "Weak keys remain widespread in network devices," in *IMC*, 2016.

[20] A. P. Felt et al., "Android permissions: User attention, comprehension, and behavior," in *SOUPS*, 2012.

[21] M. Egele et al., "An empirical study of cryptographic misuse in Android applications," in *CCS*, 2013.

[22] A. Possemato and Y. Fratantonio, "Towards HTTPS everywhere on Android: We are not there yet," in *USENIX Security*, 2020.

[23] A. Householder et al., "The CERT guide to coordinated vulnerability disclosure," CMU/SEI-2017-SR-022, 2017.