

Post-Quantum Crypto in a Go Trojan: A Garble-Obfuscated ASUS DLL Ships ML-KEM, a Fabricated DigiCert Chain, and Azure + Firebase C2 — 8 Hours After First VT Submission

 intel.breakglass.tech/post/wingo-post-quantum-garble-trojan-asus-dll-sideload

Breakglass Intelligence

April 9, 2026



MEDIUM

THREAT INTEL

Post-Quantum Crypto in a Go Trojan: A Garble-Obfuscated ASUS DLL Ships ML-KEM, a Fabricated DigiCert Chain, and Azure + Firebase C2 — 8 H...

Malware analysis · C2 infrastructure · APT attribution

@BreakGlassIntel

TL;DR

On **April 9, 2026 at 04:17 UTC**, a Go-compiled Windows DLL trojan was submitted to VirusTotal by [SecuriteInfoCom](#) at 15/76 detections, tagged by ESET as [WinGo/Agent_AGen.ACA](#) and by Microsoft as [Trojan:Win32/Wacatac.B!ml](#). The DLL masquerades as an ASUSTeK "Gaming Host Detection" utility (file name [GamingHostDetection.dll](#), 3.57 MB), is obfuscated with [Garble](#), and carries a **fabricated DigiCert Authenticode signature** dated 2026-04-07.

We pulled it apart in the hours after first submission. The interesting part isn't the masquerade — that's routine. It's what's in the crypto stack:

- **ChaCha20-Poly1305** AEAD with **HKDF** key derivation
- **AES (CBC + CTR modes)** for configuration storage
- **ML-KEM** — NIST's standardized post-quantum key encapsulation mechanism — wired into a Go package that exposes [Encapsulate](#) / [Decapsulate](#)

Implementing **post-quantum key exchange** in a Windows commodity-looking trojan is genuinely unusual. Post-quantum crypto adoption in malware is essentially nonexistent in the public record right now. Seeing ML-KEM alongside ChaCha20-Poly1305 inside a Garble-obfuscated Go DLL suggests the author is implementing a **hybrid classical + post-quantum** key exchange for C2 forward secrecy — deliberately hardening against the possibility of future cryptanalysis (or current signals-intelligence intercept + store-now-decrypt-later capability).

What this report adds to the public record:

- A first static analysis of the sample (SHA256 `328e097831c74290f102dcf11c41bc4b7b82a37681d8df3690783ea8a9e3bebc`) with the crypto stack, the obfuscated-package → real-package mapping, and decoded Caesar-shifted paths
- Documentation of **ML-KEM post-quantum crypto inside commodity-framed malware** — a detail we could not find in any prior public writeup on Go trojans, Garble-based samples, or the `WinGo/Agent_AGen` family
- The fabricated DigiCert chain details (serial, thumbprint, SAN) and the dual-cloud C2 infrastructure (Azure Web Apps + Firebase Hosting) with the certificate-pinning setup embedded in the binary
- A breakdown of the three-layer configuration encoding (**custom base85** → **base64** → **ChaCha20-Poly1305 AEAD**) so other analysts can unwrap the config if they recover a sample with live C2

Hat tip to **SecuriteInfoCom** for the VT submission. If you've already published reporting on this specific sample, on ML-KEM in malware, on the `xzRkBBlpsGnp.azurewebsites.net` C2, or on the `WinGo/Agent_AGen` family more broadly, please reply or DM — we'll update and credit.

The Sample

Field	Value
Filename (meaningful)	<code>GamingHostDetection.dll</code>
SHA256	<code>328e097831c74290f102dcf11c41bc4b7b82a37681d8df3690783ea8a9e3bebc</code>
MD5	<code>85425de806a7373a22ce968eef1a561d</code>
SHA1	<code>8930835f206921ae0db300f2ea7c69655e57523a</code>
Size	3,566,624 bytes (3.57 MB)
Type	PE32+ DLL (console), x86-64
Imphash	<code>3271ee162568f50a6810be9b8973807f</code>
Authentihash	<code>54c4d3d318017005f0f9faa89a3d04dcdfa348238fb6dfe72ba40356dd6d447d</code>

Field	Value
Compile timestamp	2025-07-12 (inconsistent with the 2026-04-07 signing cert — likely falsified)
VT first submission	2026-04-09 04:17:54 UTC
VT detections	15/76
Reporter	SecuriteInfoCom

Alternate filenames seen on VT: `SecuriteInfo.com.Win64.MalwareX-gen.71844241.exe`, `ck-3d80df5d12cdf6450a782fc87bf66b444.google`, `payload.bin`, `malware.sample`, `3ztdichcw.exe`. The meaningful name `GamingHostDetection.dll` only appears in a subset of submissions — this DLL is **built for side-loading**, not standalone execution.

Structural Tells

It's Go 1.20+

`pcIntab` magic `0xFFFFFFFF1` sits at raw offset `0x279CC0`. That confirms Go 1.20 or later — the language runtime version where Garble became reliably usable against modern Go builds.

```
Sections: .text (entropy 6.52) .rdata (5.53) .data (3.72)
          .pdata (5.50) .gfids .rsrc .reloc (5.42)
Functions: 3,932 total (316 user, ~3,616 stdlib)
Source files: 162 (all paths obfuscated by Garble)
Packages: 30+ (all renamed to random strings)
```

The Garble fingerprint

Go packages in the binary have been renamed by Garble to unpronounceable strings:

```
sZWQT2 · tyjyopi · xRLot07 · kGRVel · P4sFVai · kso04J ·
tzpPTgFg5x · ivgngaXawd80 · rdDc50kIdN · rafqW5_WRmuj ·
EdyAMKC · Pa23ZEj · AAg1tAT01StX
```

Those aren't semantic names — they're the random-string output of Garble's obfuscation pass. Function names, type names, and source file paths are all stripped. The `\xff Go buildinf:` magic is stripped too, so standard tools that identify Go builds fail out of the gate.

But method signatures still reveal the underlying package identity. When you see `bTRSLc2zrTh.(*GLZvwf4e).BlockSize + .Encrypt` in the disassembly, that's `crypto/aes`. When you see `E2NMDz1.Encapsulate / E2NMDz1.Decapsulate`, that's `crypto/kem`. We walked about 30 of these back to their real package identities via method-signature matching.

Decoded Caesar-shifted paths

Garble applies a **per-string Caesar cipher** to filesystem-path constants with **varying shift values per string**. Brute-forcing the per-string shift recovers the operational targets:

Obfuscated	Decoded	Shift
E:\Wugtu\Rwdnke\vtkcig_ycnnrcrgt.lri	C:\Users\Public\triage_wallpaper.jpg	2
J:\Dpukvdz\Zfzalt32\uakss.kss	C:\Windows\System32\ntdll.dll	7
J:\Dpukvdz\Tpjyvvma.ULA	C:\Windows\Microsoft.NET	7
H:\Bnsitbx\XdxBTB64\wzsiqq32.jcj	C:\Windows\SysWOW64\rundll32.exe	5
I:\Cotjucy\YeyCUC64\jrrnuyz.kdk	C:\Windows\SysWOW64\dllhost.exe	6
M:\Gsxnygc\Cicdow32\zvknvv	C:\Windows\System32\pla.dll	10
K:\Eqvlwea\AgaEWE64\uafut3.ltts	C:\Windows\SysWOW64\msxml3.dll	8

The `triage_wallpaper.jpg` reference is the giveaway: that file exists on [Hatching Triage](#) and [Any.Run](#) sandbox VMs specifically. Checking for it is a one-line hardcoded environment check that any analyst-run sandbox trips immediately.

The two DLL exports

Ordinal 1	CfgGetCacheW	– Windows-API-style name, likely the sideload entry point
Ordinal 2	NTjmeGDYnYYCwrvZB	– Garble-randomized fallback

CfgGetCacheW is named to resemble a legitimate Windows configuration API function so a host process that's been tricked into loading the wrong DLL will call it without suspicion. Sandbox execution paths observed in VT behavior data:

```
rundll32.exe "GamingHostDetection.dll",CfgGetCacheW
rundll32.exe "GamingHostDetection.dll",NTjmeGDYnYYcWrvZB
rundll32.exe "GamingHostDetection.dll",#1
```

The ASUSTeK masquerade

CompanyName	ASUSTek COMPUTER INC.
FileDescription	Gaming Host Detection
FileVersion	3.4.12.0
InternalName	GamingHostDetection.dll
OriginalFilename	GamingHostDetection.dll
ProductName	Gaming Host Detection
LegalCopyright	©ASUSTeK Computer Inc. All rights reserved.

Version info wants you to think this ships with ASUS gaming software. The Windows file properties dialog will display this exact metadata, right down to the copyright.

The Crypto Stack

This is where the sample gets interesting. Mapped from the obfuscated Go packages back to their real identities:

Obfuscated	Real package	Purpose
bTRSLc2zrTh	crypto/aes + crypto/cipher	AES block cipher (CBC + CTR)
kXs3jAhPTI	golang.org/x/crypto/chacha20poly1305	AEAD with HKDF key derivation
B6dntbkqbRQ	crypto/sha256	Hashing for key derivation
kPrvc3	crypto/sha512	Extended hashing
Ca4m1RAHZUK	crypto/hmac	Message authentication
E2NMDz1	crypto/kem (ML-KEM)	Post-quantum key encapsulation
HBYlMK	crypto/rand	Cryptographic PRNG

ML-KEM, in plain language

ML-KEM (Module-Lattice Key Encapsulation Mechanism) is the **NIST-standardized post-quantum key encapsulation algorithm**, finalized in August 2024 as **FIPS 203**. It replaces the classical elliptic-curve + RSA key exchange that TLS, SSH, and basically every network protocol

on earth currently rely on. Its whole purpose is to resist attacks from **future quantum computers** that could break RSA and ECC but can't efficiently break lattice problems.

Implementing ML-KEM in a Go library is easy — the filippo.io/mlkem768 package and the stdlib [crypto/mlkem](#) (added in Go 1.24) both expose [Encapsulate](#) and [Decapsulate](#).

Implementing it **inside a commodity-framed malware sample** is not something we have seen in public writeups before.

Why an actor would do this

There are two plausible reasons:

1. **Store-now-decrypt-later resistance.** An operator who believes nation-state-level SIGINT agencies are capturing their C2 traffic today and might break it later with a quantum computer would want to prevent retroactive compromise of captured sessions. Hybrid ML-KEM + X25519 key exchange gives you forward secrecy **even against a future quantum adversary**.
2. **Operational signaling.** The author wants to demonstrate to a customer (or the malware research community) that they're implementing modern cryptographic best practice. Post-quantum crypto is an active topic in CTI circles; putting it in your trojan makes your trojan look sophisticated to anyone who reverses it.

Either reason points at a **well-resourced author** who reads cryptographic standards and tracks NIST publications. That's not the typical behavior pattern for commodity-crimeware authors, which normally reuse AES-256-CBC with a hardcoded key and call it a day.

The three-layer configuration encoding

The encrypted config lives at [.rdata + 0x1DB404](#) and uses a custom stack:

1. Custom base85 encoding (charset 0x21-0x75, 5 chars → 4 bytes)
2. Entries prefixed with "11:" (2 header entries, 24 bytes nonce/key material) and "12:" (46 data entries)
3. Base64 decoding yields 414 bytes of binary ciphertext
4. ChaCha20-Poly1305 AEAD decrypt with HKDF-derived key

Plus the [.data](#) section embeds an **AES S-box** at [0x33E420](#) and four **AES T-tables** at [0x33E56C–0x3401B0](#) (4 × 1020 bytes), plus ~8 KB of additional encrypted data blocks at [0x33D900–0x3435E4](#). The actor is precomputing T-tables for AES rather than computing them at runtime — another sign the build pipeline was written by someone who understands performance optimization inside a cryptographic library, not just glued one together.

The Fabricated DigiCert Chain

CN: xzRkBB1psGnp.azurewebsites.net
SAN: *.azurewebsites.net ← wildcard for every Azure App Service
Issuer: "DigiCert Global G2 TLS RSA SHA256 2020 CA1" ← self-signed fake
Created: 2026-04-07
Validity: 2026-04-07 → 2026-07-07 (90 days)
Status: INVALID – chain terminates in untrusted self-signed root

Two details worth noting:

1. **Code Signing key usage in a TLS-style cert.** Legitimate DigiCert TLS certificates don't carry **Code Signing** in **Extended Key Usage**. Including both Server Auth + Client Auth + Code Signing in one cert is something only a self-signed template generator would do — and it's a trivial single-field check for detection tooling to flag.
2. **The CN subdomain xzRkBB1psGnp matches the Garble random-string generator.** Garble produces random package names like sZWQT2, P4sFVai, tyjyopi. The Azure subdomain xzRkBB1psGnp fits the same distribution — 12 mixed-case alphanumeric characters with no vowel structure. The actor's C2 domain registrar tooling and their malware obfuscation tooling probably share a random-string source.

The timestamp-vs-cert gap is also suspicious: the PE compilation timestamp reads **2025-07-12** but the signing certificate was created **2026-04-07** — a nine-month gap. Either the PE timestamp was forged (common), or the binary is template-based and gets re-signed per deployment with a fresh per-customer cert. We favor the second reading given the operational maturity of the crypto stack.

The C2 Infrastructure

Primary: xzRkBB1psGnp.azurewebsites.net (Azure Web Apps)

Platform	Azure App Service
DNS	Not resolving as of 2026-04-09 04:30 UTC
Cert created	2026-04-07 16:56:51 UTC
Cert validity	2026-04-07 → 2026-07-07 (90 days)
TLS pinning	DER cert embedded in binary
crt.sh	No CT log entries found

Secondary: **0AfNve0a.web.app** (Firebase Hosting)

Platform	Firebase Hosting (Google Cloud)
DNS	199.36.158.100 (Google/Fastly, AS54113)
HTTP	404 – "Site Not Found"
Cert	Google Trust Services
Cert validity	2026-04-06 → 2026-07-06
Cert serial	AAD2F0B1A1DE74557FFA6D6F9EF46170
SHA1	26:44:73:DD:E1:E0:05:2B:EC:D0:46:C4:A6:8D:4F:A8:AD:11:A6:1F
TLS pinning	DER cert embedded in binary
CDN	Served by Fastly (x-served-by: cache-lga21985-LGA)

Both C2 domains have **DER-encoded TLS certificates embedded in the binary** — the trojan does cert pinning, so MITM interception by defenders with a private CA inserted into the client trust store won't work. That's a technique seen in sophisticated state-level implants and effectively blocks TLS inspection.

The infrastructure is classic **cloud-provider abuse for C2**:

- **Redundancy** — if one provider takes the C2 down, the other persists
- **Ephemeral** — 90-day cert validity windows, disposable cloud projects
- **Shared CDN/IP pools** — you can't IP-block *.azurewebsites.net or *.web.app without breaking huge swathes of legitimate traffic
- **Domain-pattern evasion** — random-string subdomains under trusted parent domains

Both domains were stood up in the **48 hours preceding the sample's first VT submission** — Firebase on Apr 6, Azure on Apr 7, sample uploaded Apr 9.

Attribution

What the evidence supports

Signal	Read
ML-KEM + ChaCha20-Poly1305 + HKDF hybrid crypto	Author reads cryptographic standards, prioritizes forward secrecy
Garble obfuscation + per-string Caesar cipher	Author understands Go-specific reverse-engineering tooling
TLS cert pinning via embedded DER	Defeats enterprise TLS inspection
Dual-cloud C2 with 90-day rotation	Operationally mature infrastructure posture
Triage / Any.Run sandbox detection	Author knows the dominant commercial sandbox ecosystem

Signal

Read

Fabricated DigiCert chain with Code Signing ECU anomaly Tooling gap — template generator left an artifact

What it doesn't

We can't confidently name a threat actor. There are **zero campaign siblings** via VT pivoting (vhash, ssdeep, imphash, authentihash, signer name). There are no matches in ThreatFox or URLhaus for the C2 domains. The **WinGo/Agent_AGen** detection name is a generic label, not a specific family with established attribution.

What we can say: this is **not** a typical cybercrime information-stealer or access-broker build. The crypto sophistication, the infrastructure posture, and the author's evident attention to forward secrecy against future quantum adversaries all point at a **well-resourced actor** — potentially a state-adjacent group, a red-team tooling vendor, or a new commercial implant framework. The **absence of campaign siblings** is consistent with a **per-target build** with unique infrastructure per victim, which is a deployment model associated with targeted intrusion rather than broad cybercrime.

Detection

YARA — build-pattern match

The combination of **Go pcIntab magic + PE64 DLL with single-character exports + Authenticode signature with **azurewebsites.net** CN + Code Signing ECU** is a tight, low-false-positive fingerprint. You can write a YARA rule against:

- PE file type + x86-64 + DLL characteristics
- Embedded bytes **0xFFFFFFFF1** (Go pcIntab magic) in **.text**
- Authenticode subject containing **.azurewebsites.net**
- Authenticode ECU set containing both **1.3.6.1.5.5.7.3.1** (Server Auth) and **1.3.6.1.5.5.7.3.3** (Code Signing)

Network

- Alert on HTTPS connections originating from DLLs loaded via **rundll32.exe** to ***.azurewebsites.net** subdomains matching a **[A-Za-z0-9]{12}** random-string pattern
- Block **xzRkBB1psGnp.azurewebsites.net** and **0AfNve0a.web.app** at the perimeter
- Monitor Firebase ***.web.app** subdomains with certs from Google Trust Services that have 90-day validity windows and no associated crt.sh history outside the ordinary Firebase provisioning pattern

Endpoint

- Hunt for `GamingHostDetection.dll` **outside** `C:\Program Files\ASUS\*` installation paths
- Alert on `rundll32.exe` loading a DLL with an Authenticode subject matching `*.azurewebsites.net`
- Detect the `C:\Users\Public\trriage_wallpaper.jpg` path being referenced by any non-analyst process — it's a strong anti-sandbox check that should never appear in legitimate software

Behavioral

- Go DLLs loaded via `rundll32.exe` with long sleep periods and environment checks
- Process creation chains: `legitimate_asus_gaming_app.exe` → `rundll32.exe` `GamingHostDetection.dll`
- TLS handshakes from processes loaded via `rundll32.exe` to cloud subdomains with DER-pinned certificates

MITRE ATT&CK

T1195.002 · T1569 · T1106 · T1036.001 · T1027 · T1140 · T1553.002 · T1497 · T1574.002 · T1071.001 · T1573 · T1102 · T1012 · T1055 · T1583.006 · T1588.003

IOCs

Sample

SHA256	328e097831c74290f102dcf11c41bc4b7b82a37681d8df3690783ea8a9e3bebc
MD5	85425de806a7373a22ce968eef1a561d
SHA1	8930835f206921ae0db300f2ea7c69655e57523a
Size	3,566,624 bytes
Type	PE32+ DLL x86-64, Go 1.20+, Garble-obfuscated

C2 infrastructure

xzRkBBlpsGnp.azurewebsites.net	Primary C2 (Azure Web Apps, DNS offline)
0AfNve0a.web.app	Secondary C2 (Firebase Hosting)
199.36.158.100	0AfNve0a.web.app → Google/Fastly (AS54113)

Fabricated signing cert

CN	xzRkBBlpsGnp.azurewebsites.net
SAN	*.azurewebsites.net
Issuer	"DigiCert Global G2 TLS RSA SHA256 2020 CA1" (self-signed fake)
Valid	2026-04-07 → 2026-07-07
EKU	Server Auth + Client Auth + Code Signing (← anomaly)

Pinned Firebase cert

Serial AAD2F0B1A1DE74557FFA6D6F9EF46170
SHA1 26:44:73:DD:E1:E0:05:2B:EC:D0:46:C4:A6:8D:4F:A8:AD:11:A6:1F
Issuer Google Trust Services
Valid 2026-04-06 → 2026-07-06

Sandbox evasion artifact

C:\Users\Public\triage_wallpaper.jpg Triage / Any.Run sandbox fingerprint

DLL masquerade metadata

CompanyName ASUSTek COMPUTER INC.
FileDescription Gaming Host Detection
FileVersion 3.4.12.0
ProductName Gaming Host Detection
Filename GamingHostDetection.dll

Exports

Ordinal 1 CfgGetCacheW
Ordinal 2 NTjmeGDYnYYcWrvZB

Disclosure

- **Microsoft Azure abuse** — xzRkBB1psGnp.azurewebsites.net
- **Google Firebase abuse** — 0AfNve0a.web.app
- **ASUS PSIRT** — for awareness that their product name + copyright is being used in a masquerade DLL

Share 

Related Reports

[high](#)
[Multi-RAT Operation Dismantled: WaterHydra APT Nexus, Five AES Keys Recovered, and Live C2 Infrastructure Mapped Across Three Continents](#)
[3/7/2026](#)