Go 1.24 Release Notes

Table of Contents

Introduction to Go 1.24

The latest Go release, version 1.24, arrives in February 2025, six months after Go 1.23. Most of its changes are in the implementation of the toolchain, runtime, and libraries. As always, the release maintains the Go 1 promise of compatibility. We expect almost all Go programs to continue to compile and run as before.

Changes to the language

Go 1.24 now fully supports generic type aliases: a type alias may be parameterized like a defined type. See the language spec for details. For now, the feature can be disabled by setting `GOEXPERIMENT=noaliastypeparams`; but the `aliastypeparams` setting will be removed for Go 1.25.

Tools

Go command

Go modules can now track executable dependencies using `tool` directives in go.mod. This removes the need for the previous workaround of adding tools as blank imports to a file conventionally named “tools.go”. The `go tool` command can now run these tools in addition to tools shipped with the Go distribution. For more information see the documentation.

The new `-tool` flag for `go get` causes a tool directive to be added to the current module for named packages in addition to adding require directives.

The new `tool` meta-pattern refers to all tools in the current module. This can be used to upgrade them all with `go get tool` or to install them into your GOBIN directory with `go install tool`.

Executables created by `go run` and the new behavior of `go tool` are now cached in the Go build cache. This makes repeated executions faster at the expense of making the cache larger. See #69290.

The `go build` and `go install` commands now accept a `-json` flag that reports build output and failures as structured JSON output on standard output. For details of the reporting format, see `go help buildjson`.

Furthermore, `go test -json` now reports build output and failures in JSON, interleaved with test result JSON. These are distinguished by new `Action` types, but if they cause problems in a test integration system, you can revert to the text build output with GODEBUG setting `gotestjsonbuildtext=1`.

The new `GOAUTH` environment variable provides a flexible way to authenticate private module fetches. See `go help goauth` for more information.

The `go build` command now sets the main module's version in the compiled binary based on the version control system tag and/or commit. A `+dirty` suffix will be appended if there are uncommitted changes. Use the `-buildvcs=false` flag to omit version control information from the binary.

The new GODEBUG setting `toolchaintrace=1` can be used to trace the `go` command's toolchain selection process.

Cgo

Cgo supports new annotations for C functions to improve run time performance. `#cgo noescape cFunctionName` tells the compiler that memory passed to the C function `cFunctionname` does not escape. `#cgo nocallback cFunctionName` tells the compiler that the C function `cFunctionName` does not call back to any Go functions. For more information, see the cgo documentation.

Cgo currently refuses to compile calls to a C function which has multiple incompatible declarations. For instance, if `f` is declared as both `void f(int)` and `void f(double)`, cgo will report an error instead of possibly generating an incorrect call sequence for `f(0)`. New in this release is a better detector for this error condition when the incompatible declarations appear in different files. See #67699.

Objdump

The objdump tool now supports dissassembly on 64-bit LoongArch (`GOARCH=loong64`), RISC-V (`GOARCH=riscv64`), and S390X (`GOARCH=s390x`).

Vet

The new `tests` analyzer reports common mistakes in declarations of tests, fuzzers, benchmarks, and examples in test packages, such as malformed names, incorrect signatures, or examples that document non-existent identifiers. Some of these mistakes may cause tests not to run. This analyzer is among the subset of analyzers that are run by `go test`.

The existing `printf` analyzer now reports a diagnostic for calls of the form `fmt.Printf(s)`, where `s` is a non-constant format string, with no other arguments. Such calls are nearly always a mistake as the value of `s` may contain the `%` symbol; use `fmt.Print` instead. See #60529. This check tends to produce findings in existing code, and so is only applied when the language version (as specified by the go.mod `go` directive or `//go:build` comments) is at least Go 1.24, to avoid causing continuous integration failures when updating to the 1.24 Go toolchain.

The existing `buildtag` analyzer now reports a diagnostic when there is an invalid Go major version build constraint within a `//go:build` directive. For example, `//go:build go1.23.1` refers to a point release; use `//go:build go1.23` instead. See #64127.

The existing `copylock` analyzer now reports a diagnostic when a variable declared in a 3-clause "for" loop such as `for i := iter(); done(i); i = next(i) { ... }` contains a `sync.Locker`, such as a `sync.Mutex`. Go 1.22 changed the behavior of these loops to create a new variable for each iteration, copying the value from the previous iteration; this copy operation is not safe for locks. See #66387.

GOCACHEPROG

The `cmd/go` internal binary and test caching mechanism can now be implemented by child processes implementing a JSON protocol between the `cmd/go` tool and the child process named by the `GOCACHEPROG` environment variable. This was previously behind a GOEXPERIMENT. For protocol details, see the documentation.

Runtime

Several performance improvements to the runtime have decreased CPU overheads by 2–3% on average across a suite of representative benchmarks. Results may vary by application. These improvements include a new builtin `map` implementation based on Swiss Tables, more efficient memory allocation of small objects, and a new runtime-internal mutex implementation.

The new builtin `map` implementation and new runtime-internal mutex may be disabled by setting `GOEXPERIMENT=noswissmap` and `GOEXPERIMENT=nospinbitmutex` at build time

respectively.

Compiler

The compiler already disallowed defining new methods with receiver types that were cgo-generated, but it was possible to circumvent that restriction via an alias type. Go 1.24 now always reports an error if a receiver denotes a cgo-generated type, whether directly or indirectly (through an alias type).

Linker

The linker now generates a GNU build ID (the ELF `NT_GNU_BUILD_ID` note) on ELF platforms and a UUID (the Mach-O `LC_UUID` load command) on macOS by default. The build ID or UUID is derived from the Go build ID. It can be disabled by the `–B none` linker flag, or overridden by the `–B 0xNNNN` linker flag with a user-specified hexadecimal value.

Bootstrap

As mentioned in the Go 1.22 release notes, Go 1.24 now requires Go 1.22.6 or later for bootstrap. We expect that Go 1.26 will require a point release of Go 1.24 or later for bootstrap.

Standard library

Directory-limited filesystem access

The new `os.Root` type provides the ability to perform filesystem operations within a specific directory.

The `os.OpenRoot` function opens a directory and returns an `os.Root`. Methods on `os.Root` operate within the directory and do not permit paths that refer to locations outside the directory, including ones that follow symbolic links out of the directory. The methods on `os.Root` mirror most of the file system operations available in the `os` package, including for example `os.Root.Open`, `os.Root.Create`, `os.Root.Mkdir`, and `os.Root.Stat`,

New benchmark function

Benchmarks may now use the faster and less error-prone `testing.B.Loop` method to perform benchmark iterations like `for b.Loop() { ... }` in place of the typical loop structures involving `b.N` like `for range b.N`. This offers two significant advantages:

- The benchmark function will execute exactly once per -count, so expensive setup and cleanup steps execute only once.
- Function call parameters and results are kept alive, preventing the compiler from fully optimizing away the loop body.

Improved finalizers

The new `runtime.AddCleanup` function is a finalization mechanism that is more flexible, more efficient, and less error-prone than `runtime.SetFinalizer`. `AddCleanup` attaches a cleanup function to an object that will run once the object is no longer reachable. However, unlike `SetFinalizer`, multiple cleanups may be attached to a single object, cleanups may be attached to interior pointers, cleanups do not generally cause leaks when objects form a cycle, and cleanups do not delay the freeing of an object or objects it points to. New code should prefer `AddCleanup` over `SetFinalizer`.

New weak package

The new `weak` package provides weak pointers.

Weak pointers are a low-level primitive provided to enable the creation of memory-efficient structures, such as weak maps for associating values, canonicalization maps for anything not covered by package `unique`, and various kinds of caches. For supporting these use-cases, this release also provides `runtime.AddCleanup` and `maphash.Comparable`.

New crypto/mlkem package

The new `crypto/mlkem` package implements ML-KEM-768 and ML-KEM-1024.

ML-KEM is a post-quantum key exchange mechanism formerly known as Kyber and specified in FIPS 203.

New crypto/hkdf, crypto/pbkdf2, and crypto/sha3 packages

The new `crypto/hkdf` package implements the HMAC-based Extract-and-Expand key derivation function HKDF, as defined in RFC 5869.

The new `crypto/pbkdf2` package implements the password-based key derivation function PBKDF2, as defined in RFC 8018.

The new `crypto/sha3` package implements the SHA-3 hash function and SHAKE and cSHAKE extendable-output functions, as defined in FIPS 202.

All three packages are based on pre-existing `golang.org/x/crypto/...` packages.

FIPS 140-3 compliance

This release includes a new set of mechanisms to facilitate FIPS 140-3 compliance.

The Go Cryptographic Module is a set of internal standard library packages that are transparently used to implement FIPS 140-3 approved algorithms. Applications require no changes to use the Go Cryptographic Module for approved algorithms.

The new `GOFIPS140` environment variable can be used to select the Go Cryptographic Module version to use in a build. The new `fips140` GODEBUG setting can be used to enable FIPS 140-3 mode at runtime.

Go 1.24 includes Go Cryptographic Module version v1.0.0, which is currently under test with a CMVP-accredited laboratory.

New experimental testing/synctest package

The new experimental `testing/synctest` package provides support for testing concurrent code.

- The `synctest.Run` function starts a group of goroutines in an isolated "bubble". Within the bubble, `time` package functions operate on a fake clock.
- The `synctest.Wait` function waits for all goroutines in the current bubble to block.

See the package documentation for more details.

The `synctest` package is experimental and must be enabled by setting `GOEXPERIMENT=synctest` at build time. The package API is subject to change in future releases. See issue #67434 for more information and to provide feeback.

Minor changes to the library

`archive`

The `(*Writer).AddFS` implementations in both `archive/zip` and `archive/tar` now write a directory header for an empty directory.

`bytes`

The `bytes` package adds several functions that work with iterators:

- `Lines` returns an iterator over the newline-terminated lines in a byte slice.
- `SplitSeq` returns an iterator over all subslices of a byte slice split around a separator.
- `SplitAfterSeq` returns an iterator over subslices of a byte slice split after each instance of a separator.
- `FieldsSeq` returns an iterator over subslices of a byte slice split around runs of whitespace characters, as defined by `unicode.IsSpace`.
- `FieldsFuncSeq` returns an iterator over subslices of a byte slice split around runs of Unicode code points satisfying a predicate.

`crypto/aes`

The value returned by `NewCipher` no longer implements the `NewCTR`, `NewGCM`, `NewCBCEncrypter`, and `NewCBCDecrypter` methods. These methods were undocumented and not available on all architectures. Instead, the `Block` value should be passed directly to the relevant `crypto/cipher` functions. For now, `crypto/cipher` still checks for those methods on `Block` values, even if they are not used by the standard library anymore.

`crypto/cipher`

The new `NewGCMWithRandomNonce` function returns an `AEAD` that implements AES-GCM by generating a random nonce during Seal and prepending it to the ciphertext.

The `Stream` implementation returned by `NewCTR` when used with `crypto/aes` is now several times faster on amd64 and arm64.

`NewOFB`, `NewCFBEncrypter`, and `NewCFBDecrypter` are now deprecated. OFB and CFB mode are not authenticated, which generally enables active attacks to manipulate and recover the plaintext. It is recommended that applications use `AEAD` modes instead. If an unauthenticated `Stream` mode is required, use `NewCTR` instead.

`crypto/ecdsa`

`PrivateKey.Sign` now produces a deterministic signature according to RFC 6979 if the random source is nil.

`crypto/md5`

The value returned by `md5.New` now also implements the `encoding.BinaryAppender` interface.

`crypto/rand`

The `Read` function is now guaranteed not to fail. It will always return `nil` as the `error` result. If `Read` were to encounter an error while reading from `Reader`, the program will irrecoverably crash. Note that the platform APIs used by the default `Reader` are documented to always succeed, so this change should only affect programs that override the `Reader` variable. One exception are Linux kernels before version 3.17, where the default `Reader` still opens `/dev/urandom` and may fail.

On Linux 6.11 and later, `Reader` now uses the `getrandom` system call via vDSO. This is several times faster, especially for small reads.

On OpenBSD, `Reader` now uses `arc4random_buf(3)`.

The new `Text` function can be used to generate cryptographically secure random text strings.

`crypto/rsa`

GenerateKey now returns an error if a key of less than 1024 bits is requested. All Sign, Verify, Encrypt, and Decrypt methods now return an error if used with a key smaller than 1024 bits. Such keys are insecure and should not be used. GODEBUG setting rsa1024min=0 restores the old behavior, but we recommend doing so only if necessary and only in tests, for example by adding a //go:debug rsa1024min=0 line to a test file. A new GenerateKey example provides an easy-to-use standard 2048-bit test key.

It is now safe and more efficient to call PrivateKey.Precompute before PrivateKey.Validate. Precompute is now faster in the presence of partially filled out PrecomputedValues, such as when unmarshaling a key from JSON.

The package now rejects more invalid keys, even when Validate is not called, and GenerateKey may return new errors for broken random sources. The Primes and Precomputed fields of PrivateKey are now used and validated even when some values are missing. See also the changes to crypto/x509 parsing and marshaling of RSA keys described below.

SignPKCS1v15 and VerifyPKCS1v15 now support SHA-512/224, SHA-512/256, and SHA-3.

GenerateKey now uses a slightly different method to generate the private exponent (Carmichael's totient instead of Euler's totient). Rare applications that externally regenerate keys from only the prime factors may produce different but compatible results.

Public and private key operations are now up to two times faster on wasm.

crypto/sha1

The value returned by sha1.New now also implements the encoding.BinaryAppender interface.

crypto/sha256

The values returned by sha256.New and sha256.New224 now also implement the encoding.BinaryAppender interface.

crypto/sha512

The values returned by sha512.New, sha512.New384, sha512.New512_224 and sha512.New512_256 now also implement the encoding.BinaryAppender interface.

crypto/subtle

The new WithDataIndependentTiming function allows the user to run a function with architecture specific features enabled which guarantee specific instructions are data value timing invariant. This can be used to make sure that code designed to run in constant time is not optimized by CPU-level features such that it operates in variable time. Currently,

`WithDataIndependentTiming` uses the PSTATE.DIT bit on arm64, and is a no-op on all other architectures. GODEBUG setting `dataindependenttiming=1` enables the DIT mode for the entire Go program.

The `XORBytes` output must overlap exactly or not at all with the inputs. Previously, the behavior was otherwise undefined, while now `XORBytes` will panic.

`crypto/tls`

The TLS server now supports Encrypted Client Hello (ECH). This feature can be enabled by populating the `Config.EncryptedClientHelloKeys` field.

The new post-quantum `X25519MLKEM768` key exchange mechanism is now supported and is enabled by default when `Config.CurvePreferences` is nil. GODEBUG setting `tlsmlkem=0` reverts the default. This can be useful when dealing with buggy TLS servers that do not handle large records correctly, causing a timeout during the handshake (see TLS post-quantum TL;DR fail).

Support for the experimental `X25519Kyber768Draft00` key exchange has been removed.

Key exchange ordering is now handled entirely by the `crypto/tls` package. The order of `Config.CurvePreferences` is now ignored, and the contents are only used to determine which key exchanges to enable when the field is populated.

The new `ClientHelloInfo.Extensions` field lists the IDs of the extensions received in the Client Hello message. This can be useful for fingerprinting TLS clients.

`crypto/x509`

The `x509sha1` GODEBUG setting has been removed. `Certificate.Verify` no longer supports SHA-1 based signatures.

`OID` now implements the `encoding.BinaryAppender` and `encoding.TextAppender` interfaces.

The default certificate policies field has changed from `Certificate.PolicyIdentifiers` to `Certificate.Policies`. When parsing certificates, both fields will be populated, but when creating certificates policies will now be taken from the `Certificate.Policies` field instead of the `Certificate.PolicyIdentifiers` field. This change can be reverted with GODEBUG setting `x509usepolicies=0`.

`CreateCertificate` will now generate a serial number using a RFC 5280 compliant method when passed a template with a nil `Certificate.SerialNumber` field, instead of failing.

`Certificate.Verify` now supports policy validation, as defined in RFC 5280 and RFC 9618. The new `VerifyOptions.CertificatePolicies` field can be set to an acceptable set of

policy `OIDs`. Only certificate chains with valid policy graphs will be returned from `Certificate.Verify`.

`MarshalPKCS8PrivateKey` now returns an error instead of marshaling an invalid RSA key. (`MarshalPKCS1PrivateKey` doesn't have an error return, and its behavior when provided invalid keys continues to be undefined.)

`ParsePKCS1PrivateKey` and `ParsePKCS8PrivateKey` now use and validate the encoded CRT values, so might reject invalid RSA keys that were previously accepted. Use GODEBUG setting `x509rsacrt=0` to revert to recomputing the CRT values.

debug/elf

The `debug/elf` package adds support for handling symbol versions in dynamic ELF (Executable and Linkable Format) files. The new `File.DynamicVersions` method returns a list of dynamic versions defined in the ELF file. The new `File.DynamicVersionNeeds` method returns a list of dynamic versions required by this ELF file that are defined in other ELF objects. Finally, the new `Symbol.HasVersion` and `Symbol.VersionIndex` fields indicate the version of a symbol.

encoding

Two new interfaces, `TextAppender` and `BinaryAppender`, have been introduced to append the textual or binary representation of an object to a byte slice. These interfaces provide the same functionality as `TextMarshaler` and `BinaryMarshaler`, but instead of allocating a new slice each time, they append the data directly to an existing slice. These interfaces are now implemented by standard library types that already implemented `TextMarshaler` and/or `BinaryMarshaler`.

encoding/json

When marshaling, a struct field with the new `omitzero` option in the struct field tag will be omitted if its value is zero. If the field type has an `IsZero() bool` method, that will be used to determine whether the value is zero. Otherwise, the value is zero if it is the zero value for its type. The `omitzero` field tag is clearer and less error-prone than `omitempty` when the intent is to omit zero values. In particular, unlike `omitempty`, `omitzero` omits zero-valued `time.Time` values, which is a common source of friction.

If both `omitempty` and `omitzero` are specified, the field will be omitted if the value is either empty or zero (or both).

`UnmarshalTypeError.Field` now includes embedded structs to provide more detailed error messages.

go/types

All `go/types` data structures that expose sequences using a pair of methods such as `Len() int` and `At(int) T` now also have methods that return iterators, allowing you to simplify code such as this:

```
params := fn.Type.(*types.Signature).Params()
for i := 0; i < params.Len(); i++ {
   use(params.At(i))
}
```

to this:

```
for param := range fn.Signature().Params().Variables() {
   use(param)
}
```

The methods are: `Interface.EmbeddedTypes`, `Interface.ExplicitMethods`, `Interface.Methods`, `MethodSet.Methods`, `Named.Methods`, `Scope.Children`, `Struct.Fields`, `Tuple.Variables`, `TypeList.Types`, `TypeParamList.TypeParams`, `Union.Terms`.

hash/adler32

The value returned by `New` now also implements the `encoding.BinaryAppender` interface.

hash/crc32

The values returned by `New` and `NewIEEE` now also implement the `encoding.BinaryAppender` interface.

hash/crc64

The value returned by `New` now also implements the `encoding.BinaryAppender` interface.

hash/fnv

The values returned by `New32`, `New32a`, `New64`, `New64a`, `New128` and `New128a` now also implement the `encoding.BinaryAppender` interface.

hash/maphash

The new `Comparable` and `WriteComparable` functions can compute the hash of any comparable value. These make it possible to hash anything that can be used as a Go map key.

log/slog

The new `DiscardHandler` is a handler that is never enabled and always discards its output.

`Level` and `LevelVar` now implement the `encoding.TextAppender` interface.

`math/big`

`Float`, `Int` and `Rat` now implement the `encoding.TextAppender` interface.

`math/rand`

Calls to the deprecated top-level `Seed` function no longer have any effect. To restore the old behavior use GODEBUG setting `randseednop=0`. For more background see proposal #67273.

`math/rand/v2`

`ChaCha8` and `PCG` now implement the `encoding.BinaryAppender` interface.

`net`

`ListenConfig` now uses MPTCP by default on systems where it is supported (currently on Linux only).

`IP` now implements the `encoding.TextAppender` interface.

`net/http`

`Transport`'s limit on 1xx informational responses received in response to a request has changed. It previously aborted a request and returned an error after receiving more than 5 1xx responses. It now returns an error if the total size of all 1xx responses exceeds the `Transport.MaxResponseHeaderBytes` configuration setting.

In addition, when a request has a `net/http/httptrace.ClientTrace.Got1xxResponse` trace hook, there is now no limit on the total number of 1xx responses. The `Got1xxResponse` hook may return an error to abort a request.

`Transport` and `Server` now have an HTTP2 field which permits configuring HTTP/2 protocol settings.

The new `Server.Protocols` and `Transport.Protocols` fields provide a simple way to configure what HTTP protocols a server or client use.

The server and client may be configured to support unencrypted HTTP/2 connections.

When `Server.Protocols` contains UnencryptedHTTP2, the server will accept HTTP/2 connections on unencrypted ports. The server can accept both HTTP/1 and unencrypted HTTP/2 on the same port.

When `Transport.Protocols` contains UnencryptedHTTP2 and does not contain HTTP1, the transport will use unencrypted HTTP/2 for http:// URLs. If the transport is configured to use

both HTTP/1 and unencrypted HTTP/2, it will use HTTP/1.

Unencrypted HTTP/2 support uses "HTTP/2 with Prior Knowledge" (RFC 9113, section 3.3). The deprecated "Upgrade: h2c" header is not supported.

net/netip

`Addr`, `AddrPort` and `Prefix` now implement the `encoding.BinaryAppender` and `encoding.TextAppender` interfaces.

net/url

`URL` now also implements the `encoding.BinaryAppender` interface.

os/user

On Windows, `Current` can now be used in Windows Nano Server. The implementation has been updated to avoid using functions from the `NetApi32` library, which is not available in Nano Server.

On Windows, `Current`, `Lookup` and `LookupId` now support the following built-in service user accounts:

- `NT AUTHORITY\SYSTEM`
- `NT AUTHORITY\LOCAL SERVICE`
- `NT AUTHORITY\NETWORK SERVICE`

On Windows, `Current` has been made considerably faster when the current user is joined to a slow domain, which is the usual case for many corporate users. The new implementation performance is now in the order of milliseconds, compared to the previous implementation which could take several seconds, or even minutes, to complete.

On Windows, `Current` now returns the process owner user when the current thread is impersonating another user. Previously, it returned an error.

regexp

`Regexp` now implements the `encoding.TextAppender` interface.

runtime

The `GOROOT` function is now deprecated. In new code prefer to use the system path to locate the "go" binary, and use `go env GOROOT` to find its GOROOT.

strings

The `strings` package adds several functions that work with iterators:

- `Lines` returns an iterator over the newline-terminated lines in a string.
- `SplitSeq` returns an iterator over all substrings of a string split around a separator.
- `SplitAfterSeq` returns an iterator over substrings of a string split after each instance of a separator.
- `FieldsSeq` returns an iterator over substrings of a string split around runs of whitespace characters, as defined by `unicode.IsSpace`.
- `FieldsFuncSeq` returns an iterator over substrings of a string split around runs of Unicode code points satisfying a predicate.

`sync`

The implementation of `sync.Map` has been changed, improving performance, particularly for map modifications. For instance, modifications of disjoint sets of keys are much less likely to contend on larger maps, and there is no longer any ramp-up time required to achieve low-contention loads from the map.

If you encounter any problems, set `GOEXPERIMENT=nosynchashtriemap` at build time to switch back to the old implementation and please file an issue.

`testing`

The new `T.Context` and `B.Context` methods return a context that's canceled after the test completes and before test cleanup functions run.

The new `T.Chdir` and `B.Chdir` methods can be used to change the working directory for the duration of a test or benchmark.

`text/template`

Templates now support range-over-func and range-over-int.

`time`

`Time` now implements the `encoding.BinaryAppender` and `encoding.TextAppender` interfaces.

Ports

Linux

As announced in the Go 1.23 release notes, Go 1.24 requires Linux kernel version 3.2 or later.

Darwin

Go 1.24 is the last release that will run on macOS 11 Big Sur. Go 1.25 will require macOS 12 Monterey or later.

WebAssembly

The `go:wasmexport` compiler directive is added for Go programs to export functions to the WebAssembly host.

On WebAssembly System Interface Preview 1 (`GOOS=wasip1 GOARCH=wasm`), Go 1.24 supports building a Go program as a reactor/library, by specifying the `-buildmode=c-shared` build flag.

More types are now permitted as argument or result types for `go:wasmimport` functions. Specifically, `bool`, `string`, `uintptr`, and pointers to certain types are allowed (see the documentation for detail), along with 32-bit and 64-bit integer and float types, and `unsafe.Pointer`, which are already allowed. These types are also permitted as argument or result types for `go:wasmexport` functions.

The support files for WebAssembly have been moved to `lib/wasm` from `misc/wasm`.

The initial memory size is significantly reduced, especially for small WebAssembly applications.

Windows

The 32-bit windows/arm port (`GOOS=windows GOARCH=arm`) has been marked broken. See issue #70705 for details.