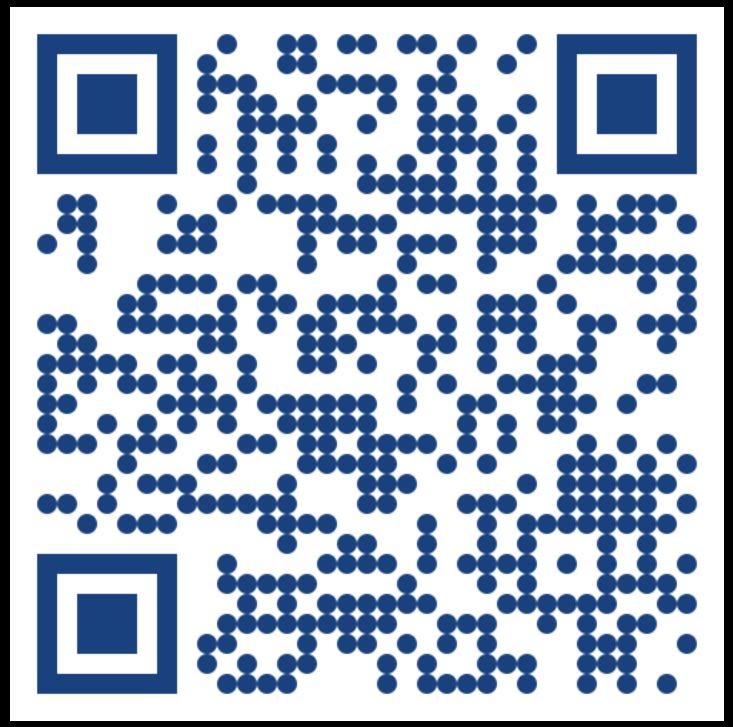
WHAT'S NEW IN GO

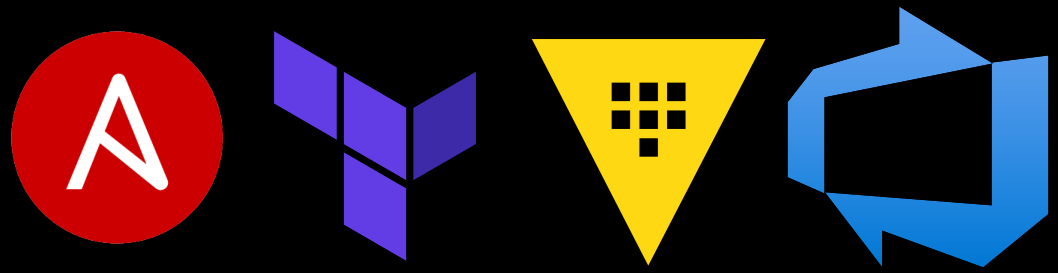
June 11, 2025

DON'T MISS…

- Security Improvements
- Traversal-resistant file API
- testing.B.Loop
- Zero-Value JSON

AWESOME TO HAVE YOU HERE!





SECURITY IMPROVEMENTS

- AES NewCipher API changes
- Stream NewCTR several times faster (but it's unauthenticated)
- Non-AEAD crypters deprecated
- RSA key must be at least 1024 bits
- PrivateKey::Precompute safety
- No more SHA-1 based signatures

Load a private key from… anywhere

```go
// BytesToPrivateKey converts arbitrary bytes to a private key
func BytesToPrivateKey(priv []byte) (any, error) {  2 usages  Aliaksei Yanchuk *
    block, _ := pem.Decode(priv)
    if block == nil {
        return nil, errors.New( text: "input is not a valid PEM-encoded block")
    }

    b := block.Bytes

    if block.Type == "EC PRIVATE KEY" {
        key, err := x509.ParseECPrivateKey(b)
        return key, err
    } else if block.Type == "PRIVATE KEY" {
        key, err := x509.ParsePKCS8PrivateKey(b)
        if rsaKey, ok := key.(*rsa.PrivateKey); ok {
            rsaKey.Precompute()
            if rsaKeyValidErr := rsaKey.Validate(); rsaKeyValidErr != nil {
                return nil, rsaKeyValidErr
            }
        }
        return key, err
    } else {
        return nil, fmt.Errorf( format: "unsupported block type %q", block.Type)
    }
}
```

Initialize AES cipher…

```go
rv := AESData{
    Key: make([]byte, aesKeySizeBits/8),
}

if _, err := rand.Reader.Read(rv.Key); err != nil {
    return nil, rv, err
}

block, blockErr := aes.NewCipher(rv.Key)
if blockErr != nil {
    return nil, rv, blockErr
}

gcm, gcmErr := cipher.NewGCM(block)
if gcmErr != nil {
    return nil, rv, gcmErr
}
```

… and use AEAD options

```go
// Additional authentication data; e.g., reference an object
// to detect unintended copying.
rv.AAD = make([]byte, gcm.NonceSize())
if _, err := rand.Reader.Read(rv.AAD); err != nil {
    return nil, rv, err
}

output := gcm.Seal(dst: nil, rv.IV, plaintext, rv.AAD)
return output, rv, nil
```

```go
func AESDecrypt(ciphertext []byte, data AESData) ([]byte, error) {
    block, blockErr := aes.NewCipher(data.Key)
    if blockErr != nil { return nil, blockErr }

    gcm, gcmErr := cipher.NewGCM(block)
    if gcmErr != nil { return nil, gcmErr }

    return gcm.Open(dst: nil, data.IV, ciphertext, data.AAD)
}
```

TRAVERSAL-RESISTANCE

- Path sanitization
  - filepath::IsLocal (Go 1.20)
  - filepath::Localize(Go 1.23)
- filepath:EvalSymlinks
- os::Root

Too easy to get PWND!

```go
func TestInsecureOpen(t *testing.T) {  new *
    name := "a/../../../secret.txt"
    txt, err := os.ReadFile(name)
    assert.Nil(t, err)
    fmt.Println(string(txt))
}
```

✓ 1 test passed 1 test total, 125 ms

```
<5 go setup calls>
=== RUN   TestInsecureOpen
Golang Meetyp: you'be been PWND!

--- PASS: TestInsecureOpen (0.00s)
PASS

Process finished with the exit code 0
```

PWND is harder…

```go
func TestOpenWithEscape(t *testing.T) {  new *
    name := "a/../../../secret.txt"
    fmt.Printf( format: "IsLocal = %t\n", filepath.IsLocal(name))

    localize, locErr := filepath.Localize(name)
    assert.Nil(t, locErr)
    if locErr != nil {
        return
    }

    fmt.Printf( format: "Localized path = %s\n", localize)
```

```
Process finished with the exit code 1   1 test total, 246 ms
> <5 go setup calls>
=== RUN   TestOpenWithEscape
IsLocal = false
    os_root_test.go:25:
            Error Trace:    /Users/aliakseiyanchuk/GolandProjects/ConfidentialKVEntries/c
            Error:          Expected nil, but got: &errors.errorString{s:"invalid path"}
            Test:           TestOpenWithEscape
--- FAIL: TestOpenWithEscape (0.00s)

FAIL

Process finished with the exit code 1
```

But still possible via symlink

```go
func TestOpenViaSymlink(t *testing.T) {  new *
    name := "blnk/secret.txt"
    fmt.Printf( format: "IsLocal = %t\n", filepath.IsLocal(name))

    localize, locErr := filepath.Localize(name)
    assert.Nil(t, locErr)
    if locErr != nil {
        return
    }

    fmt.Printf( format: "Localized path = %s\n", localize)

    txt, err := os.ReadFile(name)
    assert.Nil(t, err)
    fmt.Println(string(txt))
}
```

✓ 1 test passed 1 test total, 222 ms

```
<5 go setup calls>
=== RUN   TestOpenViaSymlink
IsLocal = true
Localized path = blnk/secret.txt
Golang Meetyp: you'be been PWND!

--- PASS: TestOpenViaSymlink (0.00s)
PASS
```

Os.Root will block symlink escape…

```go
func TestOpenWithRootWithEscapeViaSymlink(t *testing.T) {  new *
    root, rootInitErr := os.OpenRoot( name: "..")
    assert.Nil(t, rootInitErr)
    defer root.Close()

    f, err := root.Open( name: "blnk/secret.txt")
    assert.Nil(t, err)
    if err != nil {
        fmt.Println(err.Error())
        return
    }

    defer f.Close()
    fileContents := readFileFully(f)
    fmt.Println(string(fileContents.Bytes()))
}
```

Process finished with the exit code 1 1 test total, 215 ms

```
> <5 go setup calls>
=== RUN   TestOpenWithRootWithEscapeViaSymlink
    os_root_test.go:77:
            Error Trace:    /Users/aliakseiyanchuk/GolandProjects/ConfidentialKVEntries/core/os_root_test.go:77
            Error:          Expected nil, but got: &fs.PathError{Op:"openat", Path:"blnk/secret.txt", Err:0x2}
            Test:           TestOpenWithRootWithEscapeViaSymlink
openat blnk/secret.txt: no such file or directory
--- FAIL: TestOpenWithRootWithEscapeViaSymlink (0.00s)
```

b.Loop

- No manual timer management
- Shorter ramp-up
- No dead code elimination by disallowing the inlining of such loops
- Recommended way of writing benchmark tests

b.Loop GOODIES

```go
func BenchmarkOldStyle(b *testing.B) {  new *
    // let's assume I'll do the init
    b.ResetTimer() // if setup may be expensive
    for range b.N {
        outKey := azkeys.JSONWebKey{}
        inbytes := []byte(rsaPrivateKey)
        err := PrivateKeyTOJSONWebKey(inbytes, &outKey)
        assert.Nil(b, err)
    }
    b.StopTimer()
    // And here we'll need to d the tear-down
}
```

```go
func BenchmarkRSAKeyConversion(b *testing.B) {  new *
    for b.Loop() {
        outKey := azkeys.JSONWebKey{}
        inbytes := []byte(rsaPrivateKey)
        err := PrivateKeyTOJSONWebKey(inbytes, &outKey)
        assert.Nil(b, err)
    }
}

func BenchmarkECKeyConversion(b *testing.B) {  new *
    for b.Loop() {
        outKey := azkeys.JSONWebKey{}
        inbytes := []byte(prime256v1EcPrivateKey)
        err := PrivateKeyTOJSONWebKey(inbytes, &outKey)
        assert.Nil(b, err)
    }
}
```

json:omitzero

A better way of suppressing values from JSON payloads, especially for buggy integrations

json:omitzero

```go
type CrazyZeroType int  3 usages  new *

func (z CrazyZeroType) IsZero() bool {  new *
    return int(z) < 100
}

Implement interface
type CrazyZeroStruct s
    Message string
    Number  CrazyZeroT
}
```

```go
func TestCrazyZero(t *testing.T) {  new *
    v := &CrazyZeroStruct{
        Message: "this is a message",
        Number:  CrazyZeroType(50),
    }
    jsTxt, err := json.Marshal(v)
                                    t))
```

```
✓ 1 test passed  1 test total, 213 ms
<5 go setup calls>
=== RUN   TestCrazyZero
{"message":"this is a message","number":50}
--- PASS: TestCrazyZero (0.00s)
PASS
```

json:omitzero

```go
type CrazyZeroType int  3 usages  new *

func (z CrazyZeroType) IsZero() bool {  new *
    return int(z) < 100
}

Implement interface
type CrazyZeroStruct struct {  1 usage  new *
    Message string        `json:"message"`
    Number  CrazyZeroType `json:"number,omitempty,omitzero"`
}
```

```go
func TestCrazyZero(t *testing.T) {  new *
    v := &CrazyZeroStruct{
        Message: "this is a message",
        Number:  CrazyZeroType(50),
    }
    jsTxt, err := json.Marshal(v)
```

✓ 1 test passed 1 test total, 215 ms

```
> <5 go setup calls>
=== RUN   TestCrazyZero
{"message":"this is a message"}
--- PASS: TestCrazyZero (0.00s)
PASS
```

That’s all, folks!