`jwt-go` Version History

3.2.0

- Added method `ParseUnverified` to allow users to split up the tasks of parsing and validation
- HMAC signing method returns `ErrInvalidKeyType` instead of `ErrInvalidKey` where appropriate
- Added options to `request.ParseFromRequest`, which allows for an arbitrary list of modifiers to parsing behavior. Initial set include `WithClaims` and `WithParser`. Existing usage of this function will continue to work as before.
- Deprecated `ParseFromRequestWithClaims` to simplify API in the future.

3.1.0

- Improvements to `jwt` command line tool
- Added `SkipClaimsValidation` option to `Parser`
- Documentation updates

3.0.0

- **Compatibility Breaking Changes**: See MIGRATION_GUIDE.md for tips on updating your code
  - Dropped support for `[]byte` keys when using RSA signing methods. This convenience feature could contribute to security vulnerabilities involving mismatched key types with signing methods.
  - `ParseFromRequest` has been moved to `request` subpackage and usage has changed
  - The `Claims` property on `Token` is now type `Claims` instead of `map[string]interface{}`. The default value is type `MapClaims`, which is an alias to `map[string]interface{}`. This makes it possible to use a custom type when decoding claims.
- Other Additions and Changes
  - Added `Claims` interface type to allow users to decode the claims into a custom type
  - Added `ParseWithClaims`, which takes a third argument of type `Claims`. Use this function instead of `Parse` if you have a custom type you'd like to decode into.
  - Dramatically improved the functionality and flexibility of `ParseFromRequest`, which is now in the `request` subpackage
  - Added `ParseFromRequestWithClaims` which is the `FromRequest` equivalent of `ParseWithClaims`
  - Added new interface type `Extractor`, which is used for extracting JWT strings from http requests. Used with `ParseFromRequest` and `ParseFromRequestWithClaims`.

  - Added several new, more specific, validation errors to error type bitmask
  - Moved examples from README to executable example files
  - Signing method registry is now thread safe
  - Added new property to `ValidationError`, which contains the raw error returned by calls made by parse/verify (such as those returned by keyfunc or json parser)

2.7.0 This will likely be the last backwards compatible release before 3.0.0, excluding essential bug fixes.

- Added new option `-show` to the `jwt` command that will just output the decoded token without verifying
- Error text for expired tokens includes how long it's been expired
- Fixed incorrect error returned from `ParseRSAPublicKeyFromPEM`
- Documentation updates

2.6.0

- Exposed inner error within ValidationError
- Fixed validation errors when using UseJSONNumber flag
- Added several unit tests

2.5.0

- Added support for signing method none. You shouldn't use this. The API tries to make this clear.
- Updated/fixed some documentation
- Added more helpful error message when trying to parse tokens that begin with `BEARER`

2.4.0

- Added new type, Parser, to allow for configuration of various parsing parameters
  - You can now specify a list of valid signing methods. Anything outside this set will be rejected.
  - You can now opt to use the `json.Number` type instead of `float64` when parsing token JSON
- Added support for Travis CI
- Fixed some bugs with ECDSA parsing

2.3.0

- Added support for ECDSA signing methods
- Added support for RSA PSS signing methods (requires go v1.4)

2.2.0

- Gracefully handle a `nil` `Keyfunc` being passed to `Parse`. Result will now be the parsed token and an error, instead of a panic.

2.1.0 Backwards compatible API change that was missed in 2.0.0.

- The `SignedString` method on `Token` now takes `interface{}` instead of `[]byte`

2.0.0 There were two major reasons for breaking backwards compatibility with this update. The first was a refactor required to expand the width of the RSA and HMAC-SHA signing implementations. There will likely be no required code changes to support this change.

The second update, while unfortunately requiring a small change in integration, is required to open up this library to other signing methods. Not all keys used for all signing methods have a single standard on-disk representation. Requiring `[]byte` as the type for all keys proved too limiting. Additionally, this implementation allows for pre-parsed tokens to be reused, which might matter in an application that parses a high volume of tokens with a small set of keys. Backwards compatibilty has been maintained for passing `[]byte` to the RSA signing methods, but they will also accept `*rsa.PublicKey` and `*rsa.PrivateKey`.

It is likely the only integration change required here will be to change `func(t *jwt.Token) ([]byte, error)` to `func(t *jwt.Token) (interface{}, error)` when calling `Parse`.

- **Compatibility Breaking Changes**
  - `SigningMethodHS256` is now `*SigningMethodHMAC` instead of `type struct`
  - `SigningMethodRS256` is now `*SigningMethodRSA` instead of `type struct`
  - `KeyFunc` now returns `interface{}` instead of `[]byte`
  - `SigningMethod.Sign` now takes `interface{}` instead of `[]byte` for the key
  - `SigningMethod.Verify` now takes `interface{}` instead of `[]byte` for the key
- Renamed type `SigningMethodHS256` to `SigningMethodHMAC`. Specific sizes are now just instances of this type.
  - Added public package global `SigningMethodHS256`
  - Added public package global `SigningMethodHS384`
  - Added public package global `SigningMethodHS512`
- Renamed type `SigningMethodRS256` to `SigningMethodRSA`. Specific sizes are now just instances of this type.
  - Added public package global `SigningMethodRS256`
  - Added public package global `SigningMethodRS384`

  - Added public package global `SigningMethodRS512`
- Moved sample private key for HMAC tests from an inline value to a file on disk. Value is unchanged.
- Refactored the RSA implementation to be easier to read
- Exposed helper methods `ParseRSAPrivateKeyFromPEM` and `ParseRSAPublicKeyFromPEM`

1.0.2

- Fixed bug in parsing public keys from certificates
- Added more tests around the parsing of keys for RS256
- Code refactoring in RS256 implementation. No functional changes

1.0.1

- Fixed panic if RS256 signing method was passed an invalid key

1.0.0

- First versioned release
- API stabilized
- Supports creating, signing, parsing, and validating JWT tokens
- Supports RS256 and HS256 signing methods