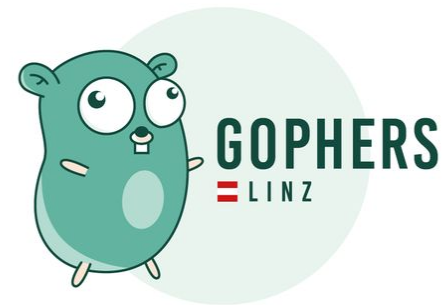


# Secure Account Management Services in Go

By Florian Harwöck at Gopher Linz #1



# Florian Harwöck

**SE-Student HTL Leonding**

**Gopher Since: 2016**

**I love:**

- ★ Open Source
- ★ Security
- ★ System architecture (“Cloud Native”)

**Free time:**

- ★ Badminton Coach for the youth team & kids at ABV Wels
- ★ Party! 🤨


**Contact me: <https://harwoeck.com>**



# But let's talk about why we are here

**My diploma thesis “Building a secure and scalable digital assets exchange”**

## Security:

- 1) Transactions
- 2) Currency-Wallets (the “storage” for the digital assets)
- 3) User Accounts 

## Scalability:

Topic for another day ;)

**Introduce key concepts  
involved into designing our  
secure “cloud-native”  
account management  
microservice**

**MUCH BUZZWORDS**

**SUCH DIGITAL. VERY WOW.**

# **Introduce key concepts involved into designing our secure “cloud-native” account management microservice**

Secure = Cryptographic tricks + Hashicorp Vault

“Cloud-native” = Go (Obviously) + Containerized  
+ GRPC + Envoy + etcd + Cockroachdb

# The 3 key parts of this talk ...

## ★ API design

- Introduction/ Showcase of GRPC
  - Demo

## ★ Security

- Password-Storage
  - How our microservice stores passwords
  - How and why we deny 550 million passwords
- User data (PII) protection
  - Cryptographic tricks

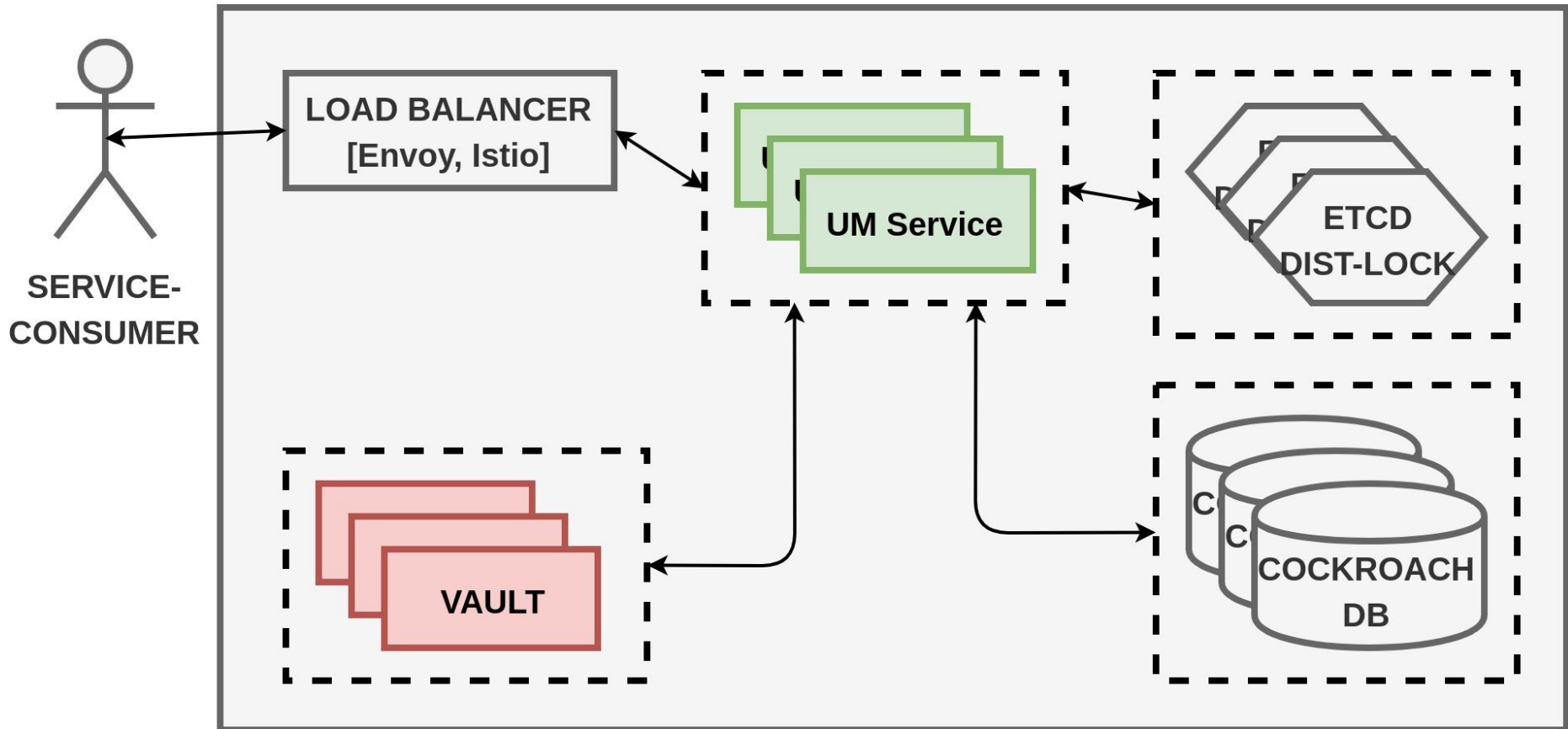
## ★ Go

- Benefits in general
- Benefits specifically in this type of application

**Before we talk  
about API Design  
let's see the  
architecture**




# Architecture



# **GRPC** Google Remote Procedure Call



# Introduction to

- High performance RPC framework
- Any environment. Support for almost all widely used languages
- Efficient way to connect microservices
- Open Source 

# Example of GRPC Service

- Simple unambiguous description of services and their capabilities
- Strongly typed! :)

```
syntax = "proto3";  
package um;  
  
service Um {  
    rpc Login(LoginRequest) returns (LoginResponse) {}  
}  
  
message LoginRequest {  
    string email_address = 1;  
    string password = 2;  
}  
  
message LoginResponse {  
    string user_id = 1;  
    string session_token = 2;  
}
```

**It's a Go Meetup!!**  
**Show us Go Code**



**OH MIGHTY DEMO GODS**



**PLEASE, PLEASE, LET THIS DEMO  
WORK**

# Critic on GRPC

- GRPC has 16 predefined status codes

```
// EmailResendVerification resends the email containing the verification
// token, used to verify the email in question.
//
// Possible exceptions:
// * InvalidArgument - The format of the token is invalid. Should be 64
//   bytes encoded with base64, resulting in 88 characters (already
//   including padding).
// * Unauthenticated - The provided token couldn't be found in our database
//   or failed the cryptographic checks.
// * ResourceExhausted - Token used too often. Wait at least 1h. Then try
//   again.
// * Unavailable - Try again
// * Internal
rpc EmailResendVerification(EmailResendVerificationRequest) returns
(google.protobuf.Empty) {}
```

**Let's talk security**



**I'm not a studied  
cryptographer/ security  
expert so do your own  
research too.**


















































# Security Goals

- **Strong user authentication**
  - Password Protection
  - Deny weak, known passwords
- **PII data confidentiality**
- **Make persistent data useless** (full database leak should have no effect)
- **Don't limit usability/ scalability/ maintainability**
- **Verify our suggestions**

# About Cryptography in Go

# Thanks Google!

- They invested heavily in this!
- Not many languages provide that many algorithms, primitives and protocols in the std-lib
- Widely used and (till now) without \*big\* security issues

 aes	 rand	 acme	 cryptobyte	 otr	 tea
 cipher	 rc4	 argon2	 curve25519	 pbkdf2	 twofish
 des	 rsa	 bcrypt	 ed25519	 pkcs12	 xtea
 dsa	 sha1	 blake2b	 hkdf	 poly1305	 xts
 ecdsa	 sha256	 blake2s	 internal	 ripemd160	
 elliptic	 sha512	 blowfish	 md4	 salsa20	
 hmac	 subtle	 bn256	 nacl	 scrypt	
 internal	 tls	 cast5	 ocsp	 sha3	
 md5	 x509	 chacha20poly1305	 openpgp	 ssh	

# Really good performance

- Heavily optimized by Cloudflare and Google itself
- Lot of the cryptographic primitives are written in GoASM

```
5  #include "textflag.h"
6
7  // func encryptBlockAsm(nr int, xk *uint32, dst, src *byte)
8  TEXT ·encryptBlockAsm(SB),NOSPLIT,$0
9      MOVQ nr+0(FP), CX
10     MOVQ xk+8(FP), AX
11     MOVQ dst+16(FP), DX
12     MOVQ src+24(FP), BX
13     MOVUPS 0(AX), X1
14     MOVUPS 0(BX), X0
15     ADDQ $16, AX
16     PXOR X1, X0
17     SUBQ $12, CX
18     JE Lenc196
19     JB Lenc128
20 Lenc256:
21     MOVUPS 0(AX), X1
22     AESENC X1, X0
23     MOVUPS 16(AX), X1
24     AESENC X1, X0
25     ADDQ $32, AX
```

# **How our microservice stores passwords**



**Brief fresh-up  
first!**



# Password hashing basics

- User passwords get hashed using cryptographic hash functions

```
hash("user password") =
```

```
2989961bb41d694cc5ee6e79174455082b591606
```

- Because lots of different users may use the same password, each user gets his own random salt

```
hash("user password" + randomSalt) =
```

```
8f8960de943473e4200c44aa57a2b4047097b101
```



**Don't MD5/SHA1/SHA2 hash your passwords (even with salts).**



# And yes, these things happen:

## unsalted MD5:

Last.fm (43M, 2012)

## salted/ unsalted SHA1:

MySpace (360M, 2008)

Dropbox (70M, 2012)

LinkedIn (160M, 2016)

## Encrypted with a single 3DES key in ECB mode:

Adobe (150M, 2013)

# The “funny” thing about ECB



Original image



Encrypted using ECB mode



Modes other than ECB result in pseudo-randomness

**The Best of Password Security 2018 goes to  
an Austrian company ... 🏆**



# Our friends at T-Mobile ...



Claudia Pellegrino @c\_pellegrino · 3d



T-Mobile Austria   
@tmobileat

Re  
@T  
He  
ag  
pa



Claudia Pellegrino @c\_pellegrino · 2d



T-Mobile Austria  @tmobileat · 2d

Hi @c\_pellegrino, I really do not get why this is a problem. You have so many passwords for every app, for every mail-account and so on. We



T-Mobile Austria  @tmobileat · 1d

@Korni22 What if this doesn't happen because our security is amazingly good? ^Käthe

 369

 527

 644





▲ T-Mobile confirms they store passwords in plaintext, don't see why it's a problem (twitter.com)  
140 points by dsr12 9 months ago | [hide](#) | [past](#) | [web](#) | [favorite](#) | 44 comments

PRIVACY AND SECURITY

# Did T-Mobile Austria Really Just Admit It Stores Customer Passwords in Plaintext?

TECH CYBERSECURITY

## T-Mobile Austria is working to implement a basic security measure 'as quickly as possible'

PCMag UK | News & Analysis | [News](#)  
@russellbrandom | Apr 10, 2018, 12:08pm EDT

# T-Mobile Austria is OK with Storing Passwords Text

BY MICHAEL KAN 7 APR 2018, 1:22 A.M.

Security

## T-Mobile Austria stores passwords as plain text, Outlook gets message crypto, and more

Warning: Contains extreme stupidity

By [Iain Thomson](#) in [San Francisco](#) 7 Apr 2018 at 11:36

24 [SHARE](#) ▼

**Eine kurze Frage an T-Mobile Österreich endete für den Mobilfunkanbieter im Fiasko**  
Der österreichische Mobilfunkprovider T-Mobile Austria steckt seit Tagen im Shitstorm. Heute gab das Unternehmen zu, Kundenpasswörter unverschlüsselt zu speichern. Ein Drama in zehn Akten.

**Since then: #amazinglygood**



# Why am I telling you all this ...

Because it would be *\*easy\** to do it right (especially for you Gophers)

1. Generate cryptographically secure random salts

```
// import "crypto/rand"
salt := make([]byte, 32)
_, err := io.ReadFull(rand.Reader, salt)
```

2. Use one of the currently recommended KDFs

```
// import "golang.org/x/crypto/argon2"
password := []byte("user password")
hash := argon2.IDKey(password, salt, 1, 32*1024, 4, 32)
```

3. Encrypt hash and salt
4. Store it

**FINISHED!**



# Verification is also simple ...

- Just calculate the hash again and do a (constant-time) comparison with the saved value in your database

```
password := []byte("user password")
```

```
var (  
    salt          = saltFromDB()  
    correctHash = hashFromDB()  
)
```

```
hashAttempt := argon2.IDKey(password, salt, 1, 32*1024, 4, 32)
```

```
if subtle.ConstantTimeCompare(correctHash, hashAttempt) == 1 {  
    // ok  
}
```

# A short note about these “KDF”s

- KDF = Key Derivation Function
- Fairly simplified: Like normal hashes, but for passwords
- Currently recommended (by OWASP):
  - PBKDF2 (Use if FIPS certification is needed)  
`pbkdf2.Key(password, salt, 4096, 32, sha256.New)`
  - Bcrypt/ Scrypt  
`bcrypt.GenerateFromPassword(password, bcrypt.DefaultCost)`  
`scrypt.Key(password, salt, 32768, 8, 1, 32)`
  - Argon2  
`argon2.IDKey(password, salt, 1, 64*1024, 4, 32)`


# More information about password storage (Shameless self promotion)

- ★ Search online for “**Password and Credential Management in 2018**”

 Password and Credential Management in 2018 – Florian Harwöck ...

<https://medium.com/.../password-and-credential-management-in-2018-56f43669d588> ▼

Aug 15, 2018 - Password and Credential Management in 2018. State of the art security for the most valuable secrets. Go to the profile of Florian Harwöck.

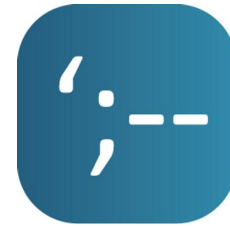
- ★ Got featured from Medium in “Cyber Security” and distributed by few security newsletter -> seems it’s worth the read
  - 12K reads
  - 1,5K claps  from 220+ people

# **How and why we deny 550 million passwords**



**Who has heard  
from  
HavelBeenPwned?**

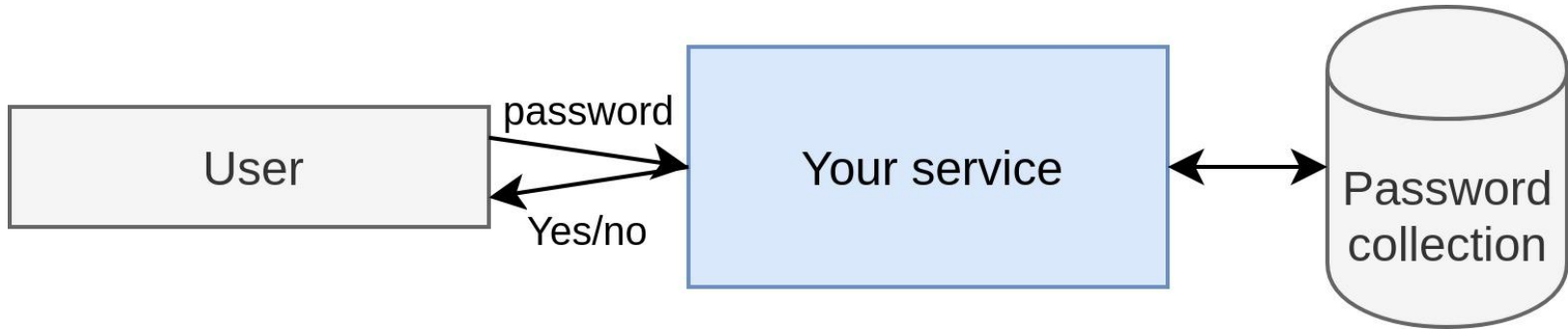
# HavelBeenPwned (HIBP)



- Data-breach collection service
- Allows you to check if your personal data was leaked during any hack (email address, passwords, etc.)
- Operated by Troy Hunt
  - Microsoft Regional Director
  - Microsoft MVP
  - International top speaker on web security
- Collected over 550 Million unique passwords from data breaches



# How does it work?



Password collection can be:

1. A public API operated by Troy Hunt and Cloudflare
  - a. You will hash your user's password with SHA1 and send a fraction of the hash to the service (k-anonymity)

SHA1("password") = **5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8**

`api.pwnedpasswords.com/range/5baa6`

```
1D72CD07550416C216D8AD296BF5C0AE8E0:10
1E2AAA439972480CEC7F16C795BBB429372:1
1E3687A61BFCE35F69B7408158101C8E414:1
1E4C9B93F3F0682250B6CF8331B7EE68FD8:3645804
1F2B668E8AABEF1C59E9EC6F82E3F3CD786:1
20597F5AC10A2F67701B4AD1D3A09F72250:3
```

# Or you download all the SHA1 hashes

	Format	File	Date	Size	SHA-1 hash of 7-Zip file
<a href="#">torrent</a> <a href="#">cloudflare</a>	SHA-1	Version 4 (ordered by prevalence)	17 Jan 2019	11.0GB	59741e11e20a3fc4f29ae597972295dcb94cef39
<a href="#">torrent</a> <a href="#">cloudflare</a>	SHA-1	Version 4 (ordered by hash)	17 Jan 2019	9.78GB	d81c649cda9cddb398f2b93c629718e14b7f2686
<a href="#">torrent</a> <a href="#">cloudflare</a>	NTLM	Version 4 (ordered by prevalence)	17 Jan 2019	8.85GB	2014695d9c4880aac69be031a1cc7c9eee4bcfb9
<a href="#">torrent</a> <a href="#">cloudflare</a>	NTLM	Version 4 (ordered by hash)	17 Jan 2019	7.58GB	ee7199ee2a1d8f23dd346d5b1fb2255e1ed8de8a



# It's not easy to search a 35GB .txt file

- Troy Hunt recommends using something like Azure TableStorage, Google BigTable, etc. for querying with low latencies
- Problem: I don't like proprietary solutions
- I like Open Source.
- Therefore I built it! :)






**hibpoffline**


Query HavelBeenPwned locally


# You can find it own Github soon


github.com/[harwoeck/hibpoffline](https://github.com/harwoeck/hibpoffline)


High-performance Service for querying an offline copy of the HIBP database (a collection of 551 million breached  passwords). [Edit](#)  
Exists because private  = .


[Manage topics](#)

 89 commits

 1 branch

 0 releases

 1 contributor

 Apache-2.0

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾











 harwoeck Update README.md

Latest commit 497c959 4 days ago

● Go 92.3%

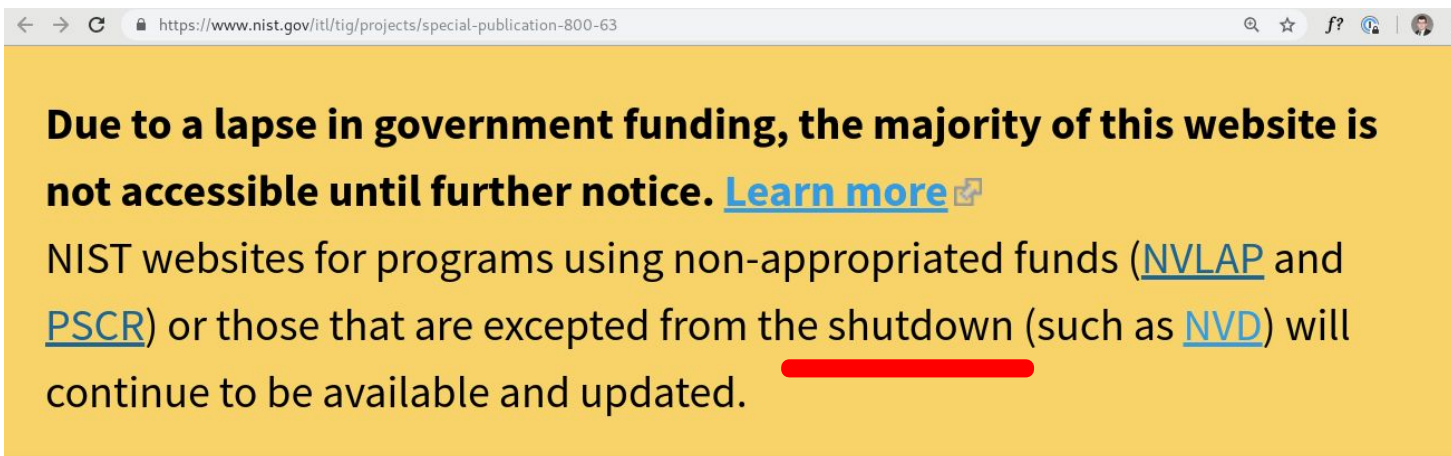
● Makefile 6.2%

● Dockerfile 1.5%

 <a href="#">CONTRIBUTING.md</a>	Add diagrams and correct readme	3 days ago
 <a href="#">Dockerfile</a>	Move towards open source	6 days ago
 <a href="#">LICENSE</a>	Create LICENSE	4 months ago
 <a href="#">Makefile</a>	Add changes	7 days ago
 <a href="#">README.md</a>	Update README.md	4 days ago
 <a href="#">go.mod</a>	Further performance improvements	2 months ago
 <a href="#">go.sum</a>	Add diagrams and correct readme	9 days ago
 <a href="#">hibpoffline.png</a>	Change readme and picture	4 months ago
 <a href="#">README.md</a>		

# Why are we even doing this?

- Password reuse is common, although extremely risky
- Most people aren't aware of the potential impact
- Hackers take advantage reused credentials
  - They collect your password from a breach at service A
  - They use your email and password to login to service B
- NIST Guidance (SP 800-63-3: Digital Identity Guidelines)



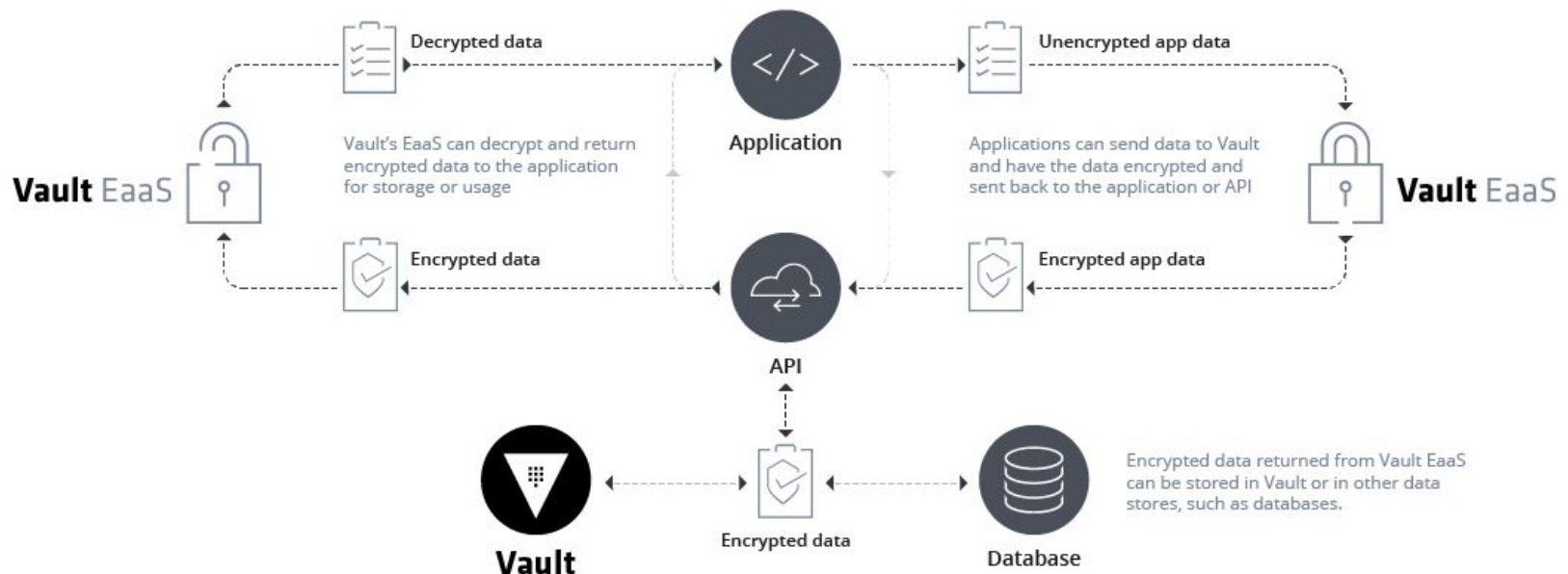
# **User data (PII) protection**



# Hashicorp Vault



- Extremely secure and highly reviewed open-source secrets management software. One of the “golden standards”
- Biggest enterprises rely on it to safeguard their production secrets
- Provides Encryption-as-a-Service capabilities



# Hashicorp builds (almost) all their tools with Go

A tool for secrets management, encryption as a service, and privileged access management <https://www.vaultproject.io/>

vault go secrets

Go 79.5% JavaScript 12.2% HTML 6.8% CSS 0.9% Shell 0.3% HCL 0.1% Other 0.2%

## consul

Consul is a distributed, highly available, and data center aware solution to connect and configure applications across dynamic, distributed infrastructure.

Go ★ 14.7k 🍴 2.5k

## nomad

Nomad is a flexible, enterprise-grade cluster scheduler designed to easily integrate into existing workflows. Nomad can run a diverse workload of micro-service, batch, containerized and non-contain...

Go ★ 4.2k 🍴 813

## packer

Packer is a tool for creating identical machine images for multiple platforms from a single source configuration.

Go ★ 8.5k 🍴 2.3k

## terraform

Terraform is a tool for building, changing, and combining infrastructure safely and efficiently.

Go ★ 15.2k 🍴 4.4k

## vault

A tool for secrets management, encryption as a service, and privileged access management

Go ★ 11.2k 🍴 1.7k

## vagrant

Vagrant is a tool for building and distributing development environments.

Ruby ★ 17.9k 🍴 3.6k

**It's a Go Meetup!!**  
**Show us Go Code**



Provided by Hashicorp

```
package vault

type Vault struct {
    client *api.Logical
}

func New(address, token string) (*Vault, error) {
    config := api.DefaultConfig()
    config.ConfigureTLS(&api.TLSConfig{
        // your certs
    })
    config.Address = address

    client, err := api.NewClient(config)
    if err != nil {
        return nil, err
    }
    client.SetToken(token)

    // return vault instance
    return &Vault{
        client: client.Logical(),
    }, nil
}
```



# Encrypt Helper

```
func (v *Vault) Encrypt(keyRing, plaintext string) (string, error) {
    secret, err := v.client.Write("transit/encrypt/"+keyRing,
    map[string]interface{}{
        "plaintext": plaintext,
    })
    if err != nil {
        return "", err
    }

    ciphertext, ok := secret.Data["ciphertext"].(string)
    if !ok {
        return "", errors.New("vault: unable to get ciphertext during
    encryption")
    }

    return ciphertext, nil
}
```

# Decrypt Helper

```
func (v *Vault) Decrypt(keyRing string, ciphertext string) (string, error) {
    plain, err := v.client.Write("transit/decrypt/"+keyRing,
        map[string]interface{}{
            "ciphertext": ciphertext,
        })
    if err != nil {
        return "", err
    }

    plaintext, ok := plain.Data["plaintext"].(string)
    if !ok {
        return "", errors.New("vault: unable to get plaintext during
decryption")
    }

    return plaintext, nil
}
```

# Vault-Helper Usage

```
v, _ := vault.New("localhost:8080", "our-access-token")  
passwordKeyRing = "password"
```

```
// During registration  
cipherText, _ := v.Encrypt(passwordKeyRing, "LrEw014lTKgUpc/F30Ytlg==")  
storePasswordInDB(cipherText)
```

```
// On Login  
password := loadPasswordFormDB()
```

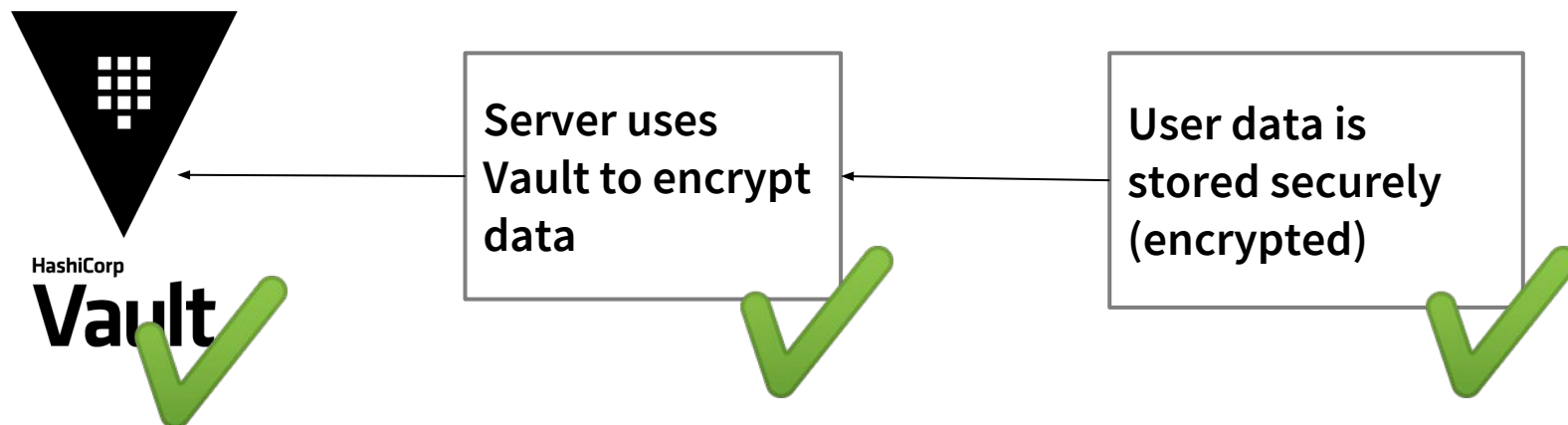
```
fmt.Println(password)  
// vault:v1:cZNHVx+sxdMErXRSuDa1q/pz49fXTn1PScKfhf+PIZPvy8xKfkytpwKcbC0fF2U=
```

```
password, err = v.Decrypt(passwordKeyRing, password)  
if err != nil {  
    log(err)  
    return ErrInternalServerError  
}
```

```
fmt.Println(password)  
// LrEw014lTKgUpc/F30Ytlg==
```

# Do we trust Vault?

- Definition:  
“In cryptographic systems with hierarchical structure, a trust anchor is an authoritative entity for which trust is assumed and not derived.”
- Hashicorp Vault is our trust anchor



# Encrypt everything

ID	EMAIL	PWD	FIRSTNAME	LASTNAME	ADDRESS	ZIPCODE	PHONENUMBER
100	florian@harwoeck.at	LrEw01G0XZMm2p0Ui2I4Qv vbHb0gs1fudIVSMGCxHIHM iBu3FjgWNsKjQUY3LSYa4l aV4c24TKgUpc/F30YtIg==	Florian	Harwöck	Wunderstr. 13	4600	+43 676 6969084

- Introduce “clearance level” and encrypt in groups

ID	PWD	CLEARANCE_L1: string	CLEARANCE_L2: string
100	<code>ENCRYPTED(LrEw01G0XZMm2p0Ui2I4Qv vbHb0gs1fudIVSMGCxHIHM iBu3FjgWNsKjQUY3LS Ya4laV4c24TKgUpc/F 30YtIg==)</code>	<code>ENCRYPTED({   "email": "florian@harwoeck.at"   "firstname": "Florian",   "lastname": "Harwöck" })</code>	<code>ENCRYPTED({   "address": "Wunderstr. 13",   "zipcode": 4600,   "phonenummer": "+43 676 6969084" })</code>

- Problem: How to do a user login? (with email + password pair)

**Who has heard of  
the concept of  
“Blind-Indexing”? 🙄**

# Blind-Indexing

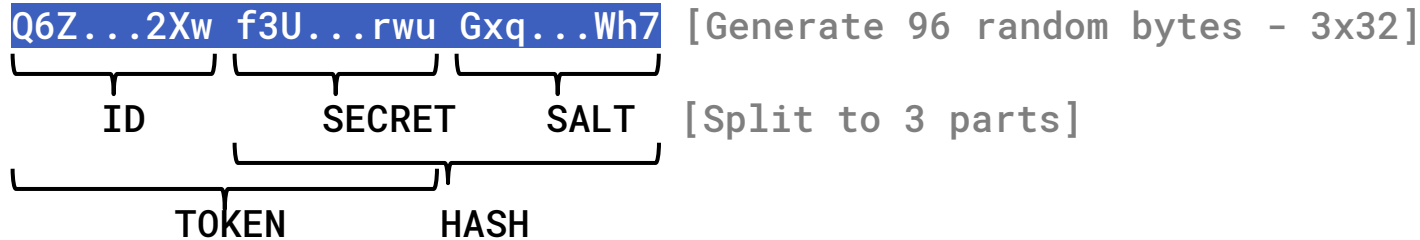
- Works by calculating a **keyed MAC** of the email address
  - The key is only known by our server software
- Use a **KDF** or a **keyed hash algorithm**: HMAC-SHA2, KMAC, Blake2b

```
h, _ := blake2b.New512(key)
h.Write([]byte("florian@harwoeck.at"))
bidx := h.Sum(nil)
```

ID	EMAIL_BIDX: string	PWD	CLEARANCE_L1: string	CLEARANCE_L2: string
100	0C3T1XZ...6p09n...SeeAYp0PRgC8 1wSeBqt...CaM...XVbzTY+0ljCmz PHVU1roh...UBWauQPz9YbsF4m 9g==	ENCRYPTED(LrE...G0XZMm2p0 Ui2I4Q...Hb...fudIVSMGCx HIHMiBu...tsKjQUY3LSYa4 1aV4c24T...pc/F30Yt1g==)	ENCRYPTED( "email": "florian@harwoeck.at" "firstname": "florian", "lastname": "Harwöck" )	ENCRYPTED({ "address": "Wur...str. 13", "zipcode": "phonenumbe..."+43 676 6969084" })

- You could get keyed MACs from **Vault -> Trust Anchor**, but it hurts performance

# Concept also applies to other secrets: example: **Session Tokens**



TOKEN = ID + SECRET  
HASH = **MAC**(SECRET, SALT)

ID_BLIND_IDX	TOKEN_INFO
 YZi0f+c9...L7S...rin+etStNctNe XYqL1yrY...i...VUtwX2B5hSXoUt Z9Y9arpM9...apECcAgxBR+rGLw= =	 <b>ENCRYPTED</b> ( { "hash": "gcW...7...j6uZNMnMC2kcRAR1CZxPzF0pKEwL0fc7TYybDqyXjEoe6+0bNUs0Y0u9PePCUYF7JHvLWEv+rF+Eg==" "salt": "3kH...cunjNrZ5KmbUG2hBFhJR/ptiFNNuZyJTnISQ==" } )

- Next time a user sends the session token you do
  - Split it to ID and SECRET parts.
  - Use ID to calculate blind index and search associated token\_info
  - Decrypt token\_info
  - Hash SECRET part provided by user with salt from DB and compare to hash



**Things we discussed here should obviously only be your last defence**



**Putting that all  
together we get**

# Goals vs Solutions

- Strong user authentication
  - Password Protection  $\Rightarrow$  Strong, modern KDF
  - Deny weak, known passwords  $\Rightarrow$  hibpoffline
- PII data confidentiality  $\Rightarrow$  content encryption
- Make persistent data useless  $\Rightarrow$  each field in the database: either encrypted or blind index (except ids)
- Don't limit usability/ scalability/ maintainability  $\Rightarrow$  blind indexing and hashicorp vault
- Kerckhoffs's Principle  $\Rightarrow$  share and verify gathered knowledge

**So we did we go with Go?**



# Benefits using Go ...

- ★ Go is open source by nature (Probably every bigger Go project is hosted on Github)
- ★ It's fun to develop
- ★ Great community and culture
- ★ Simple and non-magical
- ★ Tech is top (really fast, easy to use concurrency, free cross-platform, garbage collected)
- ★ Godoc (std lib documentation is absolutely great)
- ★ One “subconscious” way of doing things
- ★ Cloud native language is Go



**CLOUD**



**CLOUD NATIVE**

# Additional benefits in this project

- Most pieces of our tech stack are already written in Go or provide first class support
  - GRPC, Vault, etcd, CockroachDB, ...
- We could leverage the really good cryptographic libs from Go itself and didn't need to call C libraries like OpenSSL/ BoringSSL/ Libsodium/ etc.

# **Conclusion**

**... or what to remember for  
your project**



# Remember for your own project

- **GRPC** can make your life simpler and more efficient
- **HIBP** (or hibpoffline) can help improve your user's security
- **Cryptography** can be powerful, if you use it correctly
  - Get experts or at least someone who knows the stuff
  - Your user's data can be stored more secure using things like Blind Indexing or Hashicorp EaaS
- **Go** helps you in both areas
  - Really good GRPC support
  - Extensive cryptographic libraries
- Your DevSecOps feel much safer knowing you finally implemented the missing AirBag 😊

# Q&A



\*Unfortunately there was no Helicopter raising his hand. Forgive me

**Thanks for  
listening.** 🙌

**Please don't forget to give me  
feedback later**

# Google's Protocol Buffers

- Flexible, efficient, automated mechanism for serializing structured data
- *“Think XML, but smaller, faster, and simpler”*
- Define structure once, then use generated source code
- Updates to data structure don't break compatibility to deployed programs

```
syntax = "proto3";
```

```
message LoginRequest {  
    string email_address = 1;  
    string password = 2;  
}
```

```
message LoginResponse {  
    string user_id = 1;  
    string session_token = 2;  
}
```