

# Ich will doch nur verschlüsseln...

## Real world cryptography for the impatient

heise devSec 2019

Dominik Schadow  
bridgingIT





- Encrypt (user) data...
  - ... in transit ←
  - ... at rest
- ~~Encrypt user passwords~~

Typically via  
HTTPS (TLS)

Never ever encrypt user  
passwords - always hash them  
with a secure password  
hashing algorithm

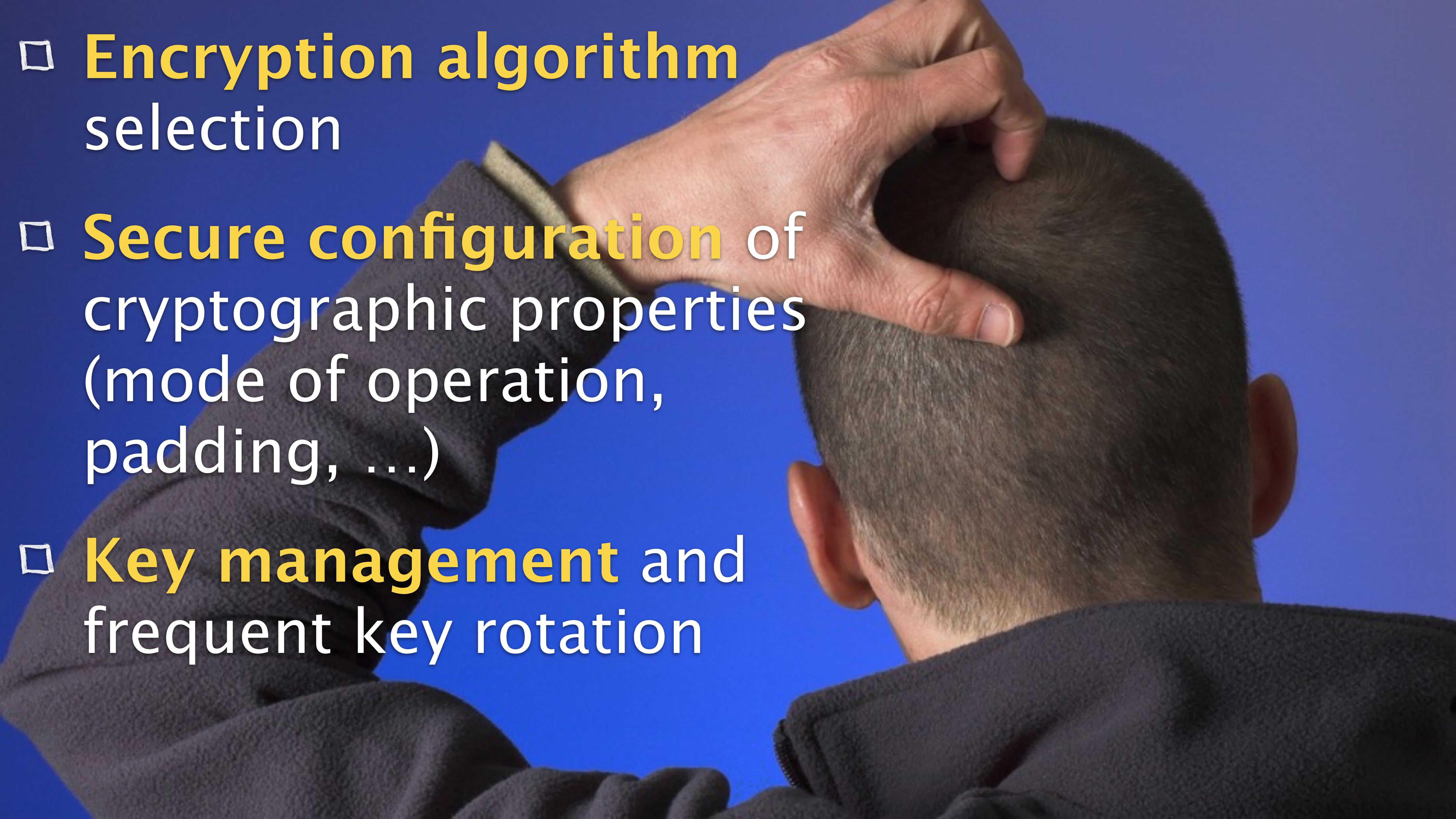
# Out of Scope

Encryption functionality  
provided by/ integrated  
in

- Operating Systems  
(disk encryption)
- Databases and other  
data stores



- **Encryption algorithm selection**
- **Secure configuration** of cryptographic properties  
(mode of operation, padding, ...)
- **Key management** and frequent key rotation



# Takeaways

- 1 Crypto usage (dev) and key management (ops) go hand in hand
- 2 A modern (Java) crypto library should provide secure defaults
- 3 Trust the cloud? Externalize your key management

# Tools and Services vs. Custom Implementations



# Google Tink

**Multi-language** - Java, Android, C++, Obj-C, Go (Python and JavaScript as early versions) and **cross-platform** open source library

**Secure, easy to use, hard(er) to misuse** - additional parameters are automatically and randomly initialized (all-in-one constructions)

**Used in Google products** like AdMob, Android Pay, and Google Android Search App

# Easier Configuration by Limitation

## Limitation of crypto algorithms (primitives)

**AEAD:** AES-EAX, AES-GCM, AES-CTR-HMAC, KMS Envelope, CHACHA20-POLY1305

**Streaming AEAD:** AES-GCM-HKDF-STREAMING, AES-CTR-HMAC-STREAMING

**Deterministic AEAD:** AES-SIV

**Hybrid Encryption:** ECIES with AEAD and HKDF

**MAC:** HMAC-SHA2

**Digital Signatures:** ECDSA over NIST curves, Ed25519, RSA-SSA-PKCS1, RSA-SSA-PSS

## Automatic initialization of their (optional) parameters

# **A**uthenticated **E**nryption with **A**sociated **D**ata

**A**sociated **D**ata is **NOT** the **k**ey - optional, not encrypted but authenticated, must be identical for decryption and is added as authentication tag

**D**etects any modification on ciphertext

# Simple Exception Handling

## Plain Java

```
catch NoSuchPaddingException,  
NoSuchAlgorithmException, KeyStoreException,  
IllegalBlockSizeException, BadPaddingException,  
CertificateException, InvalidKeyException,  
UnrecoverableKeyException
```

## Google Tink

```
catch GeneralSecurityException
```

# Full or Limited Library Integration

**Register the complete library**

```
TinkConfig.register();
```

**Register selected primitive(s) only**

```
AeadConfig.register();
HybridConfig.register();
MacConfig.register();
SignatureConfig.register();
```

# Keyset and KeysetHandle

**Keyset is a collection of keys**

All keys in a keyset correspond to a single primitive

One key is the primary key

**KeysetHandle form a wrapper around a Keyset**

Stores key data and meta data

# Instantiate a Primitive

```
KeysetHandle keysetHandle =  
    KeysetHandle.generateNew(  
        AeadKeyTemplates.AES128_GCM);
```

# Keys are Stored as Plaintext JSON

```
"primaryKeyId": 77911481,  
"key": [ {  
    "keyData": {  
        "typeUrl": "type.googleapis.com/google.crypto.tink.AesGcmKey",  
        "keyMaterialType": "SYMMETRIC",  
        "value": "GhB+1IsMGzPOsRwOfYXV2rDv"  
    },  
    "outputPrefixType": "TINK",  
    "keyId": 77911481,  
    "status": "ENABLED"  
} ]
```

# Key Rotation within the same Primitive

```
KeysetHandle keysetHandle =  
    KeysetManager  
        .withKeysetHandle(privateKeysetHandle)  
        .rotate(HybridKeyTemplates  
            .ECIES_P256_HKDF_HMAC_SHA256_AES128_GCM)  
        .getKeysetHandle();
```

# Keys are Organized in Keysets

```
"primaryKeyId": 448490911,  
"key": [ {  
    "keyData": {  
        "typeUrl": "type.googleapis.com/google.crypto.tink.EciesAeadHkdfPrivateKey",  
        "keyMaterialType": "ASYMMETRIC_PRIVATE",  
        "value": "EosBEkQKBAgCEAMSOhI4CjB0eXB1Lmdvb2dsZWFWaXMuY29tL2dvb2dsZS5jcnl..."  
    },  
    "outputPrefixType": "TINK",  
    "keyId": 976425121,  
    "status": "DISABLED"  
} , {  
    "keyData": {  
        "typeUrl": "type.googleapis.com/google.crypto.tink.EciesAeadHkdfPrivateKey",  
        "keyMaterialType": "ASYMMETRIC_PRIVATE",  
        "value": "EosBEkQKBAgCEAMSOhI4CjB0eXB1Lmdvb2dsZWFWaXMuY29tL2dvb2dsZS5jcnlwd..."  

```

# Demo



# AU L T

A tool for managing secrets.

# Transit Secrets Engine

Cryptographic functions on data in-transit - **encryption as a service**

**No data stored on server** - only keys

**Encrypt, decrypt** (*random number generator, sign, verify, hash*) for **base64 encoded input**

**Key rotation** and **versioning** - disabling of old key versions

# Taking a sledgehammer to crack a nut



# Encryption Key Types

**AES-GCM with a 256-bit AES key and a 96-bit nonce**

**ChaCha20-Poly1305 with a 256-bit key**

**2048-bit RSA key**

**4096-bit RSA key**

```
$ vault server -dev
==> Vault server configuration:

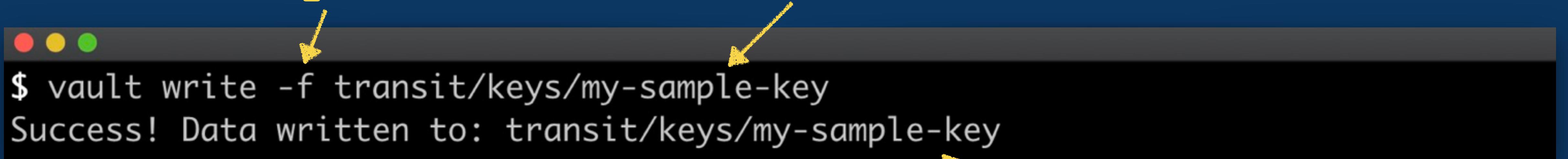
    Api Address: http://127.0.0.1:8200
        Cgo: disabled
    Cluster Address: https://127.0.0.1:8201
        Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201", max_request_duration: "1m30s", max_request_size: "33554432", tls: "disabled")
    Log Level: info
        Mlock: supported: false, enabled: false
    Storage: inmem
    Version: Vault v1.2.1
```

```
$ vault secrets enable transit
Success! Enabled the transit secrets engine at: transit/
```

# Command Line Key Generation

no key/value  
input

key name



A screenshot of a macOS terminal window. The title bar says "Terminal". The command entered is "\$ vault write -f transit/keys/my-sample-key". The response "Success! Data written to: transit/keys/my-sample-key" is displayed below the command.

```
$ vault write -f transit/keys/my-sample-key
Success! Data written to: transit/keys/my-sample-key
```

AES-GCM as default  
algorithm

[transit](#) < [my-sample-key](#)

## Create encryption key

**Name**

my-sample-key

**Type**

- ✓ aes256-gcm96
- chacha20-poly1305
- ecdsa-p256
- ed25519
- rsa-2048**
- rsa-4096



Enable convergent encryption

**Create encryption key**

**Cancel**

# Demo

# Crypto in the Cloud



# Cloud KMS Integration in Applications

**Every(?) cloud provider offers a KMS** - AWS KMS,  
Azure Key Vault, Google Cloud KMS

Typically **store a master key that never leaves the system**

Used to **encrypt and decrypt data stored someplace else**

# Secure Key Storage in External Services

**Tink supports the cloud Key Management System (KMS)** of Google Cloud KMS, Amazon KMS, and Android Keystore

- 1 Create a master key in the KMS
- 2 Provide Tink with KMS (Cloud) credentials
- 3 Tell Tink where to find the master key

# Master Key and AwsKmsClient

```
String MASTER_KEY_URI = "aws-kms://arn:aws:kms:...";  
  
KeysetHandle keysetHandle =  
    KeysetHandle.generateNew(  
        HybridKeyTemplates.  
        ECIES_P256_HKDF_HMAC_SHA256_AES128_GCM);  
  
keysetHandle.write(  
    JsonKeysetWriter.withFile(keysetFile),  
    new AwsKmsClient()  
        .withDefaultCredentials().getAead(MASTER_KEY_URI));
```

# Demo

# Summary

Use an established framework like **Google Tink** for your custom development and **integrate a Cloud KMS**

Use **Vault Transit Backend when already available in your environment**

Development is only part of the story - crypto requires **DevOps for best security**



Marienstr. 17  
70178 Stuttgart

dominik.schadow@bridging-it.de  
[www.bridging-it.de](http://www.bridging-it.de)

Blog [blog.dominiksshadow.de](http://blog.dominiksshadow.de)  
Twitter @dschadow

## **Google Tink Demo Project**

<https://github.com/dschadow/JavaSecurity>

## **Vault Demo Project**

<https://github.com/dschadow/CloudSecurity>

## **Google Tink**

<https://github.com/google/tink>

## **Vault**

<https://www.vaultproject.io>

## **Vault Transit Secrets Engine**

<https://www.vaultproject.io/docs/secrets/transit/index.html>

## **Pictures**

<https://www.dreamstime.com>

