# NEXT GENERATION INTERNET

## Key management

Christian Grothoff

13.03.2026

NGI TALER

# Learning Objectives

How can we protect confidentiality of private keys?

How does Shamir Secret Sharing work?

Key escrow and recovery: From Shamir to Anastasis

What are threshold signatures?

What does key management look like in practice?

## Part I: How can we protect confidentiality of private keys?

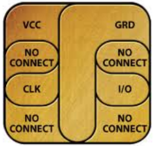
# Software based Personal Security Environments (PSE): PKCS#12

PKCS#12 is the most common format for software PSEs:

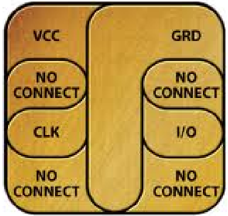
- PKCS#12 is a file container format used for storage and transport of private keys (and possibly certificates).
- Information is protected with a password-based symmetric key (e.g. a password).
- The security of a software PKCS#12 is based on the strength of the password protecting it.

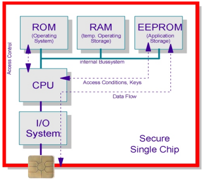
Problem: A PKCS#12 soft-token may be copied unnoticed.

# Smartcards and Cryptotokens


# Properties of Crypto-tokens/cards




- Crypto-cards have the ability of a secure container for secret data and have an executive platform for cryptographic algorithms.
- A Crypto-card looks like a “Black Box” from the outside, where some operations can only be used over a very restrictive hard- and software interface which is able to enforce specific security policies.
- Access to sensitive data areas (i.e. private keys) is physically “impossible” from the outside.

# Example: Yubikey and Personal Identity Verification (PIV)

- Yubikey provides Smart Card functionality based on the Personal Identity Verification (PIV) interface specified in NIST SP 800-73.
- Yubikeys perform RSA or ECC sign/decrypt operations using a private key stored on the token, through common interfaces such as PKCS#11.
- Supported key sizes: RSA 2048 or ECC 256/384.
- The “universal smartcard minidriver” provides “standard smart” functionality and additional certificate and PIN management features.
- Special Yubikeys obtained FIPS 140-2 security level certification.

# Hardware Security Modules (HSM)

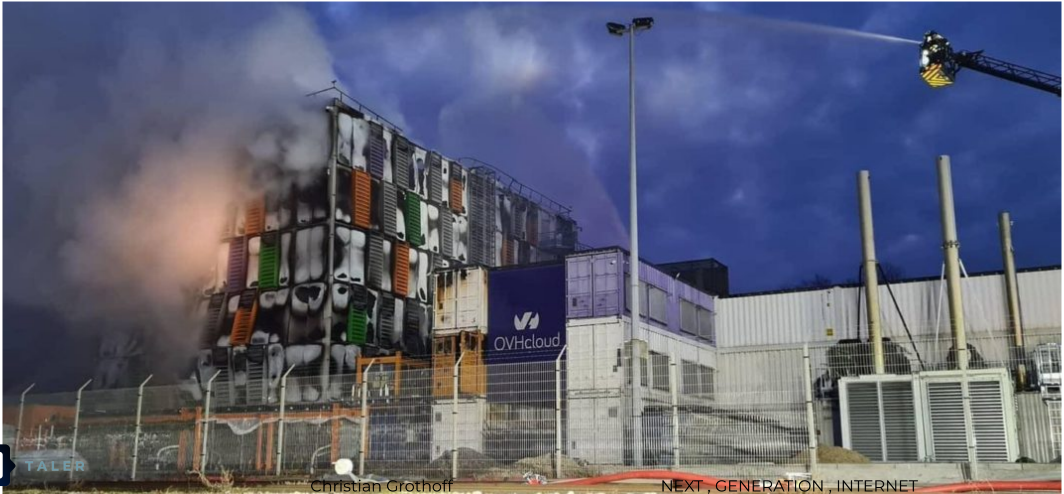
Common functionality:

- Secure storing and use of keys
- Random number generator
- Key pair generation
- Digital signing
- Key archiving
- Acceleration for crypto schemes

Should protect keys against:

- Mechanical & chemical attacks
- Temperature attacks
- Manipulation of voltage

# Availability



## Part II: How does Shamir Secret Sharing work?

# Problem 1: Availability

If you give one person (or data center) a secret, it may get lost.

# Problem 1: Availability

If you give one person (or data center) a secret, it may get lost.

$\Rightarrow$ So give it to more than one person (or data center)!

# Problem 2: Confidentiality

If you give many entities a secret, it may get disclosed.

# Problem 2: Confidentiality

If you give many entities a secret, it may get disclosed.

$\Rightarrow$ So give them only a key share!

# Problem 3: Scalability

If you want $k$ out of $n$ entities to coordinate to recover a secret, there are

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

combinations to consider.

# Problem 3: Scalability

If you want $k$ out of $n$ entities to coordinate to recover a secret, there are

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

combinations to consider.

$\Rightarrow$ Use polynominals!

# Polynominals

A polynominal of degree $k-1$ is fully determined by $k$ data points

$$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_{k-1}, y_{k-1}),$$

where no two $x_j$ may be identical.

# Lagrange interpolation

The interpolation polynominal in the Lagrange form is:

$$L(x) := \sum_{j=0}^{k} y_j \ell_j(x)$$

where

$$\ell_j(x) := \prod_{\substack{0 \le m \le k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_0)}{(x_j - x_0)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_k)}{(x_j - x_k)} \qquad (2)$$

for $0 \le j \le k$.

# Practical considerations

- Our secrets will typically be integers. Calculations with floating points are *messy*.

$\Rightarrow$ Use finite field arithmetic, not $\mathbb{R}$.

# Real world scalability

| n / k | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | | 1 | 3 | 6 | 10 | 15 |
| 3 | | | 1 | 4 | 10 | 20 |
| 4 | | | | 1 | 5 | 15 |
| 5 | | | | | 1 | 6 |
| 6 | | | | | | 1 |

Other values:

- $\binom{10}{5} = 252$
- $\binom{20}{10} = 184756$
- $\binom{30}{15} = 155117520$

# Do we have a scalability problem?

How many people do you have to share your secrets with?

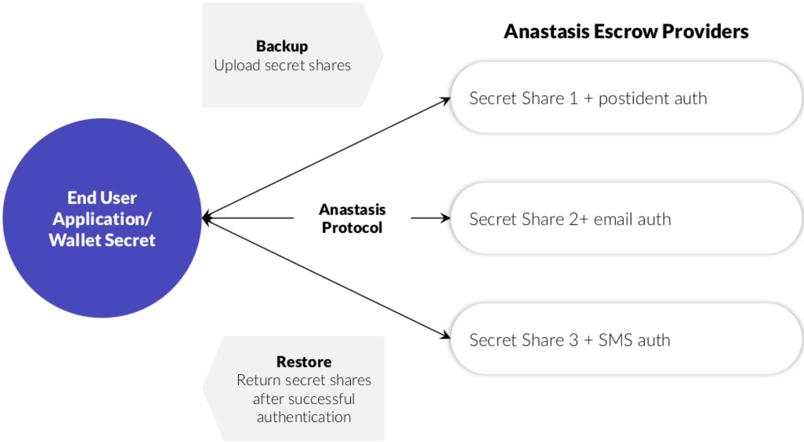
How many people realistically participate in recovery?

# Part III: Key escrow and recovery: From Shamir to Anastasis

# What is GNU Anastasis? [2]

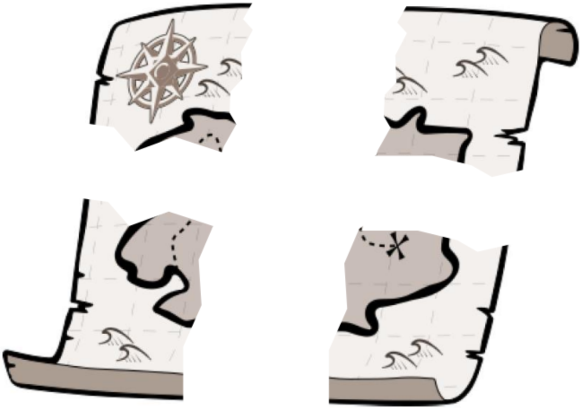
- Distributed key escrow and recovery service
- Users split their secret keys and distribute shares across multiple service providers
- Only the authorized user can recover the key by following standard authentication procedures
- Service providers learn nothing about the user, except possibly some details about how to authenticate the user

# Overview


Backup
Upload secret shares
Anastasis Escrow Providers
Secret Share 1 + postident auth
End User
Application/
Wallet Secret
Anastasis
Protocol
Secret Share 2+ email auth
Secret Share 3 + SMS auth
Restore
Return secret shares
after successful
authentication

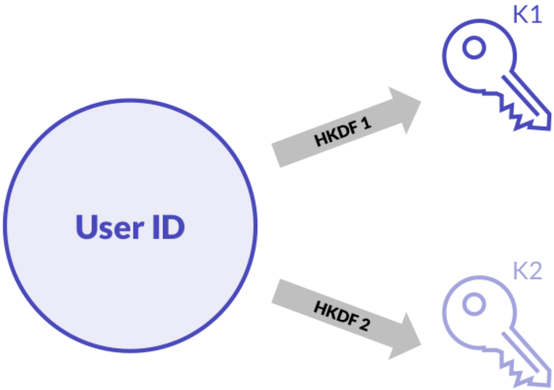
# Step 1: Enter secret information

# Step 2: Split information

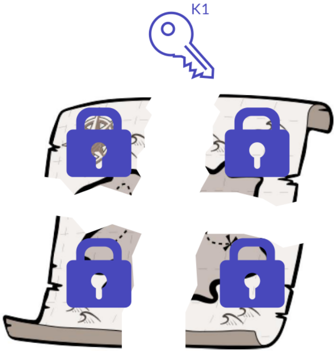


# Step 3: Hash user identification

# Step 4: Key derivation



# Step 5: Encrypt parts



# Step 6: Add truth

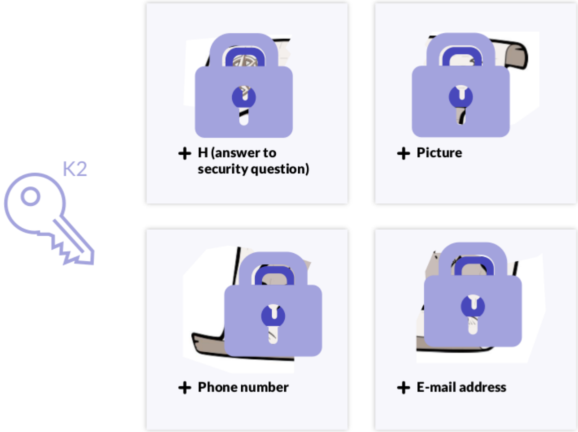
+ H (answer to security question)

+ Picture

+ Phone number

+ E-mail address

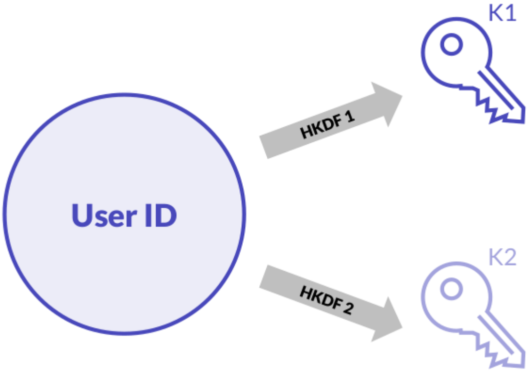
# Step 7: Encrypt truth



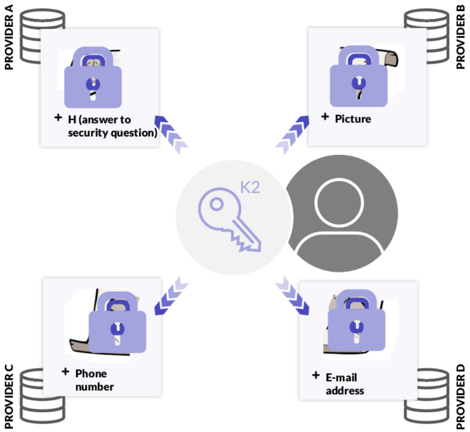
# Step 8: Store data

# Step 9: User identification

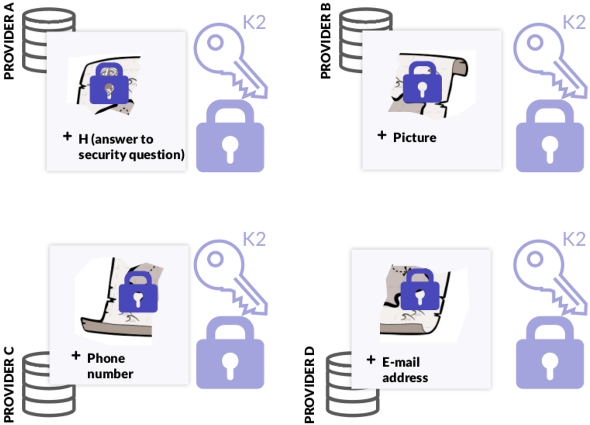
# Step 10: Key derivation



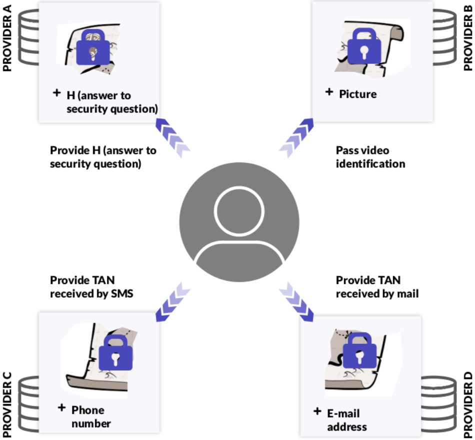
# Step 11: Provide key



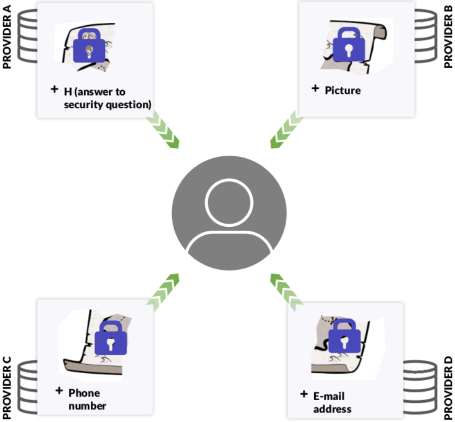
# Step 12: Decrypt truth



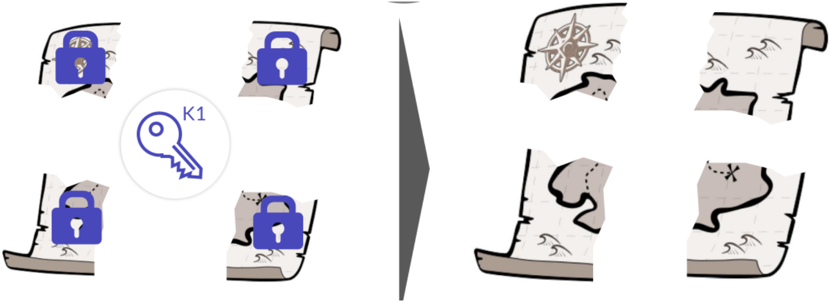
# Step 13: Authenticate



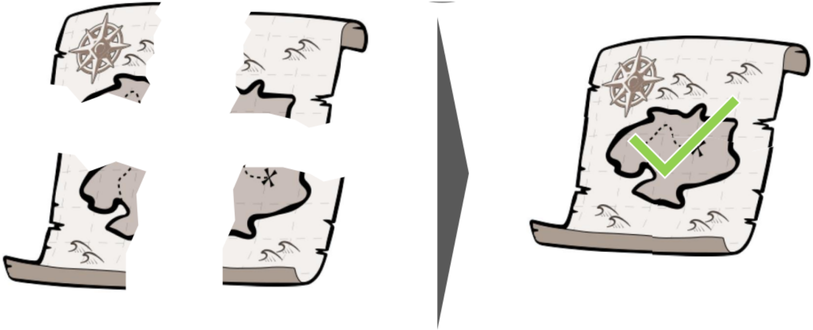
# Step 14: Receive parts



# Step 15: Decrypt parts



# Step 16: Reassembly



# Reality is more complex

- Policies to allow more flexible splitting than 4/4
- Recovery document to remember policies and providers
- Distinction between core secret and master secret
- Payment processing
- Provider salts
- Anti-DoS provisions in protocol / request limits
- Versioning
- Liability limitations

# Part IV: What are threshold signatures?

# Everything is Broken

Alice wants to create a cryptographic signature, but:

- No single piece of hardware is trusted
- No single service provider is trusted

But: Using $t$ independent signature service providers might be ok!

If we need $t$ providers, we probably should initially sign up with $n$ providers so that we can still create signatures if only $t/n$ are available...

# FROST [1]

Flexible Round-Optimized Schnorr Threshold (FROST) is a $t$-out-of-$n$ threshold signature scheme:

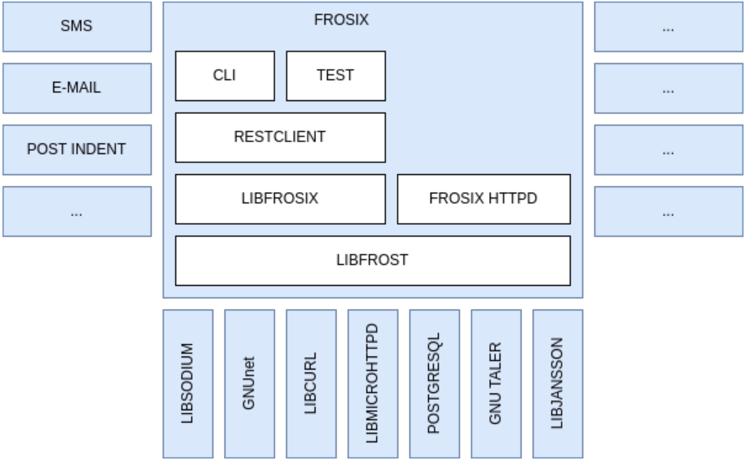
- Distributed key generation protocol can be used to ensure private key is never stored on a single device
- $t$ providers required to collaborate to create digital signature

# FROSIX

Free Software implementation for threshold signatures using FROST with:

- RESTful API to interact between signer and signing services
- Configurable authentication methods to authorize creation of signature
- Client should still use multiple devices (for authorization and to check distributed key generation) to avoid single point of failure
- Command-line tool to interact with FROSIX service providers

# System components overview



# FROSIX: Future Work

Open issues:

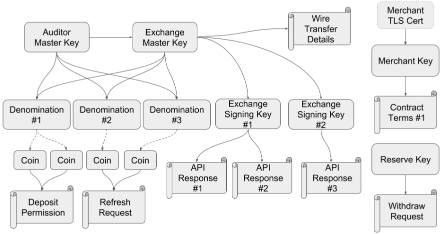
- Support additional signature schemes beyond EdDSA
- Pay signature service providers for their service
- Graphical user interfaces (Gtk+, WebUI, ...)

# Part V: What does key management look like in practice?

# Key management in GNU Taler

GNU Taler has many types of keys:

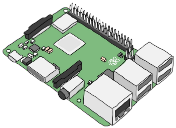
- Coin keys (EdDSA + ECDHE)
- Denomination keys (blind)
- Online message signing keys
- Offline key signing keys
- Merchant keys
- Auditor key
- Security module keys
- Transfer keys (ECDHE)
- Wallet keys
- *TLS keys, DNSSEC keys*



# Offline keys

Both exchange and auditor use offline keys.

- Those keys must be backed up and remain highly confidential!
- We recommend that computers that have ever had access to those keys to NEVER again go online.
- We recommend using a Raspberry Pi for offline key operations. Store it in a safe under multiple locks and keys.
- Apply full-disk encryption on offline-key signing systems.
- Have 3–5 full-disk backups of offline-key signing systems.



# Online keys

The exchange needs RSA and EdDSA keys to be available for online signing.

- Knowledge of these private keys will allow an adversary to mint digital cash, possibly resulting in huge financial losses.
- The corresponding public keys are certified using Taler's public key infrastructure (which uses offline-only keys).

`taler-exchange-offline` can be used to **revoke** the online signing keys, if we find they have been compromised.

# Protecting online keys

The exchange needs RSA and EdDSA keys to be available for online signing.

- `taler-exchange-secmod-*` are the only processes that must have access to the private keys. These secmod processes should run under a different UID, but share the same GID with the exchange.
- The secmods generate the keys, allow `taler-exchange-httpd` to sign with them, and eventually delete the private keys.
- Communication between secmods and `taler-exchange-httpd` is via a UNIX domain socket.
- Online private keys are stored on disk (not in database!) and should NOT be backed up (RAID should suffice). If disk is lost, we can always create fresh replacement keys!

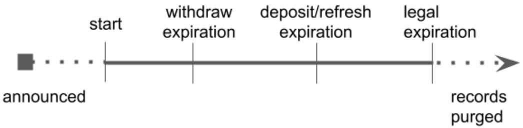
# Online keys

**What happens if private keys are disclosed or lost?**

# Private key disclosure

- Auditor and exchange can detect this once the total number of deposits exceeds the amount of digital cash put into circulation.
- At this point, signing keys are *revoked*. Users of *unspent* legitimate coins obtain a *refund*.
- The financial loss of the exchange is *bounded* by the number of legitimate coins signed with the private key.

⇒ Taler frequently rotates denomination signing keys and deletes a private key after the signing period of the respective key is over.



# References I


Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood.
Two-round threshold schnorr signatures with frost.
Technical report, IRTF, 2023.
https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/.

Dominik Samuel Meister and Dennis Neufeld.
Anastasis: Password-less key recovery via multi-factor multi-party authentication.
Master's thesis, Bern University of Applied Sciences, June 2020.

# Acknowledgements


Co-funded by the European Union (Project 101135475).

Co-funded by the European Union

Co-funded by SERI (HEU-Projekt 101135475-TALER).

**Project funded by**

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs, Education and Research EAER
**State Secretariat for Education, Research and Innovation SERI**

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union.

Neither the European Union nor the granting authority can be held responsible for them.