



The sign-if-logged tool

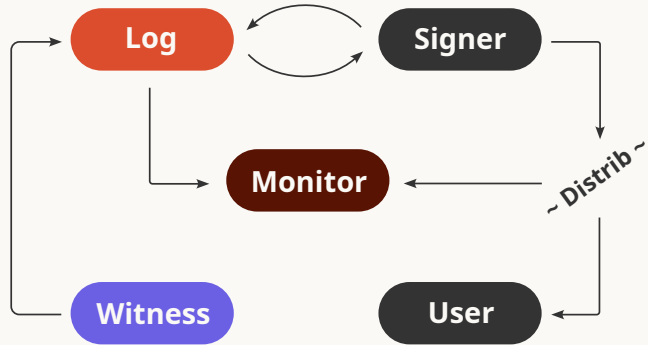
2026-10-01

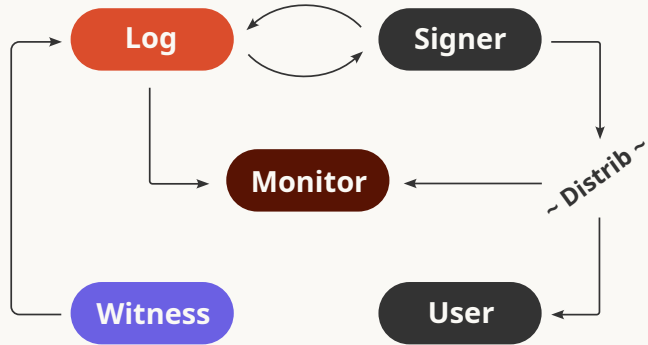
Niels Möller

Slides: <https://git.glasklar.is/nisse/tds-2026>

- 1 Introduction
- 2 How to use it
- 3 How it works
- 4 Conclusions and further work

Sigsum





Every acceptable signature is discoverable

Some verifiers don't know about Sigsum.

- OpenSSH Signatures
- PGP signatures, e.g., Debian RELEASE files
- Secure boot signatures
- Performance constrained verifiers

Some verifiers don't know about Sigsum.

- OpenSSH Signatures
- PGP signatures, e.g., Debian RELEASE files
- Secure boot signatures
- Performance constrained verifiers

Idea: Enforce transparency logging *prior* to signing

- Keep signing key in a secure hardware device
- Configure device to require proof of logging before signing

Some verifiers don't know about Sigsum.

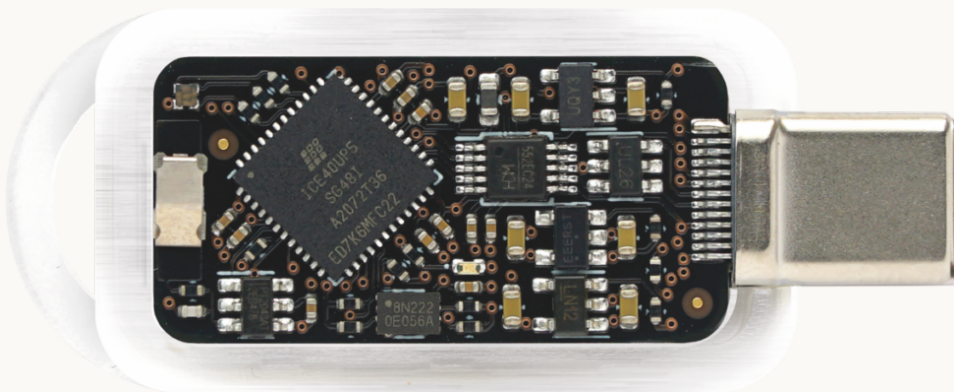
- OpenSSH Signatures
- PGP signatures, e.g., Debian RELEASE files
- Secure boot signatures
- Performance constrained verifiers

Idea: Enforce transparency logging *prior* to signing

- Keep signing key in a secure hardware device
- Configure device to require proof of logging before signing

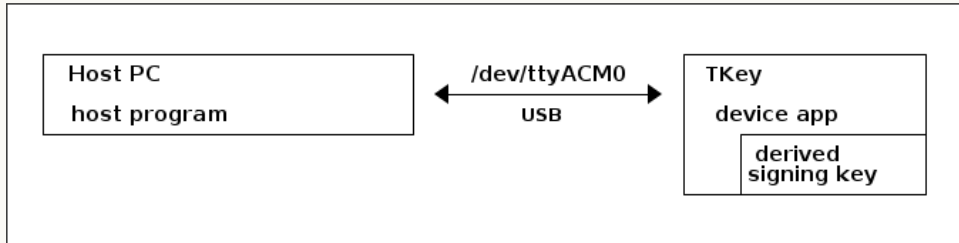
Key owner can discover all data that can be signed

Tillitis TKey

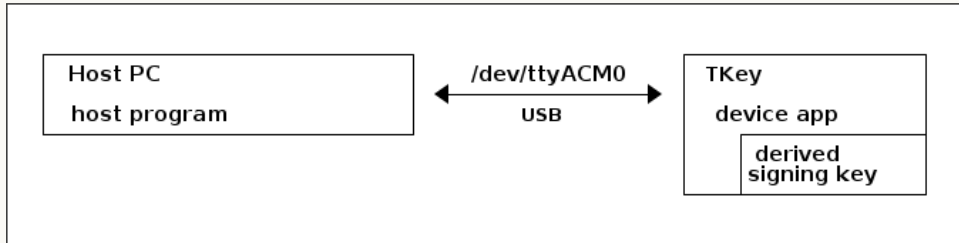


- Open hardware USB device
- Runs arbitrary apps
- Firmware provides app-specific secrets
- App's secret = $H(\text{device secret}, H(\text{app}))$

Host program and device app

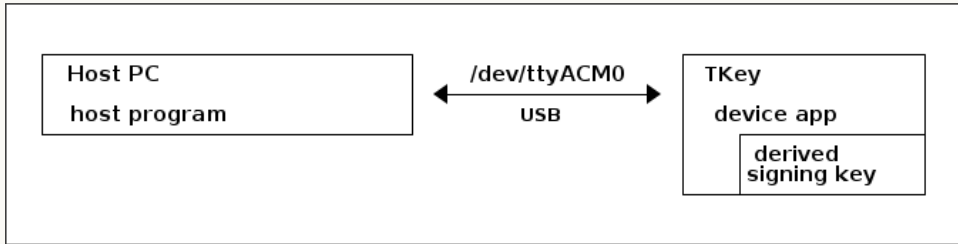


Host program and device app



```
func NewClient(p string) (...) {  
    tkey := tkeyclient.New()  
    if err := tkey.Connect(p); ...  
    return &Client{  
        port:  port,  
        tkey:  tkey,  
    }, nil  
}
```

Host program and device app

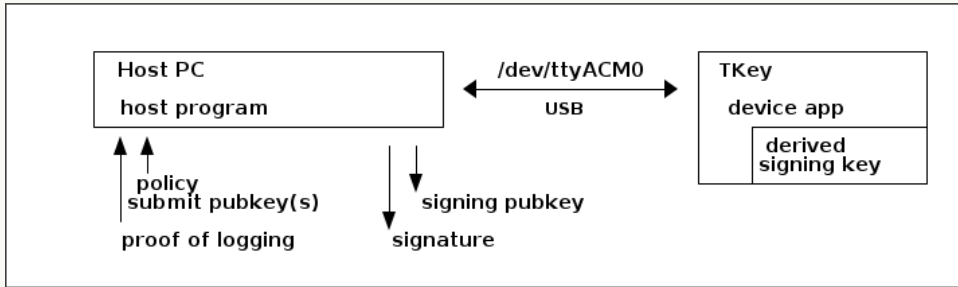


```
func NewClient(p string) (...) {  
    tkey := tkeyclient.New()  
    if err := tkey.Connect(p); ...  
    return &Client{  
        port:  port,  
        tkey:  tkey,  
    }, nil  
}
```

```
/* Timeout in seconds. */  
#define TOUCH_TIMEOUT 30  
  
static bool  
touch (void) {  
    return touch_wait (  
        LED_GREEN, TOUCH_TIMEOUT);  
}
```

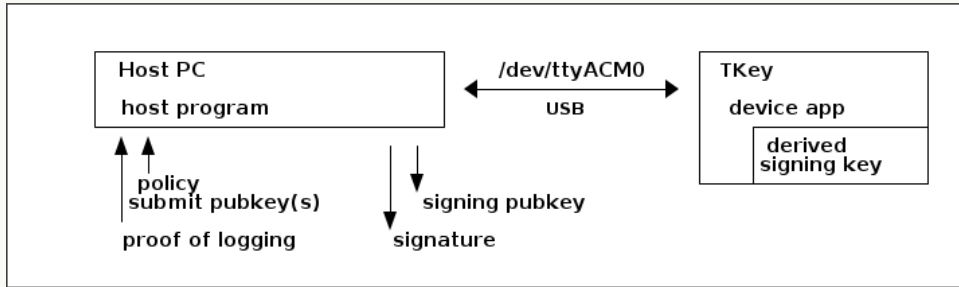
HOW TO USE IT

System setup



- Device app is a constrained signing oracle
- Host program sends device app to TKey firmware
- Host program talks to the running device app

System setup

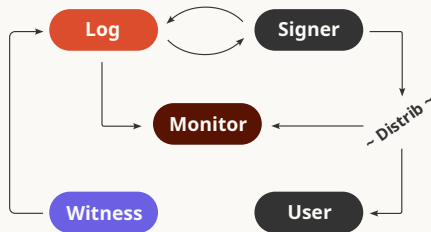


- Device app is a constrained signing oracle
- Host program sends device app to TKey firmware
- Host program talks to the running device app

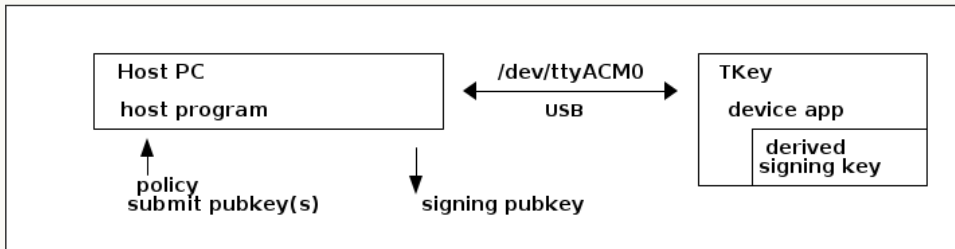
We next look closer at the UI of the host program

To submit artifacts to Sigsum:

- Generate Sigsum submit key(s)
`sigsum-key generate -o submit.key`
- Select a Sigsum policy
- Set up monitoring of the submit key(s)

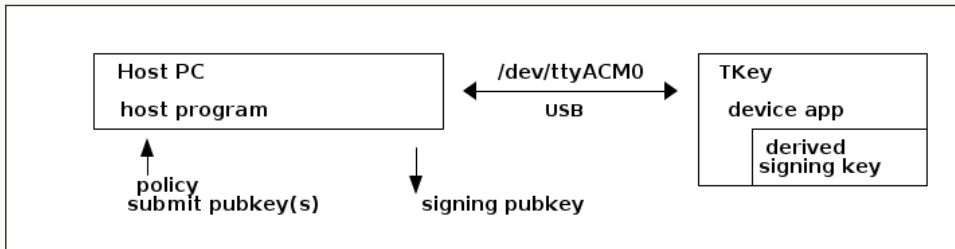


Configure sign-if-logged



- Insert TKey and configure it
`sign-if-logged init --ssh -k submit-key.pub -P sigsum-test-2025-3`
- Extract public key
`sign-if-logged pubkey -o sign-key.pub`

Configure sign-if-logged

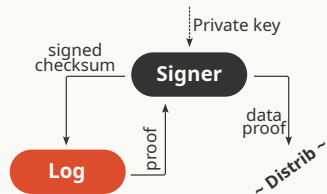


- Insert TKey and configure it
`sign-if-logged init --ssh -k submit-key.pub -P sigsum-test-2025-3`
- Extract public key
`sign-if-logged pubkey -o sign-key.pub`
- Convert to an OpenSSH `allowed_signers` file
`sed 's/^/user@example.org namespaces="file" /'
 < sign-key.pub > allowed_signers`

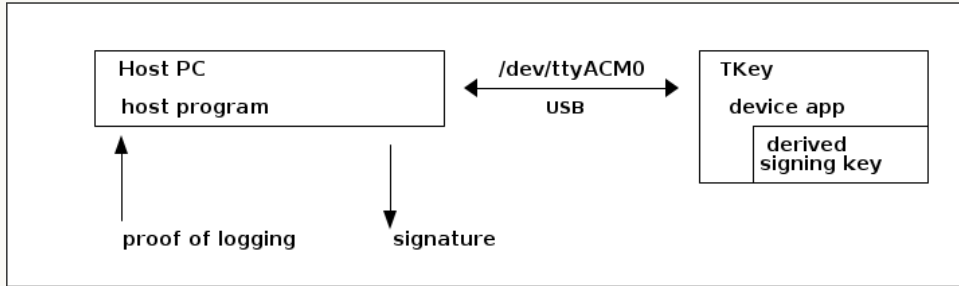
Artifact signing — first stage

Submit artifact to Sigsum:

- Produce the artifact to be signed
`echo "This is it" > the-artifact`
- Submit it to a Sigsum log, get back `the-artifact.proof`
`sigsum-submit -k submit-key -P sigsum-test-2025-3 the-artifact`



Artifact signing — second stage



- Insert TKey and configure it *identically* as before
`sign-if-logged init --ssh -k submit-key.pub -P sigsum-test-2025-3`
- Pass artifact and corresponding `.proof` to TKey device for signing
`sign-if-logged sign -o the-artifact.sig the-artifact`

1. Verify as a plain OpenSSH signature

```
ssh-keygen -Y verify -f allowed-signers -I user@example.org  
-n file -s the-artifact.sig < the-artifact
```

Discoverable signature

1. Verify as a plain OpenSSH signature

```
ssh-keygen -Y verify -f allowed-signers -I user@example.org  
-n file -s the-artifact.sig < the-artifact
```

2. Key-owner monitors log

```
sigsum-monitor -P sigsum-test-2025-3 submit-key.pub
```

Discoverable signature

1. Verify as a plain OpenSSH signature

```
ssh-keygen -Y verify -f allowed-signers -I user@example.org  
-n file -s the-artifact.sig < the-artifact
```

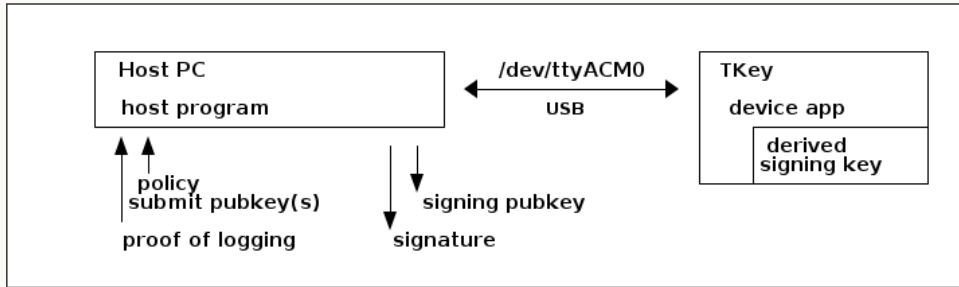
2. Key-owner monitors log

```
sigsum-monitor -P sigsum-test-2025-3 submit-key.pub
```

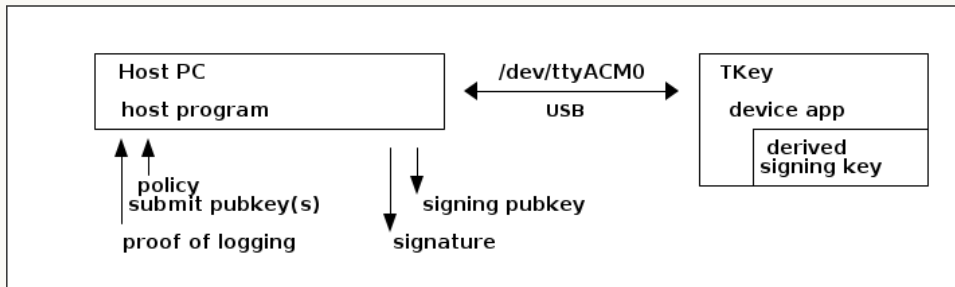
3. PROFIT!!!!1!

HOW IT WORKS

Recall system setup



Recall system setup



We next look closer at what is happening inside the TKey

TKey crash course:

- TKey inserted in a USB port of host computer
- Firmware starts, reads unique device secret (UDS)
- Host sends the firmware a TKey device app

TKey crash course:

- TKey inserted in a USB port of host computer
- Firmware starts, reads unique device secret (UDS)
- Host sends the firmware a TKey device app
- Firmware computes $H(\text{app})$ and compound device identity (CDI)
- $\text{CDI} = H(\text{UDS}, H(\text{app}))$

TKey crash course:

- TKey inserted in a USB port of host computer
- Firmware starts, reads unique device secret (UDS)
- Host sends the firmware a TKey device app
- Firmware computes $H(\text{app})$ and compound device identity (CDI)
- $\text{CDI} = H(\text{UDS}, H(\text{app}))$
- Firmware switches to app mode and starts app
- The running app has access to the CDI, but *not* the UDS.

TKey crash course:

- TKey inserted in a USB port of host computer
- Firmware starts, reads unique device secret (UDS)
- Host sends the firmware a TKey device app
- Firmware computes $H(\text{app})$ and compound device identity (CDI)
- $\text{CDI} = H(\text{UDS}, H(\text{app}))$
- Firmware switches to app mode and starts app
- The running app has access to the CDI, but *not* the UDS.

Any change to the app binary results in a different CDI

User's configuration, passed on to the TKey device app:

- Signature type (Ed25519 or ECDSA, raw or OpenSSH)
- A Sigsum policy
- A list of authorized submitter keys

User's configuration, passed on to the TKey device app:

- Signature type (Ed25519 or ECDSA, raw or OpenSSH)
- A Sigsum policy
- A list of authorized submitter keys
- If user presence should be required
- For OpenSSH, the signature namespace

User's configuration, passed on to the TKey device app:

- Signature type (Ed25519 or ECDSA, raw or OpenSSH)
- A Sigsum policy
- A list of authorized submitter keys
- If user presence should be required
- For OpenSSH, the signature namespace

The signing key depends on the configuration:

- Private key = HMAC(CDI, configuration)
- Corresponding public key is returned to the host

User's configuration, passed on to the TKey device app:

- Signature type (Ed25519 or ECDSA, raw or OpenSSH)
- A Sigsum policy
- A list of authorized submitter keys
- If user presence should be required
- For OpenSSH, the signature namespace

The signing key depends on the configuration:

- Private key = HMAC(CDI, configuration)
- Corresponding public key is returned to the host

Any change to the app binary or the configuration results in a different key

Host program:

- Packages message and a Sigsum proof, sends to TKey
- Depending on signature type, message may be prehashed with SHA256
- Gets back a signature or an error message
- On success, serializes signature in appropriate format

Host program:

- Packages message and a Sigsum proof, sends to TKey
- Depending on signature type, message may be prehashed with SHA256
- Gets back a signature or an error message
- On success, serializes signature in appropriate format

TKey device app:

- Requires that the Sigsum proof is valid
- Requires user presence if configured
- If all good, message is signed using configured algorithm

The device app uses a compiled Sigsum policy

- List of submitter pubkeys (all key lists sorted by key hash)
- List of log pubkeys
- List of witness pubkeys

The device app uses a compiled Sigsum policy

- List of submitter pubkeys (all key lists sorted by key hash)
- List of log pubkeys
- List of witness pubkeys
- Quorum rule is represented as bytecodes for a stack machine

ADD	Pop two values, push sum	00000001
X?	Push 1 if witnessed by X, otherwise 0	01xxxxxx
$\geq K$	Pop value, push 1 if $\geq K$, otherwise 0	10kkkkkk

The device app uses a compiled Sigsum policy

- List of submitter pubkeys (all key lists sorted by key hash)
- List of log pubkeys
- List of witness pubkeys
- Quorum rule is represented as bytecodes for a stack machine

ADD	Pop two values, push sum	00000001
X?	Push 1 if witnessed by X, otherwise 0	01xxxxxx
$\geq K$	Pop value, push 1 if $\geq K$, otherwise 0	10kkkkkk

Compilation from ASCII is fully reproducible

SUMMARY

Version v1.0.0 of sign-if-logged was released this summer

- Signatures become discoverable, even when verifiers are unaware of Sigsum
- Can distribute both the sign-if-logged signature and the Sigsum proof
- Only key holder can know that a TKey with sign-if-logged is used
- Open security hardware is fun!

`https://git.glasklar.is/sigsum/apps/sign-if-logged`

Software:

- Support OpenPGP signatures (in progress)
- RSA support, e.g., secure boot (on-device key generation is challenging)
- Support a separate submit key on the same TKey, for smoother UX
- Improved performance

`https://git.glasklar.is/sigsum/apps/sign-if-logged`

Software:

- Support OpenPGP signatures (in progress)
- RSA support, e.g., secure boot (on-device key generation is challenging)
- Support a separate submit key on the same TKey, for smoother UX
- Improved performance

Hardware:

- Attest that a key is managed by sign-if-logged on known hardware
- Upcoming TKey version has local storage, enables log and witness signer apps

`https://git.glasklar.is/sigsum/apps/sign-if-logged`

A small library for offline verification:

- Supports parsing Sigsum ASCII formats and OpenSSH pubkey files as an optional feature
- Supports verification of Sigsum proofs
- Defines the quorum byte codes used by `sign-if-logged`
- 1740 LOC (comments and crypto dependencies not included)
< 1000 LOC when built without ASCII parsers

`https://git.glasklar.is/sigsum/core/sigsum-c`