TKey, an open source/open hardware security token

tillitis

Michael "MC" Cardell Widerkrantz mc@tillitis.se

MC

MC

- Michael "MC" Cardell Widerkrantz, mc@tillitis.se
- Member of Technical Staff @ https://tillitis.se/
- Personal: https://hack.org/mc/ & gemini://gem.hack.org/mc/

Tillitis history

Tillitis history

TCR projects split in September 2022:

- Glasklar Teknik AB: System Transparency and the Sigsum transparency log. https://system-transparency.org/ https://sigsum.org/
- Tillitis: The hardware department! https://tillitis.se/

Tillitis Tkey - an open source hardware security token

Tillitis TKey

- A RISC-V computer for sensitive computations.
- Open source software and hardware (BSD2, CERN-OHL, some parts still GPLv2)
- https://github.com/tillitis
- https://dev.tillitis.se/

You can use it for

- Authentication.
- Digital signatures.
- Hardware root of trust.
- (Signed) random number generator.
- (Slow) encryption.
- A protected environment for sensitive computations.
- Other things... It's a general computer!

Advantages

- The client (computer/mobile) decides the function of the TKey.
- No need for new hardware for new functionality.
- Can write custom software.
- No risk for persistent threats.
- Secrets and private keys are not stored persistently on the device.

Advantages 2

- You can verify that the TKey comes from the vendor.
- You can make your own TKey, or just choose your own base hardware secret.

Basic TKey use with killer app: our SSH Agent

I want to login to something, typically a server (or Github, Gitlab or sign my Git commits):

- My client started the tkey-ssh-agent automatically.
- I insert the TKey into my client.
- `$ ssh some-server`
- The agent automatically loads the device app `signer`, Ed25519 signatures.
- TKey starts to blink the status LED.
- I touch the touch sensor.
- "I'm in!"
- Just like any security token.

But... It's a general computer!?

- How can we trust general applications sent from the client?
- What if we don't share anything between the apps?
- ...and guarantee software integrity?
- ...by measuring the apps,
- and creating new secrets for this combination of app and device.
- These secrets never leave the TKey.

Measured boot

- Use immutable code to measure the application, mix in a hardware secret: get a new identity!
- Inspired by TCG DICE (nee RIoT from Microsoft Research): Trusted boot for constrained environments.

Advantages of measured boot

- Software integrity is guaranteed.
- The measured identity can be used to create key material.
- Private keys are not stored on the TKey.
- Unlimited number of private keys.
- Secrets don't leak between device applications.

Different than verified boot

- **Forced** verified boot would lock TKey device apps to a specific vendor.
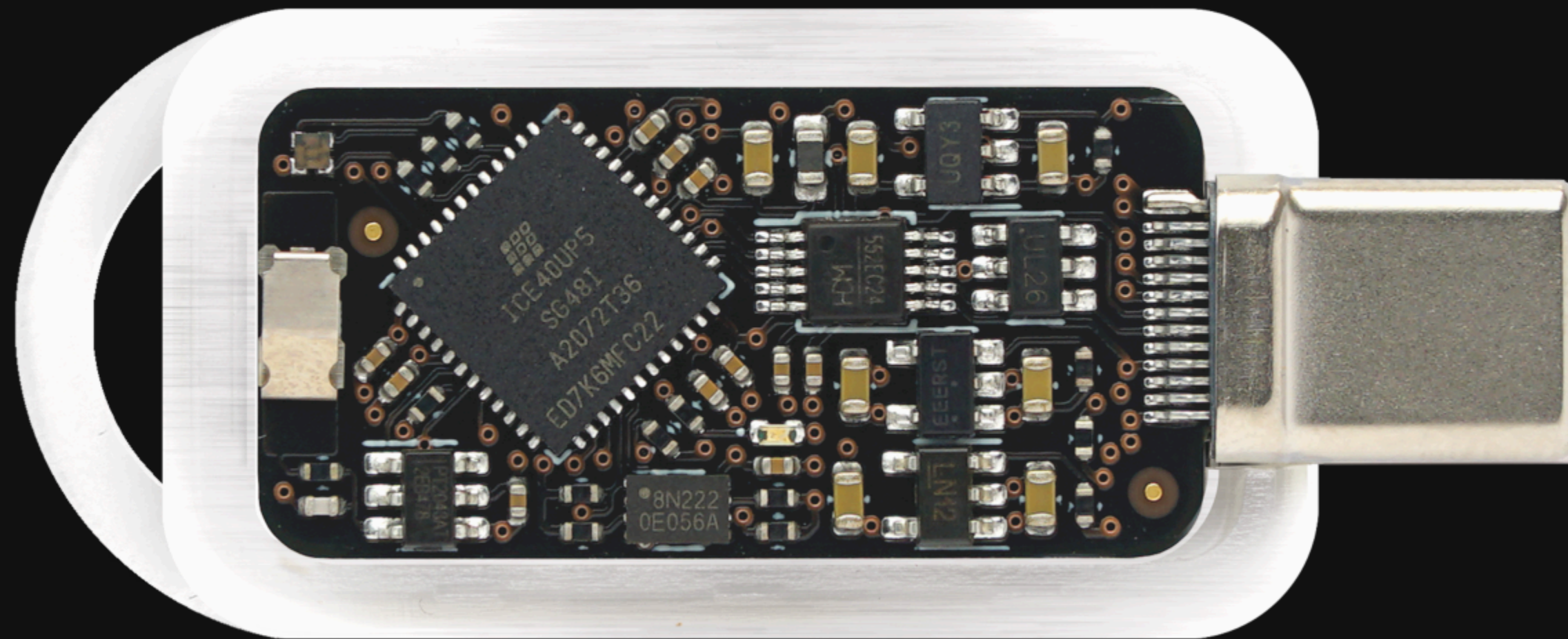- Unacceptable in an open platform.

Measured boot & Compound Device Identity

```
CDI = blake2s(UDS, blake2s(application), USS)
```

CDI is a cryptograpic mix of:

- Unique Device Secret (UDS) in hardware, something the user **has**.
- Optional User Supplied Secret (USS), something the user **knows**.
- Measurement (hash digest) of TKey device application, **integrity** of the application.

Hardware

TKey Specs

- 32 bit RISC-V PicoRV32 (Claire Wolf) softcore @ 18 MHz.
- 128 kiB RAM.
- Memory mapped hardware cores.
- Firmware mode/app mode.
- No interrupts.
- No persistent storage. (1 MiB flash usable during hardware dev)
- No OS.

Hardware design and testing tools

- Only open source tools!
- Chip design in Verilog.
- Limits choice of FPGA chip.
- Limits choice of PCB manufacturers.
- Yosys & NextPNR for synthesis, place & route, mapping and timing.
- Icestorm tools for bitstream generation.
- Developed NVCM programming tools.
- Icarus and Verilator for module and systems simulation.
- PCB design with KiCAD.
- Everything published!

FPGA chip

- Lattice iCE40 UltraPlus UP5K FPGA.
- Good support in open source tools.
- Lockable internal configuration memory (NVCM).
- Limited resources (~5 k LUTs, 120 kbit block RAM, 1024 kbit SPRAM).
- Paying for reversing other FPGA chips.

In the FPGA
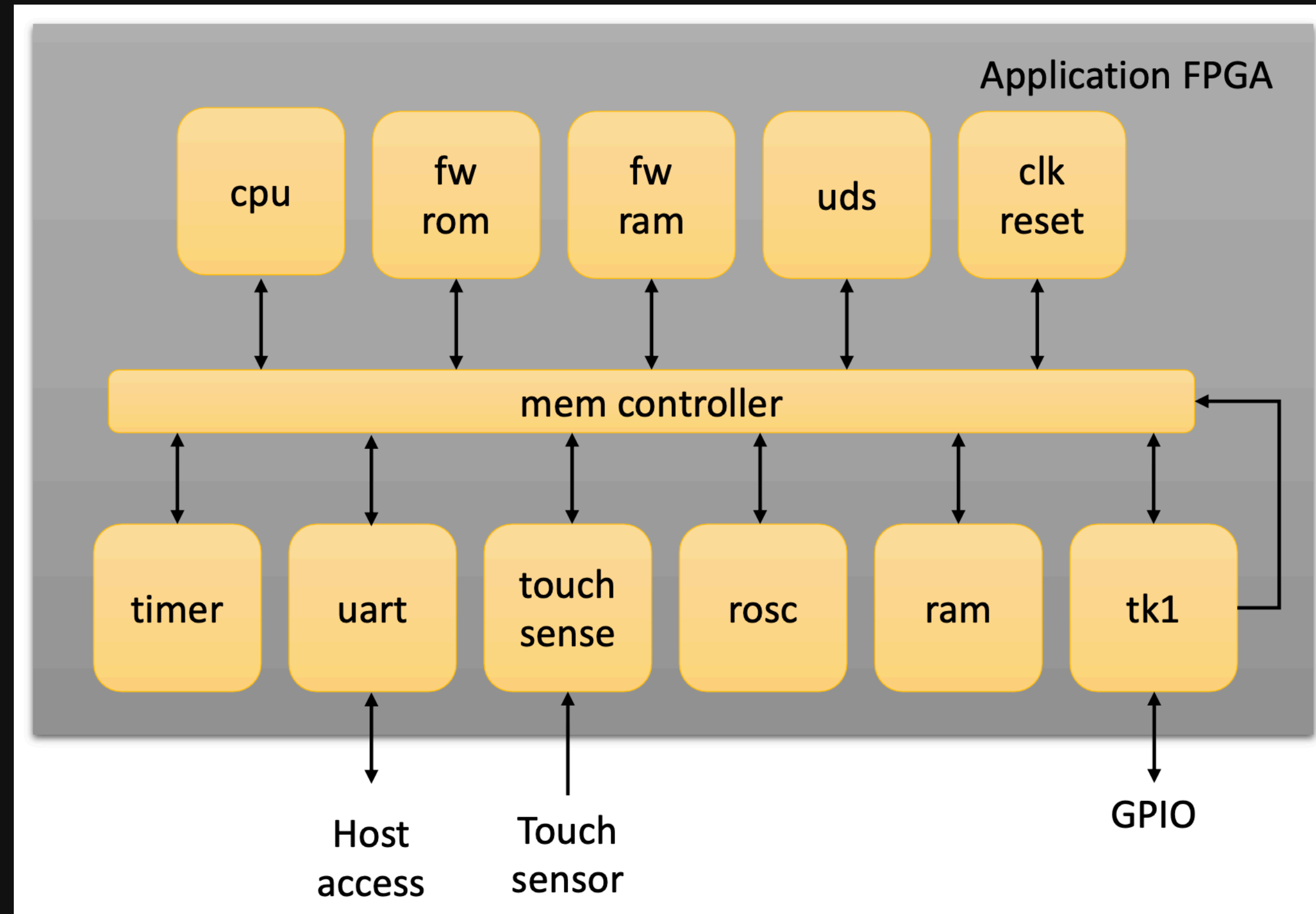
Security Monitor

- Write xor Execute (W^X) and memory access control.
- Registers to set protected memory area.
- FW_RAM always protected.
- Can be set by software, but not disabled.
- Watches program counter (PC).
- If PC enters forbidden area, feed it illegal instruction.
- Halts CPU - no exit condition.
- Outside core watches CPU and blinks LED red if halted.
- Very simple implementation compared to changing the CPU.

First ever TKey device app

```
        li a0, 0xff000024 # LED MMIO
        li a1, 0x1
loop:
        sw a1, 0(a0)
        addi a1, a1, 1

        li a2, 0
        li a3, 100000
delay:
        addi a2, a2, 1
        blt a2, a3, delay
        j loop
```

Our software

- Emulator: friendly qemu fork, also as OCI image.
- Simple firmware/boot loader. (~4 kiB)
- Some client applications: tkey-ssh-agent, tkey-verification, tkey-sign, tkey-random-generator, and their device applications.
- Client libraries: Go, Python supported. Also started Java (Imad Alihodzic).
- Device libraries: C supported, from LLVM-15.
- tkey-builder OCI image for podman/docker.

Some external software

- Device libraries: rusTkey (Danny van Heumen).
- Client libraries: Typescript using WebSerial (Mihir Patil et al), part of Cryptum.
- Experimental bringup of Zig (Morten Linderud and Mikael Ågren independently).
- Tinygo (Ron Evans).
- `tkey-age-plugin`: A plugin to the age encryption tool. (Daniel Lublin)
- All known projects: https://dev.tillitis.se/projects/

Supply chain

- PCB assembled Sweden.
- FPGA provisioned in airgapped environment at Tillitis HQ, Gothenburg.
- User verifiable firmware and identity.

TKey Unlocked & Programmer Board

- Unprovisioned TKey - empty, unlocked NVCM.
- Programming board for NVCM and SPI flash.
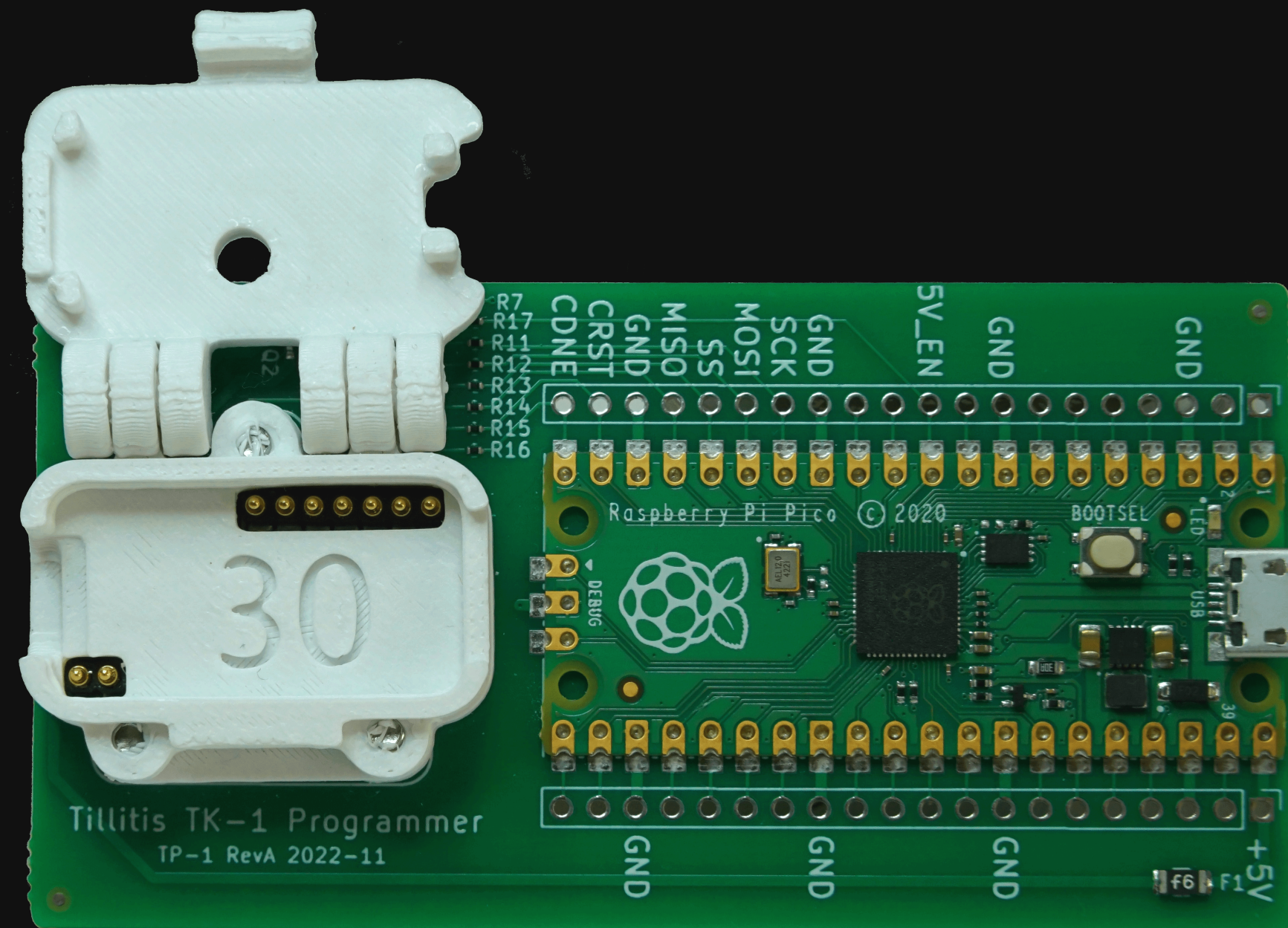
Status

- First official hardware release in March 2023: https://shop.tillitis.se/
- Several client apps available for Linux, macOS, and Windows.
- Reproducible builds:
  - For FPGA bitstream, firmware.
  - For all TKey device apps.
  - For client apps, but not on macOS (shared libs).
- Device verification service up and running.
- Everything released on Github: https://github.com/tillitis/

Future

- Writing more apps, supporting more use cases.
- Native USB HID.
- System reset from device app.
- Secure persistent storage.
- Extended memory access control.
- Faster UART (500 kBaud) & hardware flow control.
- 18 -> 24 MHz.

Later/other people

- Smaller version.
- Biometrics?
- FIDO2 app.
- Looking at other FPGA chips.
- HSM using TKey design with a Lattice ECP5 FPGA (Joachim Strömbergson)
- Trust anchor in a bigger computer?

Summary

- A new RISC-V computer with interesting hardware features
- in a USB stick form factor
- with no persistent state
- that uses measured boot to create unique identities based on what the user **has**, **knows**, and **the software integrity**.
- Client & device SDKs available.
- Custom SSH Agent in Go and other useful apps.
- Devices are user verifiable from production to end users.
- Open licences (BSD2, CERN-OHL, some parts still GPLv2)
- All known projects: https://dev.tillitis.se/projects/

The End

- Michael "MC" Cardell Widerkrantz, **mc@tillitis.se**, **mc@hack.org**
- General inquiries, **hello@tillitis.se**
- #tillitis @ irc.oftc.org, #tillitis:matrix.org
- https://tillitis.se/
- https://shop.tillitis.se/
- https://dev.tillitis.se/

Verifying the TKey

- Load a signer app, get public key and firmware digest.
- Do a challenge/response to prove possession of private key.
- Vendor signature over a message containing serial number, device public key, and firmware digest.
- Publish the vendor signature over the message, indexed by serial number.
- User verifies by querying for serial number.
- Looks up the published data, the vendor signature, for this serial number.
- Runs same device app to recreate the vendor signed messsage.
- Challenge/response - prove that the TKey app has same private key.
- Signed by vendor?
- Genuine!