

TamaGo

Bare metal Go for ARM/RISC-V SoCs

Secure embedded unikernels
with drastically reduced attack surface

Andrea Barisani

@AndreaBarisani - andrea.bio

andrea@inversepath.com

Andrea Barisani

Information Security Engineer and Researcher

Founder: **INVERSE**  **PATH** (acquired in 2017)

Head of Hardware Security:  

Maker of the USB armory

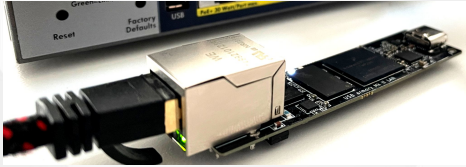


Speaker at too many conferences...

Background focused on security auditing and engineering on safety critical systems in the automotive, avionics, industrial domains.



Motivation: USB armory firmware



The USB armory is a tiny, but powerful, embedded platform for personal security applications.

Designed to fit in a pocket, laptops, PCs, server and networks (LAN model with PoE).

The USB armory targets the following primary applications:

- Encrypted storage solutions
- Hardware Security Module (HSM)
- Enhanced smart cards
- Electronic vaults (e.g. cryptocurrency wallets) and key escrow services
- Authentication, provisioning, licensing tokens
- USB firewall



open hardware



Motivation

In an ideal world **you should be free to choose the programming language you prefer.**

In an ideal world **all compilers would generate machine code with the same efficiency.**

However in real world lower specs heavily dictate language choices:

Microcontroller (MCU) firmware == unsafe¹ low level languages (C)



Examples: cryptographic tokens, cryptocurrency wallets, hardware data diodes, lower specs IoT and “smart” appliances.

¹ **Pro tip:** certification does not matter.



Motivation

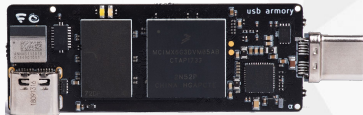
In an ideal world using **higher level languages should not entail complex dependencies.**

In an ideal world **higher level languages should reduce complexity.**

Complexity should be reduced for the entire environment, not just being shifted away.

However in real world higher specs heavily dictate OS requirements:

System-on-Chip (SoC) firmware == complex OS + safe (or unsafe¹) languages



Examples: TEE applets, infotainment units, avionics gateways, home routers, higher specs IoT and “smart” appliances.

¹ Privileged C-based apps running under Linux to “parse stuff” are very common, like your car infotainment/parking ECU.



Killing C

When security matters software and hardware optimizations matter less.

This means that less constrained hardware (e.g. SoCs in favor of MCUs) and higher level code are perfectly acceptable.

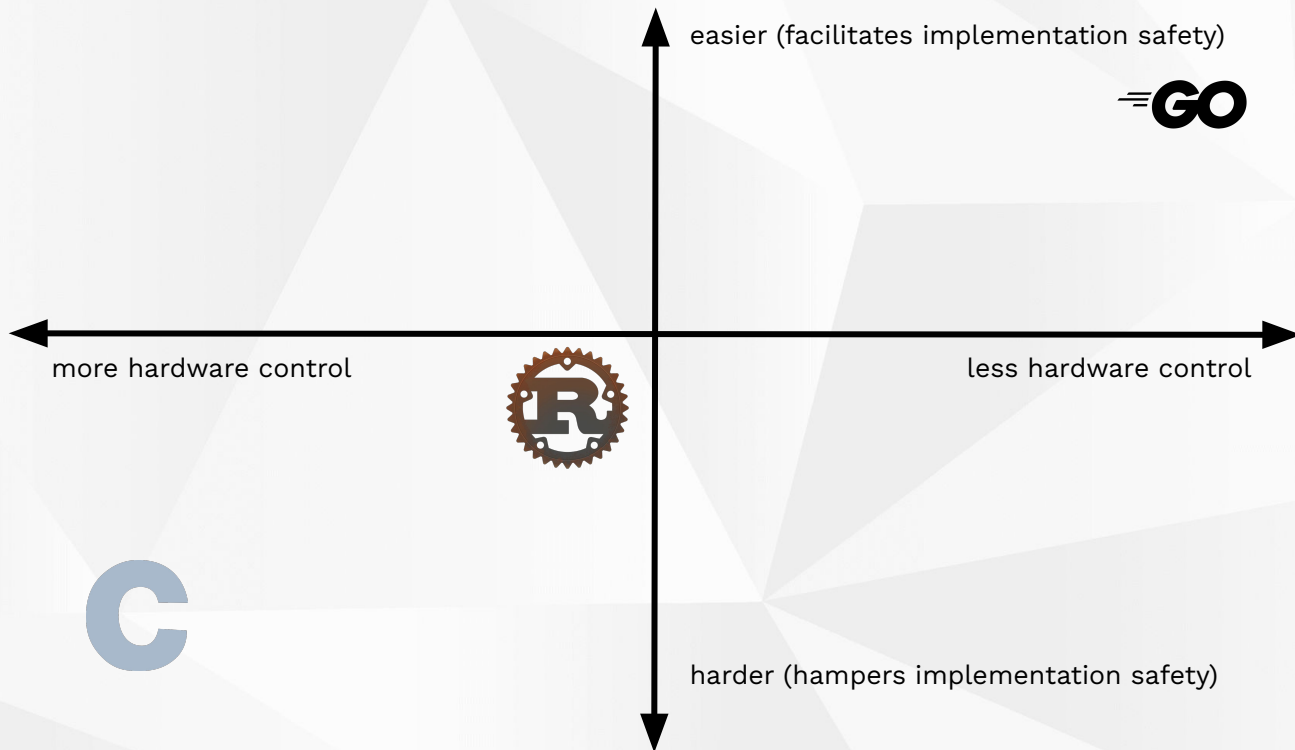
However high level programming typically entails several layers (e.g. OS, libraries) to serve runtime execution.

TamaGo spawns from the desire of **reducing the attack surface** of embedded systems firmware by **removing any runtime dependency on C code and inherently complex Operating Systems**.

In other words we want to **avoid shifting complexity around** and run a **higher level language**, such as Go in our effort, **directly on the bare metal**.



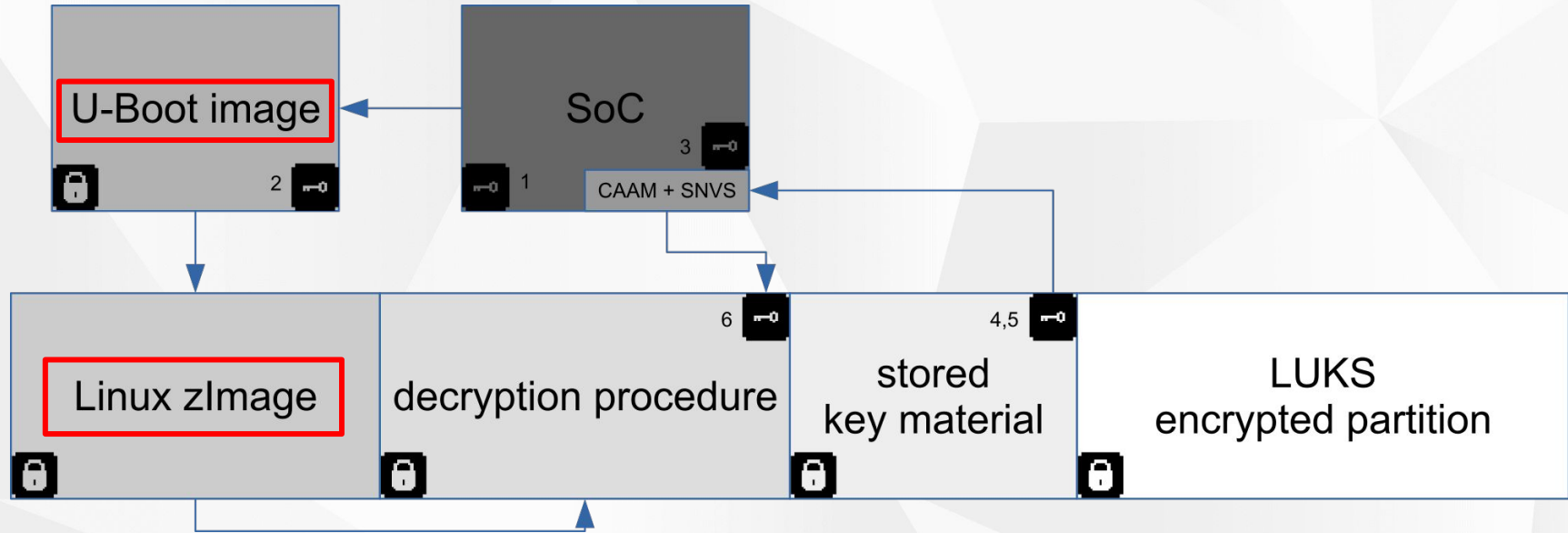
Speed vs Safety



Disclaimer: chart presented for discussion and not to claim that language X is better than language Y, also scale is subjective.



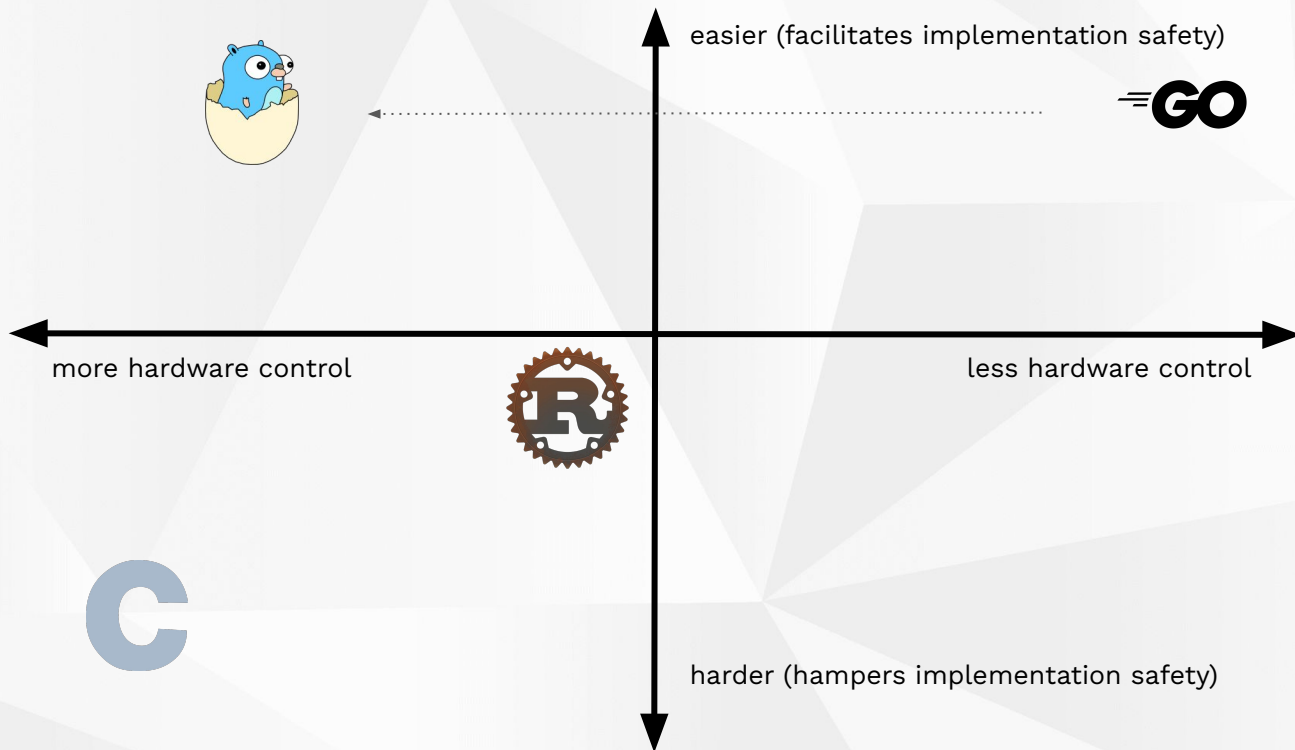
Reducing the attack surface



Typical secure booted firmware with authentication and confidentiality, taken from USB armory implementation example (NXP i.MX6UL).



Speed vs Safety



Disclaimer: chart presented for discussion and not to claim that language X is better than language Y, also scale is subjective.



Unikernels / library OS

Unikernels¹ are a single address space image to executed a “library operating system”, typically running under bare metal.

The focus is reducing the attack surface, carrying only strictly necessary code.

“True” unikernels are mostly unicorns, as a good chunk of available ones do not fit in this category and represent “fat” unikernels running under hypervisors and/or other (mini) OSES And just shift around complexity (e.g. the app is PID 1).

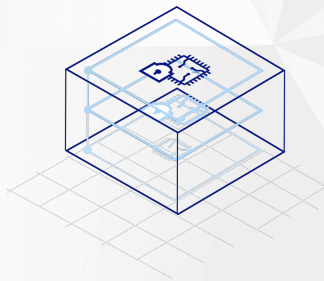
Apart for some exceptions there is always still a lot of C/dependencies involved in the underlying OS, drivers or hypervisor.

Running or importing *BSD kernels

Rump kernels (NetBSD based)
OSv (re-uses code from FreeBSD)

Running under hypervisor and 3rd party kernel

MirageOS (Solo5)
ClickOS (MiniOS)



Running under hypervisor

Nanos (Xen/KVM/Qemu) HalVM (Haskell, Xen)
LING (Erlang, Xen) RustyHermit (KVM)

Bare metal

GRISP (Erlang)
IncludeOS

¹ <https://en.wikipedia.org/wiki/Unikernel>



Unikernel security

From a security standpoint leveraging on Unikernels (whatever the kind) to run multiple applications or an individual C applications is not ideal¹.

Having an industry standard OS is necessary to support required security measures which otherwise are not present or rather primitive on most Unikernels.

Again, we want to **kill C** from the entire environment while keeping code efficiency, developing drivers having “only” to worry about interpreting reference manuals.

Unlike most unikernel projects we focus on **small embedded systems**, not the cloud.

We chose **Go** for its shallow learning curve, productivity, strong cryptographic library and standard library.

Languages like Rust have already proven they role in bare metal world, Go on the other hand needs to ... and it really can!

¹ <https://research.nccgroup.com/2019/02/04/assessing-unikernel-security/>



TamaGo in a nutshell

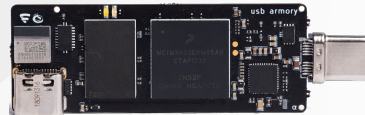
TamaGo is made of two main components.

- A **minimally**¹ patched Go distribution to enable `GOOS=tamago` support, which provides freestanding execution on `GOARCH=arm` and `GOARCH=riscv64` bare metal.
- A set of packages² to provide board support (e.g. hardware initialization and drivers).

TamaGo currently provides drivers for SoC families NXP i.MX6UL (USB armory Mk II), BCM2835 (Raspberry Pi Zero, Pi 1, Pi 2) and SiFive FU540.

On the i.MX6UL we target development of security applications, TamaGo is fully integrated with our existing open source tooling for i.MX6 Secure Boot (HAB) image signing.

TamaGo also provides full hardware initialization removing the need for intermediate bootloaders.



¹ <https://github.com/usbarmory/tamago-go>

² <https://github.com/usbarmory/tamago>



Similar efforts

Past/existing efforts

(all these projects greatly supported us in proving feasibility and identify TamaGo unique approach, diversity is good)

Biscuit (inactive) - <https://github.com/mit-pdos/biscuit>

Go kernel for non-Go software underneath, larger scope, needs two C bootloaders, hijacks `GOOS=linux`, only for `GOARCH=amd64`, redoes memory allocation and threading.

G.E.R.T (inactive) - <https://github.com/ycoroneos/G.E.R.T>

ARM adaptation of Biscuit but without non-Go software support, needs two C bootloaders, hijacks `GOOS=linux` for `GOARCH=arm`, redoes memory allocation and threading.

AtmanOS (inactive) - <https://github.com/atmanos>

Similar to TamaGo but targets the Xen hypervisor, adds `GOOS=atman` but with limited runtime support.

TinyGo (active) - <https://github.com/tinygo-org>

LLVM based compiler (not original one) aimed at MCUs and minimal footprint, does not support the entire runtime and Go language support differs from standard Go.

Embedded Go (inactive) - <https://github.com/embeddedgo>

Similar to TamaGo but targets ARMv7-M/ARMv8-M (w/ Thumb2) adding new support for it, as not native to Go. Adds `GOOS=noos` `GOARCH=thumb`.

Egg OS (inactive) - <https://github.com/icexin/eggos>

Targets x86, uses native compiler and wraps `GOOS=linux` syscalls back to Go.



Enabling trust

TamaGo not only wants to prove that it is possible to have a bare metal Go runtime, but wants to prove that it can be achieved with **clean and minimal modifications against the original Go distribution²**.

Much of the effort has been placed to understand whether Go bare metal support can be achieved without complex re-implementation of memory allocation, threading, ASM/C OS primitives that would “pollute” the Go runtime to unacceptable levels.

Less is more. Complexity is the enemy of verifiability.

The acceptance of this (and similar) efforts hinges on maintainability, ease of review, clarity, simplicity and **trust**.

- ★ Designed to achieve upstream inclusion and with commitment to always sync to latest Go release.
- ★ ~4500 LOC of changes against Go distribution with clean separation from other GOOS support.
- ★ Strong emphasis on code reuse from existing architectures of standard Go runtime, see [Internals](https://github.com/usbarmory/tamago/wiki/Internals)¹.
- ★ Requires only one import (“library OS”) on the target Go application.
- ★ Supports unencumbered Go applications with nearly full runtime availability.
- ★ In addition to the compiler, aims to provide a complete set of peripheral drivers for SoCs.

¹ <https://github.com/usbarmory/tamago/wiki/Internals>

² Which by the way is self-hosted and has reproducible builds.



Go distribution modifications¹

Glue code (~350 LOCs): patches to adds GOOS=tamago to the list of supported architectures and required stubs for unsupported operations. All changes are benign (no logic/function):

```
// +build aix darwin dragonfly freebsd js,wasm linux nacl netbsd openbsd solaris tamago
```

Re-used code (~4000 LOCs) - patches that clone original Go runtime functionality from an existing architecture to GOOS=tamago, either unmodified or with minimal changes:

- plan9 memory allocation is re-used with 2 LOC changed (brk vs simple pointer)
- js,wasm locking is re-used identically (with JS VM hooks removed)
- nacl in-memory filesystem is re-used (raw SD/MMC access implemented in imx6)

New code (~600 LOCs) - basic syscall and memory layout support:

rt0_tamago_arm.s	(LOC: ~35)	rt0_tamago_riscv64.s	(LOC: ~40)
sys_tamago_arm.go	(LOC: ~170)	sys_tamago_riscv64.go	(LOC: ~80)
os_tamago_arm.go	(LOC: ~240)	os_tamago_riscv64.go	(LOC: ~190)

<https://github.com/golang/go/compare/go1.21.0...usbarmory:tamago1.21.0>

¹ As of tamago1.21.0 against go1.21.0



TamaGo memory layout

	0000 0000
	runtime.ramStart (runtime.vecTableStart)
EXCEPTION VECTOR TABLE (16 kB)	
	runtime.ramStart + 0x4000 (16 kB)
L1 PAGE TABLE (16 kB)	
	runtime.ramStart + 0x8000 (32 kB)
EXCEPTION STACK (16 kB)	
	runtime.ramStart + 0xC000 (48 kB)
L2 PAGE TABLE (16 kB)	
	runtime.ramStart + 0x10000 (64 kB)
.text	
.noptrdata	
.data	Go application
.bss	
.noptrbss	
HEAP	
	runtime.g0.stack.lo (runtime.go.stack.hi - 0x10000)
STACK (64 kB)	
	runtime.go.stack.hi (runtime.ramStart + runtime.ramSize - runtime.ramStackOffset)
UNUSED	
	runtime.ramStart + runtime.ramSize
	FFFF FFFF

Board packages and applications are free, if required, to override:

```
vecTableStart,  
ramStart, ramSize,  
dmaStart, dmaSize
```



Go runtime support

```
// the following variables must be provided externally
var ramStart uint32
var ramStackOffset uint32
var ramSize uint32

// the following functions must be provided externally
func hwinit()
func printk(byte)
func getRandomData([]byte)
func initRNG()
func nanotime1() int64
```

MMU initialization and exception handling are all performed outside the Go runtime in tamago architecture (e.g. arm) package.

This means low-level APIs (e.g. TrustZone) can all be implemented as a regular package.

The Go runtime modification is architecture independent for the most part.

Example of separation between Go runtime, SoC and board packages with pre-defined hooks using `go:linkname`.

```
package imx6ul

//go:linkname ramStart runtime.ramStart
var ramStart uint32 = 0x80000000

// ramSize defined in board package
//go:linkname ramStackOffset runtime.ramStackOffset
var ramStackOffset uint32 = 0x100
```

```
package usbarmory

//go:linkname ramSize runtime.ramSize
var ramSize uint32 = 0x20000000 // 512 MB

//go:linkname printk runtime.printk
func printk(c byte) {
    imx6ul.UART2.Write(c)
}
```



Go runtime support

```
os_tamago_arm.go (Go runtime)
//go:linkname syscall_now syscall.now
func syscall_now() (sec int64, nsec int32) {
    sec, nsec, _ = time_now()
    return
}

imx6.go (imx6 package)
//go:linkname nanotime1 runtime.nanotime1
func nanotime1() int64 {
    return int64(ARM.TimerFn() * ARM.TimerMultiplier)
}

timer.s (arm package)
// func read_gtc() int64
TEXT ·read_gtc(SB),$0-8
    // Cortex™-A9 MPCore® Technical Reference Manual
    // 4.4.1 Global Timer Counter Registers, 0x00 and 0x04
    // p214, Table 2-1, ARM MP Global timer, IMX6DQRM
    MOVW $0x00a00204, R1
    MOVW $0x00a00200, R2

read:
    MOVW    (R1), R3
    MOVW    (R2), R4
    MOVW    (R1), R5
    CMP     R5, R3
    BNE     read

    MOVW    R3, ret_hi+4(FP)
    MOVW    R4, ret_lo+0(FP)

    RET
```

A small set of low-level functions are integrated directly with Go Assembly.

This follows existing patterns in the Go runtime.

In the example ARM Generic Timers (ARM Cortex-A7) are used to support ticks and time related functions.

Overall initialization code accounts for less than 500 lines of code.



Go low level access

```
import "github.com/usbarmory/tamago/internal/reg"

func setARMFreqIMX6ULL(hz uint32) (err error) {
    var div_select uint32
    var arm_podf uint32
    var uV uint32

    curHz := ARMFreq()

    ...

    // set bypass source to main oscillator
    reg.SetN(pll, CCM_ANALOG_PLL_ARM_BYPASS_CLK_SRC, 0b11, 0)

    // bypass
    reg.Set(pll, CCM_ANALOG_PLL_ARM_BYPASS)

    // set PLL divisor
    reg.SetN(pll, CCM_ANALOG_PLL_ARM_DIV_SELECT, 0b1111111, div_select)

    // wait for lock
    log.Printf("imx6_clk: waiting for PLL lock\n")
    reg.Wait(pll, CCM_ANALOG_PLL_ARM_LOCK, 0b1, 1)

    // remove bypass
    reg.Clear(pll, CCM_ANALOG_PLL_ARM_BYPASS)

    // set core divisor
    reg.SetN(cacrr, CCM_CACRR_ARM_PODF, 0b111, arm_podf)

    setOperatingPointIMX6ULL(uV)

    ...
}
```

Example: changing the i.MX6UL SoC ARM core clock frequency.

Go's `unsafe` can be easily identified to spot areas that require care (e.g. pointer arithmetic), it is currently used only in register and DMA memory manipulation primitives.

There are overall only 3 occurrences of `unsafe` used in `dma` and `reg` packages.

Applications are never required to use any `unsafe` function.



Go runtime support

```
//go:linkname syscall
func syscall(number, a1, a2, a3 uintptr) (r1, r2, err uintptr) {
    switch number {
    case 1: // SYS_WRITE
        r1 := write(a1, unsafe.Pointer(a2), int32(a3))
        return uintptr(r1), 0, 0
    default:
        throw("unexpected syscall")
    }

    return
}

//go:nosplit
func write1(fd uintptr, buf unsafe.Pointer, count int32) int32 {
    if fd != 1 && fd != 2 {
        throw("unexpected fd, only stdout/stderr are supported")
    }

    c := uintptr(count)

    for i := uintptr(0); i < c; i++ {
        p := (*byte)(unsafe.Pointer(uintptr(buf) + i))
        printk(*p)
    }

    return int32(c)
}
```

Only the `write` syscall is required for the overwhelming majority of basic runtime support.

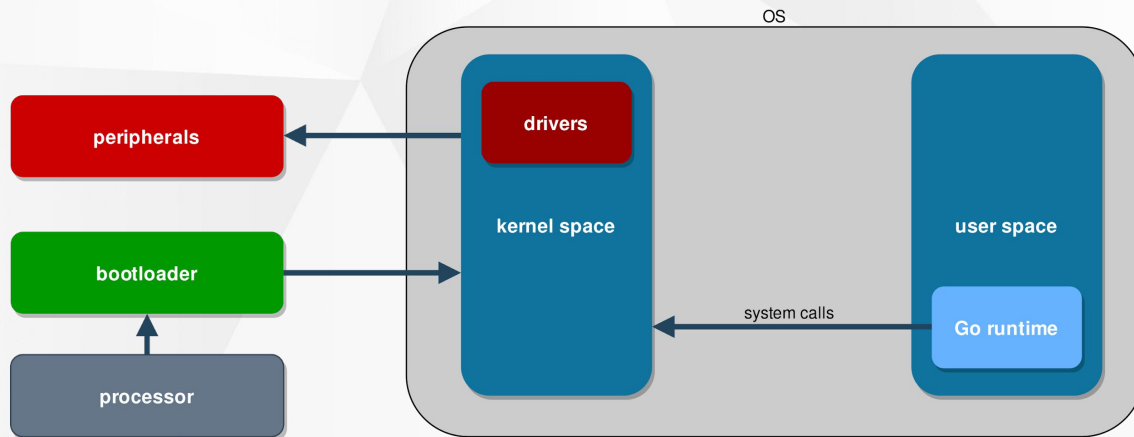
As shown before, `printk` is provided by the application to define the standard output writing function (e.g. UART).

```
imx6_clk: changing ARM core frequency to 900 MHz
imx6_clk: changing ARM core operating point to 575000 uV
imx6_clk: 450000 uV -> 575000 uV
imx6_clk: waiting for PLL lock
imx6_clk: 396 MHz -> 900 MHz
imx6_soc: i.MX6ULL (0x65, 0.1) @ freq:900 MHz - native:true
```

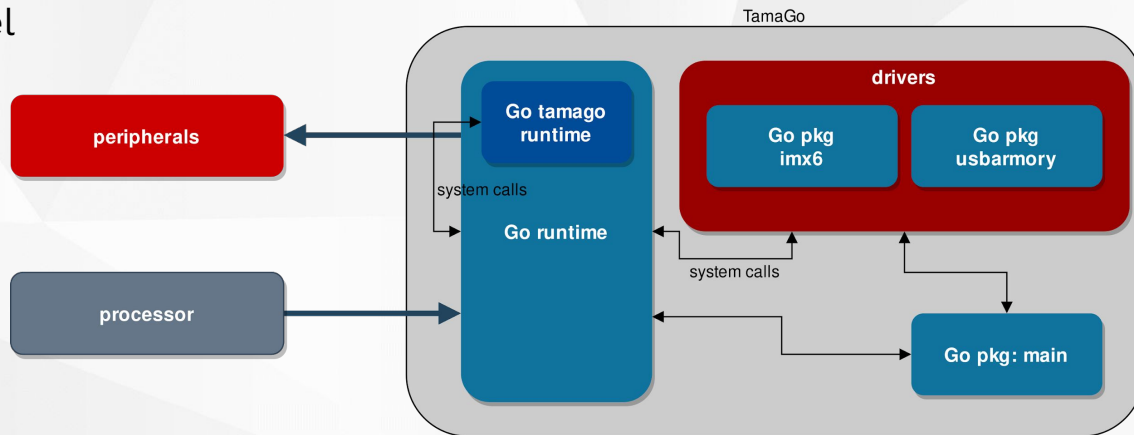


TamaGo

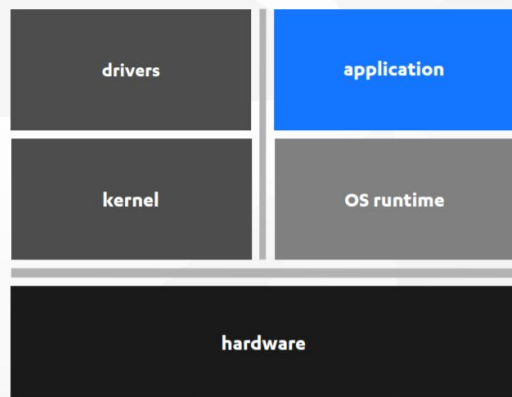
Traditional OS



TamaGo unikernel



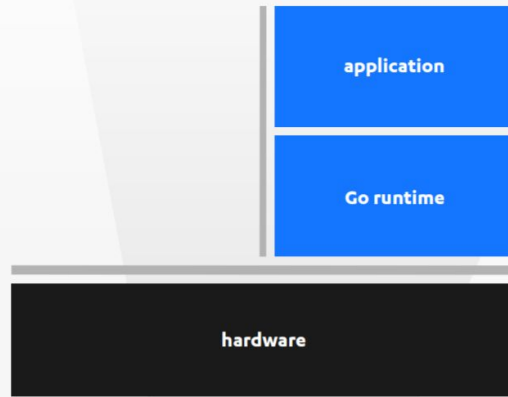
Enabling trust



Traditional OS

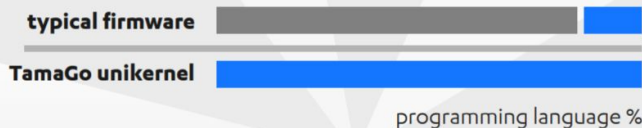


■ memory-unsafe programming language
■ memory-safe programming language



TamaGo unikernel

TamaGo allows a dramatic reduction of the attack surface by **removing any dependency on memory-unsafe languages** (e.g. C), Operating Systems and third party libraries.



Developing, building and running

The full Go runtime is supported¹ without any specific changes required on the application side (Rust on bare metal², for comparison, requires `#![no_std] pragma`).

```
package main

import (
    _ "github.com/usbarmory/tamago/board/usbarmory/mk2"
)

func main() {
    // your code
}
```

```
GO_EXTLINK_ENABLED=0 CGO_ENABLED=0 GOOS=tamago GOARM=7 GOARCH=arm \
    ${TAMAGO} build -ldflags "-T 0x80010000 -E _rt0_arm_tamago -R 0x1000"
```

All Go ecosystem features in terms of build reproducibility, dependency management, profiling, debugging, remain intact.

Firmware can be compiled just as easily on Linux, Windows, macOS.

1. The application requires a single import for the board package to enable necessary initializations.
2. Go code can be written with very few limitations and the SoC package exposes driver APIs.
3. `go build` can be used as usual (reproducible builds!) with few linker flags to define entry point.
4. The SoC package supports native loading (no bootloader required!).

¹ <https://github.com/usbarmory/tamago/wiki/Import-report>

² <https://rust-embedded.github.io/book/intro/no-std.html>



i.MX6ULZ driver: Data Co-Processor (DCP)

The DCP provides hardware accelerated crypto functions and use of the SoC unique OTPMK key for device unique encryption/decryption operations. The driver takes ~750 LOC.

```
workPacket := WorkPacket{}
workPacket.Control0 |= (1 << DCP_CTRL0_OTP_KEY)
...
workPacket.Control1 |= (AES128 << DCP_CTRL1_CIPHER_SELECT)
workPacket.Control1 |= (CBC << DCP_CTRL1_CIPHER_MODE)
workPacket.Control1 |= (UNIQUE_KEY << DCP_CTRL1_KEY_SELECT)
```

```
workPacket.BufferSize = uint32(len(diversifier))
workPacket.SourceBufferAddress = dma.Alloc(diversifier, 0)
defer dma.Free(workPacket.SourceBufferAddress)
```

```
workPacket.DestinationBufferAddress = dma.Alloc(key, 0)
defer dma.Free(workPacket.DestinationBufferAddress)
```

```
workPacket.PayloadPointer = dma.Alloc(iv, 0)
defer dma.Free(workPacket.PayloadPointer)
```

```
buf := new(bytes.Buffer)
binary.Write(buf, binary.LittleEndian, &workPacket)
```

```
pkt := dma.Alloc(buf.Bytes(), 0)
defer dma.Free(pkt)
```

```
reg.Write(HW_DCP_CH0CMDPTR, pkt)
reg.Set(HW_DCP_CH0SEMA, 0)
```

```
diversifier := []byte{0xde, 0xad, 0xbe, 0xef}
iv := make([]byte, aes.BlockSize)

key, err := imx6.DCP.DeriveKey(diversifier, iv)
```

```
-- i.mx6 dcp -----
imx6_dcp: derived test key 75f9022d5a867ad430440feec6611f0a
```

USB armory Mk II example DCP + SNVS run (w/ Secure Boot)

```
-- i.mx6 dcp -----
imx6_dcp: error, SNVS unavailable, not in trusted or secure state
```

USB armory Mk II example DCP + SNVS run (w/o Secure Boot)

Note that Go defined structs (such as `WorkPacket`) can be easily made C-compatible¹ if required.

On i.MX6UL a full CAAM driver is available.



i.MX6ULZ driver: Random Number Generator

The RNG provides a hardware True Random Number Generator, useful to gather the initial seed on embedded systems without a battery backed RTC (and not much else¹). The driver takes ~140 LOC and is hooked as provider for crypto/rand.

```
var getRandomDataFn func([]byte)

//go:linkname getRandomData runtime.getRandomData
func getRandomData(b []byte) {
    getRandomDataFn(b)
}

func (hw *rngb) getRandomData(b []byte) {
    read := 0
    need := len(b)

    for read < need {
        if reg.Get(hw.status, HW_RNG_SR_ERR, 0x1) != 0 {
            panic("imx6_rng: panic\n")
        }

        if reg.Get(hw.status, HW_RNG_SR_FIFO_LVL, 0xf) > 0 {
            val := *hw.fifo
            read = fill(b, read, val)
        }
    }
}
```

```
for i := 0; i < 10; i++ {
    rng := make([]byte, size)
    rand.Read(rng)
    fmt.Printf("%x\n", rng)
}
```

```
-- rng -----
imx6_rng: self-test
imx6_rng: seeding
f90b00053a50b9edd42df027c982769d1a7d25445e31ce98486bd4a9676bef42
56baf6ecc32bf02fb9d09c2d8c607baa487e2283b6856486b42cdf954277d4d5
49fc0c03f8cbc45f7aeb58ba71c0d561a91dbeae697d7bc511482697bf96b2f8
345db47ab3395272a9db9531f03160b3e1654b7e8b7267c1a3bc97206f3cb8c7
cb54154b105a2bd3938fbd99f1f2f5409c0be09dc5f64189f473ae905d264b25
275994ee93e0c779f3eb30d770eeabfcb5ab0b8a5da68cc28a07dfbdb46a1e08
6215cc716b9ed577d3c6cd34d57f2dc3ed93c9b6aaedf120d68a4532393e1056
d691d7f93c57a54462f90ca76528beec4bda1a0220e5d5fbf43986308f9013b
6ea213b27eb3e0e4243b3c872e7a07b7898d9f07ea205b8a50c30e62c7204602
4544d5dff957471972331532aaf34eb5644bc430f854dd6593177640e07e4f00
```

USB armory Mk II example TRNG run

¹ https://media.ccc.de/v/32c3-7441-the_plain_simple_reality_of_entropy



i.MX6ULZ driver: USB

```
func buildDTD(n int, dir int, ioc bool, addr uint32, size int) (dtd *dTD) {
    dtd = &dTD{}

    // interrupt on completion (ioc)
    if ioc {
        bits.Set(&dtd.Token, 15)
    } else {
        bits.Clear(&dtd.Token, 15)
    }

    // invalidate next pointer
    dtd.Next = 0b1
    // multiplier override (Mult0)
    bits.SetN(&dtd.Token, 10, 0b11, 0)
    // active status
    bits.Set(&dtd.Token, 7)
    // total bytes
    bits.SetN(&dtd.Token, 16, 0xffff, uint32(size))

    dtd._buf = addr
    dtd._size = uint32(size)

    for n := 0; n < DTD_PAGES; n++ {
        dtd.Buffer[n] = dtd._buf + uint32(DTD_PAGE_SIZE*n)
    }

    buf := new(bytes.Buffer)
    binary.Write(buf, binary.LittleEndian, dtd)
    dtd._dtd = dma.Alloc(buf.Bytes())[0:DTD_SIZE], DTD_ALIGN)

    return
}
```

Example of Endpoint Transfer Descriptor (dTD) configuration.

A custom DMA allocator is used to copy structures on memory reserved for DMA operation, with required alignments.

```
addr = dma.Alloc(buf, align)
defer dma.Free(addr)
```

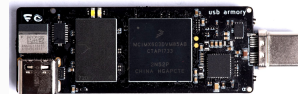
Buffers can be also reserved by the application to spare re-allocation (automatic detection of slices already in DMA memory).

Using Go goroutines, channels, mutexes, interfaces freely in low level drivers is a delight!

All in ~1000 LOC !



i.MX6ULZ driver: USB networking



```
func configureEthernetDevice(device *usb.Device) {
    // Supported Language Code Zero: English
    device.SetLanguageCodes([]uint16{0x0409})

    // device descriptor
    device.Descriptor = &usb.DeviceDescriptor{
        device.Descriptor.SetDefaults()
        device.Descriptor.DeviceClass = 0x2
        device.Descriptor.VendorId = 0x0525
        device.Descriptor.ProductId = 0xa4a2
        device.Descriptor.Device = 0x0001
        device.Descriptor.NumConfigurations = 1

        iManufacturer, _ := device.AddString(`TamaGo`)
        device.Descriptor.Manufacturer = iManufacturer

        iProduct, _ := device.AddString(`RNDIS/Ethernet Gadget`)
        device.Descriptor.Product = iProduct

        iSerial, _ := device.AddString(`0.1`)
        device.Descriptor.SerialNumber = iSerial

    // device qualifier
    device.Qualifier = &usb.DeviceQualifierDescriptor{
        device.Qualifier.SetDefaults()
        device.Qualifier.DeviceClass = 2
        device.Qualifier.NumConfigurations = 2
    }
}
```

```
func configureECM(device *usb.Device) {
    ...

    conf.Interfaces = append(conf.Interfaces, iface)

    ep1IN := &usb.EndpointDescriptor{
        ep1IN.SetDefaults()
        ep1IN.EndpointAddress = 0x81
        ep1IN.Attributes = 2
        ep1IN.MaxPacketSize = 512
        ep1IN.Function = ECMTx

        iface.Endpoints = append(iface.Endpoints, ep1IN)

    ep1OUT := &usb.EndpointDescriptor{
        ep1OUT.SetDefaults()
        ep1OUT.EndpointAddress = 0x01
        ep1OUT.Attributes = 2
        ep1OUT.MaxPacketSize = 512
        ep1OUT.Function = ECMRx

        iface.Endpoints = append(iface.Endpoints, ep1OUT)
    }
}
```

```
func ECMTx(_ []byte, lastErr error) (in []byte) {
    // gvisor tcpip channel link
    pkt := <-link.C:

    ...

    // Ethernet frame header
    in = append(in, hostMAC...)
    in = append(in, deviceMAC...)
    in = append(in, proto...)
    // packet header
    in = append(in, hdr...)
    // payload
    in = append(in, payload...)

    return
}

func ECMRx(out []byte, lastErr error) ([]byte) {
    ...

    pkt := tcpip.PacketBuffer{
        LinkHeader: hdr,
        Data:       payload,
    }

    // gvisor tcpip channel link
    link.InjectInbound(proto, pkt)

    return
}
```

Example USB Ethernet (CDC ECM) driver integrated with Google netstack (gvisor.dev/gvisor/pkg/tcpip) for pure Go networking.

Developed in less than 2 hours and ~300 LOC.



i.MX6ULL driver: Ethernet networking



```
func (ring *bufferDescriptorRing) init(rx bool, n int) uint32 {
    ring.size = n
    ring.bds = make([]*bufferDescriptor, n)

    descSize := len(&bufferDescriptor{}.Bytes())
    ptr, desc := dma.Reserve(n*descSize, bufferAlign)

    dataSize := MTU + (bufferAlign - (MTU % bufferAlign))
    addr, data := dma.Reserve(n*dataSize, bufferAlign)

    for i := 0; i < n; i++ {
        off := dataSize * i

        bd := &bufferDescriptor{
            Addr: uint32(addr) + uint32(off),
            data: data[off : off+dataSize],
        }

        ...

        off = descSize * i
        bd.desc = desc[off : off+descSize]
        copy(bd.desc, bd.Bytes())

        ring.bds[i] = bd
    }

    return uint32(ptr)
}
```

```
func (hw *ENET) Rx() (buf []byte) {
    hw.Lock()
    defer hw.Unlock()

    buf = hw.rx.pop()
    reg.Set(hw.rdar, RDAR_ACTIVE)

    return
}

func (hw *ENET) Tx(buf []byte) {
    hw.Lock()
    defer hw.Unlock()

    if len(buf) > MTU {
        return
    }

    hw.tx.push(buf)
    reg.Set(hw.tdar, TDAR_ACTIVE)
}
```

```
// enable IRQs, start interface
eth.EnableInterrupt(enet.IRQ_RXF)
eth.Start(false)

// hook interface into Go runtime
net.SocketFunc = iface.Socket
```

```
// interrupt handling
func handleEthernetInterrupt(eth *enet.ENET) {
    for buf := eth.Rx(); buf != nil; buf = eth.Rx() {
        eth.RxHandler(buf)
        eth.ClearInterrupt(enet.IRQ_RXF)
    }
}

func StartInterruptHandler(eth *enet.ENET) {
    imx6ul.GIC.Init(true, false)
    imx6ul.GIC.EnableInterrupt(eth.IRQ, true)

    arm.RegisterInterruptHandler()

    for {
        arm.WaitInterrupt()

        irq, end := imx6ul.GIC.GetInterrupt(true)

        if end != nil {
            end <- true
        }

        if irq == eth.IRQ {
            handleEthernetInterrupt(eth)
        }
    }
}
```



i.MX6ULZ driver: uSDHC (MMC/SD)

```
// p351, 35.4.5 SD card initialization flow chart, IMX6FG
// p57, 4.2.3 Card Initialization and Identification Process, SD-PL-7.10
func (hw *USDHC) initSD() (err error) {
    var arg uint32
    var bus_width uint32
    var mode uint32
    var root_clk uint32
    var clk int
    var tune bool

    if hw.LowVoltage == nil {
        hw.card.Rate = HS_Mbps
    } else if hw.card.Rate >= SDR50_Mbps {
        if err = hw.voltageSwitchSD(); err != nil {
            hw.card.Rate = HS_Mbps
        }
    }

    // CMD2 - ALL_SEND_CID - get unique card identification
    if err = hw.cmd(2, READ, arg, RSP_136, false, true, false, 0); err != nil {
        return
    }

    // CMD3 - SEND_RELATIVE_ADDR - get relative card address (RCA)
    if err = hw.cmd(3, READ, arg, RSP_48, true, true, false, 0); err != nil {
        return
    }

    ...
}
```

The uSDHC driver supports read/write operation on MMC/SD with speeds up to HS200 and SDR104 respectively.

All in ~1200 LOC !

It is used by armory-ums to allow export of the USB armory Mk II internal eMMC card as USB mass storage devices to ease firmware flashing.

In combination with packages such as go-ext4 it allows filesystem access (see armory-boot).



Demo: tamago-example

```
tamago-example $ make qemu
```

```
tamago/arm (go1.18.3) • Self6aa lcars@lambda on 2022-07-14 08:02:11 • i.MX6UL 1188 MHz (emulated)
```

```
ble          # BLE serial console
date          (time in RFC339 format)?
dcp           # show/change runtime date and time
dns           # benchmark hardware encryption
exit, quit   # resolve domain (requires routing)
help         # close session
i2c          # this help
info         # I²C bus read
kem          # device information
led          # benchmark post-quantum KEM
md           # LED control
mmc          # memory display (use with caution)
mw           # MMC/SD card read
otp          # memory write (use with caution)
rand         # OTP fuses display
reboot       # gather 32 random bytes
stack        # reset device
stackall     # stack trace of current goroutine
test         # stack trace of all goroutines
            # launch tests
```

```
> kem
```

```
Kyber1024 89248f2f33f7f4f705172911f3049c409a933ec904aedadf035f30fa5646cd5 (287.799024ms)
Kyber768  a1e122cad3c24bc51622e4c242d8b8acbcd3f618fee4220400605ca8f9ea02c2 (209.114896ms)
Kyber512  e9c2bd37133fcb40772f81559f14b1f58dcd1c816701be9ba6214d43baf4547 (149.049056ms)
```

```
> rand
```

```
db7d46647880be1e51731177b6f73645b71ca504242c97758df3a86842d93236
```

```
> md 80000000 96
```

```
00000000 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 |.....|
00000010 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 18 f0 9f e5 |.....|
00000020 04 d4 0f 80 38 d4 0f 80 6c d4 0f 80 a0 d4 0f 80 |.....|
00000030 d4 d4 0f 80 00 00 00 00 08 d5 0f 80 3c d5 0f 80 |.....|
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
> date
```

```
1970-01-01T00:03:31Z
```

```
> date 2022-07-14T08:02:11Z
```

```
2022-07-14T08:02:11Z
```

```
> info
```

```
Runtime .....: go1.18.3 tamago/arm
Board .....: UA-MKII-β
SoC .....: i.MX6UL 1188 MHz (emulated)
SDP .....: false
Secure boot ...: false
Boot ROM hash : c6aae82d3a49e6ce016ef02fa93c918d50934f93847ae371816e5fdeb79dd5
```

```
$ ssh 10.0.0.1
```

```
tamago/arm (go1.18.3) • 5e1f6aa lcars@lambda on 2022-07-14 10:00:48 • i.MX6ULL 900 MHz
```

```
> otp 0 0
```

```
OTP bank:0 word:0 val:0x00324003
```

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1
```

```
31 _____
30 _____
25 _____
23 _____
22 _____
21 _____
20 _____
19 18 _____
17 _____
16 _____
15 _____
14 _____
13 12 _____
11 10 _____
09 08 _____
06 _____
05 04 _____
03 02 _____
01 00 _____
OCOTP_LOCK
Bank:0 Word:0
R: 0x00000000
W: 0x00000000
GP3_LOCK
GP4_LOCK
PIN_LOCK
MISC_CONF_LOCK
ROM_PATCH_LOCK
OTPMK_CRC_LOCK
ANALOG_LOCK
OTPMK_LOCK
SW_GP_LOCK
GP3_LOCK
SRK_LOCK
GP2_LOCK
GP1_LOCK
MAC_ADDR_LOCK
S3C_RESP_LOCK
MEM_TRIM_LOCK
BOOT_CFG_LOCK
TESTER_LOCK
```

```
> dns www.golang.org
```

```
;; opcode: QUERY, status: NOERROR, id: 28248
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
```

```
;; QUESTION SECTION:
```

```
;www.golang.org. IN ANY
```

```
;; ANSWER SECTION:
```

```
www.golang.org. 3600 IN CNAME golang.org.
```

```
> dcp 65536 10
```

```
Doing aes-128 cbc for 10s on 65536 blocks
6201 aes-128 cbc's in 10.00086575s
```

```
> info
```

```
Runtime .....: go1.18.4 tamago/arm
Board .....: UA-MKII-β
SoC .....: i.MX6ULL 900 MHz
SDP .....: true
Secure boot ...: true
Boot ROM hash : 1727a0f46dbde555b583e9a138ae359389974b7be4369ffd4a252a8730f7e59b
```



Demo: Tailscale

```
ssh armory@10.0.0.1
```

```

tailscale $YOUKEY
tsnet --- [v1] using fake (no-op) tun device
tsnet --- [v1] using fake (no-op) OS network configurator
tsnet --- [v1] using fake (no-op) DNS configurator
tsnet --- dns: using dns.noopManager
tsnet --- link state: interfaces.State{defaultRoute= ifs={} v4=false v6=false}
tsnet --- magicsock: disco key = d:xxxxxxxxxxxxxxx
tsnet --- Creating WireGuard device...
tsnet --- Bringing WireGuard device up...
tsnet --- wg: [v2] UDP bind has been updated
tsnet --- wg: [v2] Interface state was Down, requested Up, now Up
tsnet --- Bringing router up...
tsnet --- [v1] warning: fakeRouter.Up: not implemented.
tsnet --- Clearing router settings...
tsnet --- [v1] warning: fakeRouter.Set: not implemented.
tsnet --- Starting network monitor...
tsnet --- Engine created.
tsnet --- tsnet running state path /tsnet-tamago/tailscaled.state
tsnet --- pm: migrating "daemon" profile to new format
tsnet ---
[{"HostInfo":{"IPNVersion":"1.49.0-dev20230906-t7a0be7f2c-dirty","OS":"tamago","Package":"tsnet","Hostname":"tamago","GoArch":"arm"},"GoArchVar":"7m","GoVersion":"go1.21.0"}]
tsnet --- logpolicy: using UserCacheDir, "/Tailscale"
tsnet --- [v1] netmap packet filter: (not ready yet)
tsnet --- tsnet starting with hostname "tamago", varRoot "/tsnet-tamago"
tsnet --- Start
tsnet --- generating new machine key
--
netmap self: {0xG6m} auth=machine-authorized u=xxxxxxxx@gmail.com [100.xxx.xx.82/32 fd7a:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/128]
tsnet --- control: [v1] mapRoutine: netmap received: state:synchronized
tsnet --- control: [v1] sendStatus: mapRoutine-got-netmap: state:synchronized
tsnet --- active login: xxxxxxxxxxx@gmail.com
tsnet --- [v1] netmap packet filter: 1 filters
tsnet --- [v1] magicsock: got updated network map; 3 peers
tsnet --- [v2] netstack: registered IP 100.xxx.xx.82/32
tsnet --- [v2] netstack: registered IP fd7a:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx/128

tsnet --- peerapi: serving on http://100.xxx.xx.82:63151
tsnet --- peerapi: serving on http://[[fd7a:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx]]:63151
tsnet --- netcheck: UDP is blocked, trying ICMP
tsnet --- control: [v1] HostInfo:
{"IPNVersion":"1.49.0-dev20230906-t7a0be7f2c-dirty","BackendLogID":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX9b2efd","OS":"tamago","Package":"tsnet","Hostname":"tamago","GoArch":"arm","GoArchVar":"7m","GoVersion":"go1.21.0","Services":[{"Proto":"peerapi4","Port":63151},{"Proto":"peerapi6","Port":63151}],{"Userspace":true,"UserspaceRouter":true}}
tsnet --- control: [v1] PollNetMap: stream=false ep=[]

starting web server at 100.117.90.82:80
tsnet --- control: [v1] successful lite map update in 316ms
starting ssh server (SHA256:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx) at :22
```

Machine Details

Information about this machine's network. Used to debug connection issues.

Creator	[REDACTED]	Created	Jun 15, 2023 at 1:06 PM GMT+2
Machine name	tamago Copy	Last seen	1:06 PM GMT+2
OS hostname	tamago	Key expiry	6 months from now
OS	Tamago		
Tailscale version	1.38.4-dev20230612	CLIENT CONNECTIVITY	
Tailscale IPv4	100 [REDACTED] Copy	Varies G	—
Tailscale IPv6	f67a [REDACTED] Copy	Hairpinning	—
ID	nt4 [REDACTED]	IPv6	—
Endpoints	nt4	UDP	—
Relays	—	UPnP	—
		PCP	—
		NAT-PMP	—

```
$ ping 100.xxx.xx.82
```

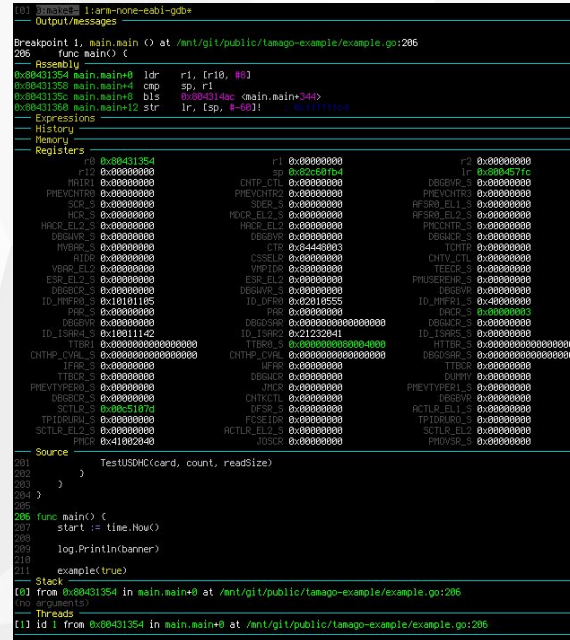
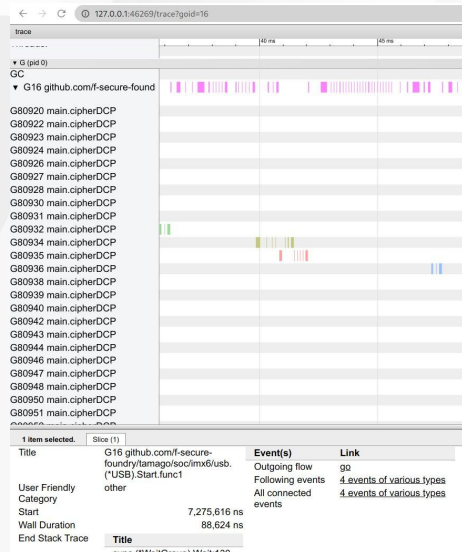
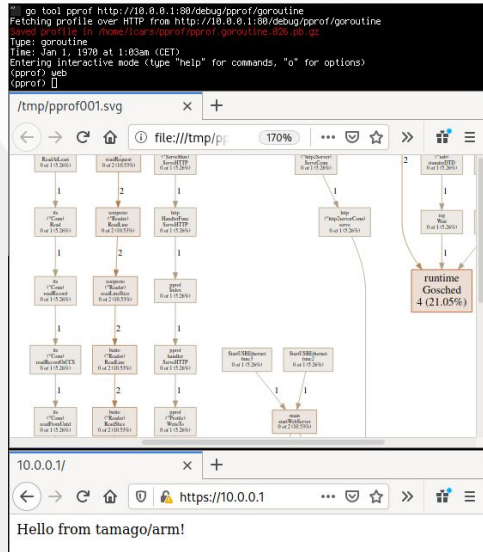
```

PING 100.xxx.xx.82 (100.xxx.xx.82) 56(84) bytes of data.
64 bytes from 100.xxx.xx.82: icmp_seq=1 ttl=64 time=1170 ms
64 bytes from 100.xxx.xx.82: icmp_seq=2 ttl=64 time=161 ms
64 bytes from 100.xxx.xx.82: icmp_seq=3 ttl=64 time=126 ms

```



Debugging



GoKey - The bare metal Go smart card

The GoKey application implements a composite USB **OpenPGP 3.4** smartcard and **FIDO U2F** token, written in pure Go (~2500¹ LOC).

It allows to implement a radically different security model for smartcards, taking advantage of TamaGo to safely mix layers and protocols not easy to combine.

For instance authentication can happen over SSH instead of plaintext PIN transmission over USB.

	Trust anchor	Data protection	Runtime	Application	Requires tamper proofing	Encryption at rest
traditional smartcard	flash protection	flash protection	JCOP	JCOP applets	Yes	No
USB armory with GoKey	secure boot	SoC security element	TamaGo	Go application	No	Yes

```
GoKey
host > gpg --card-status
Reader .....: USB armory Mk II [Smart Card Control] (0.1) 00 00
Application ID ....: D27600124010304F5ECD209320C0000
Application type ...: OpenPGP
Version .....: 3.4
Manufacturer .....: F-Secure
Serial number .....: D209320C
Name of cardholder: Alice
Language prefs ....: [not set]
Salutation .....:
URL of public key : [not set]
Login data .....: [not set]
Signature PIN .....: forced
Key attributes ....: rsa4096 rsa4096 rsa4096
Max. PIN lengths ..: 254 127 127
PIN retry counter ..: 1 0 0
Signature counter ..: 0
Signature key .....: 05EC DEB4 43FA 5C01 9C7A 51A2 E9C8 5194 3E46 C2B5
created .....: 2020-04-03 15:10:30
Encryption key .....: 656B E354 EE12 BF FB 98B8 1607 556B 9659 5A2C D776
created .....: 2020-04-03 15:01:49
Authentication key: [none]
General key info...: sub rsa4096/E9C851943E46C2B5 2020-04-03 Alice <alice@wonderl
and>
sec# rsa4096/CBBE74C25E15EA0B created: 2020-04-03 expires: 2022-04-03
ssb> rsa4096/556B96595A2CD776 created: 2020-04-03 expires: 2022-04-03
card-no: F5EC D209320C
ssb> rsa4096/E9C851943E46C2B5 created: 2020-04-03 expires: 2022-04-03
card-no: F5EC D209320C
ssb# rsa4096/2EB31B5E996EE83D created: 2020-04-03 expires: 2022-04-03
host > ssh alice@10.0.10
GoKey > tamago/arm (gol.14) > 0330e82 user@host on 2020-04-09 07:42:11 > 1.MX6ULL

exit, quit # close session
help # this help
init # initialize card
rand # gather 32 bytes from TRNG via crypto/rand
reboot # restart
status # display card status
lock (all|sig|dec) # key lock
unlock (all|sig|dec) # key unlock, prompts decryption passphrase

resizing terminal (pty-req:80x66)
> unlock all
Passphrase:
VERIFY: 05 EC DE B4 43 FA 5C 01 9C 7A 51 A2 E9 C8 51 94 3E 46 C2 B5 unlocked
VERIFY: 65 6B E3 54 EE 12 BF FB 98 B8 16 07 55 6B 96 59 5A 2C D7 76 unlocked

> exit
logout
closing ssh connection
Connection to 10.0.10 closed.
host > gpg --decrypt secret.asc
gpg: encrypted with 4096-bit RSA key, ID 556B96595A2CD776, created 2020-04-03
cheshire wrote:
"Alice <alice@wonderland>
"Where do you want to go?"

alice wrote:
"I don't know"


cheshire wrote:
"Then, it really doesn't matter, does it?"
host >
```



Demo: GoKey



GoKey - the bare metal Go smart card



It wasn't until Alice...

"You won't get caught!" the Queen declared at the top of her voice, beyond mind.

"What makes you say that?" said Alice, who had grown so her little sister by (this time). "You're putting out a piece of candy!"

At first the white paper came up into the air, and then flying round about her like a little screen, half of which was red & half blue to the top of her sister's, who was going dancing away from Alice (seeing that had finished her part of the dance soon as it was over).

"Then we'll dance!" said her sister. "Why, what a long sleep you've had!"

"Yes, I have had such a sore head!" said Alice, and she took her sister, as well as she could remember them, all down through the hall, the kitchen, the dining room, and out, "It was a curious dream, dear, certainly! But now that I'm going (at least) to bed, I'll tell you."

But her sister, who still just as she left her, leaving her head on her hand, looking at the setting sun, and thinking of (1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19) (20) (21) (22) (23) (24) (25) (26) (27) (28) (29) (30) (31) (32) (33) (34) (35) (36) (37) (38) (39) (40) (41) (42) (43) (44) (45) (46) (47) (48) (49) (50) (51) (52) (53) (54) (55) (56) (57) (58) (59) (60) (61) (62) (63) (64) (65) (66) (67) (68) (69) (70) (71) (72) (73) (74) (75) (76) (77) (78) (79) (80) (81) (82) (83) (84) (85) (86) (87) (88) (89) (90) (91) (92) (93) (94) (95) (96) (97) (98) (99) (100) (101) (102) (103) (104) (105) (106) (107) (108) (109) (110) (111) (112) (113) (114) (115) (116) (117) (118) (119) (120) (121) (122) (123) (124) (125) (126) (127) (128) (129) (130) (131) (132) (133) (134) (135) (136) (137) (138) (139) (140) (141) (142) (143) (144) (145) (146) (147) (148) (149) (150) (151) (152) (153) (154) (155) (156) (157) (158) (159) (160) (161) (162) (163) (164) (165) (166) (167) (168) (169) (170) (171) (172) (173) (174) (175) (176) (177) (178) (179) (180) (181) (182) (183) (184) (185) (186) (187) (188) (189) (190) (191) (192) (193) (194) (195) (196) (197) (198) (199) (200) (201) (202) (203) (204) (205) (206) (207) (208) (209) (210) (211) (212) (213) (214) (215) (216) (217) (218) (219) (220) (221) (222) (223) (224) (225) (226) (227) (228) (229) (230) (231) (232) (233) (234) (235) (236) (237) (238) (239) (240) (241) (242) (243) (244) (245) (246) (247) (248) (249) (250) (251) (252) (253) (254) (255) (256) (257) (258) (259) (260) (261) (262) (263) (264) (265) (266) (267) (268) (269) (270) (271) (272) (273) (274) (275) (276) (277) (278) (279) (280) (281) (282) (283) (284) (285) (286) (287) (288) (289) (290) (291) (292) (293) (294) (295) (296) (297) (298) (299) (300) (301) (302) (303) (304) (305) (306) (307) (308) (309) (310) (311) (312) (313) (314) (315) (316) (317) (318) (319) (320) (321) (322) (323) (324) (325) (326) (327) (328) (329) (330) (331) (332) (333) (334) (335) (336) (337) (338) (339) (340) (341) (342) (343) (344) (345) (346) (347) (348) (349) (350) (351) (352) (353) (354) (355) (356) (357) (358) (359) (360) (361) (362) (363) (364) (365) (366) (367) (368) (369) (370) (371) (372) (373) (374) (375) (376) (377) (378) (379) (380) (381) (382) (383) (384) (385) (386) (387) (388) (389) (390) (391) (392) (393) (394) (395) (396) (397) (398) (399) (400) (401) (402) (403) (404) (405) (406) (407) (408) (409) (410) (411) (412) (413) (414) (415) (416) (417) (418) (419) (420) (421) (422) (423) (424) (425) (426) (427) (428) (429) (430) (431) (432) (433) (434) (435) (436) (437) (438) (439) (440) (441) (442) (443) (444) (445) (446) (447) (448) (449) (450) (451) (452) (453) (454) (455) (456) (457) (458) (459) (460) (461) (462) (463) (464) (465) (466) (467) (468) (469) (470) (471) (472) (473) (474) (475) (476) (477) (478) (479) (480) (481) (482) (483) (484) (485) (486) (487) (488) (489) (490) (491) (492) (493) (494) (495) (496) (497) (498) (499) (500) (501) (502) (503) (504) (505) (506) (507) (508) (509) (510) (511) (512) (513) (514) (515) (516) (517) (518) (519) (520) (521) (522) (523) (524) (525) (526) (527) (528) (529) (530) (531) (532) (533) (534) (535) (536) (537) (538) (539) (540) (541) (542) (543) (544) (545) (546) (547) (548) (549) (550) (551) (552) (553) (554) (555) (556) (557) (558) (559) (560) (561) (562) (563) (564) (565) (566) (567) (568) (569) (570) (571) (572) (573) (574) (575) (576) (577) (578) (579) (580) (581) (582) (583) (584) (585) (586) (587) (588) (589) (590) (591) (592) (593) (594) (595) (596) (597) (598) (599) (600) (601) (602) (603) (604) (605) (606) (607) (608) (609) (610) (611) (612) (613) (614) (615) (616) (617) (618) (619) (620) (621) (622) (623) (624) (625) (626) (627) (628) (629) (630) (631) (632) (633) (634) (635) (636) (637) (638) (639) (640) (641) (642) (643) (644) (645) (646) (647) (648) (649) (650) (651) (652) (653) (654) (655) (656) (657) (658) (659) (660) (661) (662) (663) (664) (665) (666) (667) (668) (669) (670) (671) (672) (673) (674) (675) (676) (677) (678) (679) (680) (681) (682) (683) (684) (685) (686) (687) (688) (689) (690) (691) (692) (693) (694) (695) (696) (697) (698) (699) (700) (701) (702) (703) (704) (705) (706) (707) (708) (709) (710) (711) (712) (713) (714) (715) (716) (717) (718) (719) (720) (721) (722) (723) (724) (725) (726) (727) (728) (729) (730) (731) (732) (733) (734) (735) (736) (737) (738) (739) (740) (741) (742) (743) (744) (745) (746) (747) (748) (749) (750) (751) (752) (753) (754) (755) (756) (757) (758) (759) (760) (761) (762) (763) (764) (765) (766) (767) (768) (769) (770) (771) (772) (773) (774) (775) (776) (777) (778) (779) (780) (781) (782) (783) (784) (785) (786) (787) (788) (789) (790) (791) (792) (793) (794) (795) (796) (797) (798) (799) (800) (801) (802) (803) (804) (805) (806) (807) (808) (809) (810) (811) (812) (813) (814) (815) (816) (817) (818) (819) (820) (821) (822) (823) (824) (825) (826) (827) (828) (829) (830) (831) (832) (833) (834) (835) (836) (837) (838) (839) (840) (841) (842) (843) (844) (845) (846) (847) (848) (849) (850) (851) (852) (853) (854) (855) (856) (857) (858) (859) (860) (861) (862) (863) (864) (865) (866) (867) (868) (869) (870) (871) (872) (873) (874) (875) (876) (877) (878) (879) (880) (881) (882) (883) (884) (885) (886) (887) (888) (889) (890) (891) (892) (893) (894) (895) (896) (897) (898) (899) (900) (901) (902) (903) (904) (905) (906) (907) (908) (909) (910) (911) (912) (913) (914) (915) (916) (917) (918) (919) (920) (921) (922) (923) (924) (925) (926) (927) (928) (929) (930) (931) (932) (933) (934) (935) (936) (937) (938) (939) (940) (941) (942) (943) (944) (945) (946) (947) (948) (949) (950) (951) (952) (953) (954) (955) (956) (957) (958) (959) (960) (961) (962) (963) (964) (965) (966) (967) (968) (969) (970) (971) (972) (973) (974) (975) (976) (977) (978) (979) (980) (981) (982) (983) (984) (985) (986) (987) (988) (989) (990) (991) (992) (993) (994) (995) (996) (997) (998) (999) (1000) (1001) (1002) (1003) (1004) (1005) (1006) (1007) (1008) (1009) (1010) (1011) (1012) (1013) (1014) (1015) (1016) (1017) (1018) (1019) (1020) (1021) (1022) (1023) (1024) (1025) (1026) (1027) (1028) (1029) (1030) (1031) (1032) (1033) (1034) (1035) (1036) (1037) (1038) (1039) (1040) (1041) (1042) (1043) (1044) (1045) (1046) (1047) (1048) (1049) (1050) (1051) (1052) (1053) (1054) (1055) (1056) (1057) (1058) (1059) (1060) (1061) (1062) (1063) (1064) (1065) (1066) (1067) (1068) (1069) (1070) (1071) (1072) (1073) (1074) (1075) (1076) (1077) (1078) (1079) (1080) (1081) (1082) (1083) (1084) (1085) (1086) (1087) (1088) (1089) (1090) (1091) (1092) (1093) (1094) (1095) (1096) (1097) (1098) (1099) (1100) (1101) (1102) (1103) (1104) (1105) (1106) (1107) (1108) (1109) (1110) (1111) (1112) (1113) (1114) (1115) (1116) (1117) (1118) (1119) (1120) (1121) (1122) (1123) (1124) (1125) (1126) (1127) (1128) (1129) (1130) (1131) (1132) (1133) (1134) (1135) (1136) (1137) (1138) (1139) (1140) (1141) (1142) (1143) (1144) (1145) (1146) (1147) (1148) (1149) (1150) (1151) (1152) (1153) (1154) (1155) (1156) (1157) (1158) (1159) (1160) (1161) (1162) (1163) (1164) (1165) (1166) (1167) (1168) (1169) (1170) (1171) (1172) (1173) (1174) (1175) (1176) (1177) (1178) (1179) (1180) (1181) (1182) (1183) (1184) (1185) (1186) (1187) (1188) (1189) (1190) (1191) (1192) (1193) (1194) (1195) (1196) (1197) (1198) (1199) (1200) (1201) (1202) (1203) (1204) (1205) (1206) (1207) (1208) (1209) (1210) (1211) (1212) (1213) (1214) (1215) (1216) (1217) (1218) (1219) (1220) (1221) (1222) (1223) (1224) (1225) (1226) (1227) (1228) (1229) (1230) (1231) (1232) (1233) (1234) (1235) (1236) (1237) (1238) (1239) (1240) (1241) (1242) (1243) (1244) (1245) (1246) (1247) (1248) (1249) (1250) (1251) (1252) (1253) (1254) (1255) (1256) (1257) (1258) (1259) (1260) (1261) (1262) (1263) (1264) (1265) (1266) (1267) (1268) (1269) (1270) (1271) (1272) (1273) (1274) (1275) (1276) (1277) (1278) (1279) (1280) (1281) (1282) (1283) (1284) (1285) (1286) (1287) (1288) (1289) (1290) (1291) (1292) (1293) (1294) (1295) (1296) (1297) (1298) (1299) (1300) (1301) (1302) (1303) (1304) (1305) (1306) (1307) (1308) (1309) (1310) (1311) (1312) (1313) (1314) (1315) (1316) (1317) (1318) (1319) (1320) (1321) (1322) (1323) (1324) (1325) (1326) (1327) (1328) (1329) (1330) (1331) (1332) (1333) (1334) (1335) (1336) (1337) (1338) (1339) (1340) (1341) (1342) (1343) (1344) (1345) (1346) (1347) (1348) (1349) (1350) (1351) (1352) (1353) (1354) (1355) (1356) (1357) (1358) (1359) (1360) (1361) (1362) (1363) (1364) (1365) (1366) (1367) (1368) (1369) (1370) (1371) (1372) (1373) (1374) (1375) (1376) (1377) (1378) (1379) (1380) (1381) (1382) (1383) (1384) (1385) (1386) (1387) (1388) (1389) (1390) (1391) (1392) (1393) (1394) (1395) (1396) (1397) (1398) (1399) (1400) (1401) (1402) (1403) (1404) (1405) (1406) (1407) (1408) (1409) (1410) (1411) (1412) (1413) (1414) (1415) (1416) (1417) (1418) (1419) (1420) (1421) (1422) (1423) (1424) (1425) (1426) (1427) (1428) (1429) (1430) (1431) (1432) (1433) (1434) (1435) (1436) (1437) (1438) (1439) (1440) (1441) (1442) (1443) (1444) (1445) (1446) (1447) (1448) (1449) (1450) (1451) (1452) (1453) (1454) (1455) (1456) (1457) (1458) (1459) (1460) (1461) (1462) (1463) (1464) (1465) (1466) (1467) (1468) (1469) (1470) (1471) (1472) (1473) (1474) (1475) (1476) (1477) (1478) (1479) (1480) (1481) (1482) (1483) (1484) (1485) (1486) (1487) (1488) (1489) (1490) (1491) (1492) (1493) (1494) (1495) (1496) (1497) (1498) (1499) (1500) (1501) (1502) (1503) (1504) (1505) (1506) (1507) (1508) (1509) (1510) (1511) (1512) (1513) (1514) (1515) (1516) (1517) (1518) (1519) (1520) (1521) (1522) (1523) (1524) (1525) (1526) (1527) (1528) (1529) (1530) (1531) (1532) (1533) (1534) (1535) (1536) (1537) (1538) (1539) (1540) (1541) (1542) (1543) (1544) (1545) (1546) (1547) (1548) (1549) (1550) (1551) (1552) (1553) (1554) (1555) (1556) (1557) (1558) (1559) (1560) (1561) (1562) (1563) (1564) (1565) (1566) (1567) (1568) (1569) (1570) (1571) (1572) (1573) (1574) (1575) (1576) (1577) (1578) (1579) (1580) (1581) (1582) (1583) (1584) (1585) (1586) (1587) (1588) (1589) (1590) (1591) (1592) (1593) (1594) (1595) (1596) (1597) (1598) (1599) (1600) (1601) (1602) (1603) (1604) (1605) (1606) (1607) (1608) (1609) (1610) (1611) (1612) (1613) (1614) (1615) (1616) (1617) (1618) (1619) (1620) (1621) (1622) (1623) (1624) (1625) (1626) (1627) (1628) (1629) (1630) (1631) (1632) (1633) (1634) (1635) (1636) (1637) (1638) (1639) (1640) (1641) (1642) (1643) (1644) (1645) (1646) (1647) (1648) (1649) (1650) (1651) (1652) (1653) (1654) (1655) (1656) (1657) (1658) (1659) (1660) (1661) (1662) (1663) (1664) (1665) (1666) (1667) (1668) (1669) (1670) (1671) (1672) (1673) (1674) (1675) (1676) (1677) (1678) (1679) (1680) (1681) (1682) (1683) (1684) (1685) (1686) (1687) (1688) (1689) (1690) (1691) (1692) (1693) (1694) (1695) (1696) (1697) (1698) (1699) (1700) (1701) (1702) (1703) (1704) (1705) (1706) (1707) (1708) (1709) (1710) (1711) (1712) (1713) (1714) (1715) (1716) (1717) (1718) (1719) (1720) (1721) (1722) (1723) (1724) (1725) (1726) (1727) (1728) (1729) (1730) (1731) (1732) (1733) (1734) (1735) (1736) (1737) (1738) (1739) (1740) (1741) (1742) (1743) (1744) (1745) (1746) (1747) (1748) (1749) (1750) (1751) (1752) (1753) (1754) (1755) (1756) (1757) (1758) (1759) (1760) (1761) (1762) (1763) (1764) (1765) (1766) (1767) (1768) (1769) (1770) (1771) (1772) (1773) (1774) (1775) (1776) (1777) (1778) (1779) (1780) (1781) (1782) (1783) (1784) (1785) (1786) (1787) (1788) (1789) (1790) (1791) (1792) (1793) (1794) (1795) (1796) (1797) (1798) (1799) (1800) (1801) (1802) (1803) (1804) (1805) (1806) (1807) (1808) (1809) (1810) (1811) (1812) (1813) (1814) (1815) (1816) (1817) (1818) (1819) (1820) (1821) (1822) (1823) (1824) (1825) (1826) (1827) (1828) (1829) (1830) (1831) (1832) (1833) (1834) (1835) (1836) (1837) (1838) (1839) (1840) (1841) (1842) (1843) (1844) (1845) (1846) (1847) (1848) (1849) (1850) (1851) (1852) (1853) (1854) (1855) (1856) (1857) (1858) (1859) (1860) (1861) (1862) (1863) (1864) (1865) (1866) (1867) (1868) (1869) (1870) (1871) (1872) (1873) (1874) (1875) (1876) (1877) (1878) (1879) (1880) (1881) (1882) (1883) (1884) (1885) (1886) (1887) (1888) (1889) (1890) (1891) (1892) (1893) (1894) (1895) (1896) (1897) (1898) (1899) (1900) (1901) (1902) (1903) (1904) (1905) (1906) (1907) (1908) (1909) (1910) (1911) (1912) (1913) (1914) (1915) (1916) (1917) (1918) (1919) (1920) (1921) (1922) (1923) (1924) (1925) (1926) (1927) (1928) (1929) (1930) (1931) (1932) (1933) (1934) (1935) (1936) (1937) (1938) (1939) (1940) (1941) (1942) (1943) (1944) (1945) (1946) (1947) (1948) (1949) (1950) (1951) (1952) (1953) (1954) (1955) (1956) (1957) (1958) (1959) (1960) (1961) (1962) (1963) (1964) (1965) (1966) (1967) (1968) (1969) (1970) (1971) (1972) (1973) (1974) (1975) (1976) (1977) (1978) (1979) (1980) (1981) (1982) (1983) (1984) (1985) (1986) (1987) (1988) (1989) (1990) (1991) (1992) (1993) (1994) (1995) (1996) (1997) (1998) (1999) (2000) (2001) (2002) (2003) (2004) (2005) (2006) (2007) (2008) (2009) (2010) (2011) (2012) (2013) (2014) (2015) (2016) (2017) (2018) (2019) (2020) (2021) (2022) (2023) (2024) (2025) (2026) (2027) (2028) (2029) (2030) (2031) (2032) (2033) (2034) (2035) (2036) (2037) (2038) (2039) (2040) (2041) (2042) (2043) (2044) (2045) (2046) (2047) (2048) (2049) (2050) (2051) (2052) (2053) (2054) (2055) (2056) (2057) (2058) (2059) (2060) (2061) (2062) (2063) (2064) (2065) (2066) (2067) (2068) (2069) (2070) (2071) (2072) (2073) (2074) (2075) (2076) (2077) (2078) (2079) (2080) (2081) (2082) (2083) (2084) (2085) (2086) (2087) (2088) (2089) (2090) (2091) (2092) (2093) (2094) (2095) (2096) (2097) (2098) (2099) (2100) (2101) (2102) (2103) (2104) (2105) (2106) (2107) (2108) (2109) (2110) (2111) (2112) (2113) (2114) (2115) (2116) (2117) (2118) (2119) (2120) (2121) (2122) (2123) (2124) (2125) (2126) (2127) (2128) (2129) (2130) (2131) (2132) (2133) (2134) (2135) (2136) (2137) (2138) (2139) (2140) (2141) (2142) (2143) (2144) (2145) (2146) (2147) (2148) (2149) (2150) (2151) (2152) (2153) (2154) (2155) (2156) (2157) (2158) (2159) (2160) (2161) (2162) (2163) (2164) (2165) (2166) (2167) (2168) (2169) (2170) (2171) (2172) (2173) (2174) (2175) (2176) (

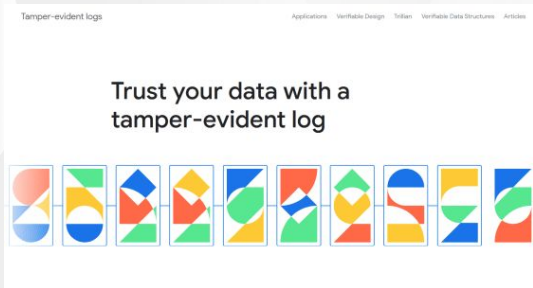
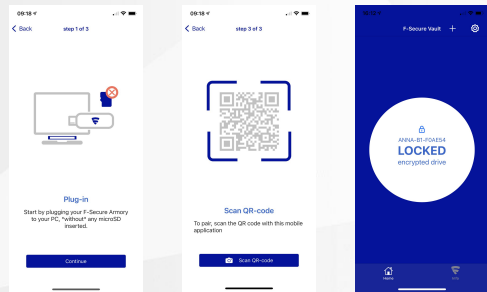
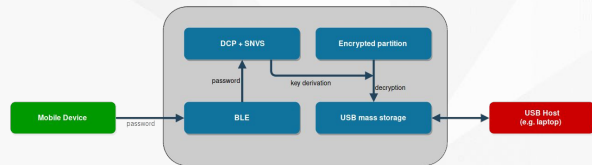
Armory Drive - Encrypted USB Mass Storage

The application implements the **easiest to use encrypted drive solution** allowing secure access to any microSD card (no storage limit).

Unlike existing encrypted drive solutions the key is unlocked
With **3 factors** (user + mobile phone + armory) and **over Bluetooth**.

No trust (or driver requirements) are delegated to the host.

It consists of **~2400 LOC** of pure TamaGo code and an iOS app.



It uses Google Firmware Transparency framework to enable firmware update authentication on the installer as well as the device itself.



Armory Drive - Demo

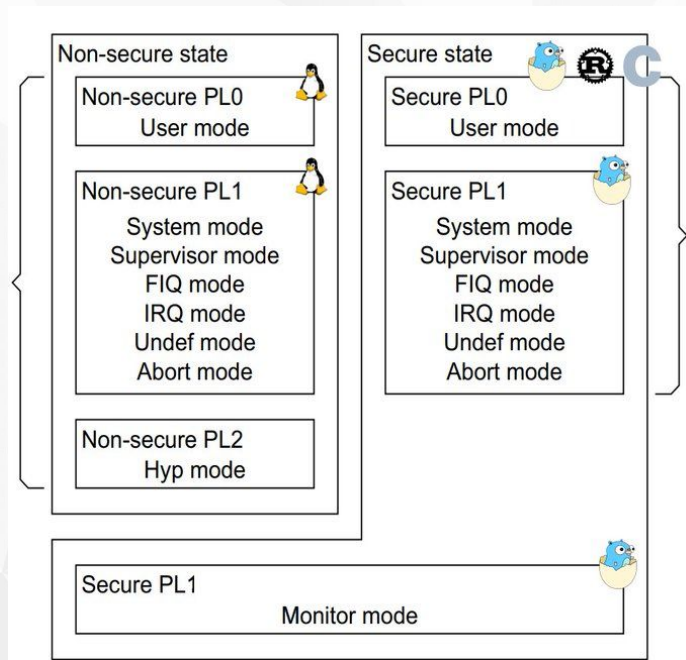


GoTEE - Trusted Execution Environment

The GoTEE framework implements concurrent instantiation of TamaGo based unikernels in privileged and unprivileged modes, interacting with each other through monitor mode and custom system calls.

With these capabilities GoTEE implements a pure Go Trusted Execution Environment (TEE) bringing Go memory safety, convenience and capabilities to bare metal execution within TrustZone Secure World.

It supports any freestanding user mode applets (e.g. TamaGo, C, Rust) and any “rich” OS running in NonSecure World (e.g. Linux).



Demo: GoTEE

```
> gotee
PL1 tamago/arm (go1.18.3) • TEE system/monitor (Secure World)
PL1 loaded applet addr:0x9c000000 size:4719809 entry:0x9c06f188
PL1 loaded kernel addr:0x80000000 size:4384184 entry:0x8006db70
PL1 starting mode:SYS ns:true sp:0x00000000 pc:0x8006db70
PL1 starting mode:USR ns:false sp:0x9e000000 pc:0x9c06f188
PL1 tamago/arm (go1.18.4) • system/supervisor (Normal World)
PL1 in Normal World is about to perform DCP key derivation
PL1 in Normal World successfully used DCP (df3eed2a50c9dd22daf7cf864f27bb90)
PL1 in Normal World is about to yield back
r0:00000000 r1:814243f0 r2:00000001 r3:00000000
r4:00000000 r5:00000000 r6:00000000 r7:8146bf14
r8:00000007 r9:00000034 r10:814040f0 r11:802e9b21 cpsr:60001d6 (MON)
r12:00000000 sp:8146bf54 lr:80185518 pc:80185648 spsr:60001df (SYS)
PL1 stopped mode:SYS ns:true sp:0x8146bf54 lr:0x80185518 pc:0x80185648 err:exit
PL0 tamago/arm (go1.18.4) • TEE user applet (Secure World)
PL0 obtained 16 random bytes from PL1: 10e742f0dad15db3f00aea14ee4a5acc
PL1 loaded kernel addr:0x80000000 size:4384184 entry:0x8006db70
PL1 re-launching kernel with TrustZone restrictions
PL1 starting mode:SYS ns:true sp:0x00000000 pc:0x8006db70
PL1 tamago/arm (go1.18.4) • system/supervisor (Normal World)
PL1 in Normal World is about to perform DCP key derivation
r0:02280000 r1:814683a0 r2:8143c588 r3:00000001
r4:00000000 r5:00000000 r6:00000000 r7:8146bf14
r8:00000007 r9:00000044 r10:814040f0 r11:802e9b21 cpsr:20001d6 (MON)
r12:00000000 sp:8146bf28 lr:80180398 pc:80011340 spsr:20001df (SYS)
PL1 stopped mode:SYS ns:true sp:0x8146bf28 lr:0x80180398 pc:0x80011340 err:DATA_ABORT
PL1 in Secure World is about to perform DCP key derivation
PL1 in Secure World World successfully used DCP (df3eed2a50c9dd22daf7cf864f27bb90)
```

```
$ ssh 10.0.0.1
```

```
PL1 tamago/arm (go1.18.3) • TEE system/monitor (Secure World)
```

```
help                # this help
reboot              # reset the SoC/board
stack               # stack trace of current goroutine
stackall            # stack trace of all goroutines
md <hex offset> <size> # memory display (use with caution)
mw <hex offset> <hex value> # memory write (use with caution)

gotee                # TrustZone test w/ TamaGo unikernels
linux <uSD|eMMC>      # boot NonSecure USB armory Debian image

dbg                  # show ARM debug permissions
csl                  # show config security levels (CSL)
csl <periph> <slave> <hex csl> # set config security level (CSL)
sa                    # show security access (SA)
sa <id> <secure|nonsecure> # set security access (SA)
```

```
> dbg
| type                | implemented | enabled |
|-----|-----|-----|
| Secure non-invasive | 1 | 0 |
| Secure invasive    | 1 | 0 |
| Non-secure non-invasive | 1 | 1 |
| Non-secure invasive | 1 | 0 |
```

```
> linux eMMC
armory-boot: loading configuration at /boot/armory-boot-nonsecure.conf
PL1 loaded kernel addr:0x80000000 size:7603616 entry:0x80800000
PL1 launching Linux
PL1 starting mode:SVC ns:true sp:0x00000000 pc:0x80800000
Booting Linux on physical CPU 0x0
Linux version 5.15.52-0 (usbarmory@usbarmory) arm-linux-gnueabi-hf-gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1)
```



GoTEE in the wild: space!

On May 22nd 2023 (UTC 05:00:32) the USB armory Mk II got a lift to space!

GoTEE supervised a TamaGo based unikernel (acting as Trusted Applet) and a full Linux instance isolated in NonSecure World to test Post Quantum Key exchanges.

For this occasion TamaGo and GoTEE have been updated with full watchdog and interrupt support.

The payload remained operation for the entire flight duration performing exactly 400 PQC key exchanges. As far as we know this is the first time bare metal Go executed in space.



Andrea Barisani
@AndreaBarisani

Yesterday (UTC 05:00:32) @WithSecure USB armory got a lift (~225 km apogee) from the MAPHEUS-13 rocket launched from the Esrange Space Center.

Thanks to @DLR_de @adesso_SE for this amazing collaboration!

Our bare metal GoTEE performed Post Quantum key exchange in space!



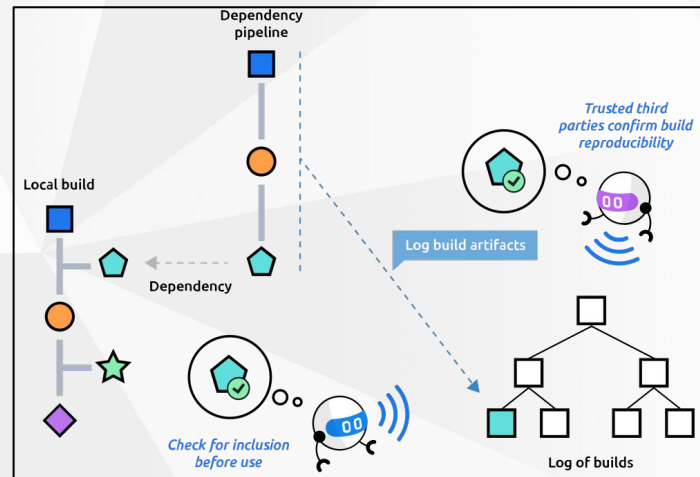
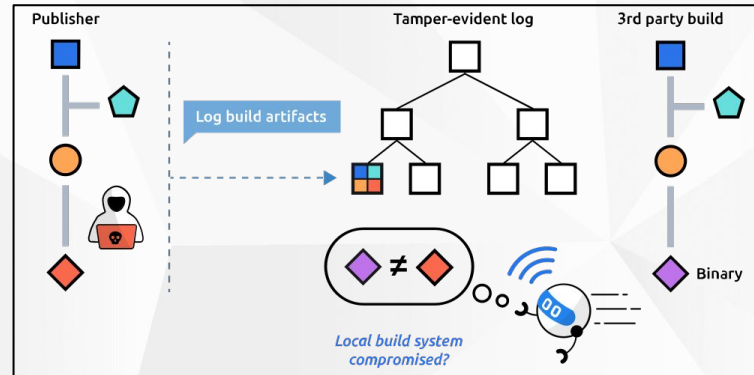
GoTEE in the wild: Armored Witness

WithSecure and Google jointly developed an Armored Witness firmware.

Multi-party signature verification using a bootloader > Trusted OS > Trusted Applet trust chain.

The Trusted OS holds the network device driver (USB or Ethernet). The Trusted Applet holds the TCP/IP stack and networking code, including DHCP (all Go).

The applet monitors tamper-evident logs of public open source repositories to detect misbehaviour and supply chain attacks.



Demo: Armored Witness - Trusted OS / Trusted Applet

```
boot: tamago/arm • i.MX6UL
boot: starting kernel@1007dee8
os: tamago/arm • TEE security monitor (Secure World system/monitor)
os: loading applet from MMC storage
os: SM applet verification pub:RWQiFth4tAgsVQT5caaZGJGgUzZFnwCdeVHe5XpobGwc9XzCJmjJ56t0
os: SM applet loaded addr:0x20000000 entry:0x20081700 size:15323136
os: SM applet started mode:USR sp:0x30000000 pc:0x20081700 ns:false
ta: tamago/arm • TEE user applet
ta: SM starting network
ta: SNVS - Deriving hardware key
ta: Opening storage - CardInfo: {BlockSize:512 Blocks:8388608}
os: SM registering applet event handler g:0x21803900 p:0x21826000
ta: MAC:26:76:04:d4:4d:db IP:10.0.0.1/24 GW:10.0.0.0/24 DNS:8.8.8.8:53
ta: Starting witness...
ta: TA starting ssh server (SHA256:IH0WYXV66dvixxOPDhtAa1AvWg+sQC...) at 10.0.0.1:22
ta: Feeder "go.sum database tree" goroutine started
ta: Feeder "Armory Drive Prod 2" goroutine started
ta: Feeder "rekor.sigstore.dev - 3904496407287907110" goroutine started
ta: Feeder "rekor.sigstore.dev - 2605736670972794746" goroutine started
ta: Feeder "developers.google.com/android/binary_transparency/0" goroutine started
ta: "sum.golang.org: go.sum database tree" grew - @0: → @19895786: 3c9b8f49f56cb...
ta: "armory-drive-log: Armory Drive Prod 2" grew - @0: → @2: 02a14ca4a7313d868a4...
ta: "rekor.sigstore.dev: 2605736670972794746" grew - @0: → @36541297: 0d491271b9...
ta: "pixel_transparency_log: .../binary_transparency/0" grew - @0: → @235: f98458...
ta: "rekor.sigstore.dev: 3904496407287907110" grew - @0: → @4163431: 4d006aa46ef...
ta: "lvfs: lvfs" grew - @0: → @12749: 6ac429c550b8b28b7c65b6e61c99c9c76303d14012...
ta: No checkpoint
```

```
$ ssh 10.0.0.1
```

```
TA tamago/arm (go1.18.3) • TEE user applet (User Mode)
```

date	(time in RFC339 format)?	# show/change runtime date and time
dns	<fqdn>	# resolve domain (requires routing)
exit, quit		# close session
hab	<hex SRK hash>	# secure boot activation (*irreversible*)
help		# this help
led	(white blue yellow green) (on off)	# LED control
mmc	<hex offset> <size>	# MMC card read
reboot		# reset device
stack		# stack trace of current goroutine
stackall		# stack trace of all goroutines
status		# status information

```
> status
```

```
----- Trusted Applet ----
Runtime .....: go1.21.1 tamago/arm
----- Trusted OS ----
Serial number .....: 3bee6cda358f0c33
Secure Boot .....: false
Revision .....: 70ffeda
Build .....: lcars@lambda on 2023-10-05 08:50:01
Version .....: 1696495801 (2023-10-05 08:50:01 +0000 UTC)
Runtime .....: go1.21.1 tamago/arm
Link .....: true
Witness/Identity .....: DEV:ArmoredWitness-still-tree+e1f17ea8+AZvDSL1C0...
Witness/IP .....: 10.0.0.1
```



armory-boot - USB armory boot loader

A primary signed boot loader (~300 LOC) to launch authenticated Linux kernel images on secure booted¹ USB armory boards, replacing U-Boot.

```
func boot(kernel []byte, dtb []byte, cmdline string) {
    dma.Init(dmaStart, dmaSize)
    mem, _ := dma.Reserve(dmaSize, 0)

    dma.Write(mem, kernel, kernelOffset)
    dma.Write(mem, dtb, dtbOffset)

    image := mem + kernelOffset
    params := mem + dtbOffset

    arm.ExceptionHandler = func(n int) {
        if n != arm.SUPERVISOR {
            panic("unhandled exception")
        }
    }

    usbarmory.LED("blue", false)
    usbarmory.LED("white", false)

    imx6ul.ARM.DisableInterrupts()
    imx6ul.ARM.FlushDataCache()
    imx6ul.ARM.Disable()

    exec(image, params)
})

svc()
}
```

```
func verifySignature(buf []byte, s []byte) (valid bool, err error) {
    sig, err := DecodeSignature(string(s))

    if err != nil {
        return false, fmt.Errorf("invalid signature, %v", err)
    }

    pub, err := NewPublicKey(PublicKeyStr)

    if err != nil {
        return false, fmt.Errorf("invalid public key, %v", err)
    }

    return pub.Verify(buf, sig)
}

func verifyHash(buf []byte, s string) bool {
    // use hardware acceleration
    sum, _ := imx6ul.DCP.Sum256(buf) {

    if hash, err := hex.DecodeString(s); err != nil {
        return false
    }

    return bytes.Equal(sum[:], hash)
}
```



Performance

Go code runs (expectedly) with identical, or improved, speed compared to the same code executed under a full blown OS.

TamaGo drivers operates comparably to their Linux counterparts, no serious overhead is present and anyway absolute performance is not a main focus of the effort, which remains security oriented.

Go ECDSA testsuite ¹	TamaGo	Linux
ECDSA sign+verify p224	115 ms	116 ms
ECDSA sign+verify p256	48 ms	46 ms
ECDSA sign+verify p384	1.85 s	1.89 s
ECDSA sign+verify p521	3.48 s	3.60 s

AES-128-CBC encryption w/ DCP	TamaGo	OpenSSL (afalg)
65536 blocks for 10s	6208	4528
4096 blocks for 10s	70326	60204

Go standard libraries run with comparable performance, while TamaGo hardware drivers highlight increased performance.

¹ https://github.com/golang/go/blob/go1.13.6/src/crypto/ecdsa/ecdsa_test.go#L124



Current limitations

The TamaGo runtime is single threaded therefore:

- avoid¹ tight loops without function calls
- avoid deadlocks (e.g. do not sleep in `main()` if nothing else is happening)

Packages/applications which rely on unsupported system calls do not compile (e.g. terminal prompt packages that require `syscall.SYS_IOCTL`), though usually such packages do not make sense in the context of OS-less unikernel operations.

Importing libraries that require `cgo` can only be done with internal linking, integrating C code with `cgo` is possible as long as such code is free standing.

There is no OS, there are no users, there are no signals, there are no environment variables. This is a feature, not a bug.

With the exception of few limitations² Go is surprisingly adept to run on bare metal.

¹ or just force `runtime.Gosched`

² <https://github.com/usbarmory/tamago/wiki/Internals#go-application-limitations>



Applications and future

TamaGo `imx6ul` package supports a wide variety of i.MX6 SoC drivers, Raspberry Pi and RISC-V support is also available.

**TamaGo lays out the foundation for development of pure Golang
HSMs,
cryptocurrency wallets,
authentication tokens,
TrustZone secure monitors,
and much more...**

It is our policy to keep comments and references (document title and page number) for all low level interactions within drivers.

TamaGo source code is a great tool to learn low level SoC development!



What have we¹ learned?

Bare metal applications can play a big role in the future of secure embedded systems and can be built by **reducing complexity**.

We feel the need for a paradigm shift and think there is no place for C code in complex drivers or applications anymore.

Go is a language that, among others, can definitely play a role in this.

To achieve trust we proved that Go distribution modifications can be minimal to achieve bare metal execution.

We completely **killed C**².

It's all about enabling choice and building trust.

¹ “We” as in the authors, but maybe the audience as well.

² The SoC boot ROM jumps directly to Go runtime.



Repository: <https://github.com/usbarmory/tamago>
Documentation: <https://github.com/usbarmory/tamago/wiki>
API: <http://pkg.go.dev/github.com/usbarmory/tamago>

Q & A

Andrea Barisani

@AndreaBarisani - andrea.bio

andrea@inversepath.com