Runtime Verification for Protocol Implementation

Secure Communication in the Quantum Era (SPS G5448)
Project Meeting, September 26th, 2019

Christian Colombo
Mark Vella

L-Università ta' Malta

NATO OTAN

Steps led by UM

2B - Identify protocol-level security mechanisms

(March 2020 → March 2021)

3B - Deploy implementation-level security mechanisms

(October 2020 → October 2021)

Progress

Identification of protocol-level security mechanisms **(2B)**

Identified different level at which RV can be useful

Design of runtime verification architecture at these various levels **(2B)**

Including enforcement of a Trusted Domain through RV

Preliminary implementation of the top level **(3B)**

L-Università ta' Malta

NATO OTAN

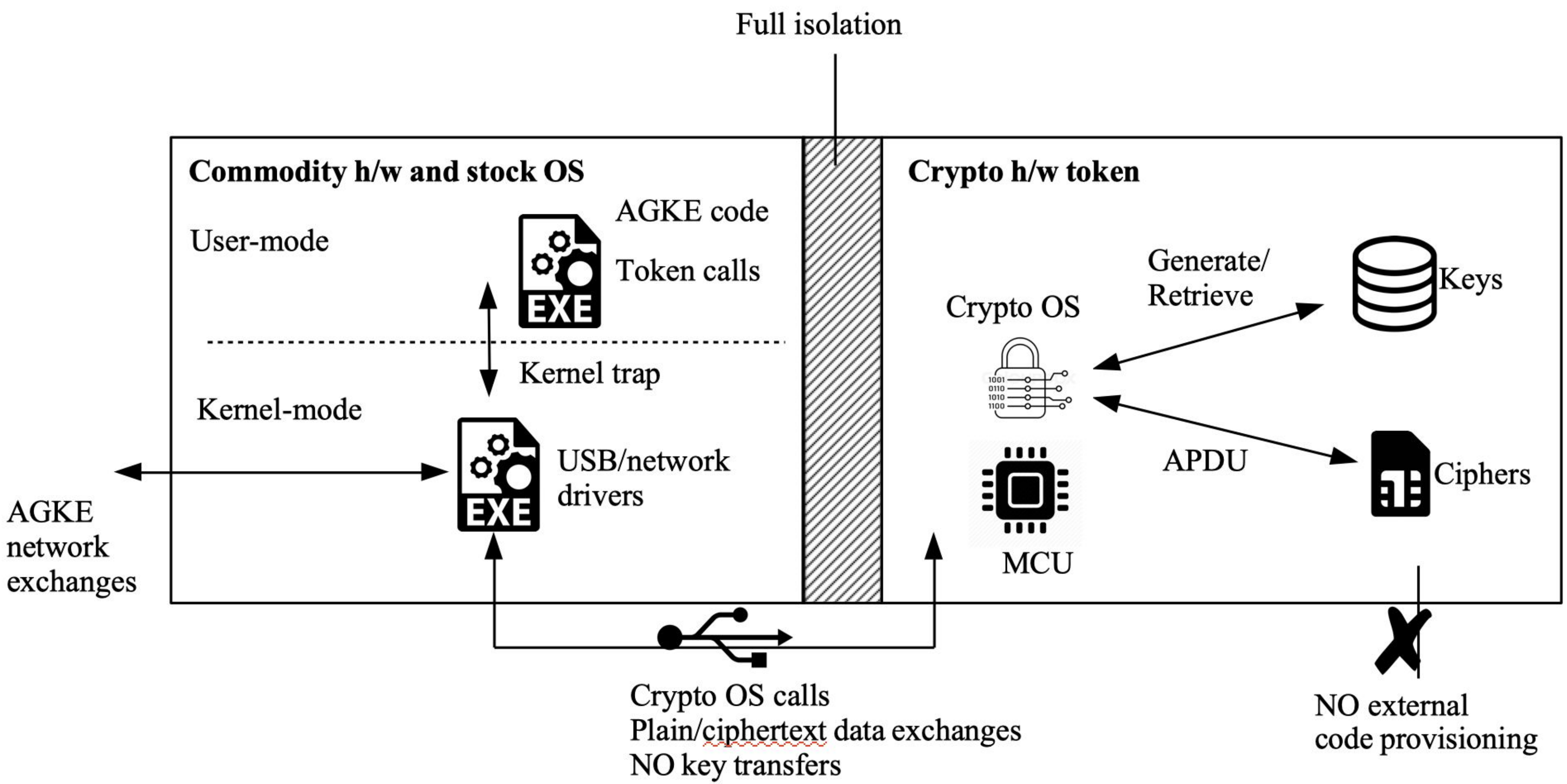
Levels of abstraction of security threats

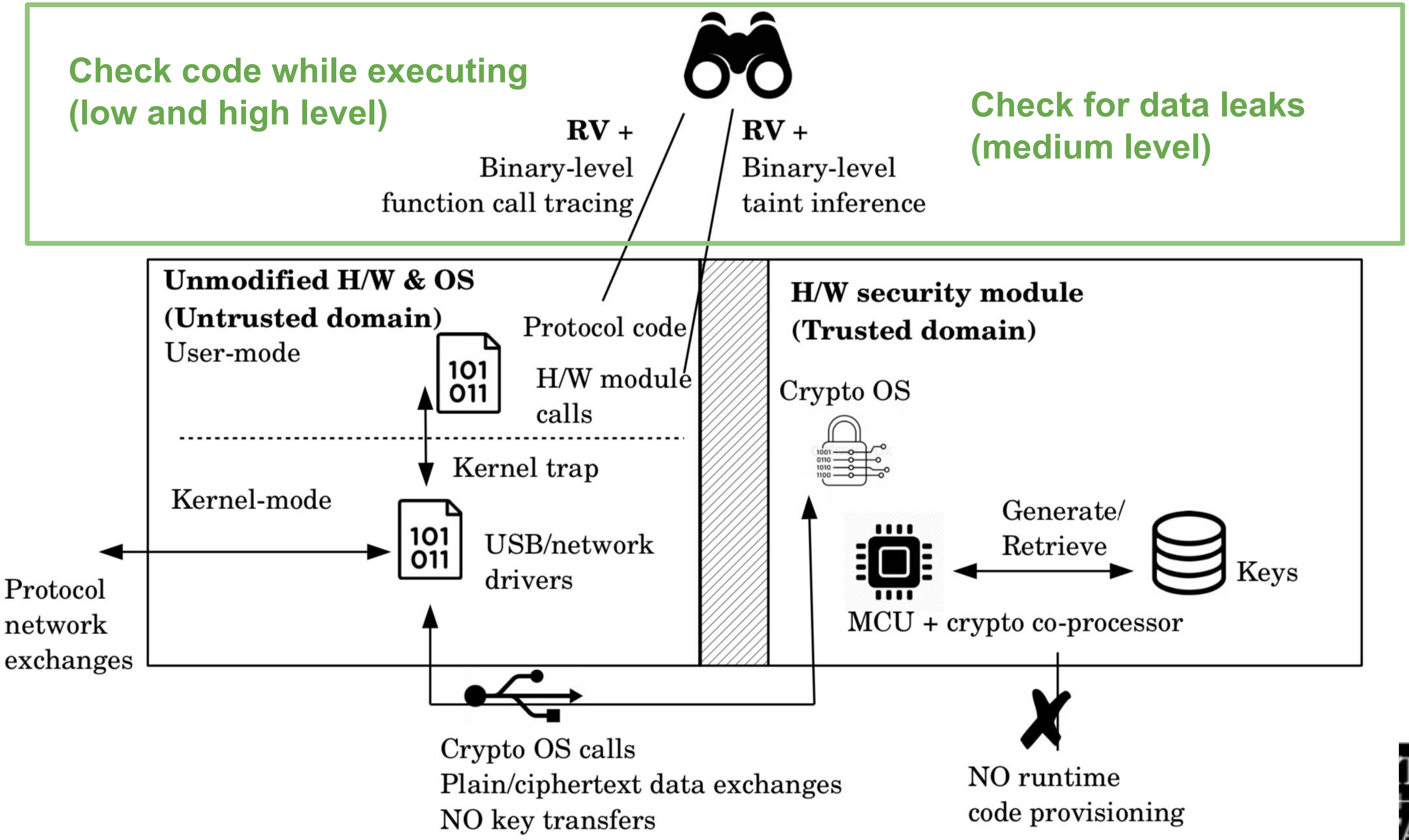
(High level) Wrong protocol implementation	The protocol implementation might deviate from the verified (theoretical) design
Medium level threats	Malware, Data leaks, etc
Low level threats	Arithmetic overflows, undefined downcasts, and invalid pointer references
Hardware	Can hardware be trusted? Side Channel attacks?

Design using RV

Use some specialised hardware to isolate sensitive processes

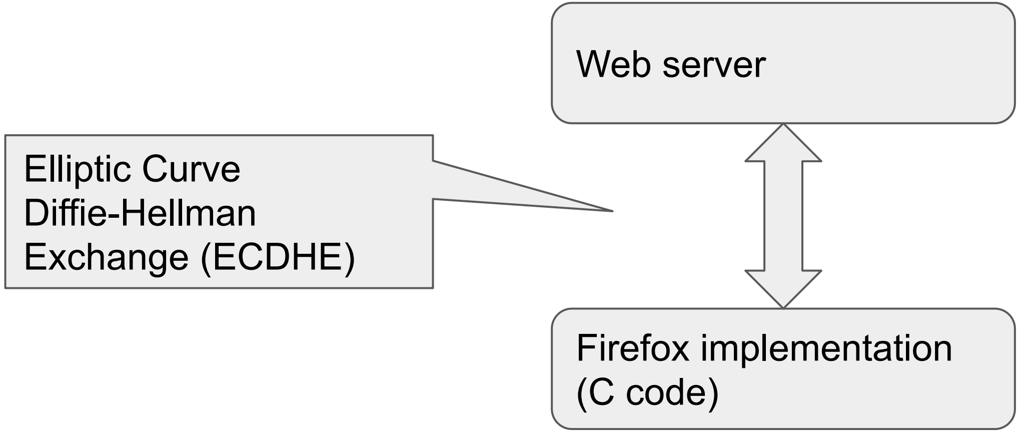
Place monitors at strategic points


Full isolation
Commodity h/w and stock OS
AGKE code
User-mode
Token calls
Kernel trap
Kernel-mode
USB/network
drivers
AGKE
network
exchanges
Crypto h/w token
Generate/
Retrieve
Keys
Crypto OS
APDU
Ciphers
MCU
Crypto OS calls
Plain/ciphertext data exchanges
NO key transfers
NO external
code provisioning



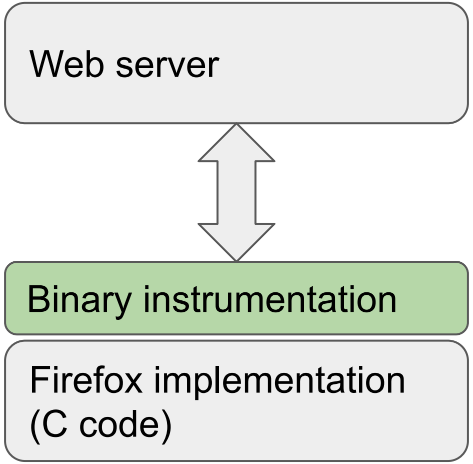
Check code while executing
(low and high level)
Check for data leaks
(medium level)
RV +
Binary-level
function call tracing
RV +
Binary-level
taint inference
Unmodified H/W & OS
(Untrusted domain)
User-mode
101
011
Protocol code
H/W module
calls
Kernel trap
Kernel-mode
101
011
USB/network
drivers
Protocol
network
exchanges
H/W security module
(Trusted domain)
Crypto OS
1001
0110
1010
1100
Generate/
Retrieve
Keys
MCU + crypto co-processor
Crypto OS calls
Plain/ciphertext data exchanges
NO key transfers
NO runtime
code provisioning

Preliminary implementation case study



Preliminary implementation

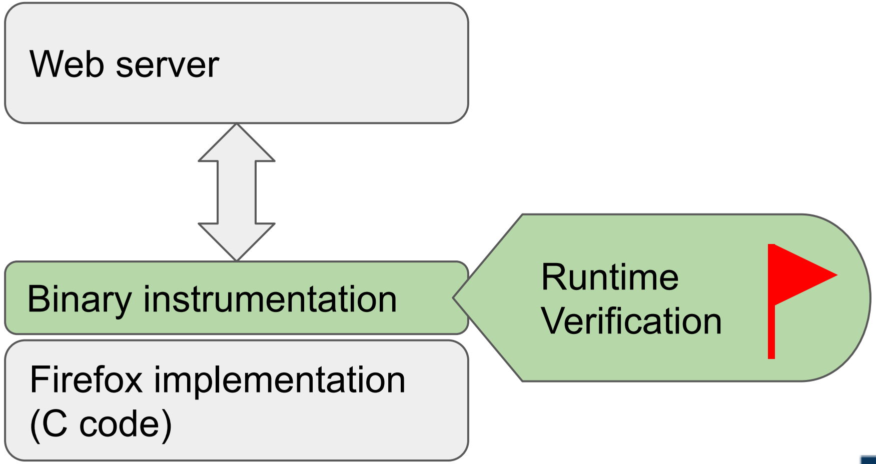
Setup using Binary-level instrumentation



Preliminary implementation

Setup using Binary-level instrumentation

Through which monitors can gain visibility



L-Università ta' Malta

NATO OTAN

Properties verified (High level) on ECDHE

Digital certificate verification is done (in order to authenticate public keys sent by peers)

Validation of remote peer's **public key** on each exchange is done (unless the session is aborted)

Once master secret is established, private keys should be **scrubbed from memory** (to limit the impact of memory leak attacks such as Heartbleed, irrespective of whether the session is aborted)

L-Università ta' Malta

NATO OTAN

Feasibility study of approach

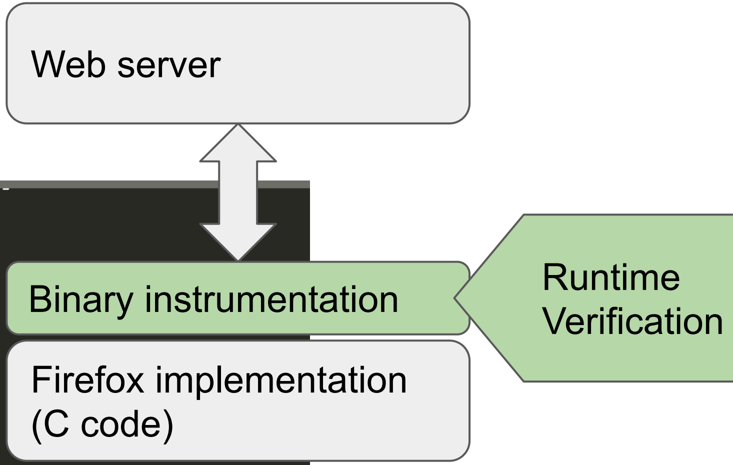
Is the approach possible for a realistic code base?

Is the approach feasible in terms of overheads?

Used the Firefox case study on top 100 Alexa sites

Feasibility study

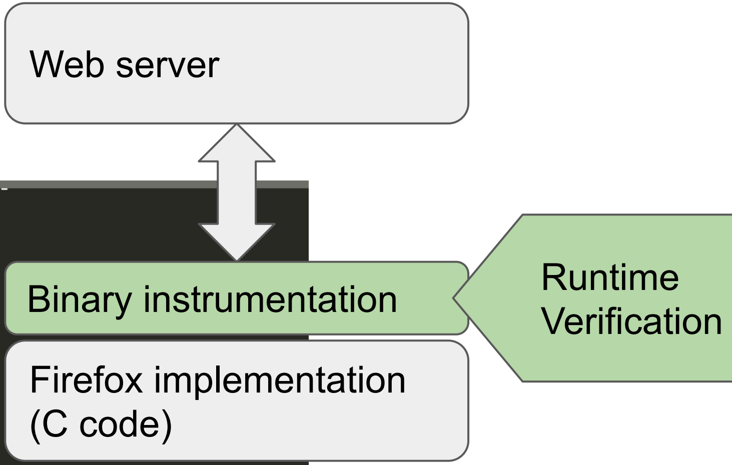
```
            /* TID 0x1003 */
200 ms  0x1003 PR_Close()
200 ms  0x1003 fd:0x7faa3ded6e20
            /* TID 0xffb */
312 ms  0xffb SSL_ImportFD()
312 ms  0xffb ret:0x7faa43591940
312 ms  0xffb SSL_AuthCertificateHook()
312 ms  0xffb fd:0x7faa43591940
312 ms  0xffb ret:0x0
312 ms  0xffb PR_Connect()
312 ms  0xffb fd:0x7faa43591940
531 ms  0xffb SECKEY_CreateECPrivateKey()
531 ms  0xffb cx:0x7faa3deda988
532 ms       | 0xffb EC_ValidatePublicKey()
532 ms       | 0xffb ret:0x0
532 ms  0xffb ret:0x7faa3dd66020::7faa3dd66020  c0 fe d9
3d                                           ...=
533 ms  0xffb SECKEY_CreateECPrivateKey()
533 ms  0xffb cx:0x7faa3deda988
534 ms       | 0xffb EC_ValidatePublicKey()
539 ms       | 0xffb ret:0x0
539 ms  0xffb ret:0x7faa3dd68020::7faa3dd68020  00 f1 10
```



Feasibility study

```
          /* TID 0x1003 */
200 ms  0x1003 PR_Close()
200 ms  0x1003 fd:0x7faa3ded6e20
          /* TID 0xffb */
312 ms  0xffb SSL_Imp      rtFD()
312 ms  0xffb ret:0x7f        3591940
312 ms  0xffb SSL_Auth            ficateHook()
312 ms  0xffb fd:0x7faa
312 ms  0xffb ret:0x0
312 ms  0xffb PR_
312 ms  0xffb fd:
531 ms  0xffb SEC
531 ms  0xffb cx:
532 ms      | 0xff
532 ms      | 0xff
532 ms  0xffb ret:0x7faa3dd66020::7faa3dd66020  c0 fe d9
3d                                          ...=
533 ms  0xffb SECKEY_CreateECPrivateKey()
533 ms  0xffb cx:0x7faa3deda988
534 ms      | 0xffb EC_ValidatePublicKey()
539 ms      | 0xffb ret:0x0
539 ms  0xffb ret:0x7faa3dd68020::7faa3dd68020  00 f1 10
```

Challenge: Threads didn’t correspond to sessions



Challenge: efficiency vs precision

How do you keep track which method calls belong to which session?

Firefox is built for efficiency not monitorability

Two options:

- Trace all method calls
- Change Firefox implementation

Challenge: efficiency vs precision

How do you keep track which method calls belong to which session?

Another option:

Trace only the methods of interest

Use a heuristic (around 98% effectiveness)

What does the specification language look like?

```
Transitions {
 start –> newsession                              [sslimport]
 newsession –> server_connect                     [prconnect]
 server_connect –> failed_cert_auth    [sslauthcertcompl]
 failed_cert_auth –> close       [prclose\\mcParent=mc;]
 close –> certerr_ok  [destroypk\mc.hasParent(mcParent)]

 failed_cert_auth –> certerr_bad                      [eot]
      close –> certerr_bad                            [eot]
}
```

L-Università ta' Malta

NATO OTAN

Overheads measurement

Configuration	Pages	Page load time (ms)	
		mean	***std. dev.***
No RV	1,000	6,918.37	24,870.86
With RV	1,000	7,282.35	27,328.9
Mean overhead	0.05		
Wilcoxon signed-rank test	p=0.281		

L-Università ta' Malta

NATO OTAN

Overheads measurement

0.05 ms per page

Configuration	Pages	Page lo	
		mean	*d. dev.*
No RV	1,000	6,918.37	24,870.86
With RV	1,000	7,282.35	27,328.9
Mean overhead	0.05		
Wilcoxon signed-rank test	p=0.281		

Lessons learnt

Good start with promising results - approach seems feasible

Beware:

> **Program comprehension is required**, both for setting up function hooks as well as to enable individual TLS session monitoring
>
> Real-world code tends to be written in a manner to **favor efficient execution rather than monitorability** (eg, was difficult to keep track of particular sessions on the server)

L-Università ta' Malta

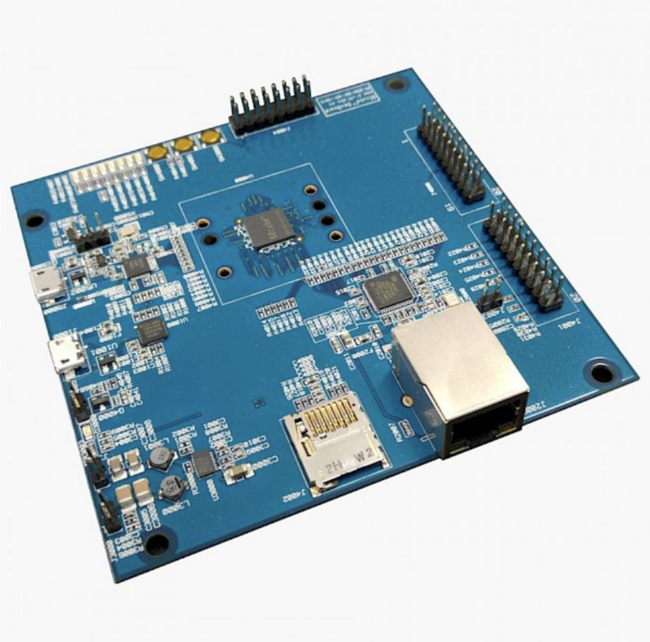
NATO OTAN

Moving forward

Implementation on SEcube Development Kit

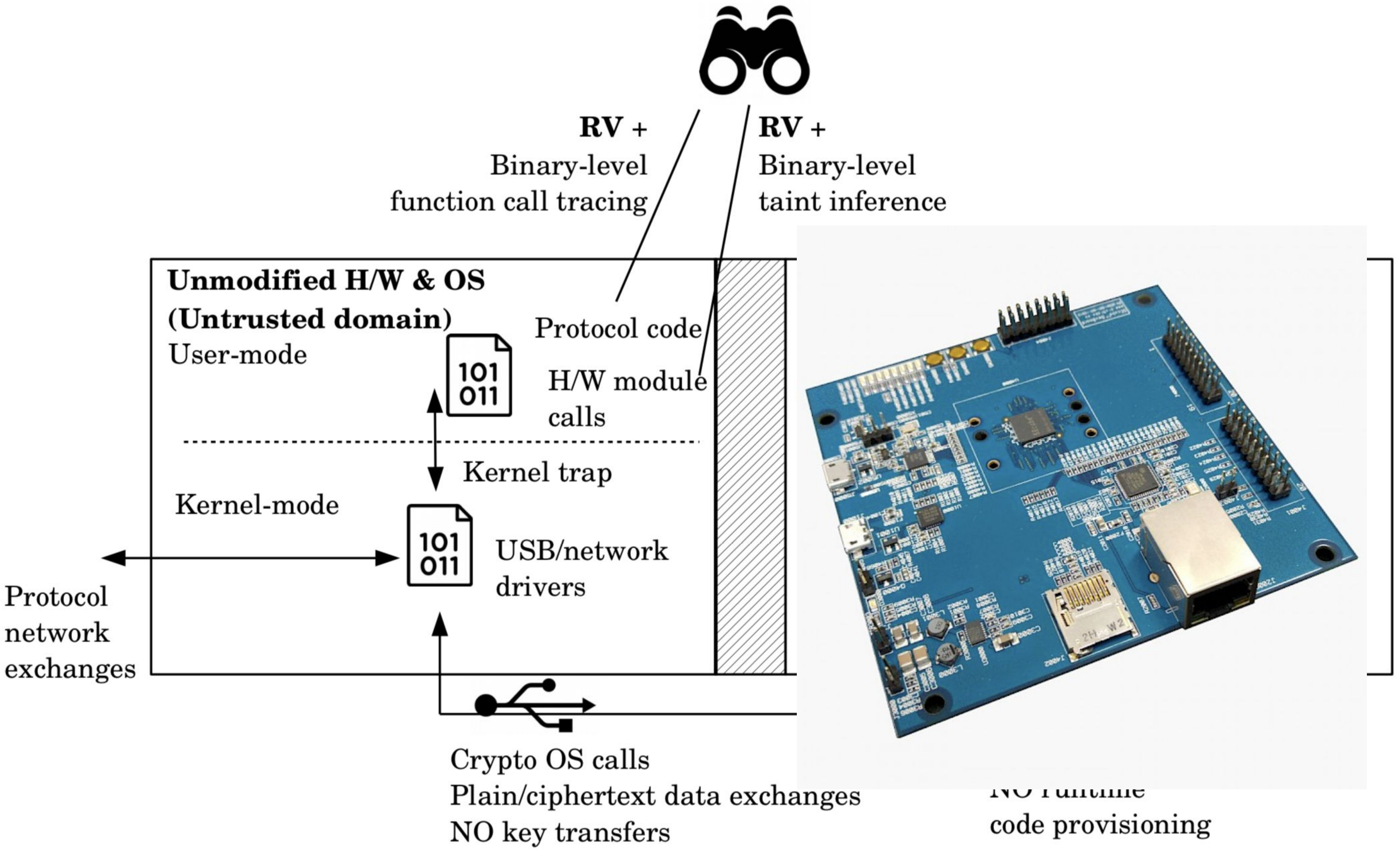
Key generation will take place on dedicated HW

While still monitoring the protocol execution



L-Università ta' Malta

NATO OTAN



RV +
Binary-level
function call tracing
RV +
Binary-level
taint inference
Unmodified H/W & OS
(Untrusted domain)
User-mode
Protocol code
H/W module
calls
Kernel trap
Kernel-mode
USB/network
drivers
Protocol
network
exchanges
Crypto OS calls
Plain/ciphertext data exchanges
NO key transfers
code provisioning

Plan

Identification of (the actual) protocol-level properties **(D1) deadline Dec 2019**

Implementation

- Setup with SEcube hardware **(next step with Peter)**
- Monitoring our “quantum” protocol with this setup
- Low level runtime verification (using existing libraries)
- Taint inference

L-Università ta' Malta

NATO OTAN