What's Necessary to Establish Malware Freedom Unconditionally?

Virgil D. Gligor

ECE and CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

FCS Workshop
Boston
June 22, 2020

Outline

I. Background

- adversary: persistent malware & its remote controller
- malware-free state? unconditionally ?
- a **sufficient solution** for the ***cWRAM model***

II. What's necessary on *real systems*?

- external verifiers and challenge functions

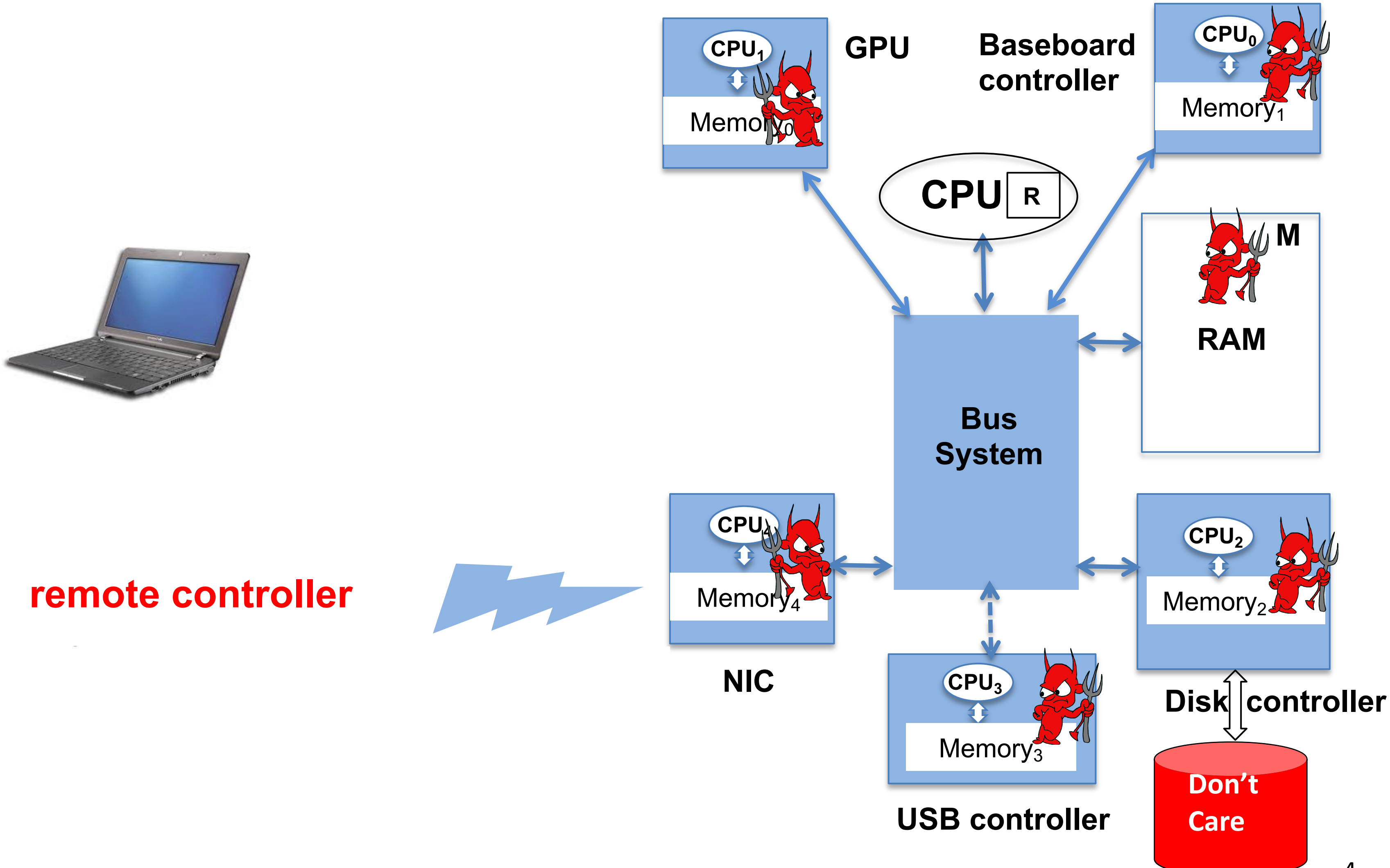
challenge functions:
- optimal space-time bounds (m. t)
- unique (m, t) bounds for code
- target claw free within (m, t) bounds

III. Q & A

I. Background

V. Gligor and M. Woo, "*Establishing Software Root of Trust Unconditionally*,"
in **Proc. of NDSS**, San Diego, CA. 2019.
(full length paper - **CyLab TR 2018 -003**, Nov. 2018)

V. Gligor, "*A Rest Stop on the Unending Road to Provable Security*"
in **Proc. of SPW**, Cambridge University, UK, 2019 (article and transcript of discussion)


remote controller
GPU
Baseboard controller
CPU_1
$Memory_0$
CPU_0
$Memory_1$
CPU R
M
RAM
Bus System
CPU_4
$Memory_4$
NIC
CPU_3
$Memory_3$
USB controller
CPU_2
$Memory_2$
Disk controller
Don't Care

Adversary:
persistent malware & its remote controller

persistent malware *can*

- extract *all software secrets* stored on its computer
- modify all SW/FW; e.g., at system initialization
- read/write all I/O channels & communicate with ***remote controller***
- adaptively modify programs and data & execute any function on chosen input

but

- *cannot access the processors & storage (e.g., random bits) of a* ***connected system***

remote controller *can*

- exercise *all attacks that implant* ***persistent malware*** on remote system
- *communicate with & control* ***persistent malware***
- *use unbounded computation power: e.g.,* break *all complexity-based crypto*

but

- *cannot predict* Nature's throw of fair dice . . . or random bits of an QRNG
- *cannot modify* a system's *HW*

Malware-free states? Unconditionally?

Persistent malware has *no* externally observable (hyper)properties

Q: How can malware-free states be established (w/o taking the system apart)**?**

A: RoT state (“*all and only chosen content*”) **=> malware-free state**

RoT failure => detect **malware execution** or **unaccounted content**

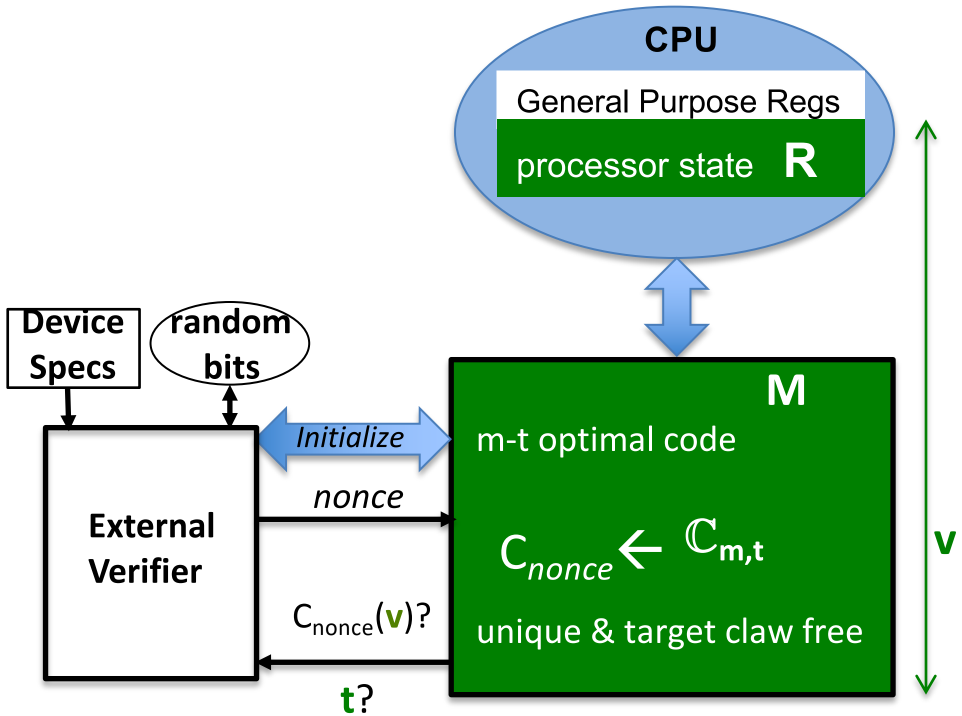
` (e.g., malware caused), or **both**

Unconditional Establishment of RoT State

- **no** secrets, **no** trusted HW modules, **no** bounds on remote adversary’s power

- need **only** *truly random bits* & *HW specifications*

A Sufficient Solution on the cWRAM



OK => *RoT* on malware-free Device

Overview: cWRAM ISA++

- ***Constants:*** ***w****-bit word*, up to ***2 operands****/instruction*
  instructions execute in ***unit time***; no cycles, frequency, voltage, current, …

- ***Memory***: ***M*** words
- **Processor registers**: GPRs, PC, PSW, Special Processor Registers ***R***
- ***Addressing***: immediate, relative, direct, indirect
- ***Architecture features:*** caches, virtual memory, TLBs, pipelining, multi-core processors

- ***ISA: all (un)signed integer instructions***
  - All Loads, Stores, Register transfers
  - All Unconditional & Conditional Branches, all branch types
    - *all predicates with 1 or 2 operands*
  - *Halt*
  - All *Computation* Instructions:
    - addition, subtraction, logic, $shift_{r/l}(R_i, \alpha)$, $rotate_{r/l}(R_i, \alpha)$, . . .
    - *variable* $shift_{r/l}(R_i, R_j)$, *variable* $rotate_{r/l}(R_i, R_j)$, . . .
    - multiplication (1 register output). . .
    - *mod* (aka., division-with-remainder) . . .

What is a ***nonce?***

$\mathbb{C}_{\mathbf{m,t}}$ on **cWRAM?**

random bits → $\{ r_0 \ldots r_{k-1}, x \} \xleftarrow{\$} \mathbf{Z_p}$ — ***nonce***

$$\mathbf{H}r_0\ldots r_{k-1},x(\mathbf{v}) = \sum_{i=d}^{0} (\mathbf{s_i} \oplus \mathbf{v_i}) \cdot x^i \ (\mathrm{mod}\ p), \quad \mathbf{s_i} = \sum_{j=0}^{k-1} r_j (i+1)^j \ (\mathrm{mod}\ p)$$

$$d = |\mathbf{v}|-1$$

k-independent (almost) universal hash functions

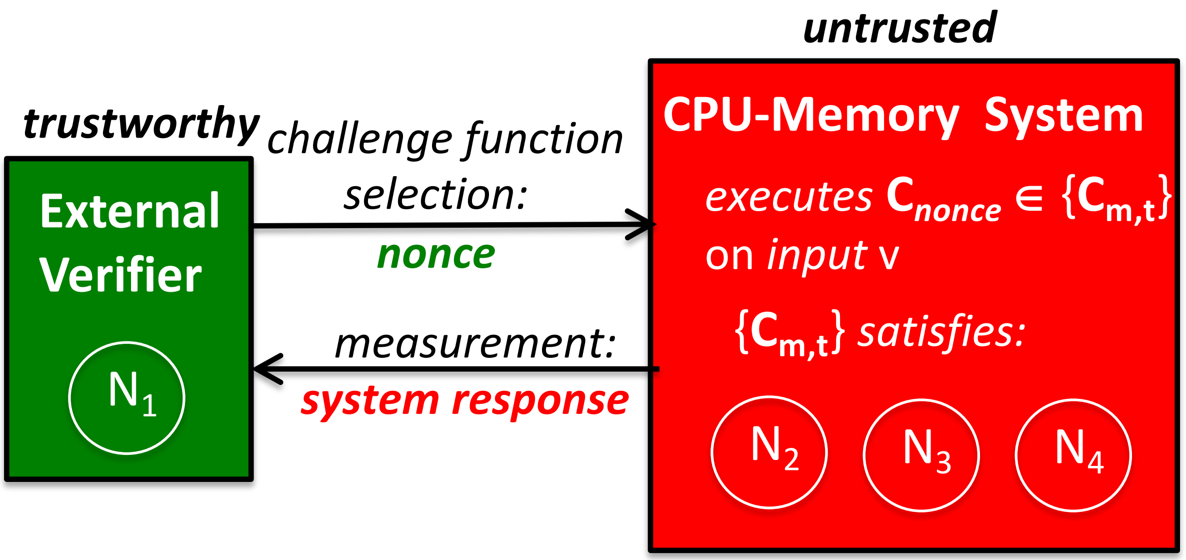
$$\mathbf{H}r_0\ldots r_{k-1},x(\mathbf{v}) = \mathbf{H}_{d,k,x}(\mathbf{v})$$

unique m-t *optimal* bounds on **cWRAM code:** $\mathbf{m} = \mathbf{k} + 22$, $\mathbf{t} = (6\mathbf{k} - 4)6\mathbf{d}$

$$\exists\ (m',t')\ \text{“<“}\ (\mathbf{m}, \mathbf{t}) \Rightarrow \mathbf{Pr}\ [nonce,\ \exists \mathbf{f,y} : \mathbf{f(y)} = \mathbf{H}_{d,k,x}(\mathbf{v})\ |\ (m',t')\ ] \le \frac{\mathbf{3}}{\mathbf{p}}$$

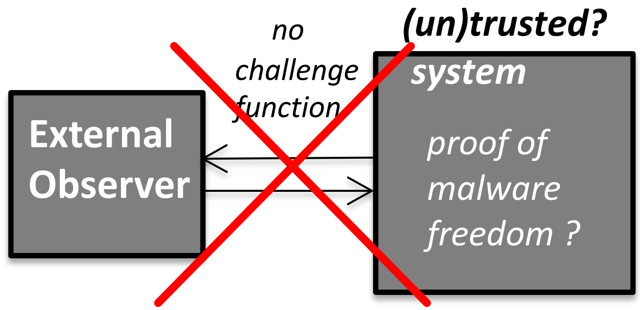
target claw free within the m-t bounds

II. What's necessary on *real systems*?



N_1: existence of external verifier & challenge function
N_2: find a concrete space-time optimal bound: (m,t)
N_3: (m,t) is unique for program code
N_4: target claw free within (m,t)

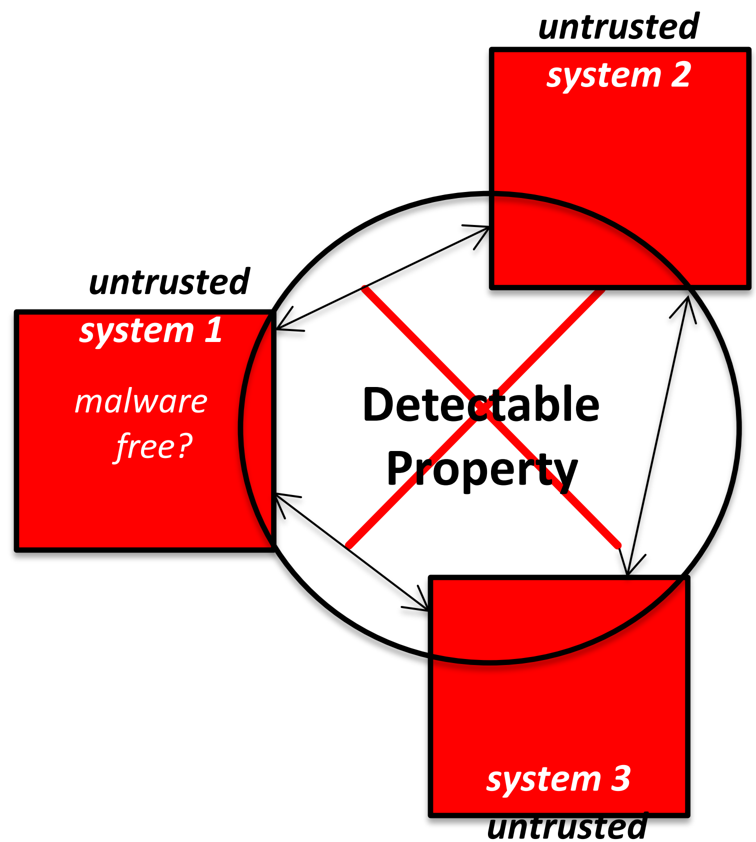
1. external verifiers
& challenge functions



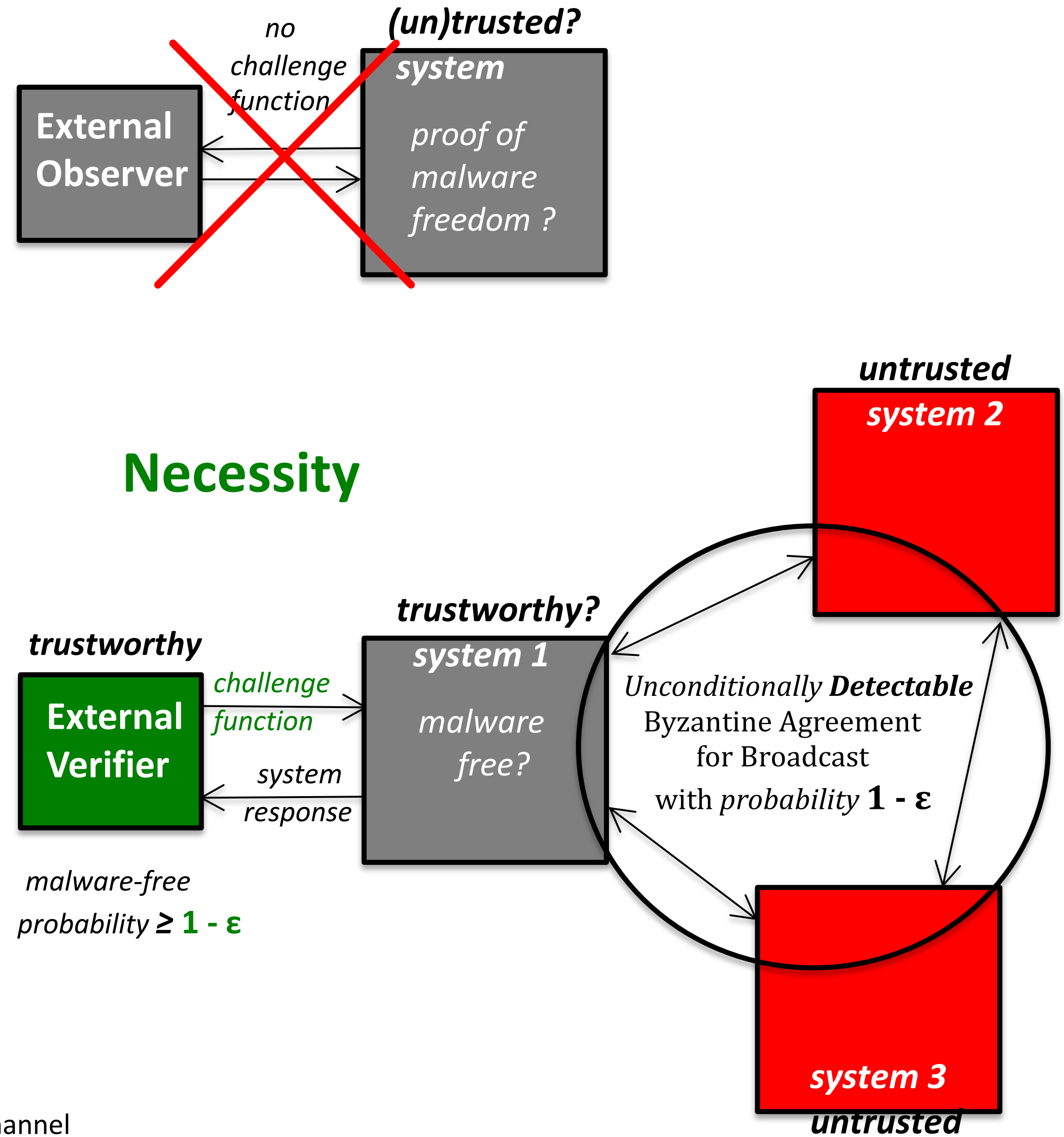
Protocols for **n Detectable Properties**

establish => **all n** systems are **trusted**

abort => **≤ n -1** systems are **untrusted**

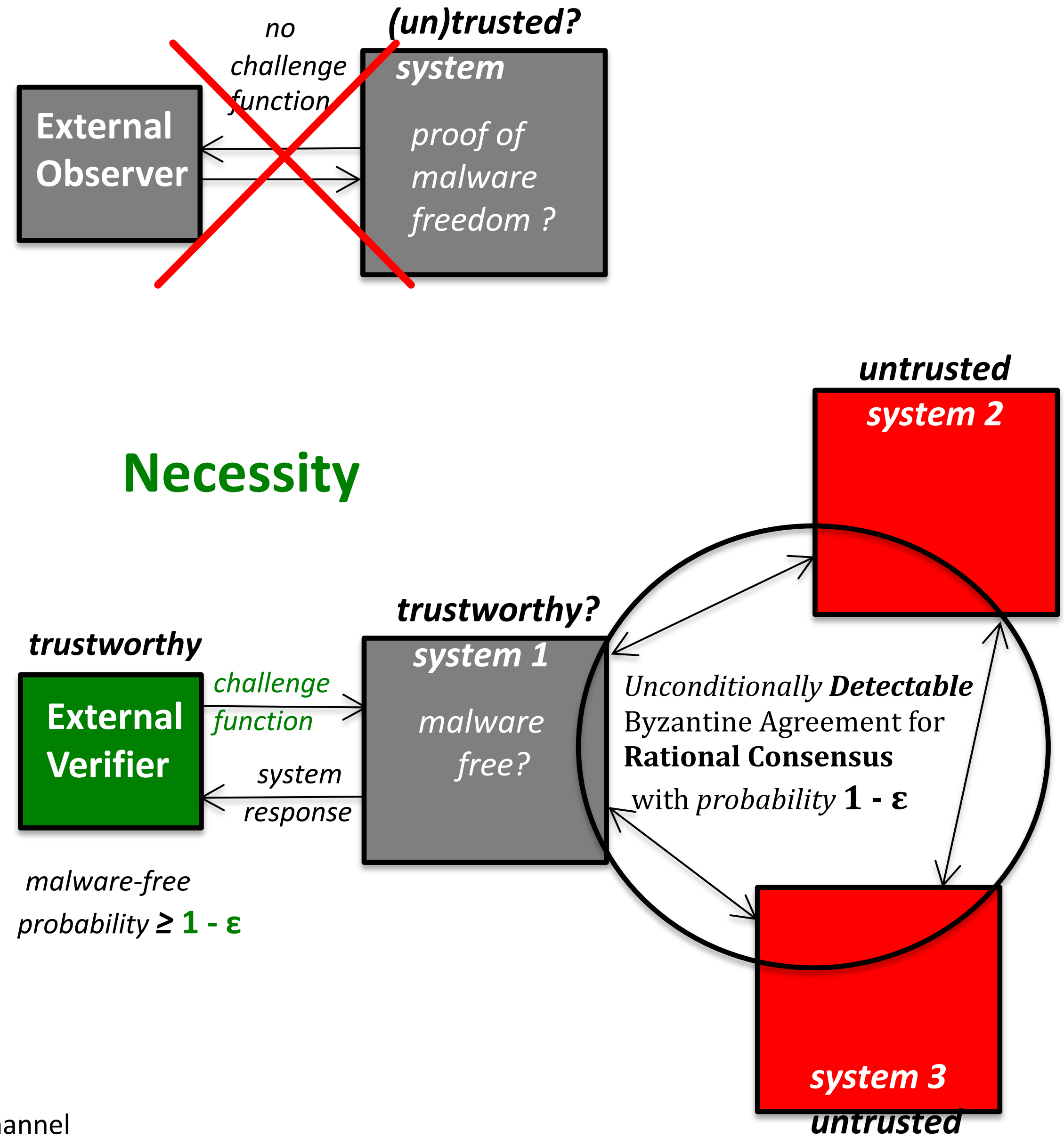


1. external verifiers
& challenge functions



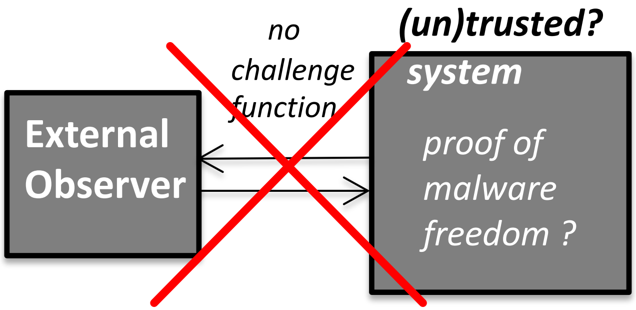
Legend: ⟷ synchronous private channel

1. external verifiers & challenge functions

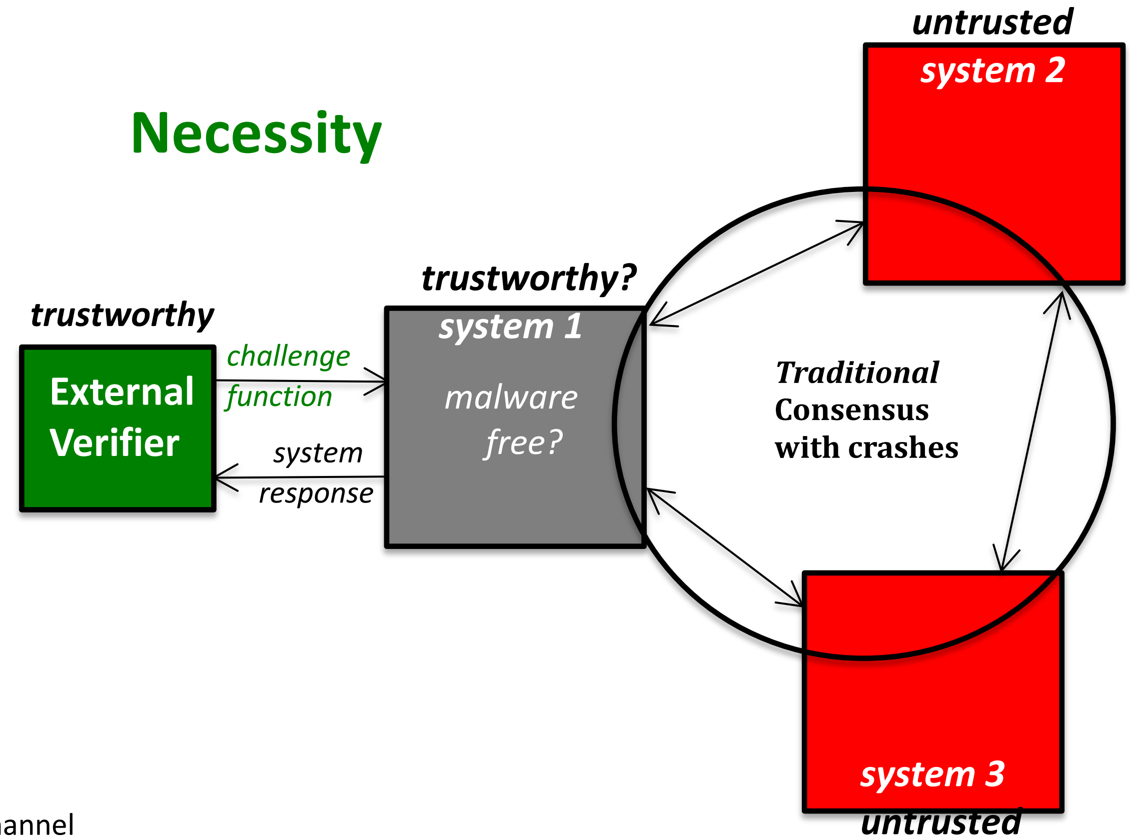


Legend: ⟷ synchronous private channel

1. external verifiers
& challenge functions

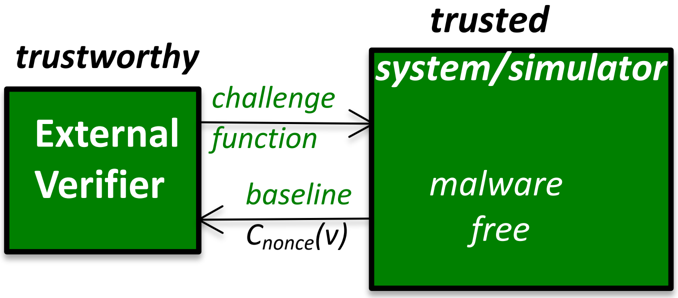


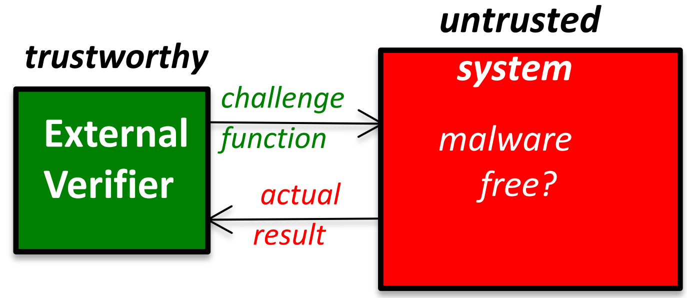
Necessity



Legend: ⟷ synchronous private channel

2. *find space-time* bounds





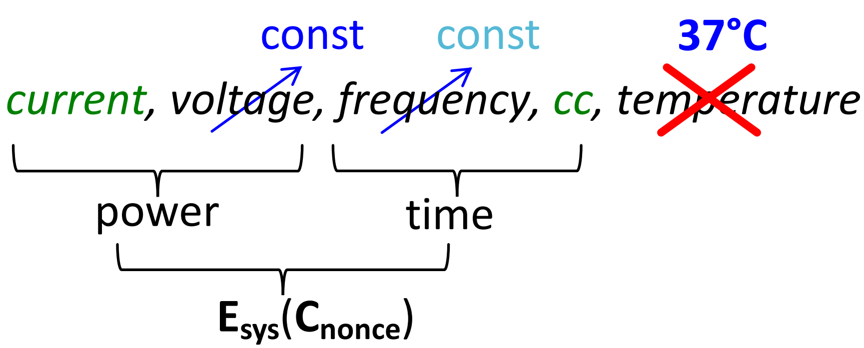
baseline measurement

= minimum amount of resources used by C_{nonce}

to prevent malware running or hiding

$C_{nonce}(v)$ = result &

baseline = actual?

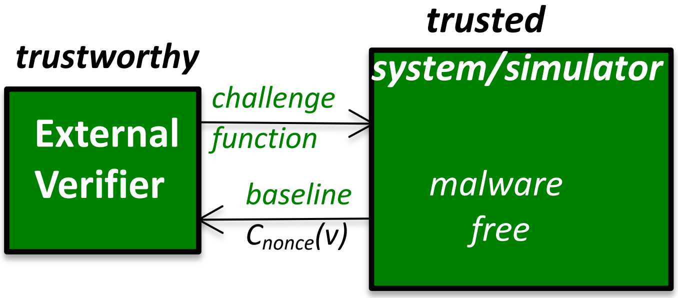


$E_{sys}(C_{nonce})$ ***measurement accuracy*** *=> a specific system initialization & choice of* C_{nonce}

min $E_{sys}(C_{nonce})$ => min. space-time bounds **=>** *lower (m,t) bounds* = ***optimal (m,t) bounds***

min $E_{sys}(C_{nonce})$ <≠ optimal (m,t) bounds

2. *find space-time* bounds



baseline measurement

min E_{sys} for single core CPUs [DeVogeleer, *et al.* 2017]

~mem size; 0; const; 0; const **ε**

$$E_{sys,i} = (P_{cpu,i} + P_{drop,i} + P_{back}) \cdot cc_i \cdot (1/(f - f_k) + \beta).$$

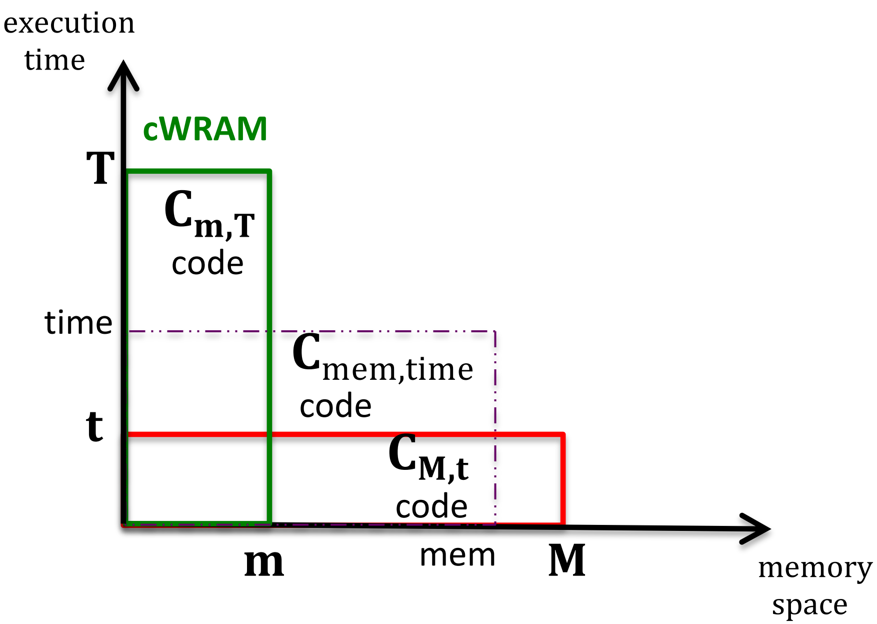
for specific **system initialization** & **choice of C_{nonce}**

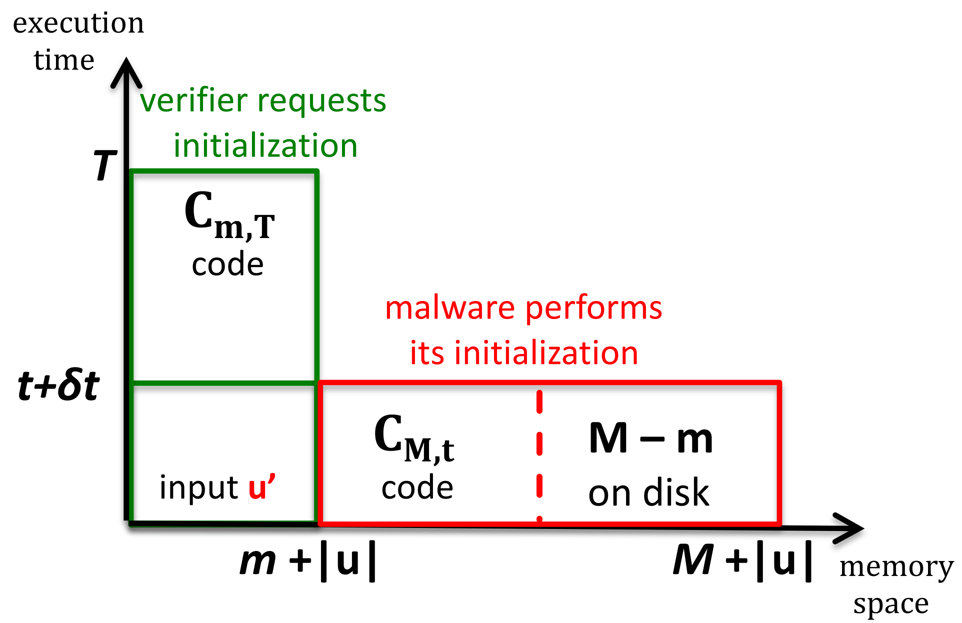
$$\mathbf{E_{sys}(C_{nonce})} = \Sigma_i E_{sys,i} = (\mathbf{P_{cpu,i}} + P_{back}) \cdot \mathbf{cc_i} \cdot \underbrace{(1/f + \boldsymbol{\varepsilon})}_{\text{const}}$$

min $E_{sys}(C_{nonce})$ => min cc_i & **min** mem size **=>** ***lower (m,t) bounds*** = ***optimal (m,t) bounds***

min $E_{sys}(C_{nonce})$ <≠ optimal (m,t) bounds of *C_{nonce}*

3. *unique m-t bounds for* $C_{m,t}$ *program code*





3 space-time optimal program families $C_{M,t}$

δt **=** time to transfer **M – m** to/from disk

$T/t > 1 + \delta$, $0 < \delta < 1$; $T/t > 3$ in practice

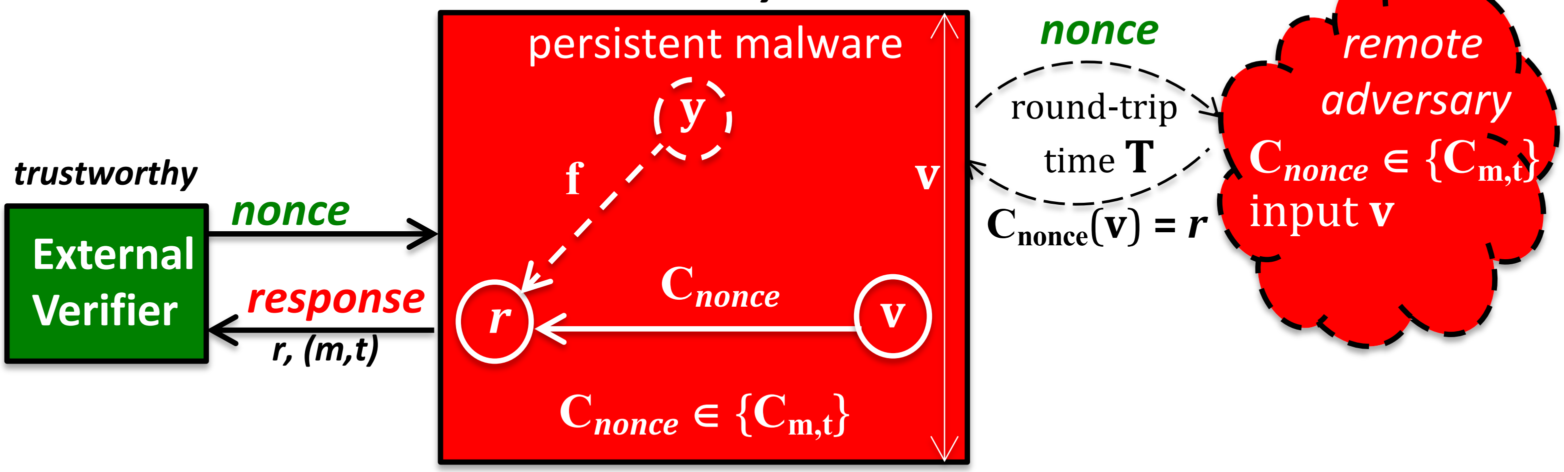
a) **single choice**: $C_{m,t}$; e.g., (**M,t**)

b) $C_{m,t}$ **= second pre-image free:** u' ≠ u => C_{nonce}(u') ≠ C_{nonce}(u), *whp*.

c) $C_{m,t}$ **code identity in (m,t)**: C_{nonce} code in **v** => C_{nonce}(**v**) is unique in (m,t), *whp*.

4. *target claw-free in (m,t)*

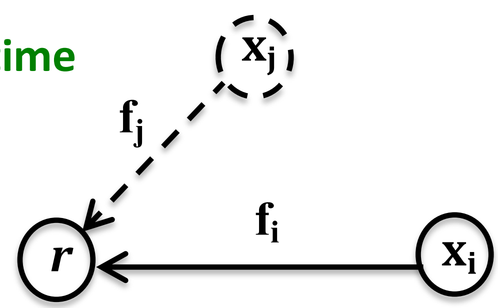
untrusted system



$f_i, f_j \in \{F\}$, **not arbitrary**

poly time

=> **hardness conjectures** and/or **secrets**



on any system

III. Q & A

1. How can we tell that the *untrusted system is initialized correctly*?
e.g., how are asynchronous events *verifiably disabled*?

2. OK, *zero false negatives* cannot exist in *RoT*… But why are they *negligible*?
Sure, the cWRAM model has *zero false positives for RoT*…
How about in *real systems*?

3. Is the energy model used *realistic*? Is there *any advantage* in using energy measurements? If so, how are the *sensors protected* from manipulation?

4. Is this paper formal enough for a productive discussion at FCS?
(Are there *any formal models* of security that *do not require* secure initial state and implicitly *persistent-malware freedom*?)