Introduction to

Physically Observable Cryptography

by Sebastian Faust, ESAT/COSIC

Outline

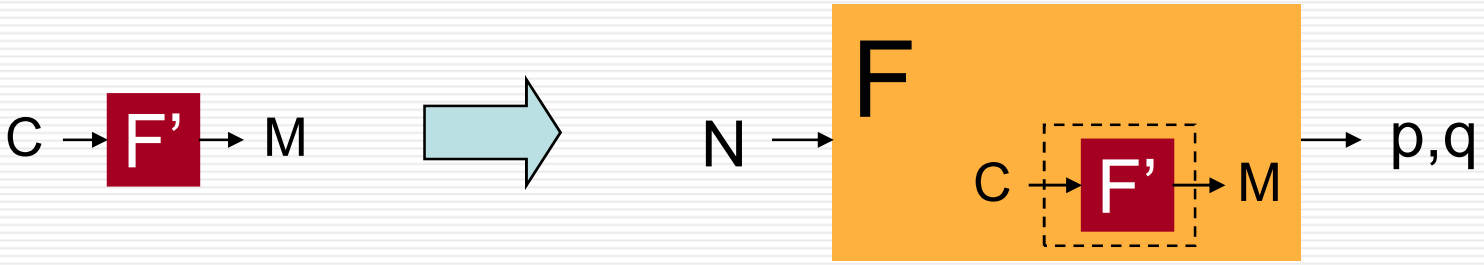
1. Gap in Provable Security
2. Approach of Micali & Reyzin
3. Axioms of Micali & Reyzin Model
4. Security Model of M&R
5. Assumptions and definitions of POC
6. A proof in the M&R Model
7. Outlook

Approach in provable security

- Develop adversarial model
- Define what is understood under the security of algorithm
- Prove that no adversary can exist under reasonable assumptions

Proof by contradiction

- Given F' against the algorithm, build F against assumption
- F uses F' ➔ F has to simulate a „real-looking“ environment for F'



Traditional Provable Security

- Cryptographic algorithms are modeled as black boxes
- Adversary may have access to inputs and outputs
- Inner workings during computation are not revealed

Gap to real-world implementations

- Practical attacks on provably secure systems exist
- Not due to failure in proof but by taking a step outside of the model
- Physical devices do not behave as black-boxes

  ➔ Side-channel attacks provide the adversary with a partial view on the inner working of implementations

Physically Observable Cryptography

1. Define axioms specifying the physical world and the therein existent physical devices
2. Define a formal security model for physical devices incorporating physical observers
3. Develop new assumptions and definitions
4. Prove security properties for more enhanced constructions against **all** observing adversaries.

Goal:

Given a physical device P computing a function f(x) s.t. only some information $L_{P,f(x)}$ is leaked, can we use it to build more complex schemes?

3. Axioms of Micali & Reyzin Model

1. Computation, and only computation leaks information
   - Unaccessed memory does not leak any information
2. Computation can have different leakages on different computers
   - Real-world implementation of an algorithm may vary
     - ➔ E.g. shielded hardware will leak different than non-shielded
3. Information leakage depends on the chosen measurement
   - Not all leakage can be observed simultanously

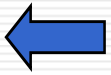
4. Information leakage is local

- The leakage of a device is independent of computation that takes place before the device is invoked or after it halts.
- No modular design would be possible if the behavior of components changes depending on the context of usage
  - ➔ E.g. if A produced a properly shielded device used in computers of company B, then B should not damage the shielding

5. All leaked information is efficiently computable from the computer's internal configuration

- Leakage is a polynomial-time computable function of algorithm's internal configuration, chosen measurement and randomness
- Implies that leakage is efficiently simulatable knowing all inputs

Security Model of Micali & Reyzin:

1. Abstract notion to model computation 
2. Physical security model & adversary

1. Abstract notion to model computation

Can we use traditional Turing machines? **No! Why?**

i. Axiom 1: unaccessed memory leaks no information

➔ device has to seperate memory that is used from memory that is not

But: traitional TM accesses tape sequentially

Solution: augment TM with random access memory

ii. Axiom 4: leakage is independent of computation that follows or precedes

➔ abstract notion has to isolate one portion of computation from another

But: Traditional notion of computation uses a single TM

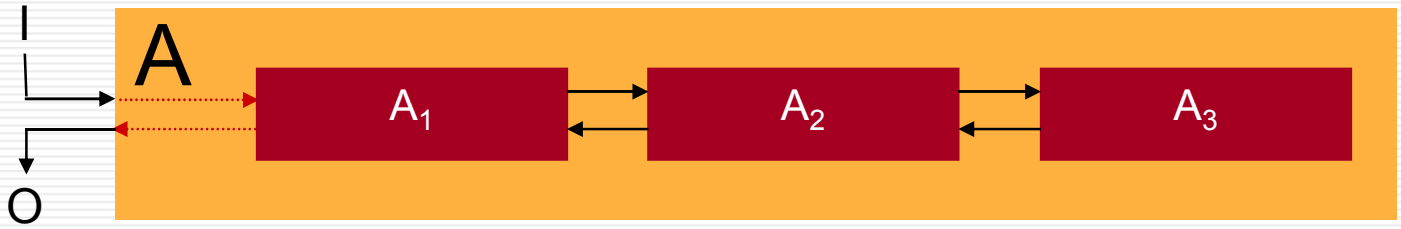
➔ internal configuration of TM incorporates all future computation

Solution: use series of TMs each with own memory space

Abstract virtual-memory computer $A=(A_1, \ldots, A_n)$

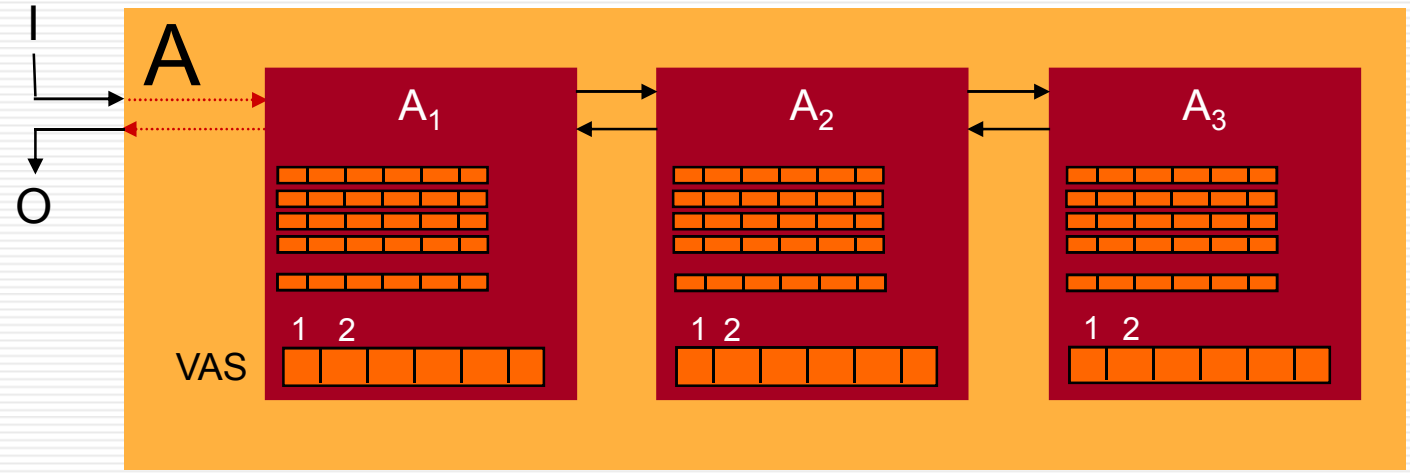
Collection of special TMs (VTM) A_i invoking each other as subroutines

Abstract virtual-memory computer $A=(A_1, \ldots, A_n)$



A_1 is invoked first and its input/output = input/output of A

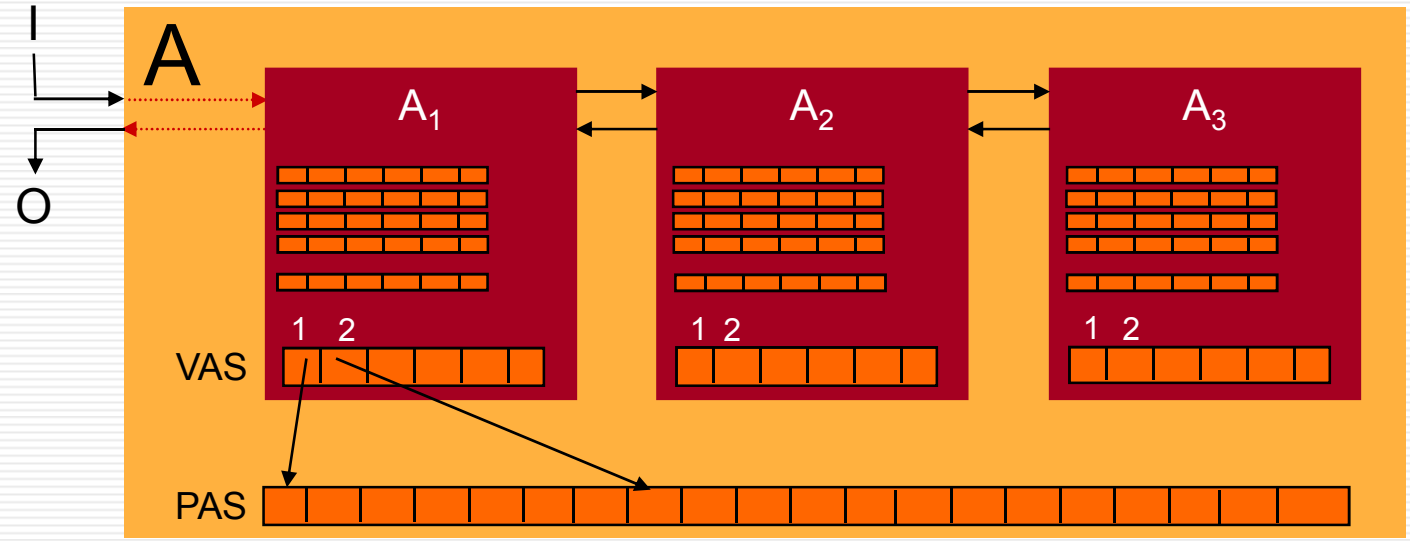
Abstract virtual-memory computer A=(A_1, …, A_n)



Each VTM A_i has access to

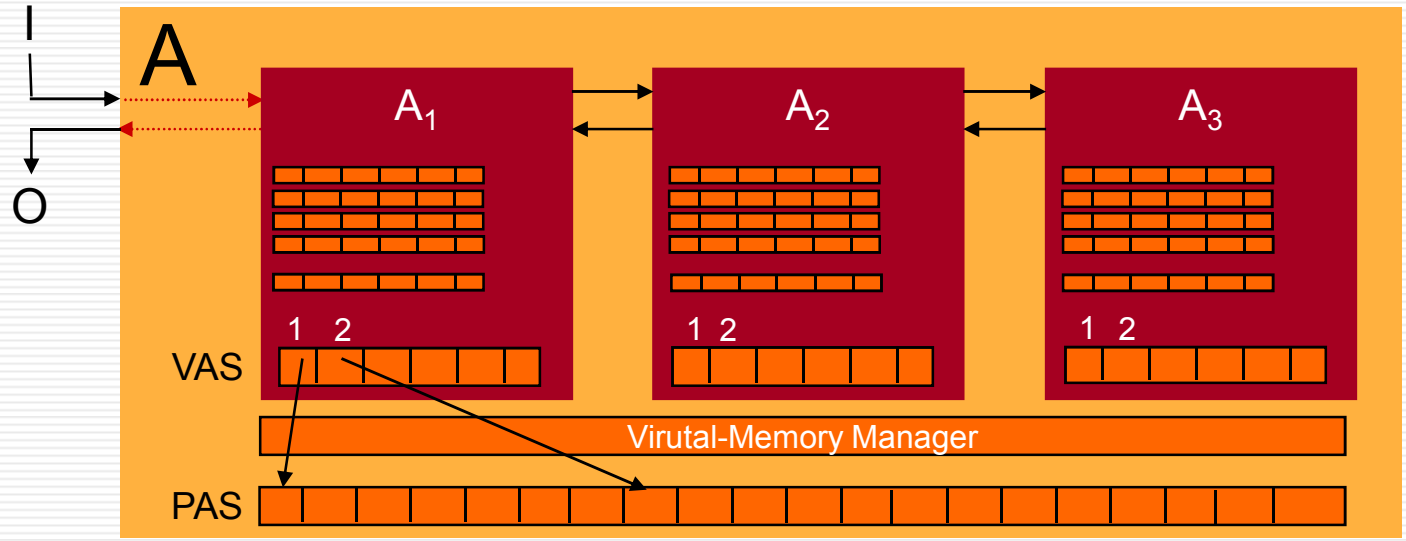
- traditional input, output, work and random tape of a TM
- random access to a *virtual address space* (VAS): unbounded array of bits that start at address 1 and goes to indefinite
  - ➔ VAS_i can only be accessed by VTM_i
- VAS-access tape to access VAS

Abstract virtual-memory computer A=(A_1, …, A_n)



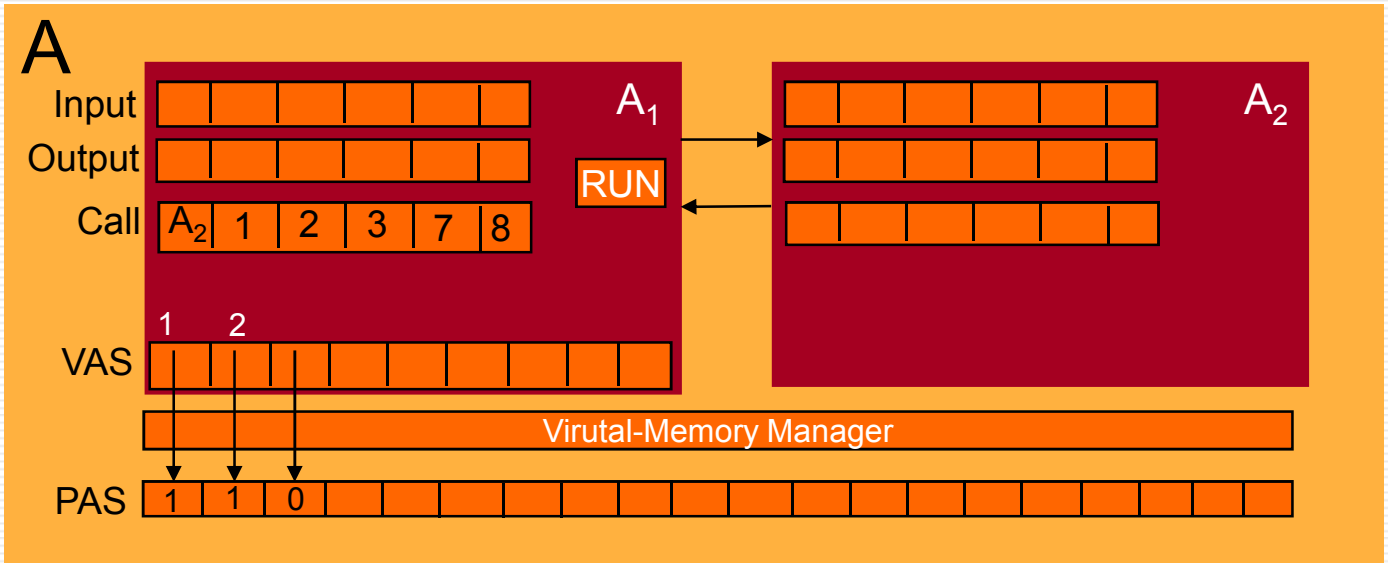
- VTM A_i only „thinks“ that it has own individual VAS ➔ reality: all share a single *physical address space* (PAS). **Why?**
  - Parameter passing
  - Axiom 1: Access real content as little as possible

Abstract virtual-memory computer $A=(A_1, \ldots, A_n)$



- Virtual-Memory Manager (VMM) maps individual VASes to unique PAS:
  - never accesses content of memory, only remapping of addresses
  - VMM allows for parameter passing among different VTMs
  - Generates new VAS initialized with 0 when VTM is invoked

Calling VTMs as subroutines:



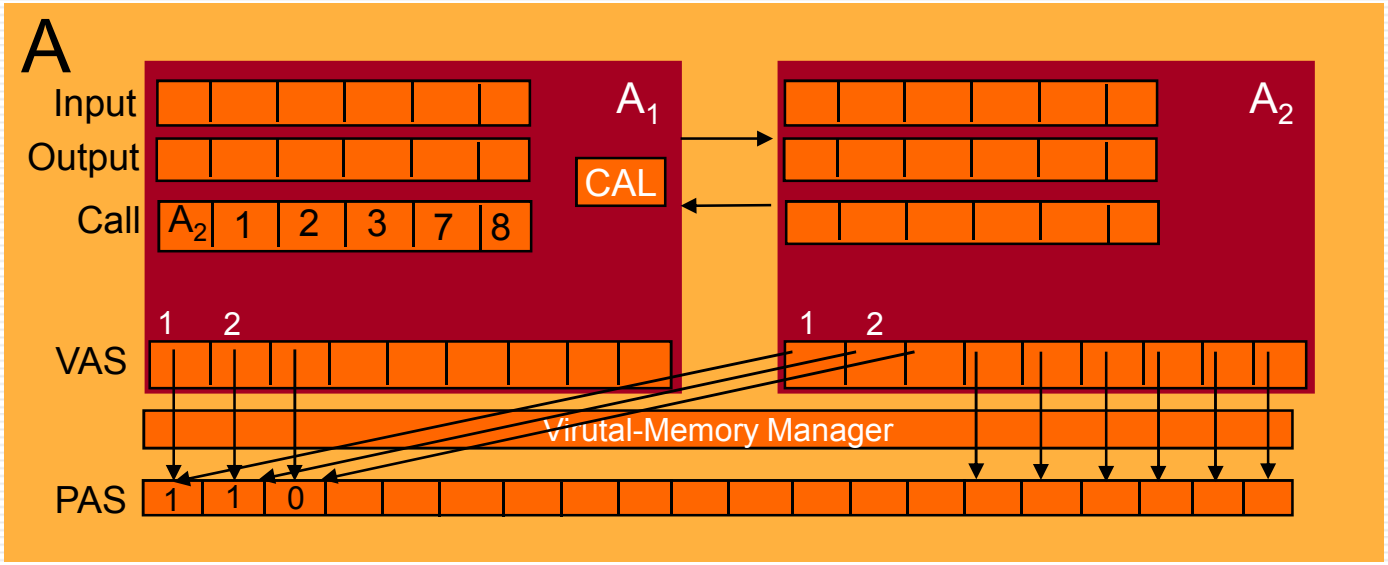
Example:

A_1 calls A_2 on input 110 (I=3) and A_2 has ouput 10 (L=2)

A_1 writes down on it subroutine call tape:

- name of A_2
- a sequence of I addresses $a_1,\ldots, a_I$ in its VAS for the input of A_2
- a sequence of L addresses $b_1,\ldots, b_L$ in its VAS to store the output of A_2

Calling VTMs as subroutines:



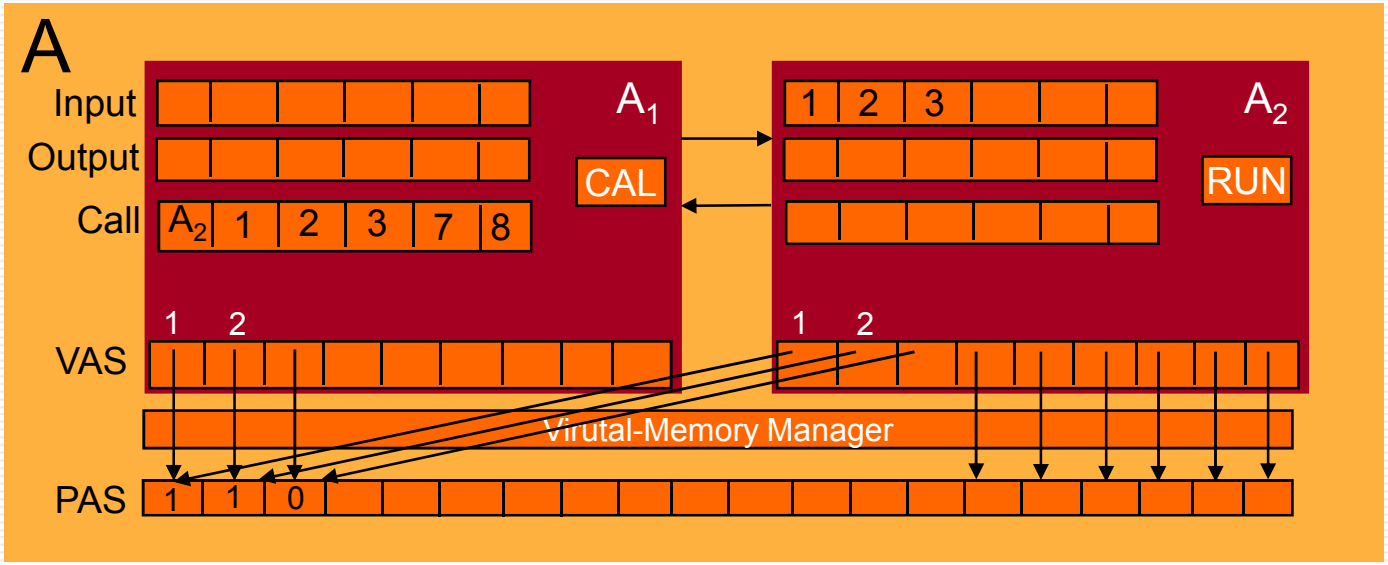
Example:

A_1 calls A_2 on input 110 (l=3) and A_2 has ouput 10 (L=2)

A_1 goes in CAL (call) state and suspends its computation

- VMM creates new VAS for A_2 and ensures that:
  - Maps first l VAS location of A_2 to the same PAS location as a_i in VAS of A_1
  - all other locations in the VAS of A_2 map to blank PAS locations

Calling VTMs as subroutines:



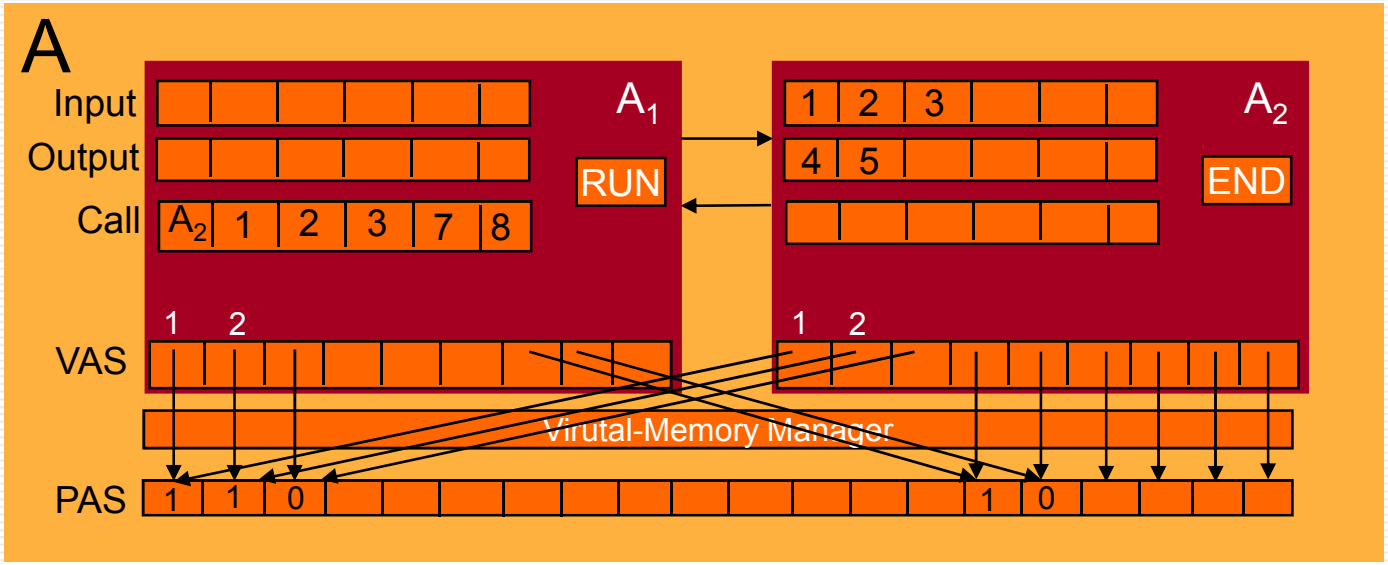
Example:

A_1 calls A_2 on input 110 (I=3) and A_2 has ouput 10 (L=2)

A_2 enters RUN state with

- input tape contains location where to find input for A_2
- other tapes point to blank

Calling VTMs as subroutines:



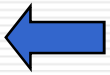
Example:

A_1 calls A_2 on input 110 (l=3) and A_2 has ouput 10 (L=2)

When A_2 ends (END):

- output tape of A_2 contains addresses of its VAS mapping to the output in PAS
- VMM remaps location $b_1,\ldots, b_L$ of VAS from A_1 to the same PAS location where A_2 has stored ist output.
- A_1 resumes operation

Security Model of Micali & Reyzin:

1. Abstract notion to model computation
2. Physical security model and adversary 

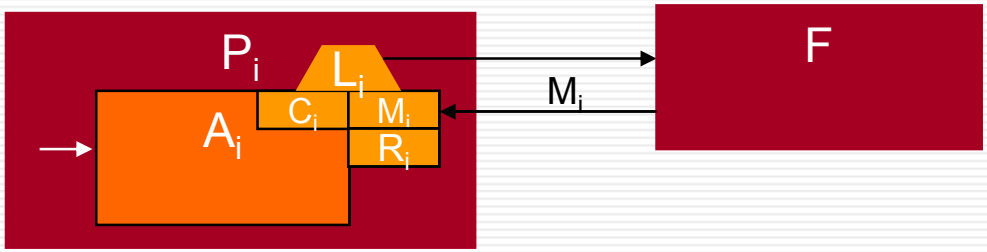
2. Physical security model

- *Goal:* Incorporate the leakage of an implementation
- *Problem:* Abstract virtual-memory computer may have different physical implementations ➔ different leakage
- *Idea:* Augement abstract VTM with a leakage function to cover all possible leakages

➔ Abstract VTM A + Leakage function L = physical VTM P=(L,A)

- P runs computation specified by abstract VTM A and incorporates leakage L.

Physical VTM $P_i = (L_i, A_i)$:



- L_i: leakage funtion with three inputs L(C,M,R):
  1. current internal configuration C: binary string that includes
     - information of the touched elements of all tapes of A_i
     - locations of all heads of A
     - current state of A_i
  2. setting/specification of the measuring system M
  3. random noise R of the measurement

Physical virtual-memory computer $P = (P_1, \ldots, P_n)$:

- Combination of physical VTMs P_i
- If $A = (A_1, \ldots, A_n)$ is abstract computer then P is a physical implementation of A
- Notation $f_P(x)$: function computed by P on input x

Adversary F:

- F can observe the computation of every single physical VTM P_i
- F can adaptively specify a measurement M for each step of the computation

Definition 1:

A physical VTM is *trivial* if its leakage function reveals its entire configuration and *non-trivial* otherwise.

Fundamental Assumption of POC:

There exists a non-trivial physical VTM.

Definition (physical world):

A polynomial-time deterministic physical computer P is a *PO one-way function* if for any polynomial-time adversary F, the following probability is negligible in k:

$$Pr[x \leftarrow^R \{0,1\}^k; y \leftarrow P(x) \hookrightarrow F(1^k) \rightarrow state; z \leftarrow F(state, y) : f_P(z) = y]$$

P is a *PO one-way permutation*: f_P is length-preserving + bijective

Intuition: P is a PO one-way function if it computes a function f_P that is hard to invert despite the leakage from P's computation.

Notation: $y_P \leftarrow P(x_P) \hookrightarrow F(x_F) \rightarrow y_F$

- F runs on input x_F and observs a physical computer P on input x_P
- P halts with output y_P ➔ F halts with output y_F

Claim 1:

If P is a PO one-way permutation then P' with $f_{P'}(\cdot) = f_P(f_P(\cdot))$ is also a PO one-way permutation.

Proof (Idea):

1. Construction of P'=(P_0,P):
   - Note that P is PO owp and P_0 is a trivial VTM (➔ leaks everything!)
   - P_0 calls P twice on different inputs and manages the parameter passing. In more details we have:
     1. P_0 prepares tapes for subroutine call to y_1 = P (x)
     2. P computes $y_1 = f_P(x)$
     3. P_0 prepares tapes for subroutine call P (y_1)
     4. P computes $y_2 = f_P(y_1)$
     5. P_0 places the address of y_2 on the output tape

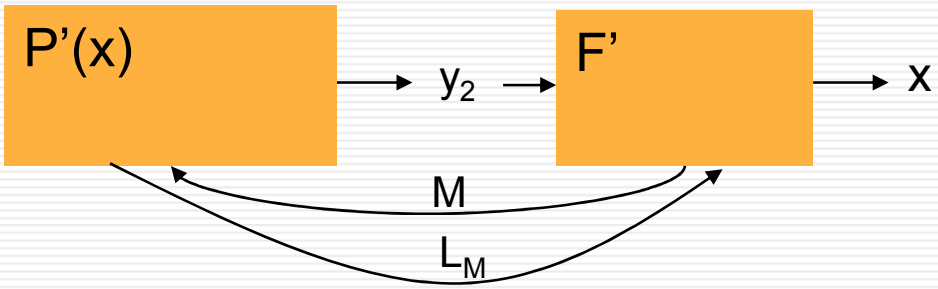
2. Prove one-wayness of P'

Show that given $y_2 = f_P(f_P(x))$ computed by P', it is difficult to find x despite the leakage of P'

Reduction (1):

Assume the existence of an adversary F' that

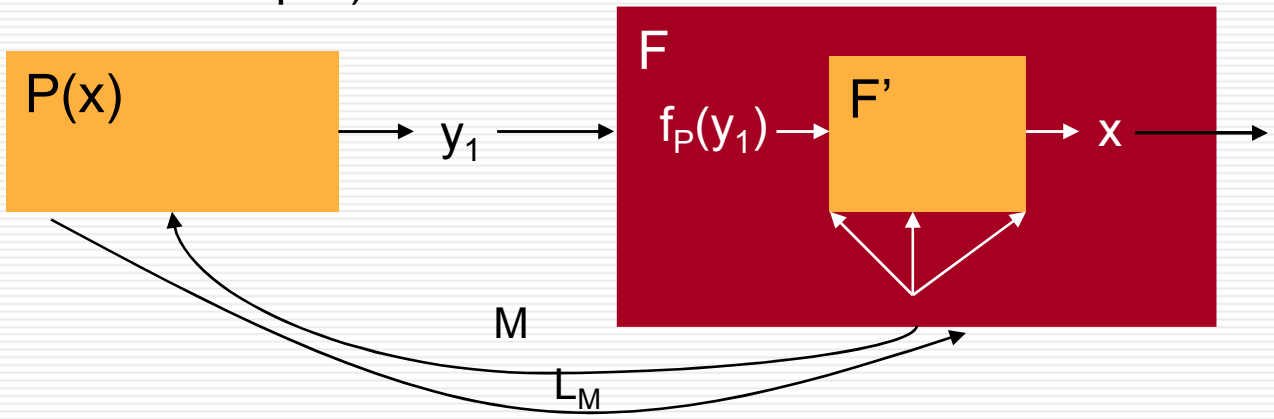
- observes P' ➔ provides measurement M and obtains leakage L_M
- gets y_2 as input
- outputs x (i.e. inverts P')



Reduction (2):

Build adversary F that attacks the assumption (i.e. one-wayness of P):

- Observes target PO one-way permutation P
- Gets $y_1 = f_P(x)$ as input
- Uses F‘ as subroutine and simulates environment by
  - producing a consistent input $y_2 = f_P(y_1)$
  - producing indistinguishable answers to measurements of F’
- outputs x (i.e. inverts owp P)



1. P_0 prepares tapes for subroutine call to $y_1 = P(x)$
   - F answers measurement queries with entire configuration of P_0
     - content of call and input tape
   - Why is this indistinguishable?
     - P_0 is trivial ➔ leaks everything in real world as well
   - Why can we compute the leakage?
     - P_0 only reassigns VAS pointers but no access to content of memory
     - Axiom 1: Unaccessed memory does not leak
       ➔ Content of P_0's VAS (in particular x) is not part of leakage because it is only remapped by VMM
     - Leakage of P_0 contains only addresses

2. P computes $y_1 = f_P(x)$

- F starts observing P(x)
- If F‘ chooses measurement M then F will use M as his measurement while running P(x)
- F forwards results of his own measurement to F’
- Why is this indistinguishable?
  - Axiom 4: Leakage is local
    - ➔ P(x) run in „isolation“ has the same leakage distribution as P(x) introduced by P_0

3. P_0 prepares tapes for subroutine call to $P(y_1)$

Simulation is done as in stage 1

4. P computes $y_2 = f_P(y_1)$

- F runs abstract computer $A(y_1)$
- For measurement M of F‘ return leakage L(C,M,R), where C is configuration of A.
- Why is this indistinguishable?
  - Axiom 4: Leakage is local
    - ➔ Leakage of running P from scratch with input y_1 is the same as the leakage running P after y_1 is computed
- Simulation efficient?
  - Axiom 5: Leakage is efficiently computable knowing all inputs for L
    - ➔ F was not observing the real $P(y_1)$, but knows all inputs for L

5. P_0 places the address of y_2 on the output tape

- F answers measurement queries with entire configuration of P_0
- Why is this indistinguishable?
  - P_0 is trivial ➔ leaks everything
- Why is the simulation efficient?
  - Axiom 1: unaccessed memory does not leak

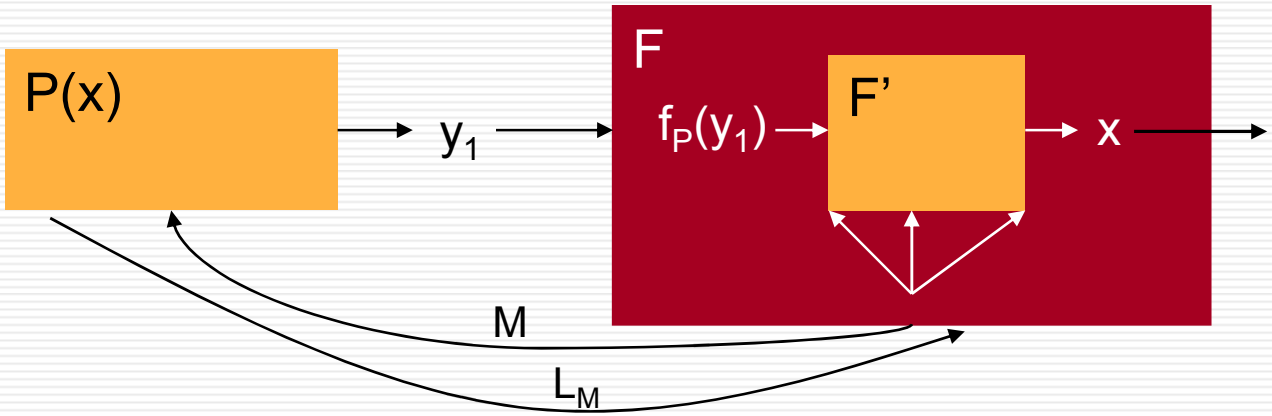
    ➔ content of P_0's VAS (in particular x) is not part of the leakage

Finally…

- F computes $y_2 = f_P(y_1)$ and gives it to F‘

  ➔ F‘ answers with x

- F outputs x



Showing existence of further cryptographic primitives:

- Unpredictable PO generator from PO one-way permutation
- Digital Signature scheme from PO one-way permutation
- Pseudorandom Function from multiple observable PRNG

Analyze PO security of real-world algorithms and figure out necessary assumptions to prove security:

- RSA FDH Signatures in PO Random Oracle Model
- RSA CPA Encryption in PO Random Oracle Model
- OAEP Encryption, PSS Signatures,...

Questions?

Remarks:

- Axiom 2: Different implementations have different leakage
  - Trivial machines leaking everything certainly exist
  - Using them to compute f(x) from x would make it easy to find an inverse
    - ➔ If leakage would be the same for all implementations, a PO one-way permutations would not exist
- Axiom 3: Information leakage depends on the chosen measurement

  Incorporated into the model: F‘ has the power of choosing its own measurements at every step of the computation

Overview

1. Gap between real-world implementations and provable security
2. Physically Observable Cryptography
    1. Define axioms specifying the physical world and the therein existent physical devices
    2. Define a formal security model for physical devices incorporating physical observers
    3. Assumption and definitions
    4. Proof

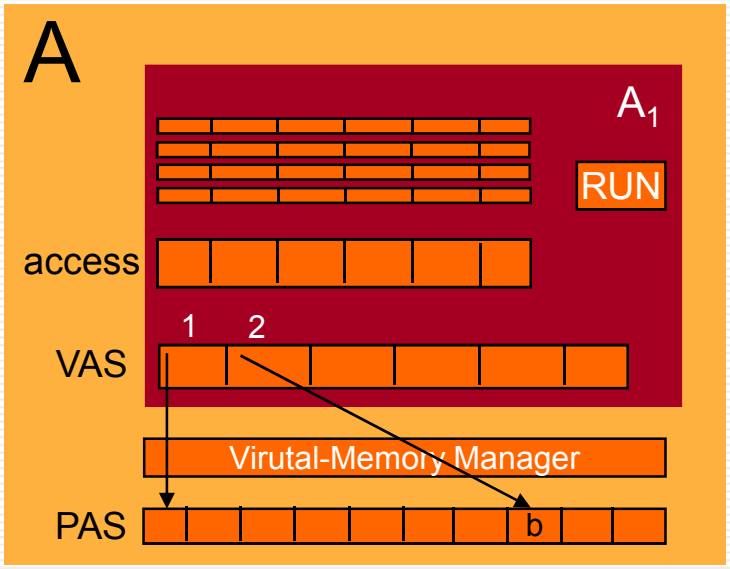
Implications of the model:

- Implementations of cryptographic concstructions are build from physical VTMs each having an own leakage function L.
- F has access to L and by querying for M obtains L(M,C,R).
- Parameter passing between VTMs can be done without leakage managed by trivial VTM.

F‘ expects to observe a 5 stage computation:

1. P_0 prepares tapes for subroutine call to $y_1 = P_1(x)$
2. P_1 and its subroutines compute $y_1 = f_P(x)$
3. P_0 prepares tapes for subroutine call $P_1(y_1)$
4. P_1 and its subroutines compute $y_2 = f_P(y_1)$
5. P_0 places the address of y_2 on the output tape

Accessing Virtual-Memory



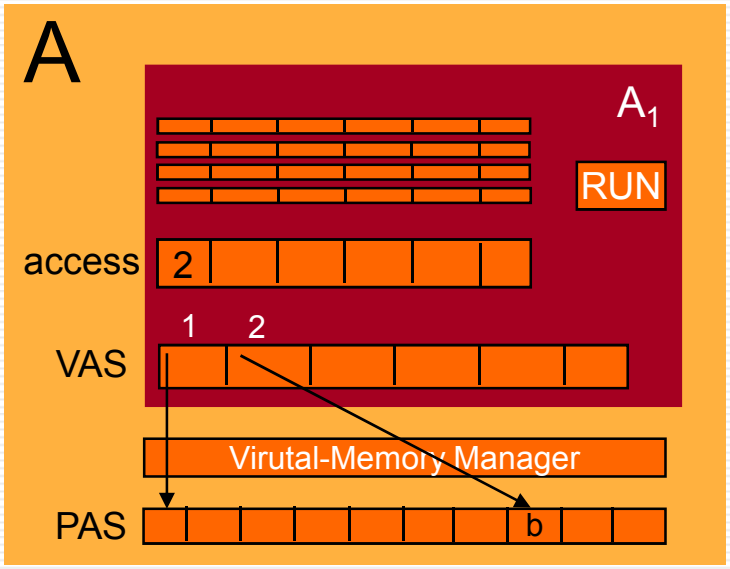
Notation:

m_{A_i} : content of A_i's VAS

$m_{A_i}[j]$: bit value stored at location j

Read the bit $m_{A_i}[2]$:

Accessing Virtual-Memory



Notation:

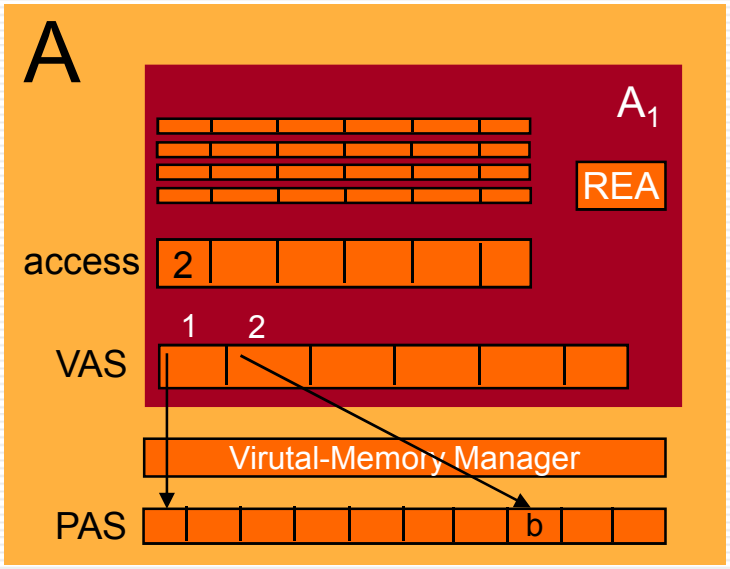
m_{A_i} : content of A_i 's VAS

$m_{A_i}[j]$: bit value stored at location j

Read the bit $m_{A_i}[2]$:

- Write location on VAS-access tape

Accessing Virtual-Memory



Notation:

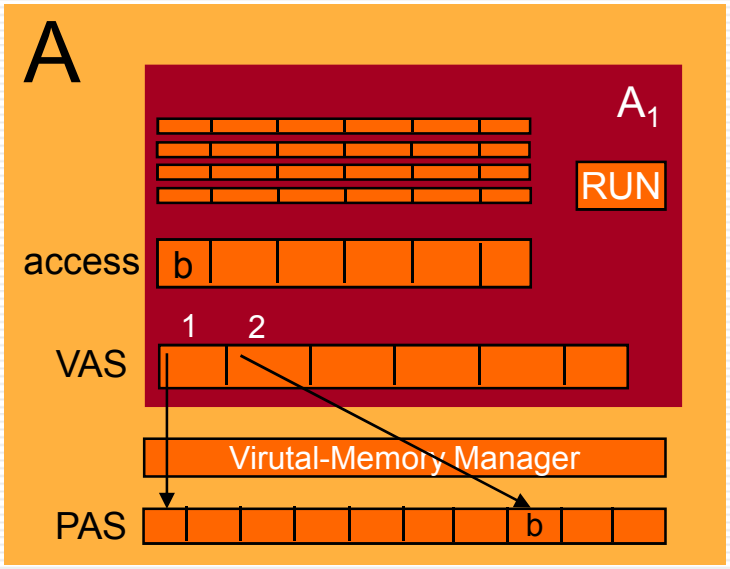
m_{A_i} : content of A_i 's VAS

$m_{A_i}[j]$: bit value stored at location j

Read the bit $m_{A_i}[2]$:

- Write location on VAS-access tape
- Enter special state (REA = read)

Accessing Virtual-Memory



Notation:

m_{A_i} : content of A_i 's VAS

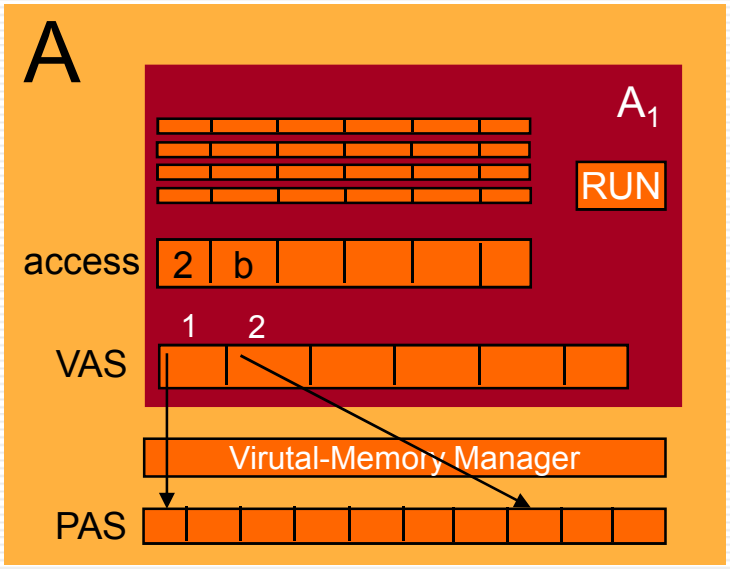
$m_{A_i}[j]$: bit value stored at location j

Read the bit $m_{A_i}[2]$:

- Write location on VAS-access tape
- Enter special state (REA = read)

➔ $m_{A_i}[2]$ appears on VAS-access tape

Accessing Virtual-Memory



Notation:

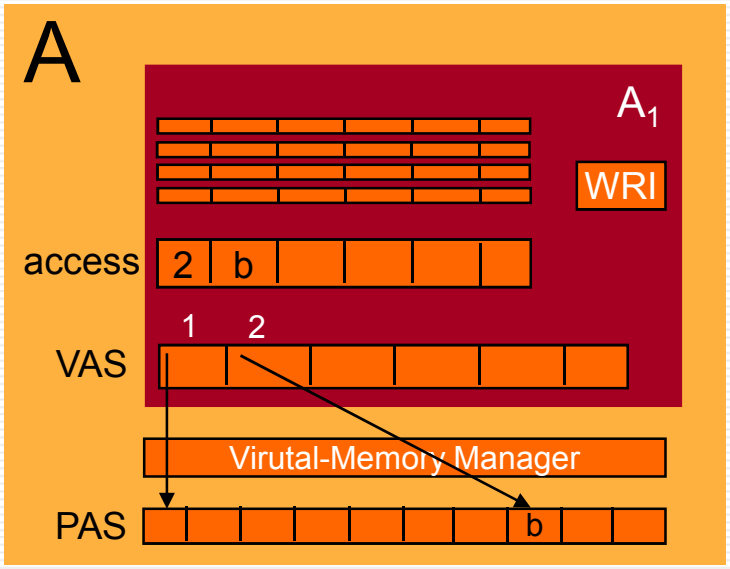
m_{A_i} : content of A_i 's VAS

$m_{A_i}[j]$: bit value stored at location j

Write bit b to position 2 in VAS:

- Write (2,b) on VAS-access tape

Accessing Virtual-Memory



Notation:

m_{A_i} : content of A_i 's VAS

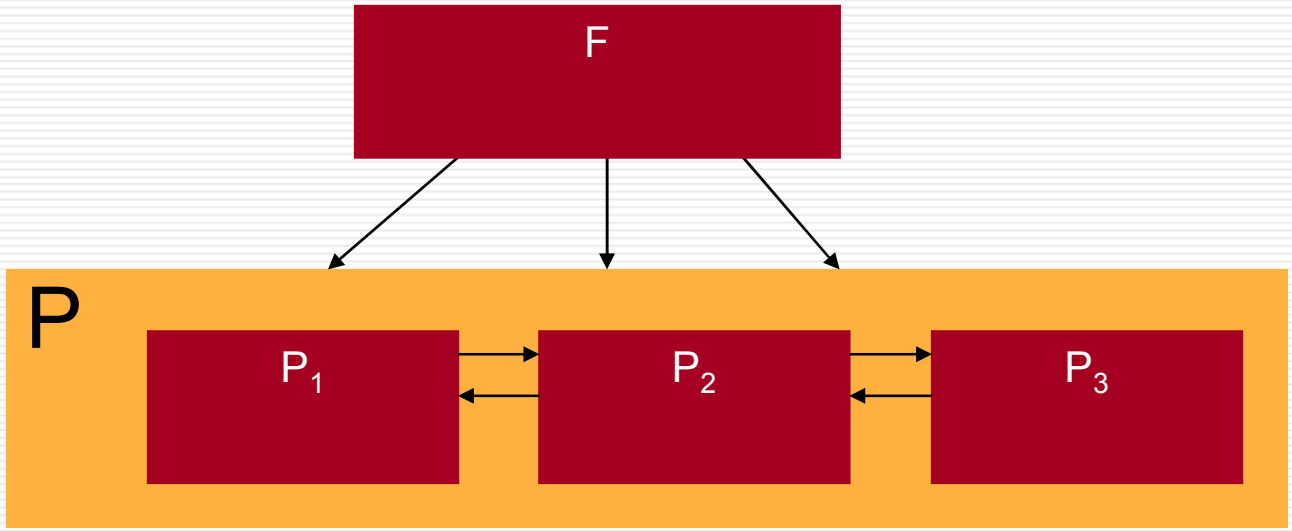
$m_{A_i}[j]$: bit value stored at location j

Write bit b to position 2 in VAS:

- Write (2,b) on VAS-access tape
- Enters special state (WRI = write)

  ➔ $m_{A_i}[2] = b$

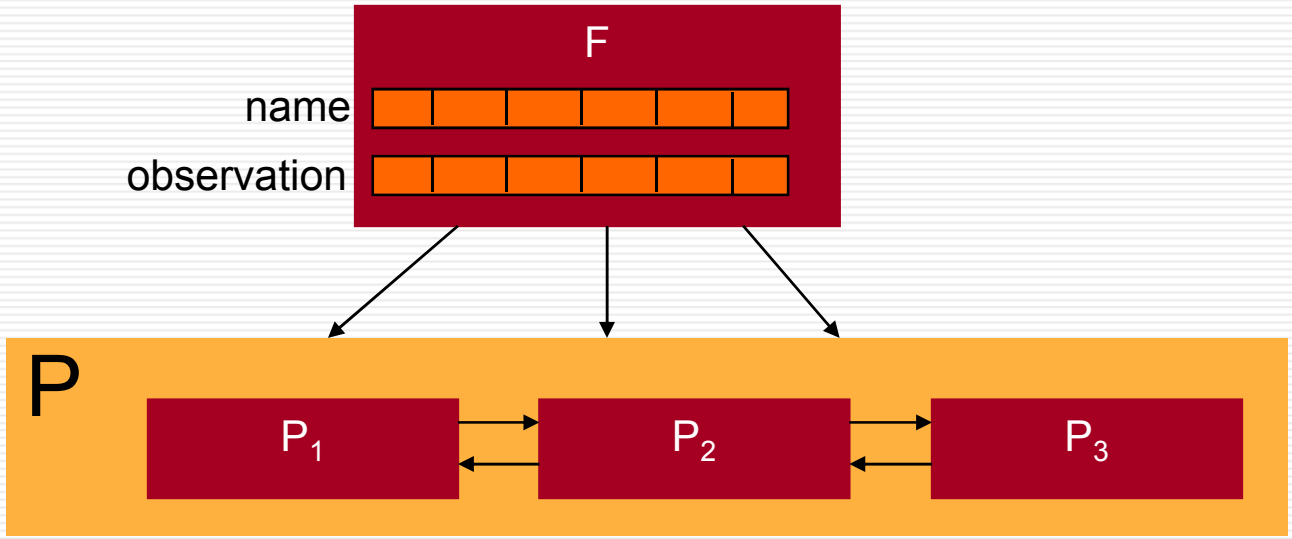
3. Adversary F:



F has an additional ability to *observe* computation of a physical computer P

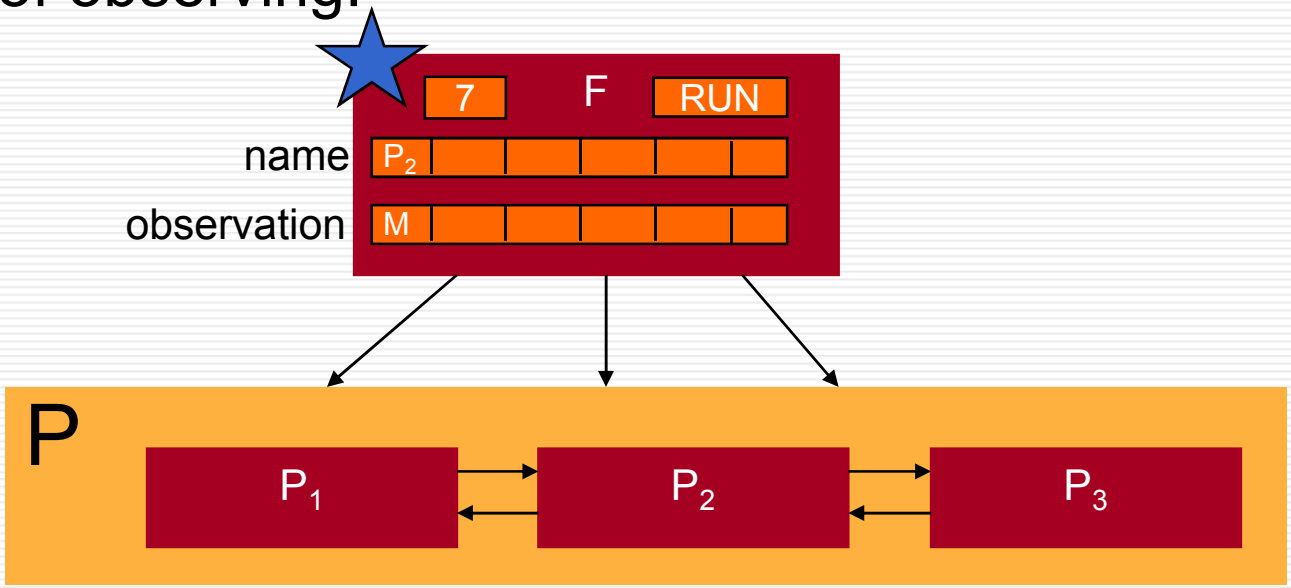
➔ F gets leakage function L

Process of observing:

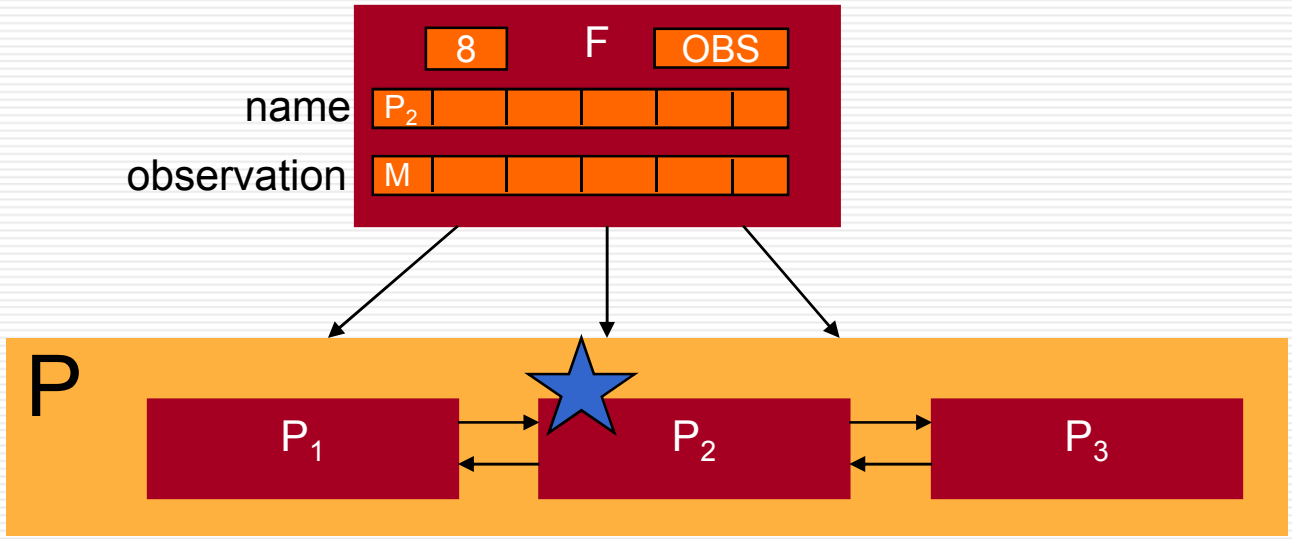


- Read-only name tape: contains name of the currently active VTM
- Observation tape: F writes down M ➔ leakage L will appear
- F is invoked before each step of a physical VTM of P
- F keeps configuration between invocations

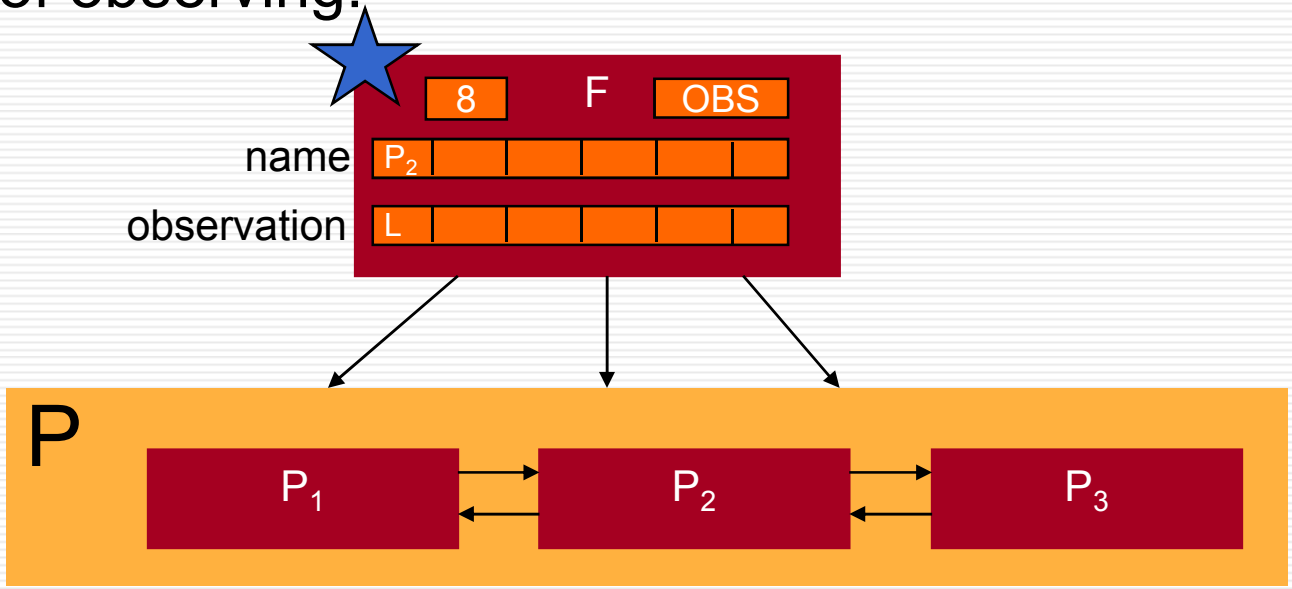
Process of observing:



Process of observing:



Process of observing:



- Repeated for each single step of the physical VTMs of P
- If P halts: F is invoked again with name tape containing 0

Definition 2 (traditional world):

A one-way function is a function f: {0,1}* → {0,1}* **such that there exists** a polynomial-time Turing machine T that computes f and, for any polynomial-time adversary F, the following probability is negligible in k:

$$Pr[x \leftarrow^R \{0,1\}^k; y \leftarrow T(x); z \leftarrow F(1^k, y) : f(z) = y]$$

Definition attempt (physical world):

A physically observable (PO) one-way function is a function f: {0,1}* → {0,1}* **such that there exists** a polynomial-time physical computer P that computes f and, for any polynomial-time adversary F, the following probability is negligible in k:

$$Pr[x \leftarrow^R \{0,1\}^k; y \leftarrow P(x) \hookrightarrow F(1^k) \rightarrow state; z \leftarrow F(state, y) : f(z) = y]$$

Problem?

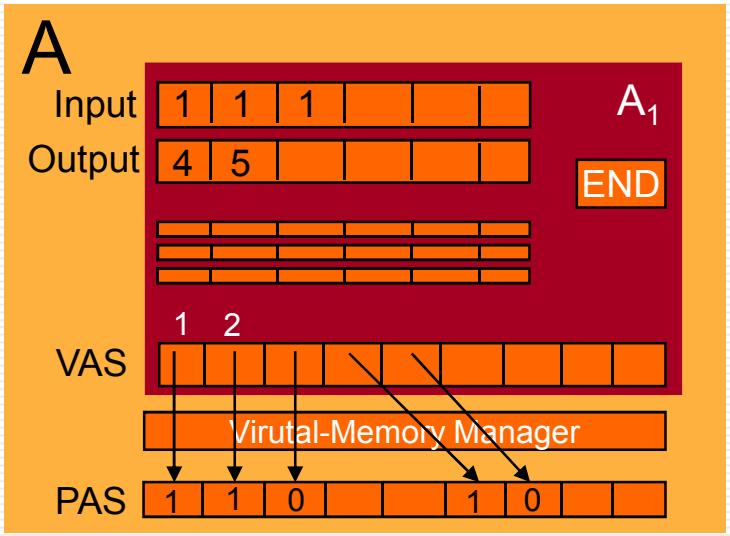
Problem:

- Definitions should not rely on any assumption
- Definition attempt relies on the fundamental assumption of the existence of a non-trivial physical computer

Solution:

Define not what it means for a function f to be one-way but for a particular physical computer P computing f.

Inputs and outputs of VTM:



Example:

Input = 110 ➔ I = 3

Output = 10 ➔ L = 2

Inputs and outputs of VTM are binary strings always residing in memory:

- Input tape contains 1^I, the unary representation of the input length
- Input in the first I bit positions of A_1's VAS
- End of computation:
  - Output tape: sequence of L addresses: $b_1, \ldots, b_L$
  - Output itself in VAS: $o = m_{A_1}[b_1] \ldots m_{A_1}[b_L]$