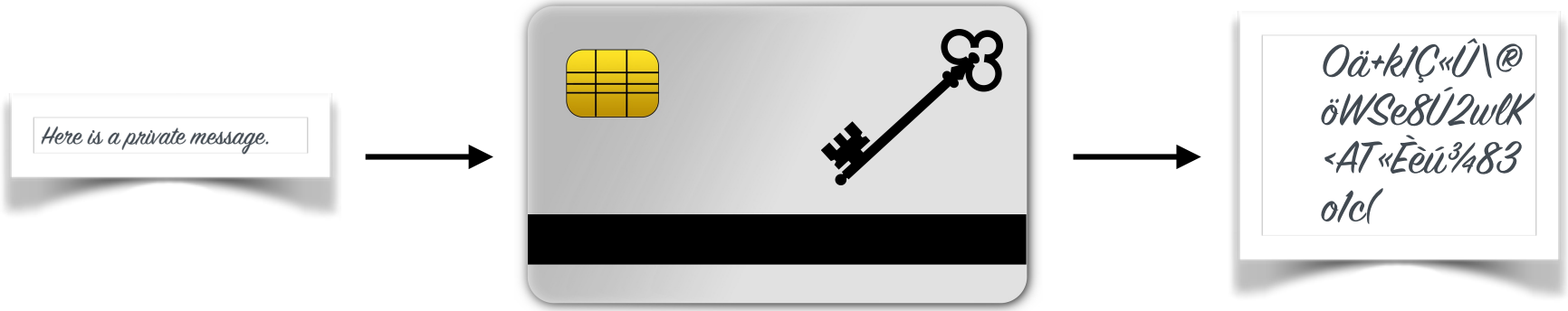
Security of Masked Implementations

Sonia Belaïd

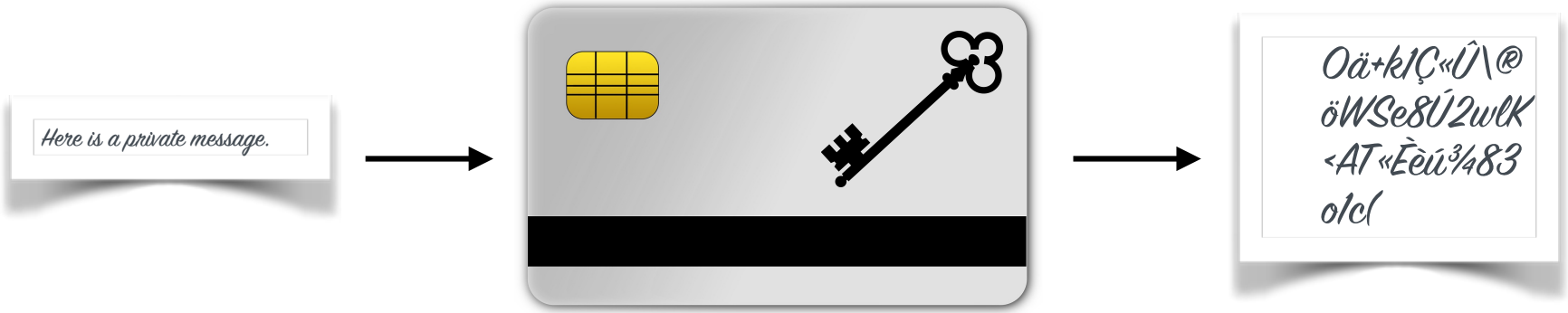
LatinCrypt 2019

Side-Channel Attacks

Side-Channel Attacks

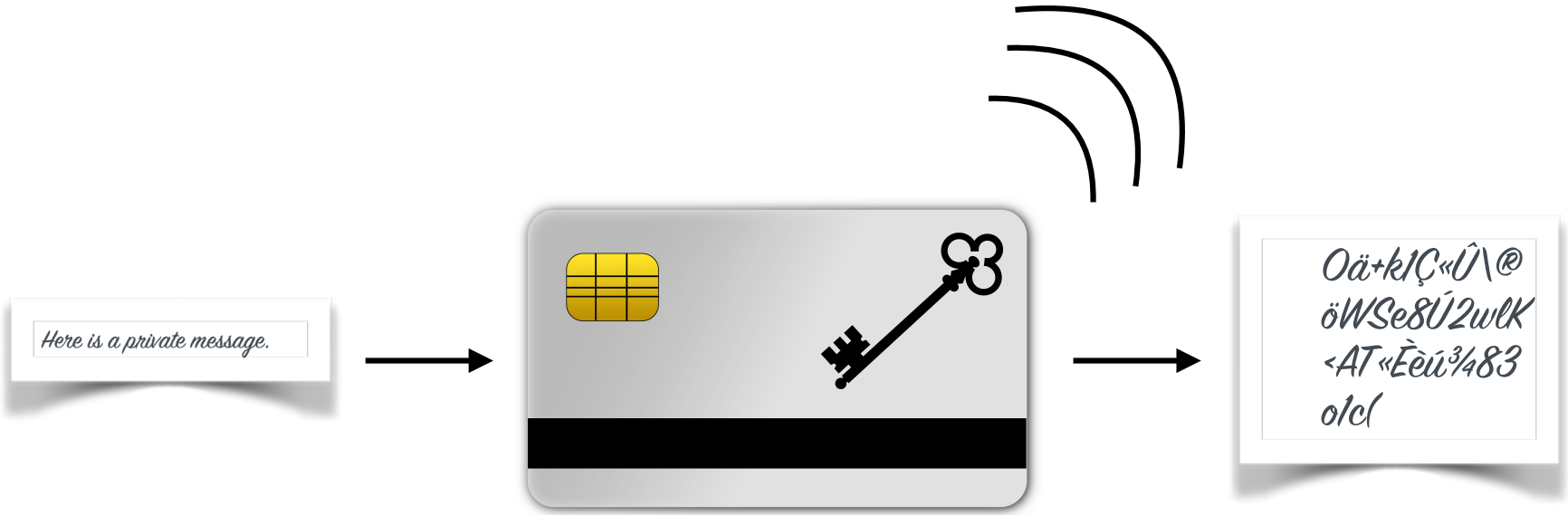


Side-Channel Attacks



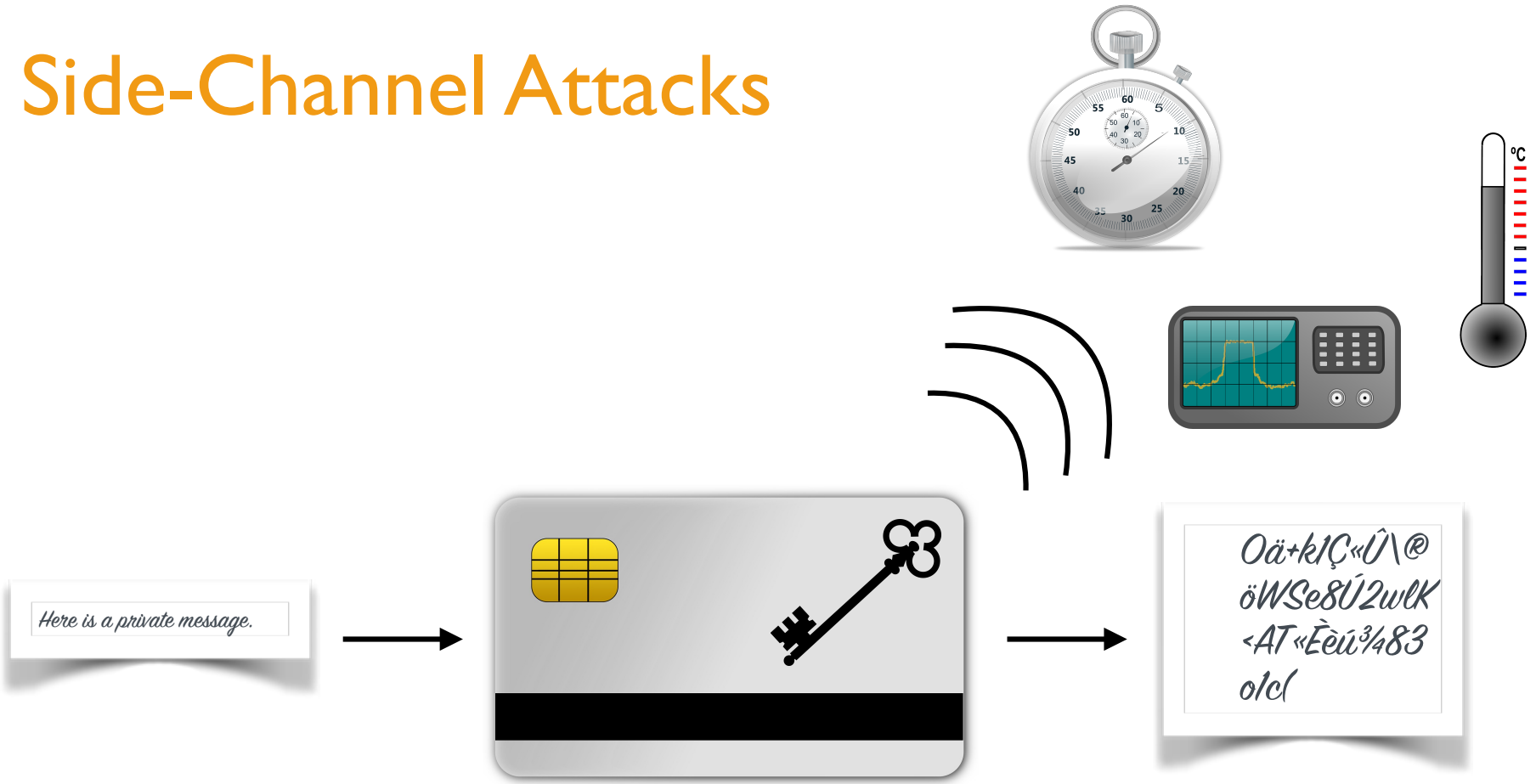
- Black-box cryptanalysis: $\mathcal{A} \leftarrow (m, c)$

Side-Channel Attacks



- Black-box cryptanalysis: $\mathscr{A} \leftarrow (m, c)$
- Side-channel analysis: $\mathscr{A} \leftarrow (m, c, \mathscr{L})$

Side-Channel Attacks



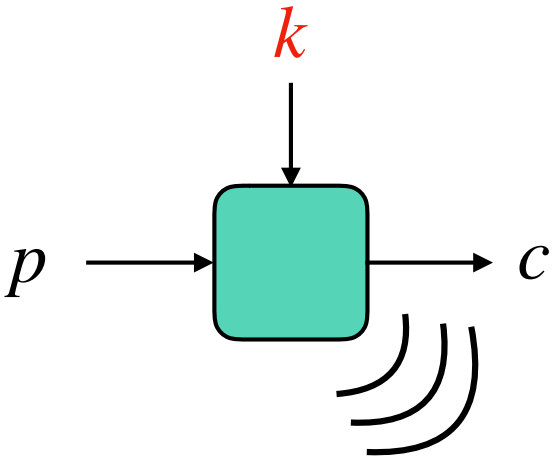
- Black-box cryptanalysis: $\mathscr{A} \leftarrow (m, c)$
- Side-channel analysis: $\mathscr{A} \leftarrow (m, c, \mathscr{L})$

Overview of this talk

- Masking Countermeasure
  - Definition and implementation
- Leakage Models
  - Definitions, pros, and cons
- Verification of Small Implementations
  - Example of tools to verify small implementations
- Composition
  - How to compose small implementations into larger secure ones

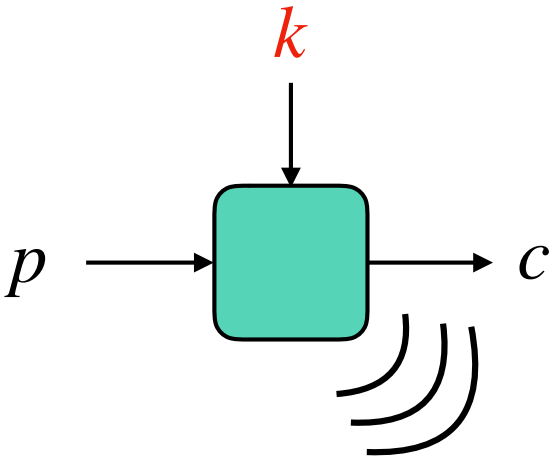
Masking Countermeasure

How to thwart SCA?



Problem: the leakage is key-dependent

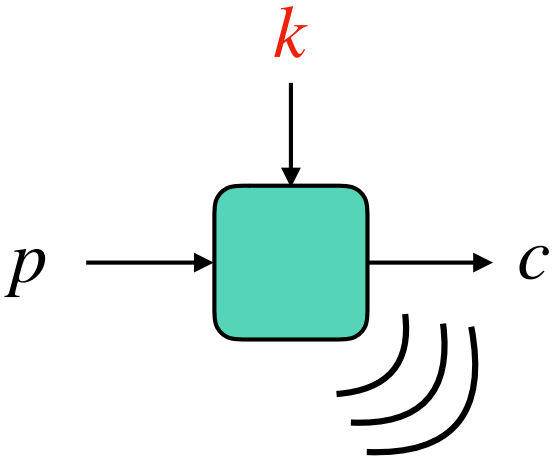
How to thwart SCA?



Problem: the leakage is key-dependent

Solution: Masking (make the leakage random)

How to thwart SCA?

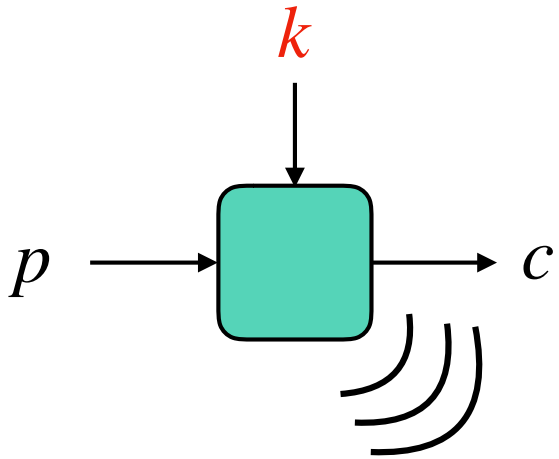


Problem: the leakage is key-dependent

Solution: Masking (make the leakage random)

for each sensitive value $v \leftarrow f(p, k)$

How to thwart SCA?



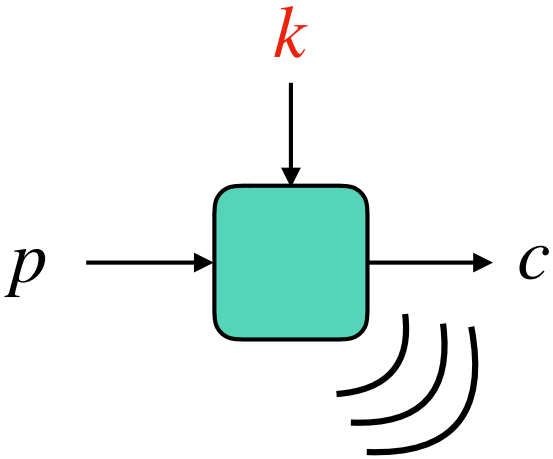
Problem: the leakage is key-dependent

Solution: Masking (make the leakage random)

for each sensitive value $v \leftarrow f(p, k)$

$$v_1 \leftarrow \$ \qquad v_2 \leftarrow \$ \qquad \cdots \qquad v_{n-1} \leftarrow \$$$

How to thwart SCA?



Problem: the leakage is key-dependent

Solution: Masking (make the leakage random)

for each sensitive value $v \leftarrow f(p,k)$

$$v_0 \leftarrow v \oplus \left(\bigoplus_{i=1}^{n-1} v_i \right) \qquad v_1 \leftarrow \$ \qquad v_2 \leftarrow \$ \qquad \cdots \qquad v_{n-1} \leftarrow \$$$

Masking in Practice

- Masking linear operations

$$z \leftarrow x \oplus y$$

$$x = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$$

$$y = y_0 \oplus y_1 \oplus \ldots \oplus y_{n-1}$$

Masking in Practice

- Masking linear operations

$$z \leftarrow x \oplus y$$

$$x = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$$

$$y = y_0 \oplus y_1 \oplus \ldots \oplus y_{n-1}$$

$$\mathbf{z} = (x_0 \oplus y_0, x_1 \oplus y_1, \ldots, x_{n-1} \oplus y_{n-1})$$

Masking in Practice

- Masking linear operations

$$z \leftarrow x \oplus y$$

$$x = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$$

$$y = y_0 \oplus y_1 \oplus \ldots \oplus y_{n-1}$$

$$\mathbf{z} = (x_0 \oplus y_0, x_1 \oplus y_1, \ldots, x_{n-1} \oplus y_{n-1})$$

- Masking non linear operations
  - Cannot be done share by share

for $i = 0$ to t

  for $j = i + 1$ to t

    $r_{i,j} \leftarrow \$$

    $r_{j,i} \leftarrow (r_{i,j} \oplus x_i y_j) \oplus x_j y_i$

for $i = 0$ to t

  $z_i \leftarrow x_i y_i$

  for $j = 0$ to t, $j \neq i$

    $z_i \leftarrow z_i \oplus r_{i,j}$

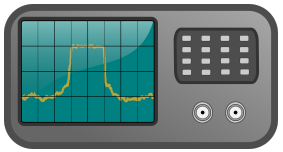
Leakage Models

Security of an implementation

- How to evaluate the security of an implementation?

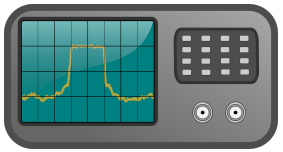
Security of an implementation

- How to evaluate the security of an implementation?
  - Integrate it on a device and try to attack it
    - Not always possible

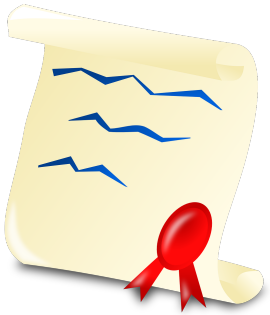


Security of an implementation

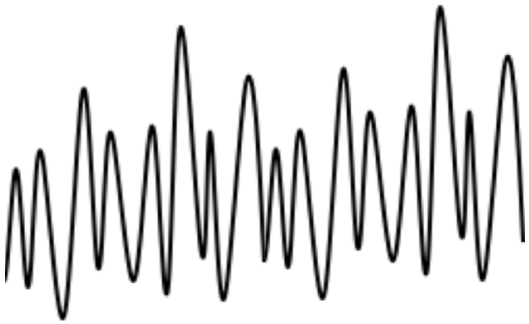
- How to evaluate the security of an implementation?
  - Integrate it on a device and try to attack it
    - Not always possible



  - Model the leakage and prove its security or exhibit an attack



Noisy Leakage Model

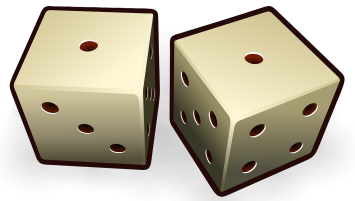


- Leakage
  - Every variable leaks
  - Leakage = noisy function of the value

S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power- analysis attacks. CRYPTO'99

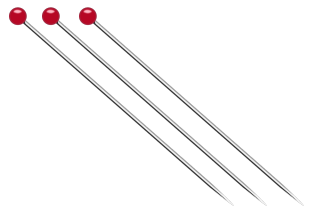
E. Prouff and M. Rivain. Masking against side-channel attacks: A formal security proof. EUROCRYPT 2013

Random Probing Model



- Leakage
  - Every variable leaks with probability p
  - Leakage = exact value

Probing Model



- Leakage
  - Only *t* variables leak in the implementation
  - Leakage = exact value

- Security in the *t*-probing model
  - Implementation such that any set of *t* intermediate variables is independent from the secret

$\underline{\mathsf{ExpReal}(\mathcal{A}, C)}$:

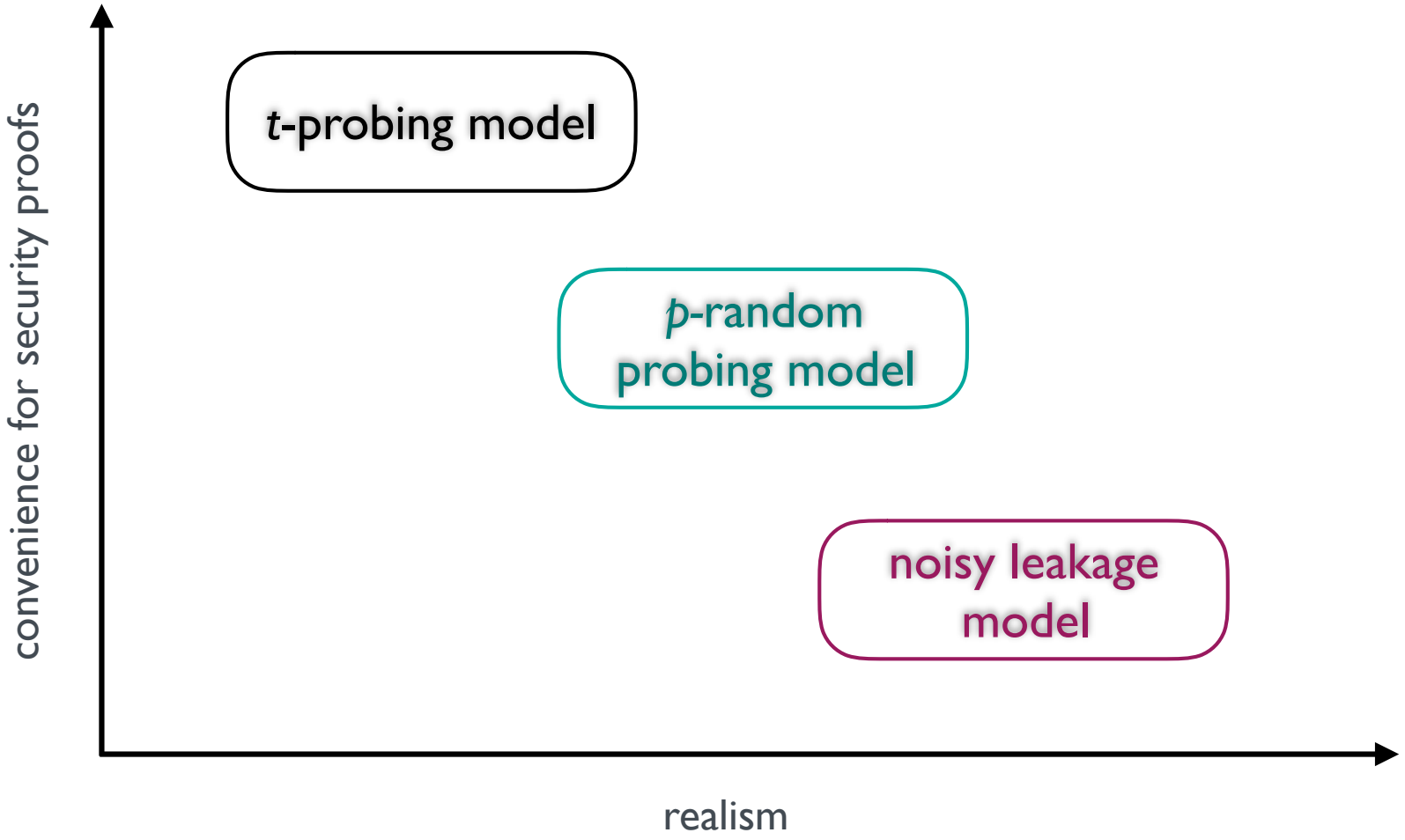
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_n] \leftarrow \mathsf{Enc}(x_n)$
3. $(v_1, \ldots, v_t) \leftarrow C([x_1], \ldots, [x_n])_{\mathcal{P}}$
4. Return $(v_1, \ldots, v_t)$

$\underline{\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)}$:

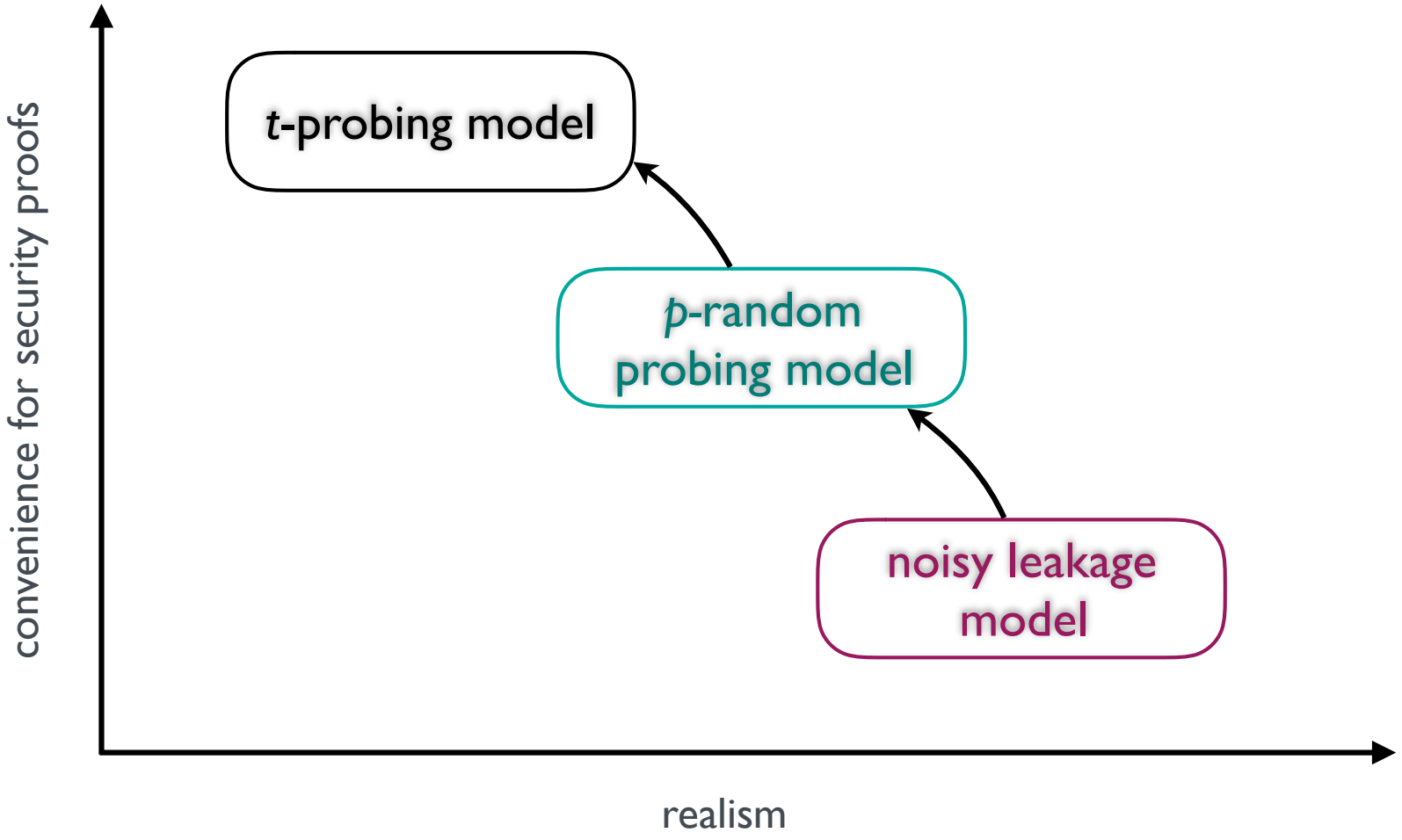
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $(v_1, \ldots, v_t) \leftarrow \mathcal{S}(\mathcal{P})$
3. Return $(v_1, \ldots, v_t)$

Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. CRYPTO 2003

Reductions


convenience for security proofs
t-probing model
p-random probing model
noisy leakage model
realism

Reductions



A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. EUROCRYPT 2014

Verification of Small Implementations

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

2 shares

1-probing secure?

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$y \leftarrow w \oplus r$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

Independent from secrets?

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$y \leftarrow w \oplus r$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

Independent from secrets?

$$w = v \oplus x$$
$$w = a_1 \cdot b_1 \oplus a_0 \cdot b_1$$
$$w = a \cdot b_1$$

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$y \leftarrow w \oplus r$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

3 shares

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
$\ldots$

return (c_0, c_1, c_2)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

3 shares

33 intermediate variables

$\binom{33}{2} = 528$ couples to verify

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
$\ldots$
return (c_0, c_1, c_2)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

3 shares

33 intermediate variables

$\binom{33}{2} = 528$ couples to verify

$\binom{n}{t}$ tuples to verify

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
$\dots$
return (c_0, c_1, c_2)

Proof in the Random Probing Model

- Reminder: an implementation is p-random probing secure iff the probability to get a tuple dependent from the secret is negligible given that each variable leaks with probability p

$$\sum_{i=1}^{n} \binom{n}{i}$$ tuples to verify

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
$\ldots$
return (c_0, c_1, c_2)

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

- Two methods to verify *t*-probing security of small implementations
  - Theoretical proof from the structure of the algorithm
  - Automatic proofs with a tool

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

- Two methods to verify *t*-probing security of small implementations
  - Theoretical proof from the structure of the algorithm
  - Automatic proofs with a tool

for $i = 1$ to t **do**
$r \leftarrow \$$
$x_0 \leftarrow x_0 + r$
$x_i \leftarrow x_i + r$
end for

Proof in the Probing Model

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret

- Two methods to verify *t*-probing security of small implementations
  - Theoretical proof from the structure of the algorithm
  - Automatic proofs with a tool

Recent Automatic Tools

- maskVerif [1,2]
  - Originally built in 2015, then extended in 2019
  - Probing security
- CheckMasks [3]
  - In CommonLisp
  - Faster with some details on the algorithm structure
  - Probing security
- Bloem et al. [4]
  - Probing security with physical defaults

[1] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub. Verified proofs of higher-order masking. EUROCRYPT 2015

[2] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, and F.-X. Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. ESORICS 2019

[3] J.-S. Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. ACNS 2017

[4] R. Bloem, H. Groß, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter. Formal verification of masked hardware implementations in the presence of glitches. EUROCRYPT 2018

Focus on maskVerif

Security order t

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
...
return (c_0, c_1, c_2)

maskVerif

Focus on maskVerif

Security order t

function example$(a_0, a_1, a_2, b_0, b_1, b_2)$

$r_{00}, r_{01}, r_{02}, r_{12} \leftarrow \$$
$t \leftarrow a_0 \cdot b_0$
$c_0 \leftarrow t \oplus r_{00}$
$t \leftarrow a_0 \cdot b_1$
$t \leftarrow t \oplus r_{01}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_0 \cdot b_2$
$t \leftarrow t \oplus r_{02}$
$c_0 \leftarrow c_0 \oplus t$
$t \leftarrow a_1 \cdot b_0$
$c_1 \leftarrow t \oplus r_{01}$
$t \leftarrow a_1 \cdot b_1$
$c_1 \leftarrow c_1 \oplus t$
...
return (c_0, c_1, c_2)

maskVerif

Security proof or potential attacks

Focus on maskVerif

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets

$$w = a_1 \cdot b_1 \oplus r \oplus a_0 \cdot b_1$$

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets
  - Rule 1: secrets?

$$w = a_1 \cdot b_1 \oplus r \oplus a_0 \cdot b_0$$

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets
  - Rule 1: secrets?
  - Rule 2: random values?

$$w = a_1 \cdot b_1 \oplus r \oplus a_0 \cdot b_1$$

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets
  - Rule 1: secrets?
  - Rule 2: random values?

$w = r$

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets
  - Rule 1: secrets?
  - Rule 2: random values?

$w = r$

- Go through all tuples
  - Verify bigger sets

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

Focus on maskVerif

- Determine whether a tuple is independent from the secrets
  - Rule 1: secrets?
  - Rule 2: random values?

$w = r$

- Go through all tuples
  - Verify bigger sets
- Extensions
  - Extended probing model

function example(a_0, a_1, b_0, b_1)

$r \leftarrow \$$

$u \leftarrow a_0 \cdot b_0$

$c_0 \leftarrow u \oplus r$

$v \leftarrow a_1 \cdot b_1$

$v \leftarrow v \cdot r$

$x \leftarrow a_0 \cdot b_1$

$w \leftarrow v \oplus x$

$z \leftarrow a_1 \cdot b_0$

$c_1 \leftarrow y \oplus z$

return (c_0, c_1)

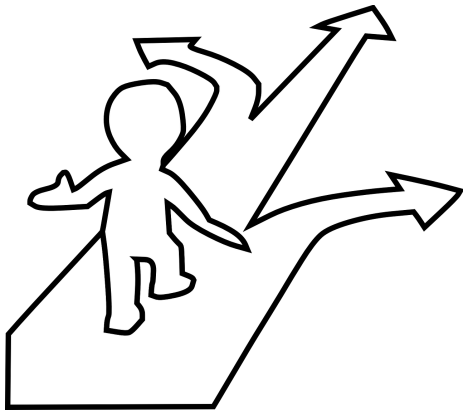
Focus on maskVerif

- Examples of classical implementations
- Until 6 shares
- In the presence of glitches (HW)

	# obs		probing	
	HW	SW	HW	SW
first-order masking (2 shares)				
Trichina AND [33]	2	13	0.01s ✗	0.01s ✗
ISW AND [25]	1	13	0.01s ✗	0.01s
TI AND [32]	3	21	0.01s	0.01s
DOM AND [22]	4	13	0.01s	0.01s
DOM AND SNI	6	13	0.01s	0.01s
PARA AND [5]	6	16	0.01s	0.01s
DOM Keccak S-box [23]	20	76	0.01s	0.01s
DOM AES S-box [22]	96	571	0.06s	0.6s
TI Fides-160 S-box [8]	192	6657	0.3s	2.8s
TI Fides-192 APN [8]	128	69281	2.3s	3m49s
second-order masking (3 shares)				
DOM AND [22]	12	30	0.01s	0.01s
DOM AND SNI	15	30	0.01s	0.01s
PARA AND [5]	15	30	0.01s	0.01s
DOM Keccak S-box [23]	60	165	0.03s	0.03s
DOM AES S-box [21]	168	1205	10.7s	15m45s
third-order masking (4 shares)				
DOM AND [22]	20	54	0.02s	0.03s
DOM AND SNI	24	54	0.03s	0.03s
PARA AND NI [5]	20	48	0.02s	0.02s
PARA AND SNI [5]	28	53	0.02s	0.02s
DOM Keccak S-box [23]	100	290	0.49s	0.68s
DOM AES S-box [21]	296	2011	12m36s	∞
fourth-order masking (5 shares)				
DOM AND [22]	30	87	0.1s	0.1s
PARA AND NI [5]	35	75	0.18s	0.15s
PARA AND SNI [5]	40	85	0.16s	0.16s
DOM Keccak S-box [23]	150	450	20s	41s
fifth-order masking (6 shares)				
DOM Keccak S-box [23]	210	618	3m59s	14m6s

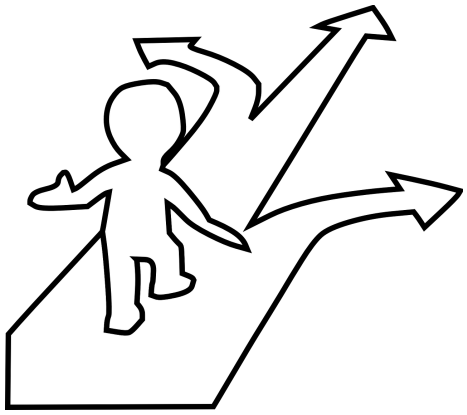
Main Challenges

- More efficient automatic verification tools



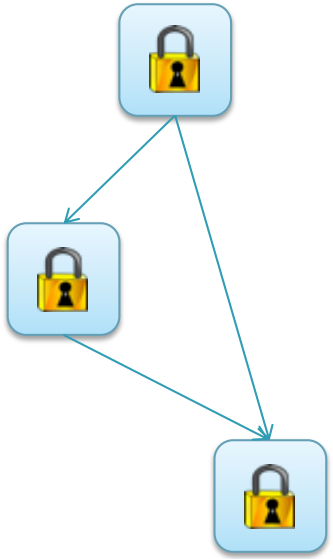
Main Challenges

- More efficient automatic verification tools
- Closer to the reality of embedded devices
  - Takes implementations in Assembly language
  - Proof in more accurate models



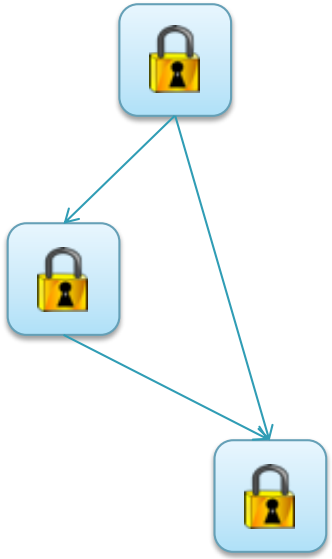
Composition

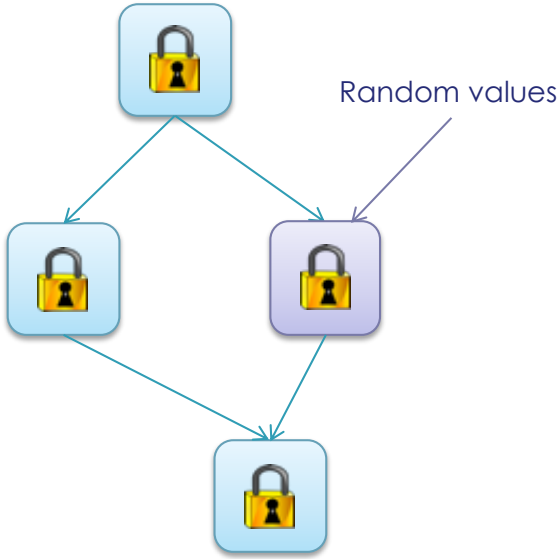
Composition of gadgets



[1] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. CRYPTO 2003

Composition of gadgets

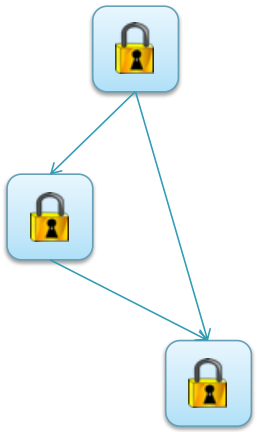




[1] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. CRYPTO 2003
[2] M. Rivain and E. Prouff. Provably secure higher-order masking of AES. CHES 2010

Composition of gadgets

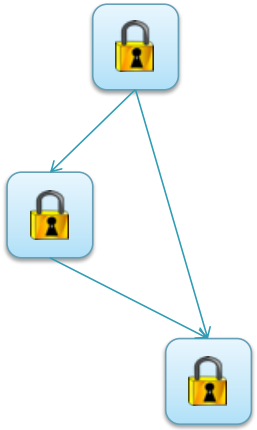
- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret
- How to reason on composition?



Composition of gadgets

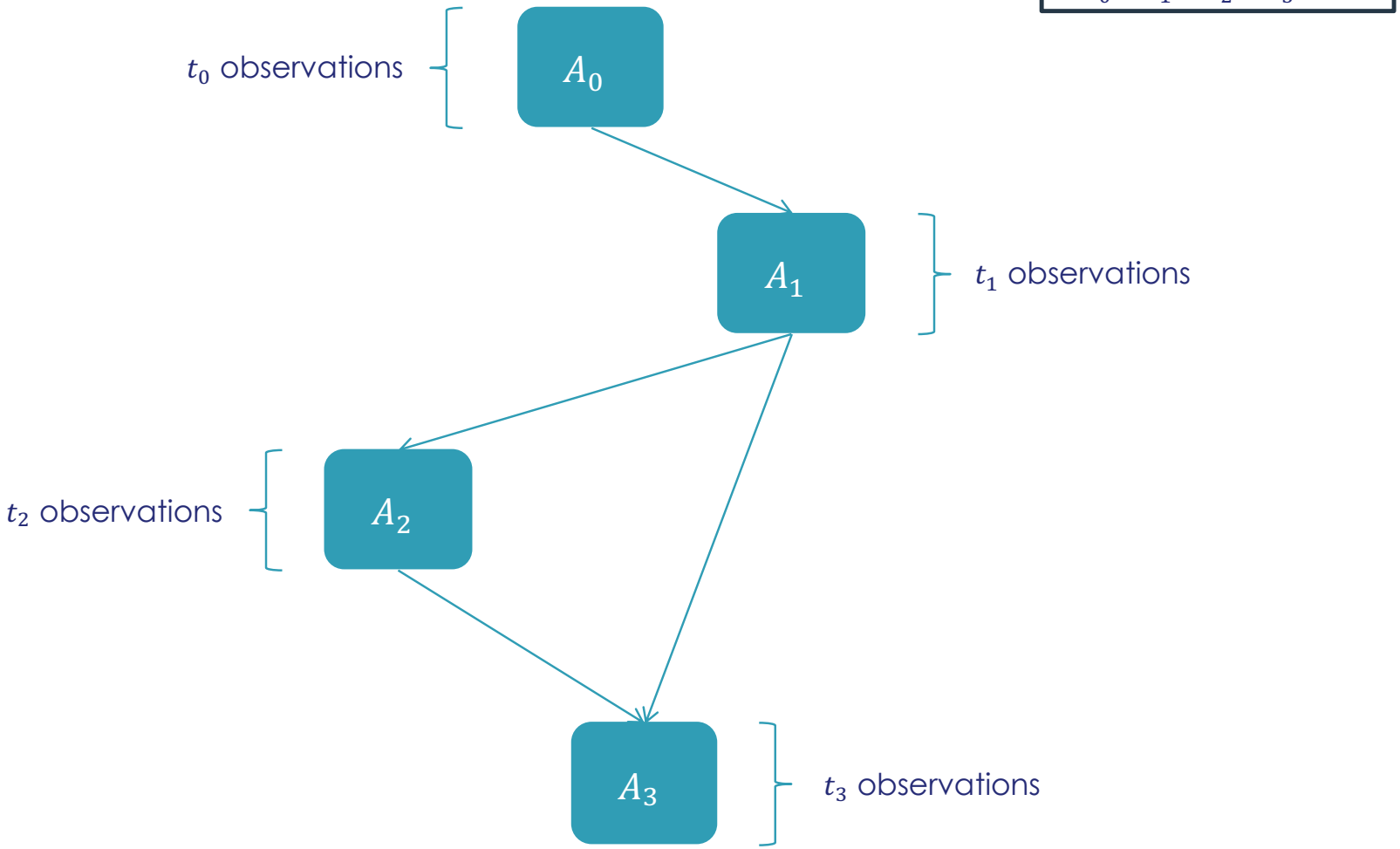
- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret
- How to reason on composition?
  - Stronger property: non-interference

An implementation is t-non-interfering iff any set of at most t variables can be simulated with at most t input shares

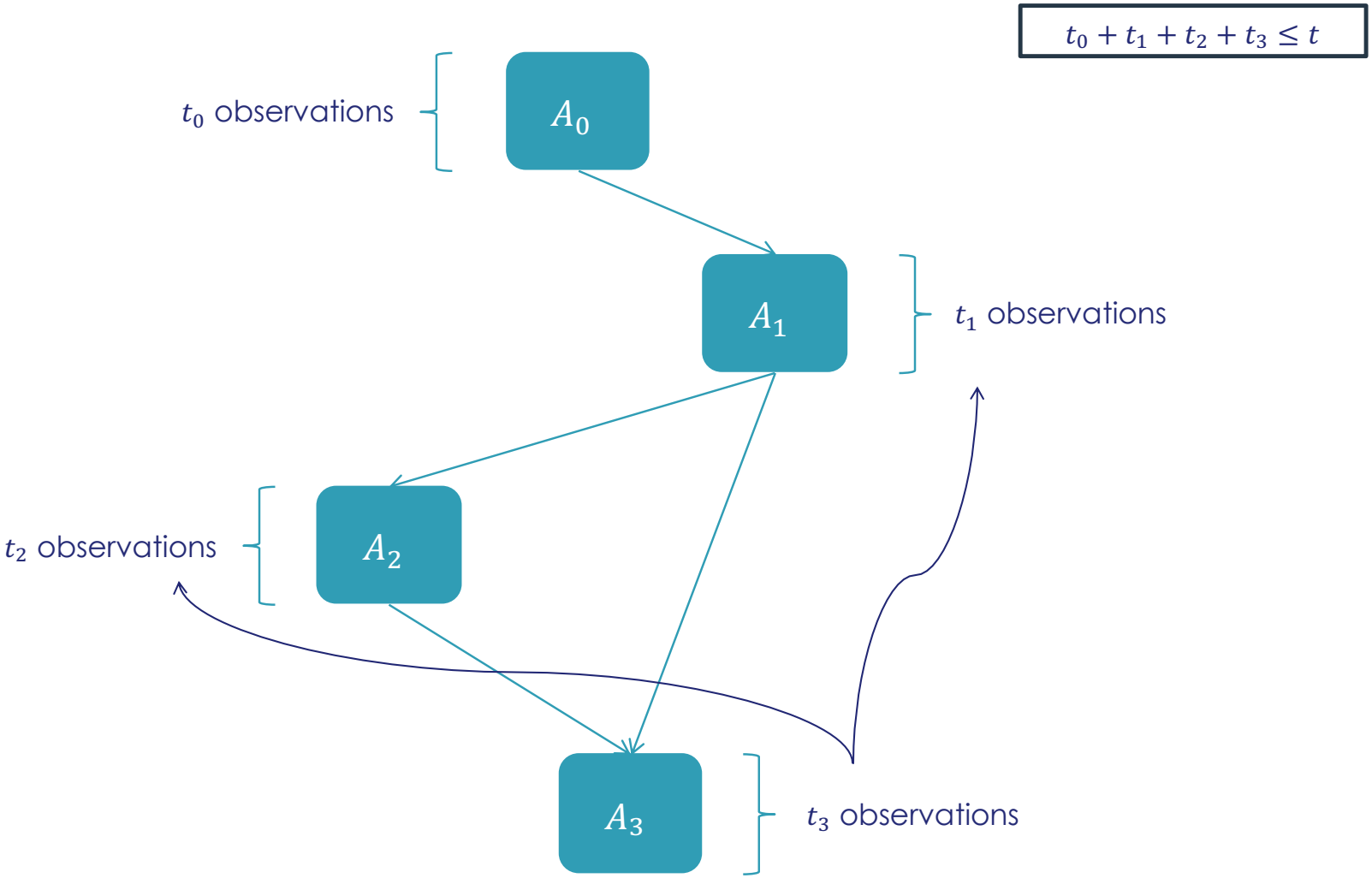


Composition of gadgets

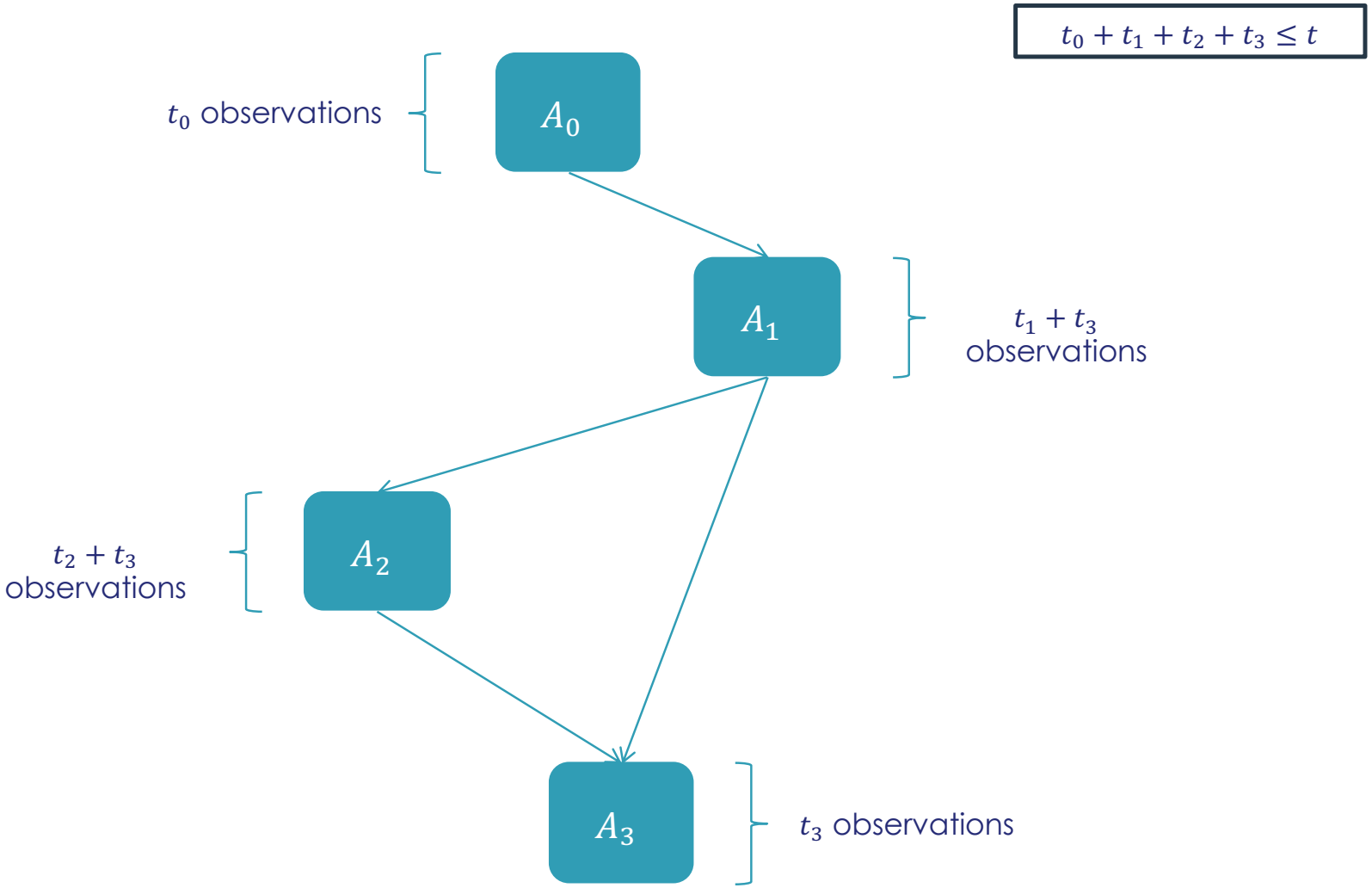
$t_0 + t_1 + t_2 + t_3 \leq t$



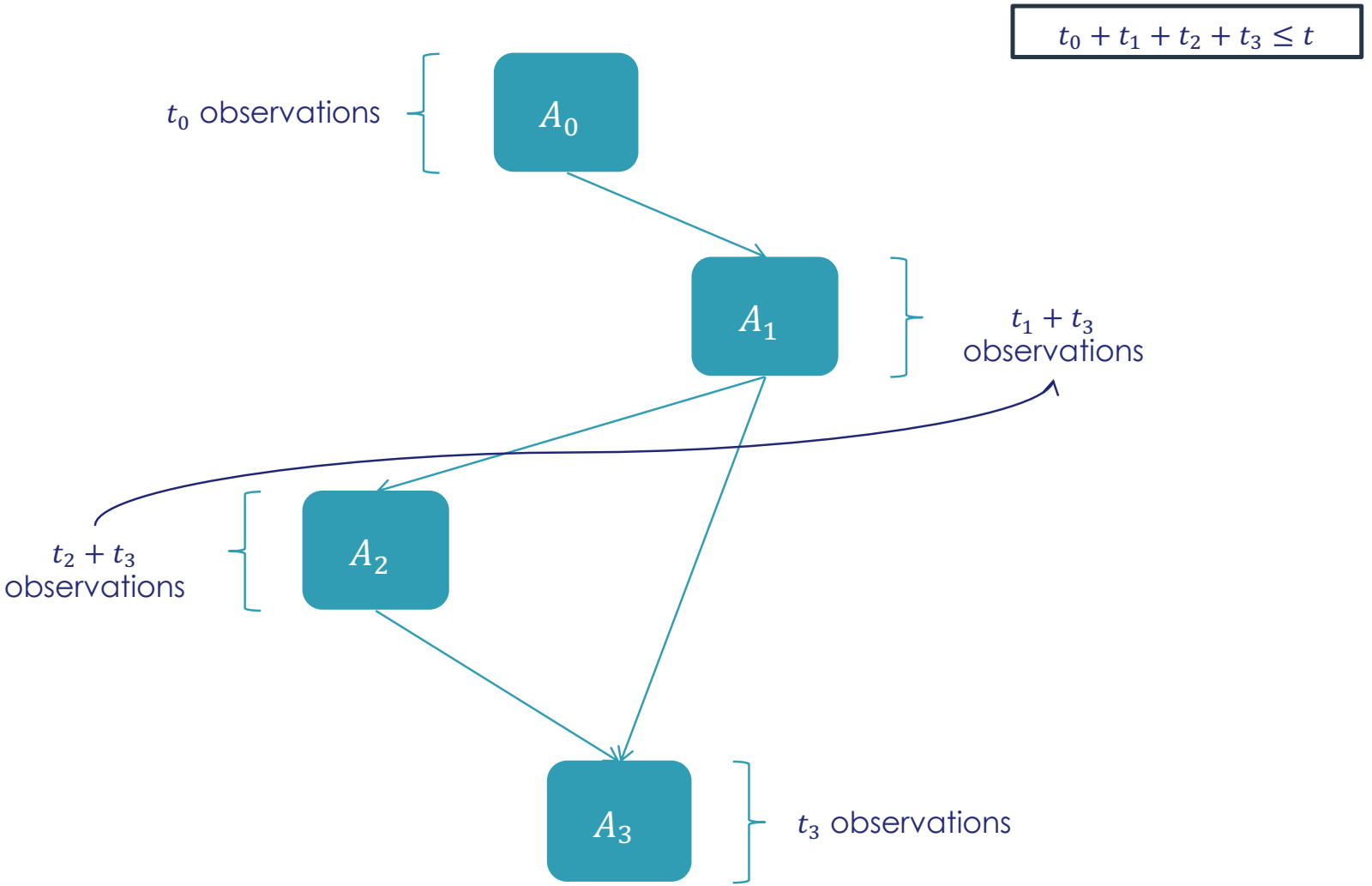
Composition of gadgets



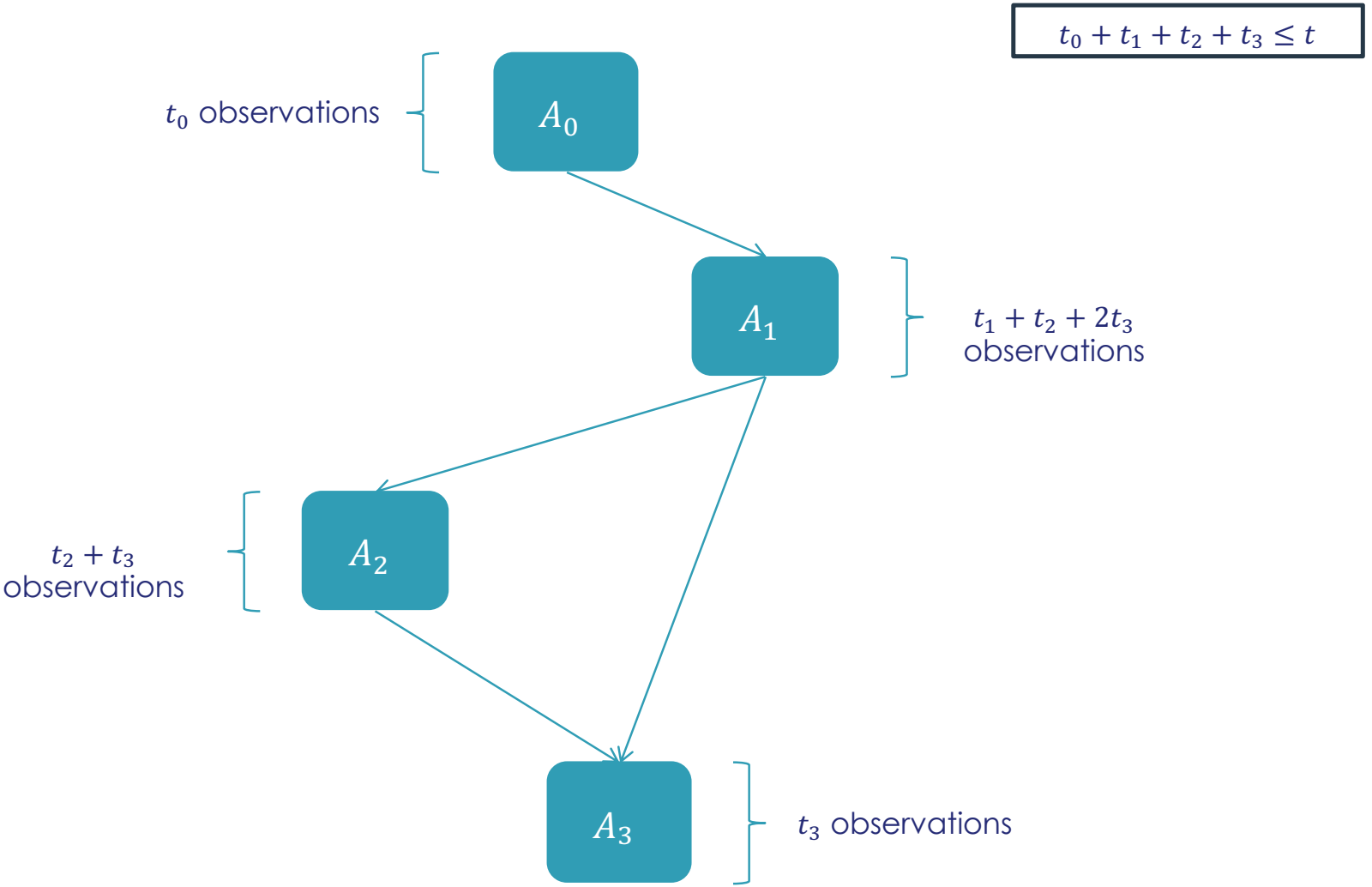
Composition of gadgets



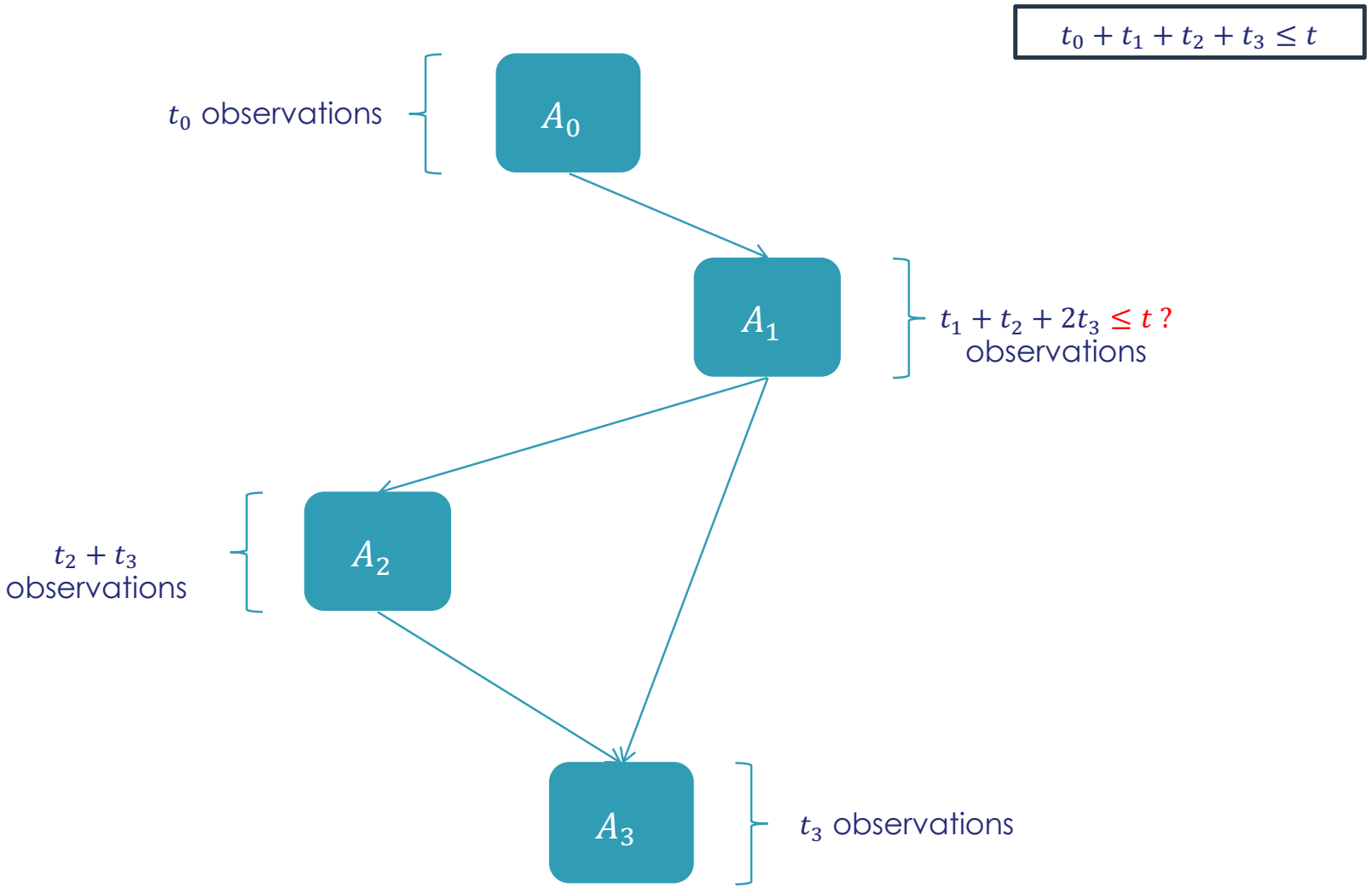
Composition of gadgets



Composition of gadgets



Composition of gadgets



Composition of gadgets

- Reminder: an implementation is *t*-probing secure iff any set of at most *t* variables is independent from the secret
- How to reason of composition?
  - Stronger property: non-interference

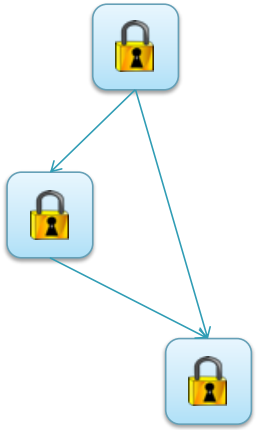
An implementation is t-non-interfering iff any set of at most *t* variables can be simulated with at most t input shares

  - Stronger property: strong non-interference

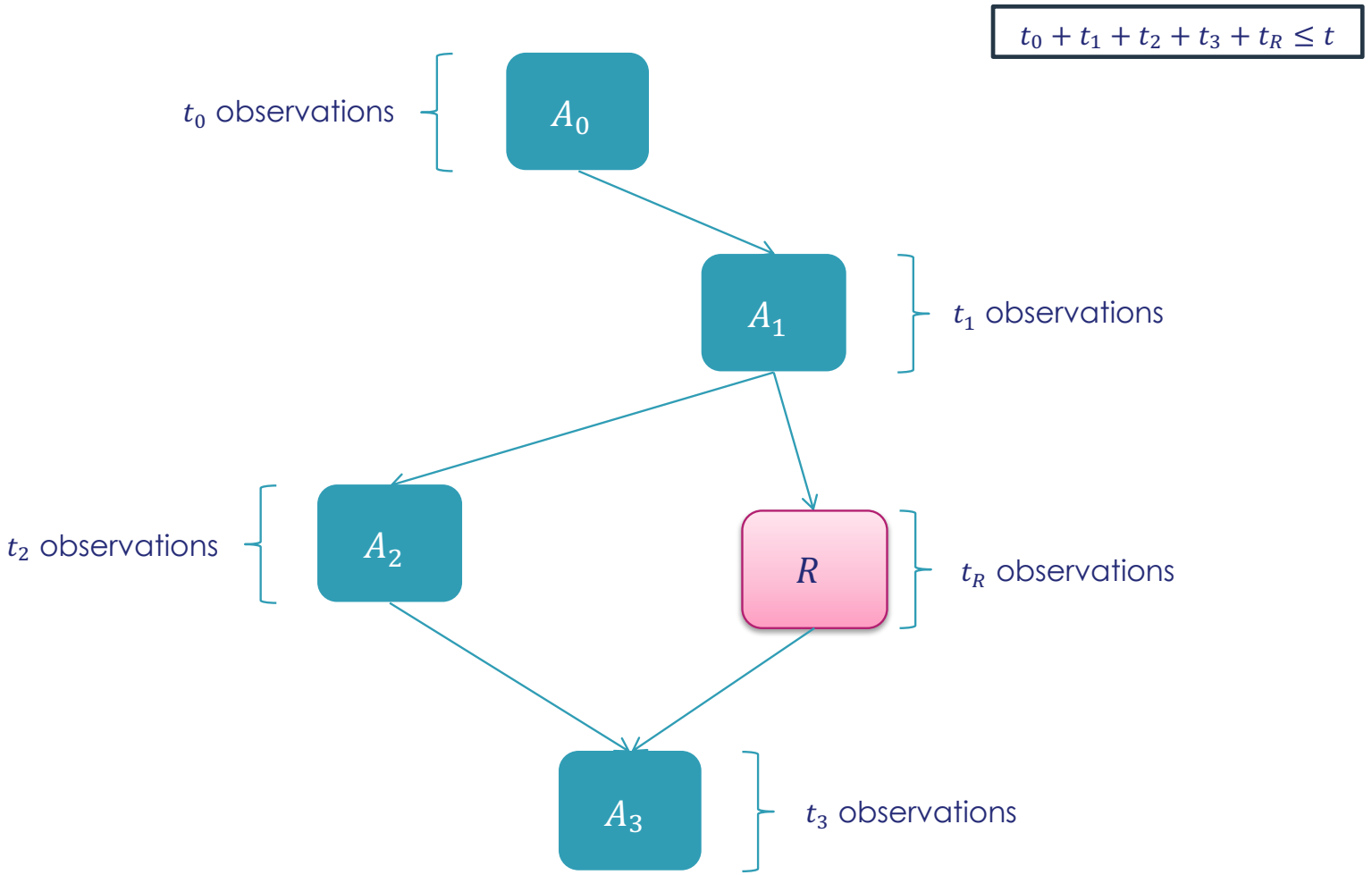
An implementation is t-strong non-interfering iff any set of

- *t1* internal variables
- *t2* output variables

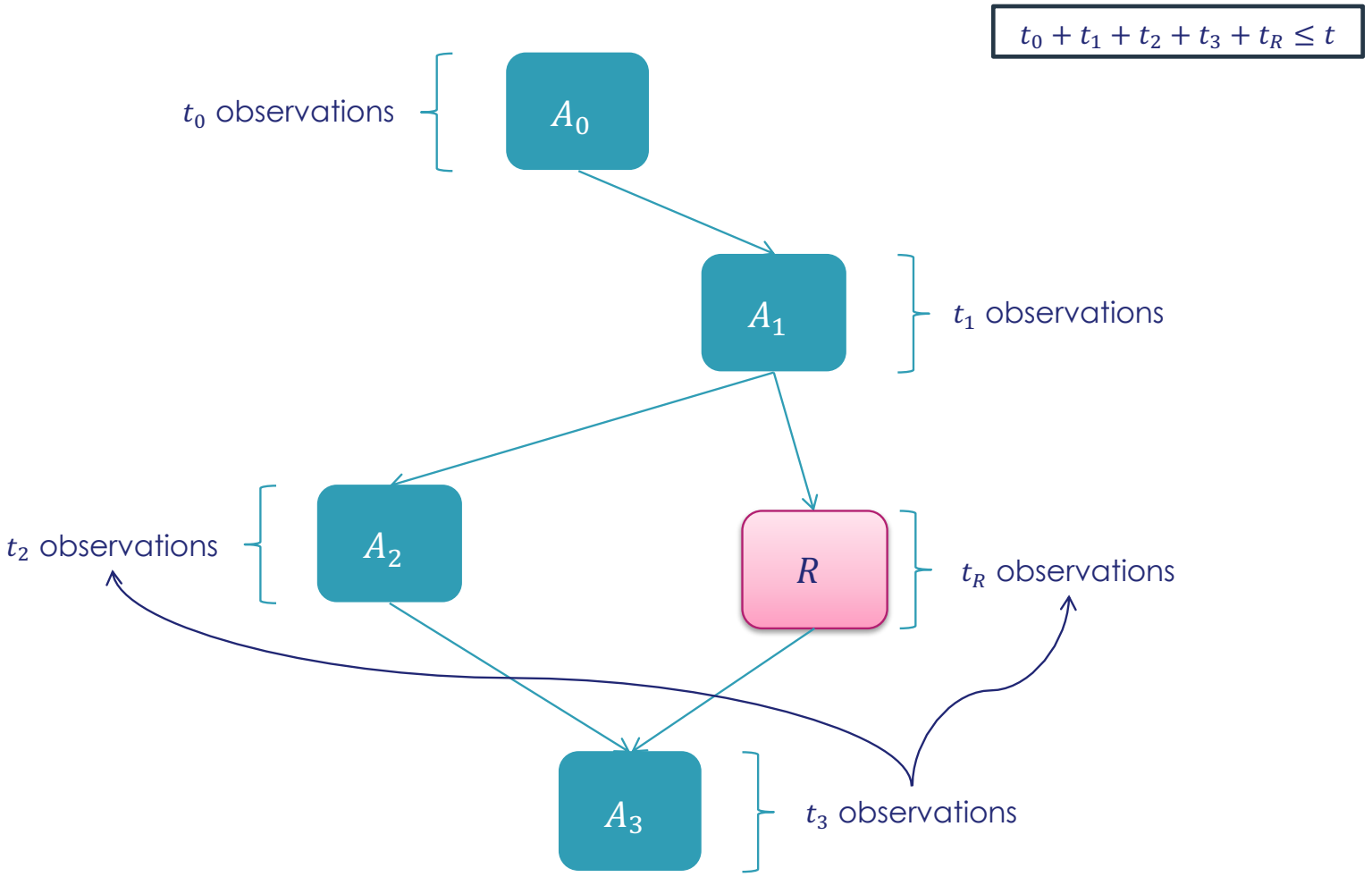
can be simulated with at most *t1* input shares



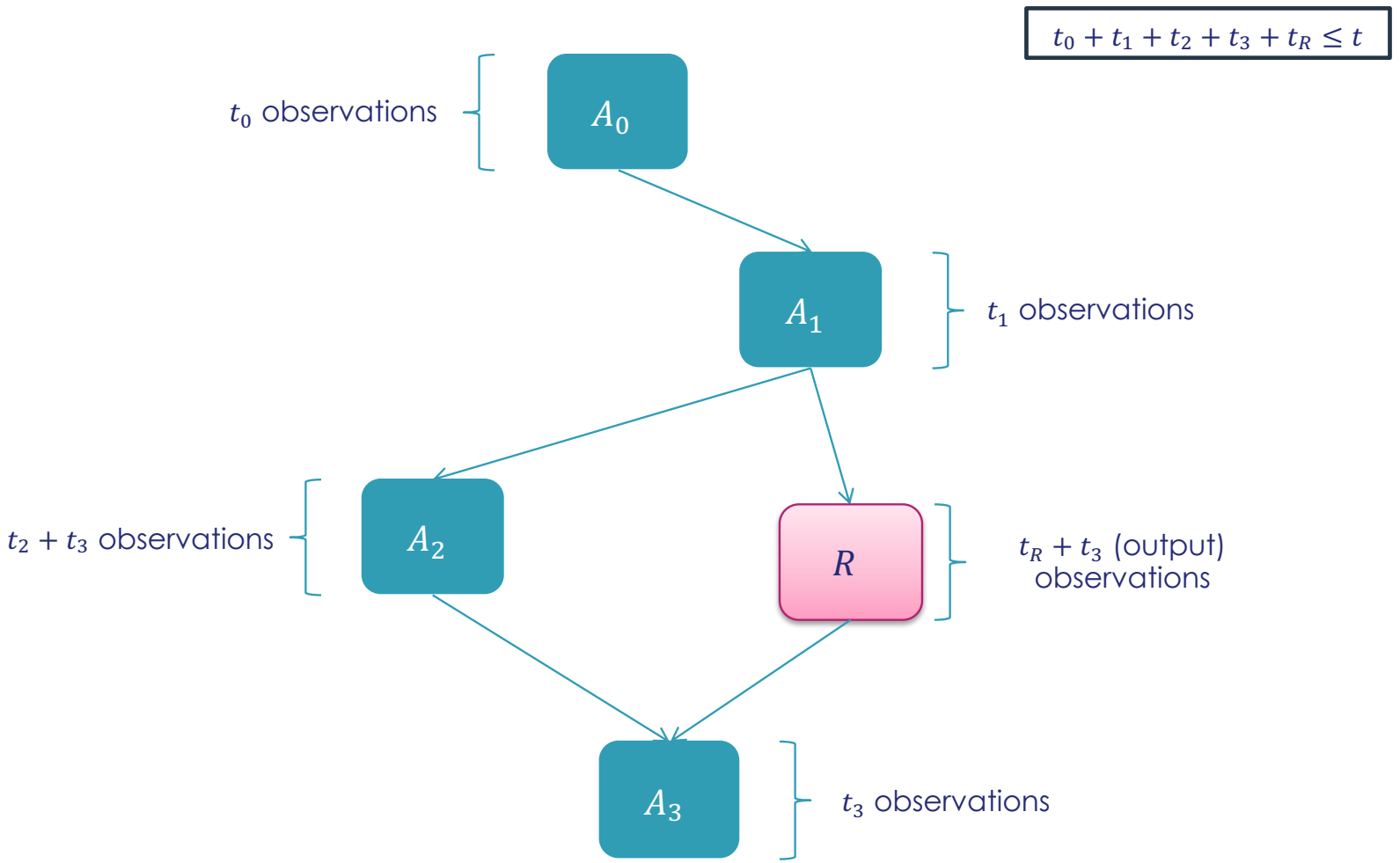
Composition of gadgets



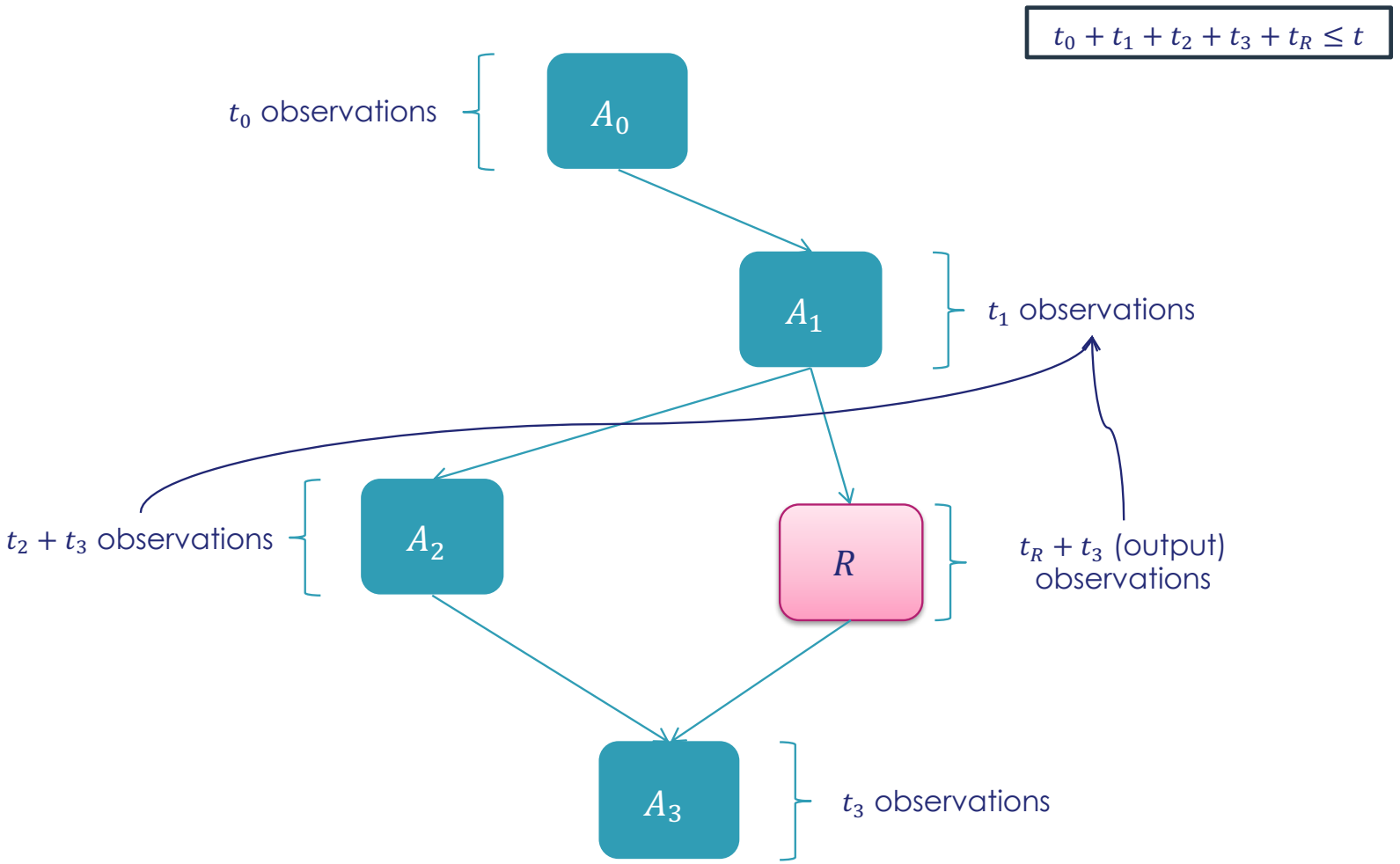
Composition of gadgets



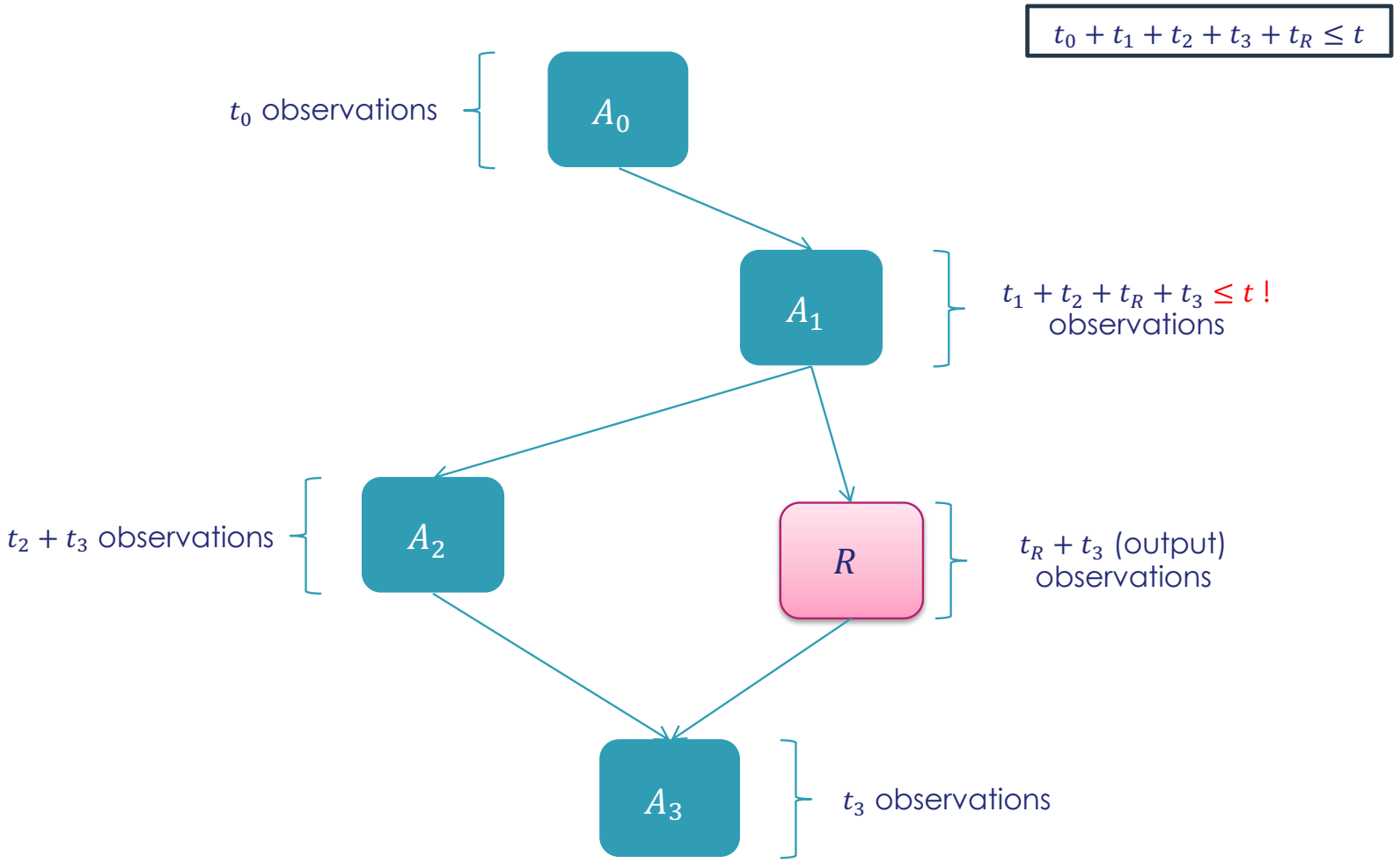
Composition of gadgets



Composition of gadgets



Composition of gadgets

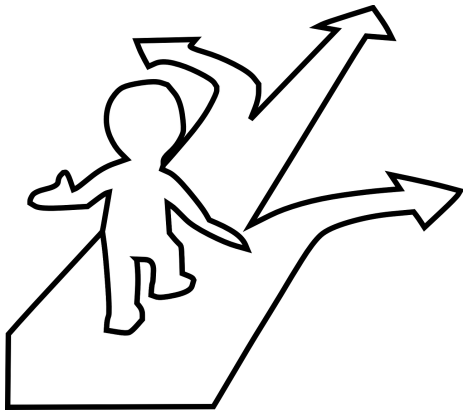


Composition techniques

- Compose NI/SNI gadgets as shown
  - Tool maskComp
- Compose standard circuits in the probing model
  - Tool tightPROVE
  - Exact methods restricted to circuits from addition, ISW multiplications, and refresh gadgets
- Compose gadgets with stronger properties
  - Example: PINI

Main Challenges

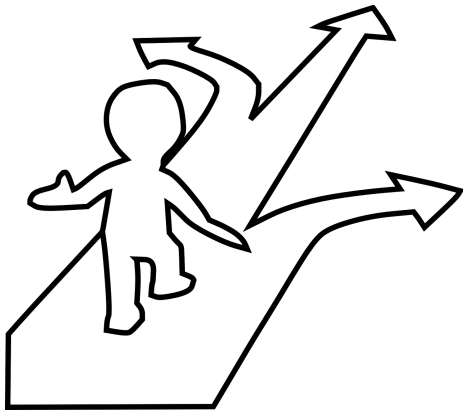
- Being able to compose any kind of gadgets without loss of efficiency



Main Challenges

- Being able to compose any kind of gadgets without loss of efficiency

- Compose in more realistic leakage models



Conclusion

Summary

- Side-channel attacks are very powerful
  - Few seconds to recover the key on some software devices
  - Cheap equipments

Summary

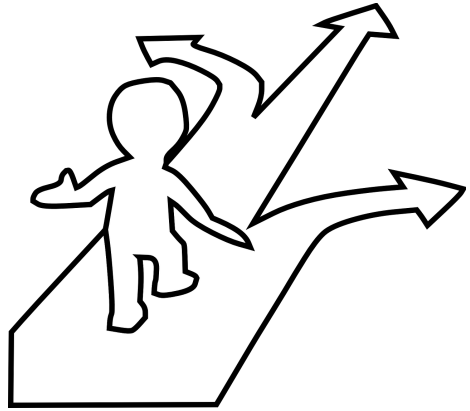
- Side-channel attacks are very powerful
  - Few seconds to recover the key on some software devices
  - Cheap equipments
- Masking is the most widely deployed countermeasure
  - Numerous works
  - Difficult to build secure constructions

Summary

- Side-channel attacks are very powerful
  - Few seconds to recover the key on some software devices
  - Cheap equipments
- Masking is the most widely deployed countermeasure
  - Numerous works
  - Difficult to build secure constructions
- Verification
  - Automatic tools
  - Composition

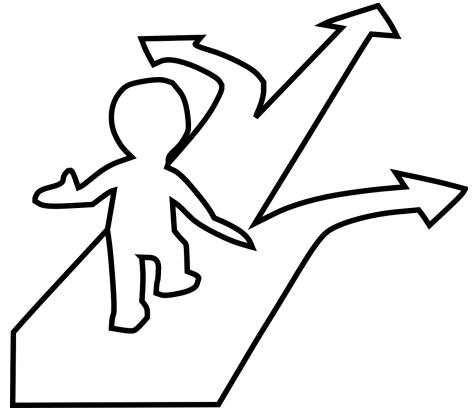
Challenges

- Efficiency
  - The least possible randomness
  - The least possible operations



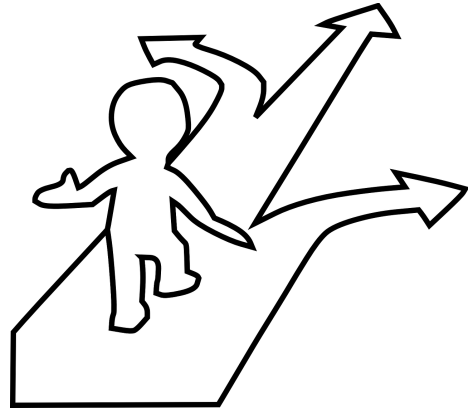
Challenges

- Efficiency
  - The least possible randomness
  - The least possible operations



- Security
  - Theoretical proofs of existing schemes
  - Automatic tools to verify the security of implementations

Challenges



- Efficiency
  - The least possible randomness
  - The least possible operations
- Security
  - Theoretical proofs of existing schemes
  - Automatic tools to verify the security of implementations
- Practicality
  - Security of implementations under leakage models as close as possible to the reality

Thank you.
Questions?