RUB

Leakage Assessment Methodology

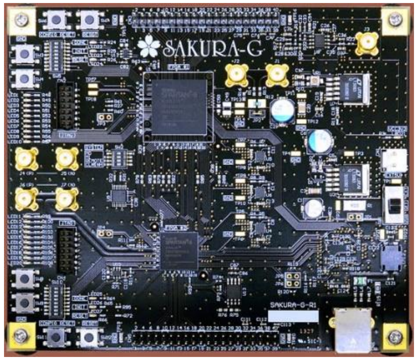
- a clear roadmap for side-channel evaluations -

Tobias Schneider and Amir Moradi

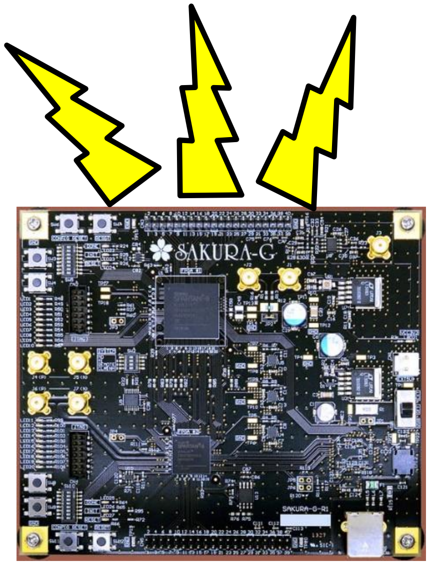
Wednesday, September 16th, 2015

hgi Horst Görtz Institute for IT-Security
RUHR-UNIVERSITÄT BOCHUM

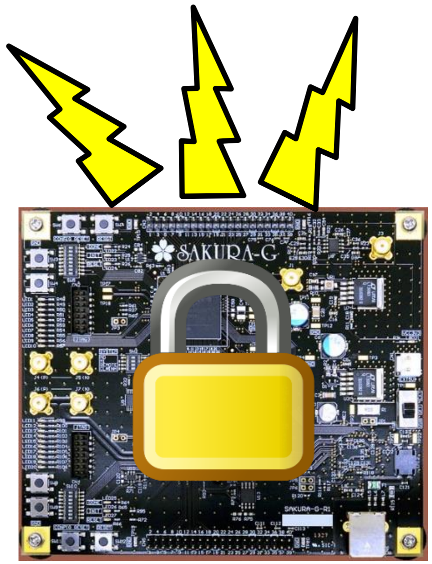
Motivation Security Evaluation



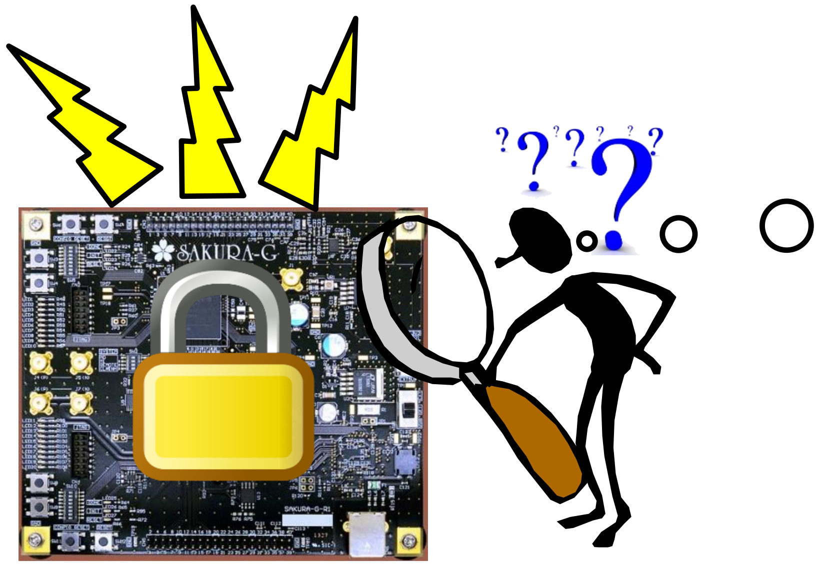
Motivation Security Evaluation



Motivation Security Evaluation



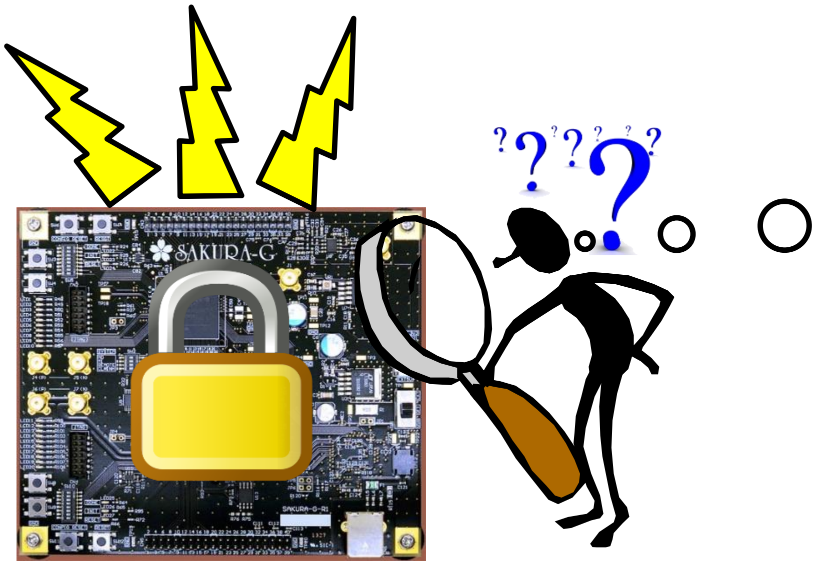
Motivation Security Evaluation



Does the chip leak information?

Problem: Evaluation is not trivial.

Motivation Security Evaluation



Does the chip leak information?

Problem: Evaluation is not trivial.

NIST *Non-Invasive Attack Testing Workshop, 2011*

Goal: Establish testing methodology capable of robustly assessing the physical vulnerability of cryptographic devices.

Motivation Attack-based Testing

Perform state-of-the-art attacks on the device under test (DUT)

Attacks Types:
- DPA
- CPA
- MIA
- …

×

Intermediate Values:
- Sbox In
- Sbox Out
- Sbox In/Out
- …

×

Leakage Models:
- HW
- HD
- Bit
- …

Motivation Attack-based Testing

Perform state-of-the-art attacks on the device under test (DUT)

Attacks Types:
- DPA
- CPA
- MIA
- …

×

Intermediate Values:
- Sbox In
- Sbox Out
- Sbox In/Out
- …

×

Leakage Models:
- HW
- HD
- Bit
- …

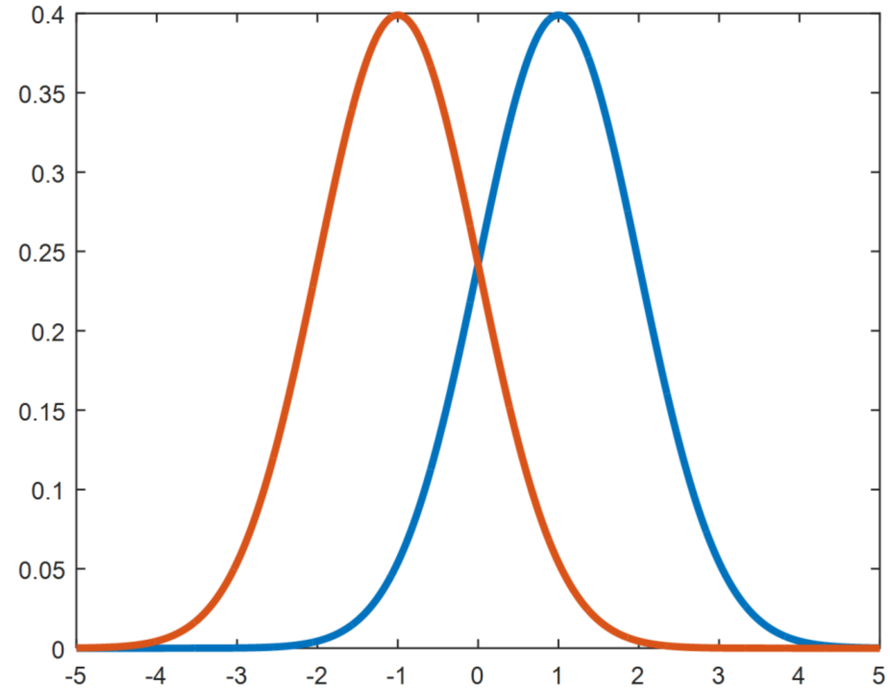
Problems:
- High computational complexity
- Requires lot of expertise
- Does not cover all possible attack vectors

Motivation Testing based on *t*-Test

Tries to detect any type of leakage at a certain order

CRYPTOGRAPHY™ RESEARCH
a division of Rambus

- Proposed by CRI at NIST workshop



Advantages:

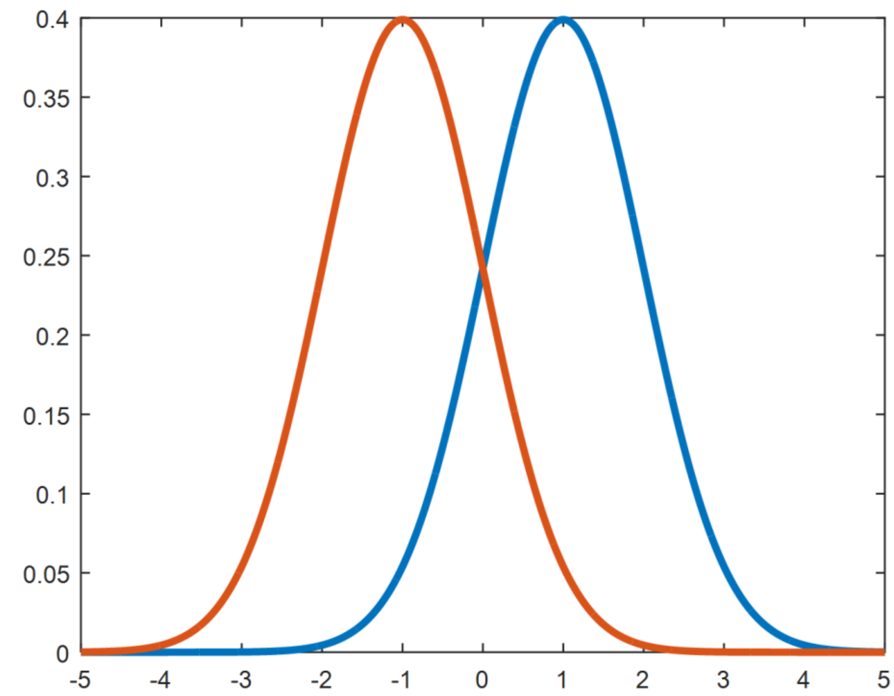
- Independent of architecture
- Independent of attack model
- Fast & simple
- Versatile

Motivation Testing based on *t*-Test

Tries to detect any type of leakage at a certain order

CRYPTOGRAPHY™ RESEARCH
a division of Rambus

- Proposed by CRI at NIST workshop



Advantages:

- Independent of architecture
- Independent of attack model
- Fast & simple
- Versatile

Problems:

- No information about hardness of attack
- Possible false positives if no care about evaluation setup

Contribution

1. Explain statistical background in a (hopefully) more understandable way

2. More detailed discussion of higher-order testing

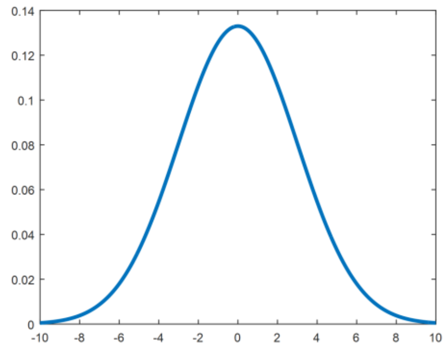
3. Hints how to design fast & correct measurement setup

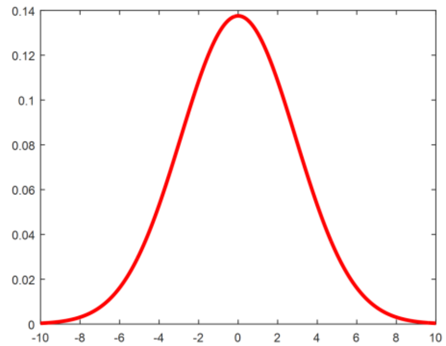
4. Optimization of analysis phase

Statistical Background

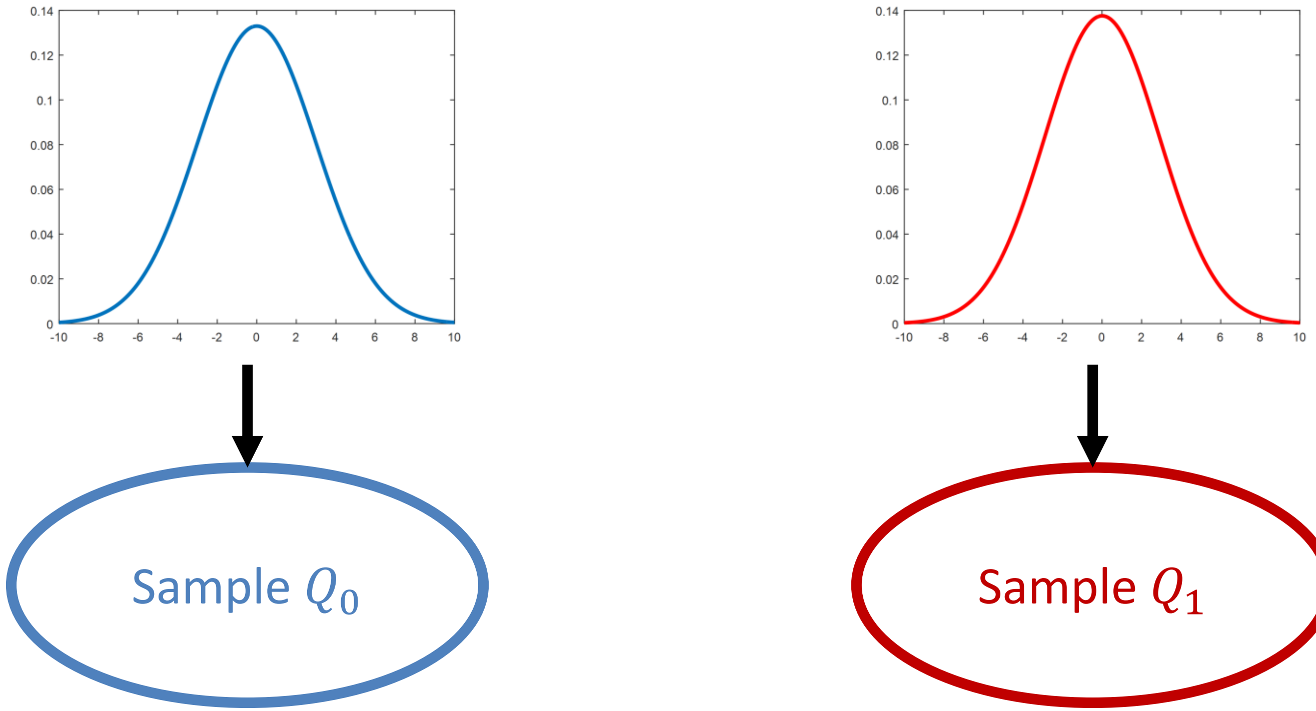
- *t*-Test

Statistical Background *t*-Test

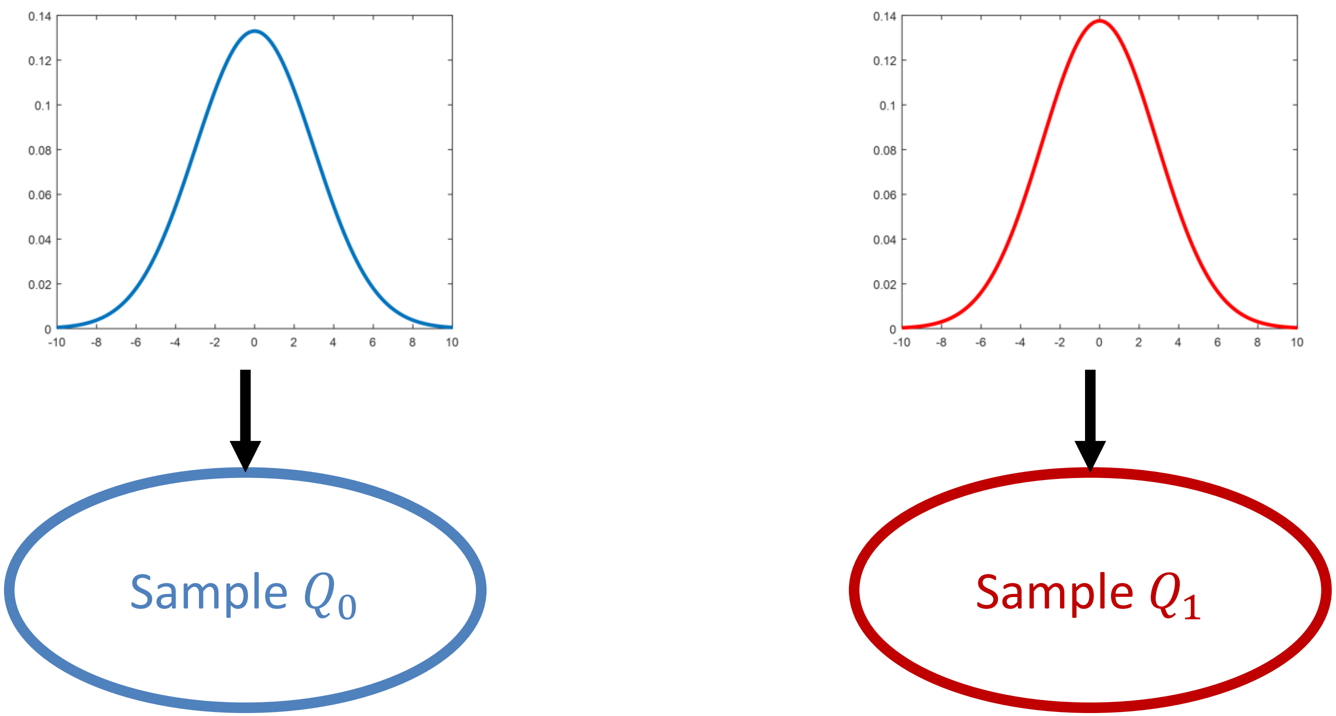




Statistical Background *t*-Test



Statistical Background *t*-Test



Null Hypothesis: Two population means are equal.

Statistical Background *t*-Test

Sample Q_0

Sample Q_1

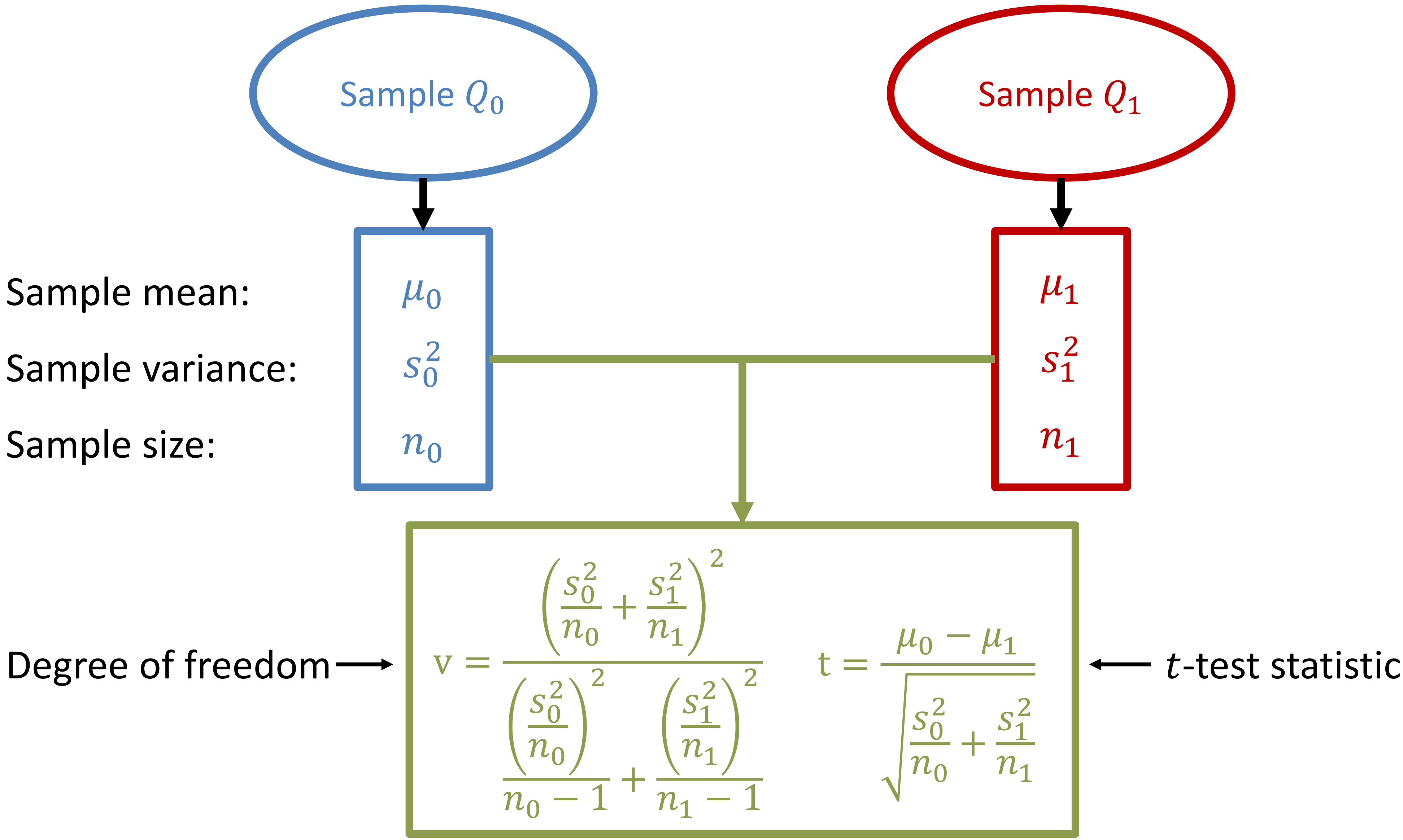
Statistical Background *t*-Test

Sample mean:

Sample variance:

Sample size:

Statistical Background *t*-Test



$$\mathrm{v} = \frac{\left(\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}\right)^2}{\frac{\left(\frac{s_0^2}{n_0}\right)^2}{n_0 - 1} + \frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1}}$$

$$\mathrm{t} = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$$

Statistical Background *t*-Test

t $\quad$ v

Estimate the probability to accept null hypothesis with Student's t distribution:

$$f(t,v) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{\pi v}\,\Gamma\left(\frac{v}{2}\right)}\left(1+\frac{t^2}{v}\right)^{-\frac{v+1}{2}}$$

Compute:

$$p = 2\int_{|t|}^{\infty} f(\mathrm{t},\mathrm{v})\,dt$$

Statistical Background *t*-Test

t $\quad$ v

Estimate the probability to accept null hypothesis with Student's t distribution:

$$f(t,v) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{\pi v}\,\Gamma\left(\frac{v}{2}\right)}\left(1+\frac{t^2}{v}\right)^{-\frac{v+1}{2}}$$

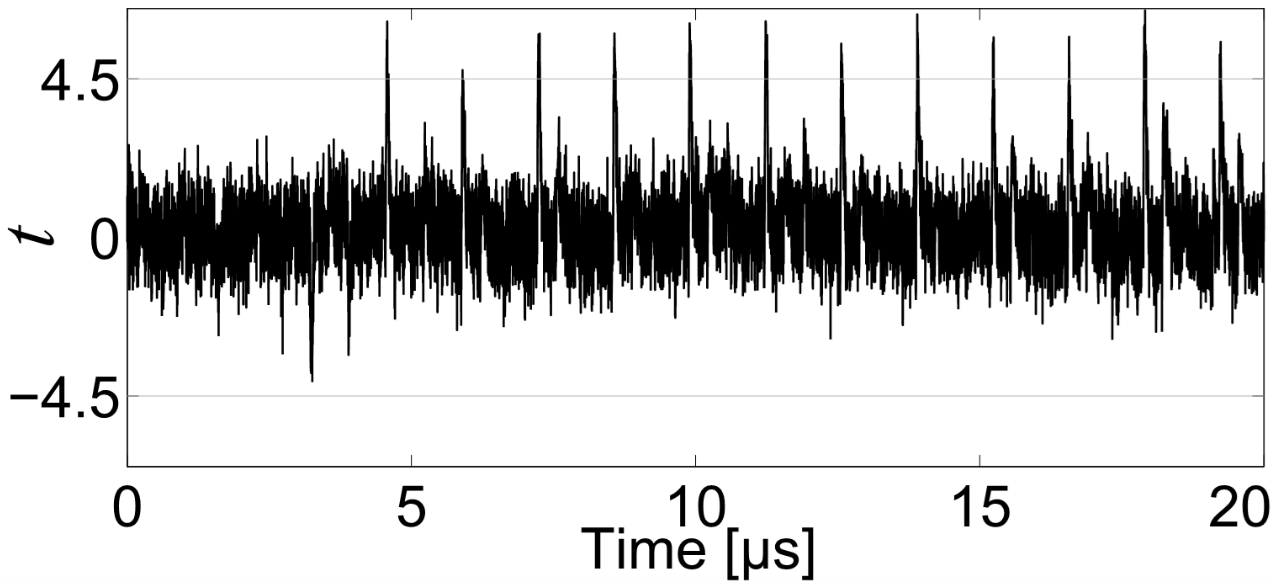
Compute:

$$p = 2\int_{|t|}^{\infty} f(\mathrm{t},\mathrm{v})\,dt$$

Small p values give evidence to reject the null hypothesis

Statistical Background *t*-Test

- For testing usually only the t-value is estimated
- Compared to a threshold of $|t| > 4.5$
  - $p = 2F(-4.5, v > 1000) < 0.00001$
  - Confidence of > 0.99999 to reject the null hypothesis



Testing Methodology

- Specific t-Test
- Non-Specific t-Test

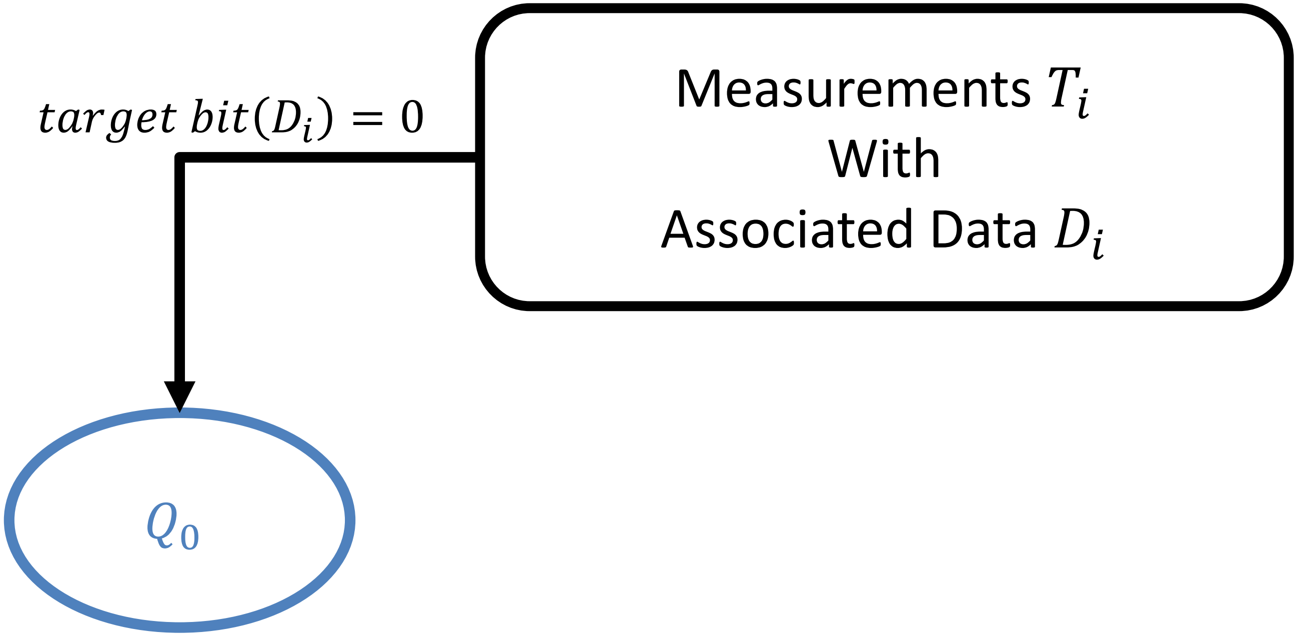
Testing Methodology Specific *t*-Test

Measurements T_i
With
Associated Data D_i

Specific *t*-Test:

- Key is known to enable correct partitioning
- Test is conducted at each sample point separately (univariate)
- If corresponding t-test exceeds threshold ⇒ DPA probable

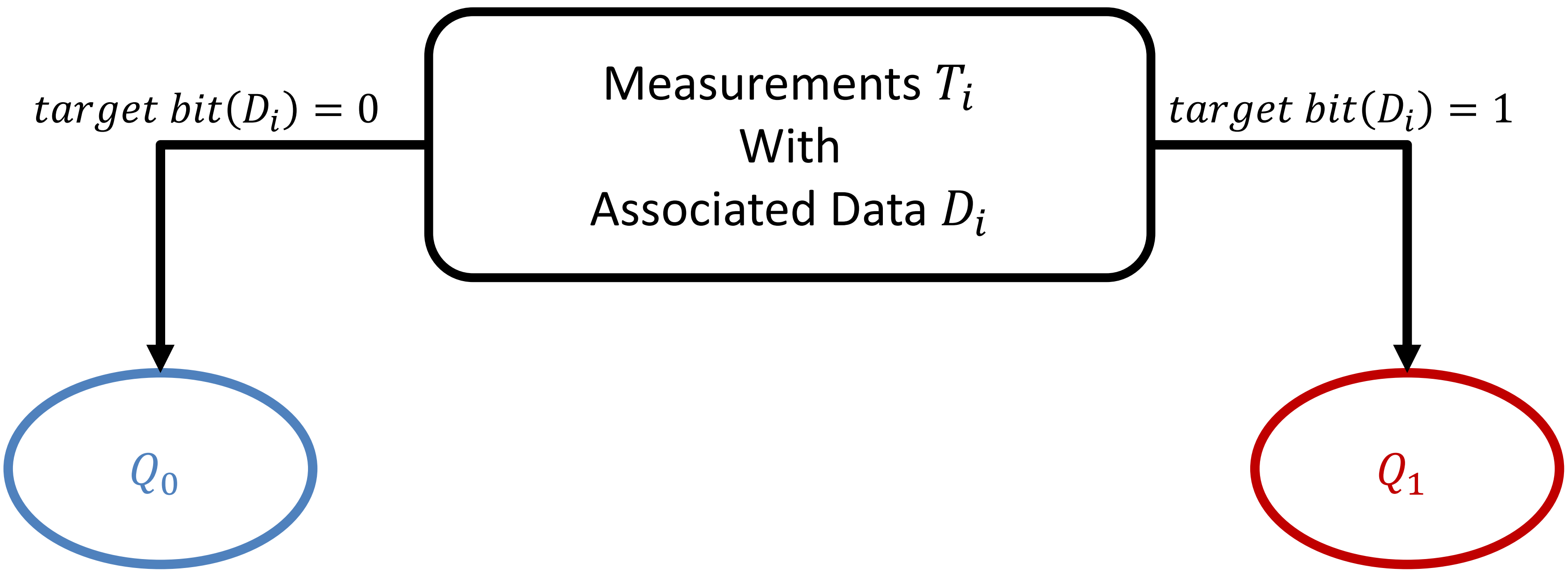
Testing Methodology Specific *t*-Test



Specific *t*-Test:

- Key is known to enable correct partitioning
- Test is conducted at each sample point separately (univariate)
- If corresponding t-test exceeds threshold ⇒ DPA probable

Testing Methodology Specific *t*-Test



Specific *t*-Test:

- Key is known to enable correct partitioning
- Test is conducted at each sample point separately (univariate)
- If corresponding t-test exceeds threshold ⇒ DPA probable

Testing Methodology Non-Specific *t*-Test

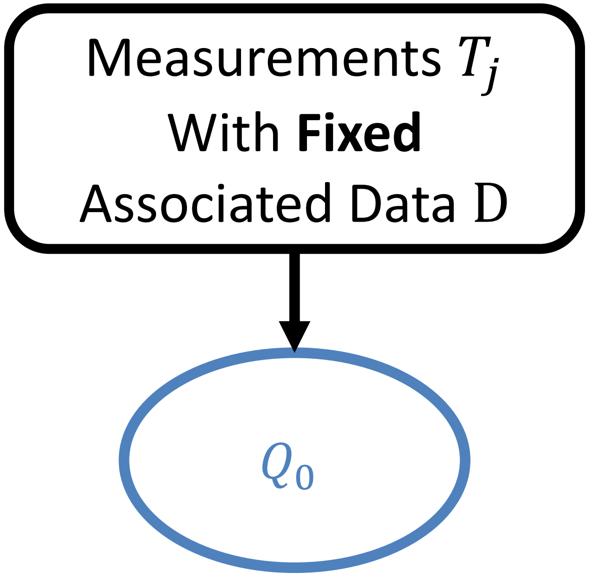
Non-Specific *t*-Test:

- *fixed vs. random t*-test
- Avoids being dependent on any intermediate value/model
- Detected leakage of single test is not always exploitable
- *Semi-fixed vs. random t*-test useful in certain cases

Testing Methodology Non-Specific *t*-Test

Non-Specific *t*-Test:

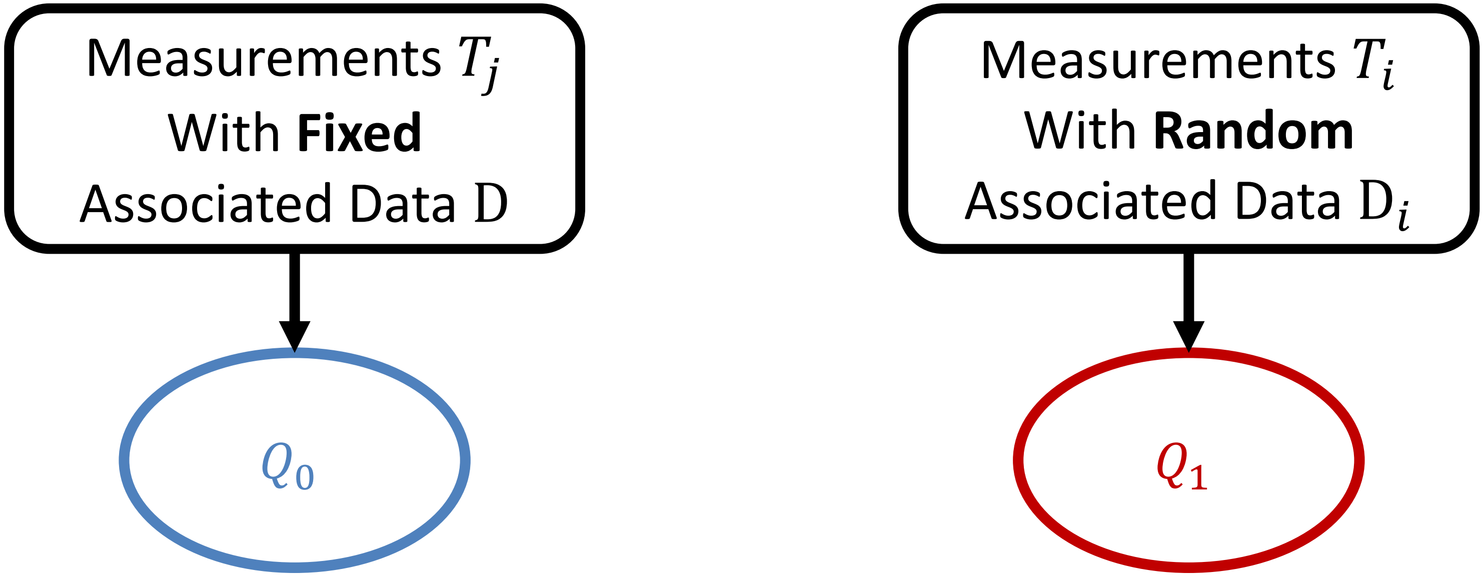
- *fixed vs. random t*-test
- Avoids being dependent on any intermediate value/model
- Detected leakage of single test is not always exploitable
- *Semi-fixed vs. random t*-test useful in certain cases



Testing Methodology Non-Specific *t*-Test

Non-Specific *t*-Test:

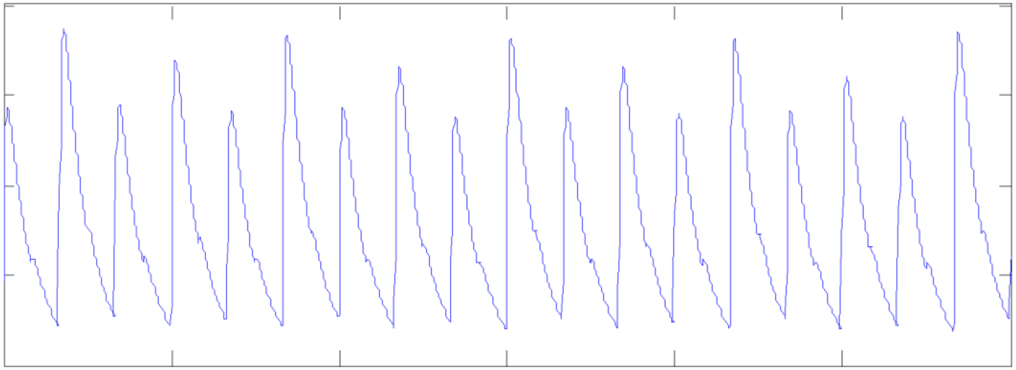
- *fixed vs. random t*-test
- Avoids being dependent on any intermediate value/model
- Detected leakage of single test is not always exploitable
- *Semi-fixed vs. random t*-test useful in certain cases



Higher-Order Testing

- Multivariate
- Univariate

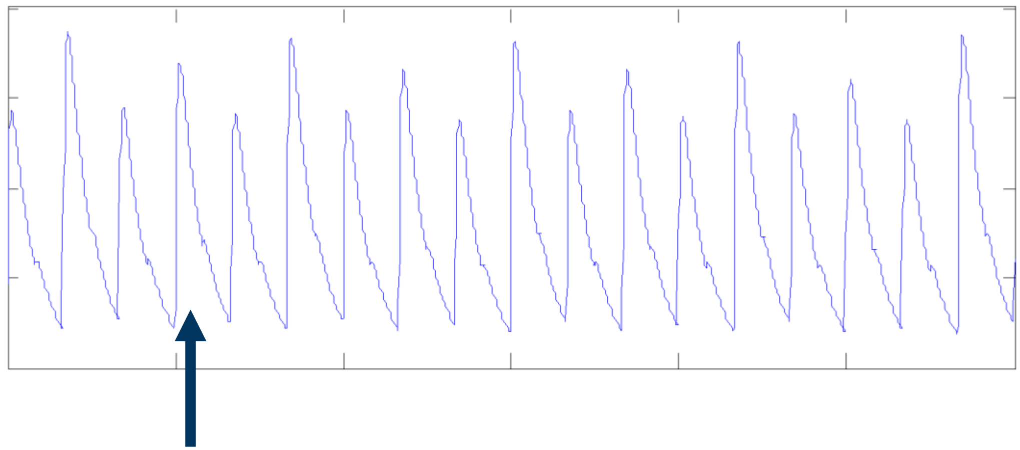
Higher-Order Testing Multivariate



Multivariate:

- Sensitive variable is shared: $S = S_1 \circ S_2$
- Shares are processed at different time instances (SW)
- Leakages at different time instances need to be combined first

Higher-Order Testing Multivariate

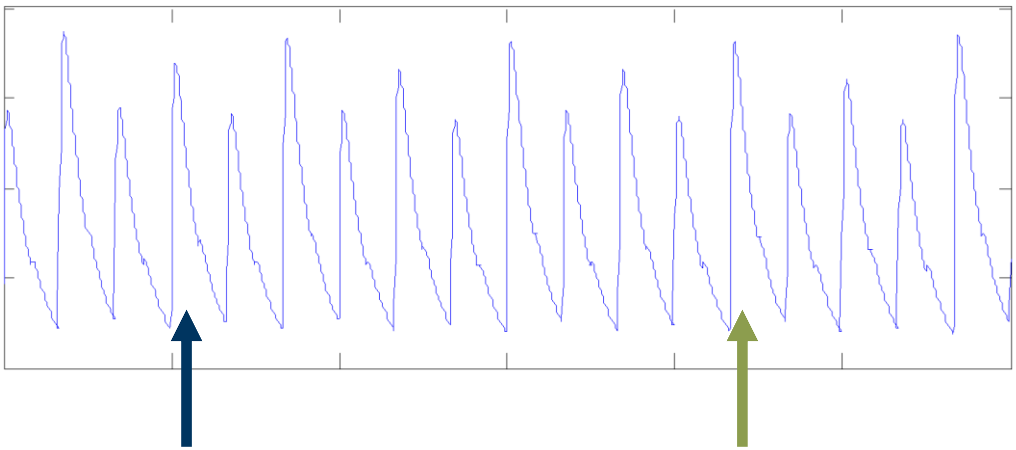


S_1

Multivariate:

- Sensitive variable is shared: $S = S_1 \circ S_2$
- Shares are processed at different time instances (SW)
- Leakages at different time instances need to be combined first

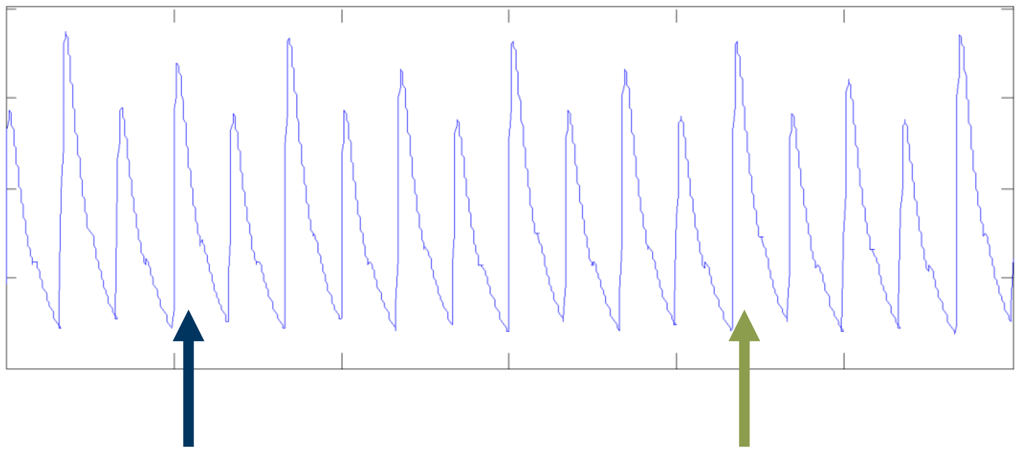
Higher-Order Testing Multivariate



Multivariate: S_1 S_2

- Sensitive variable is shared: $S = S_1 \circ S_2$
- Shares are processed at different time instances (SW)
- Leakages at different time instances need to be combined first

Higher-Order Testing Multivariate

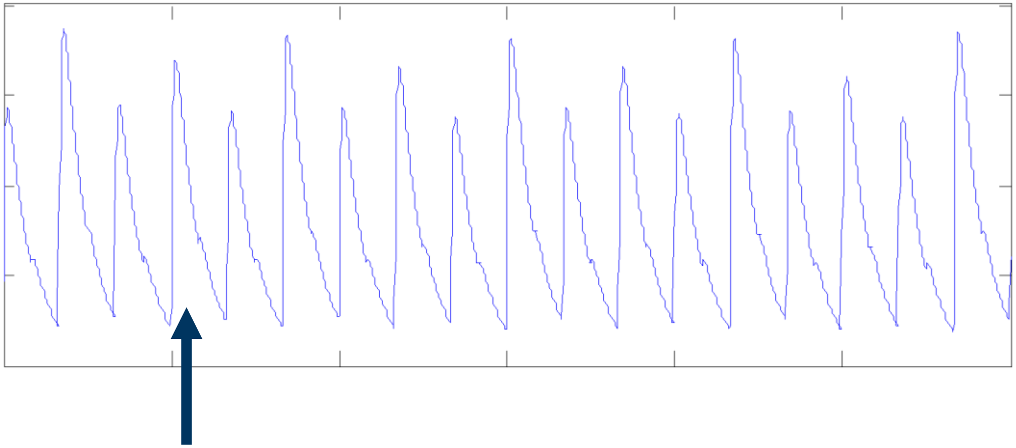


Multivariate:

- Sensitive variable is shared: $S = S_1 \circ S_2$
- Shares are processed at different time instances (SW)
- Leakages at different time instances need to be combined first

Centered Product: $x' = (x_1 - \mu_1) \cdot (x_2 - \mu_2)$

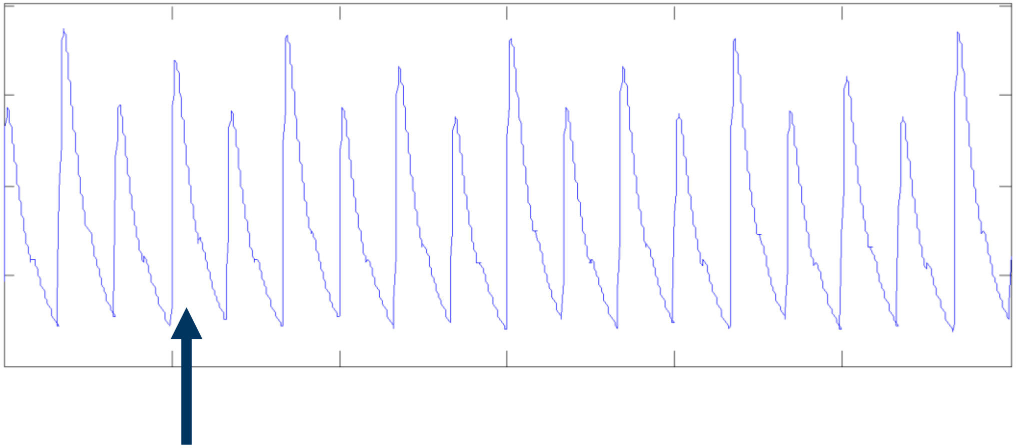
Higher-Order Testing Univariate



Univariate: S_1 S_2

- Shares are processed in parallel (HW)
- Leakages at the same time instance need to be combined first

Higher-Order Testing Univariate



Univariate: S_1 S_2

- Shares are processed in parallel (HW)
- Leakages at the same time instance need to be combined first

Variance: $x' = (x - \mu)^2$

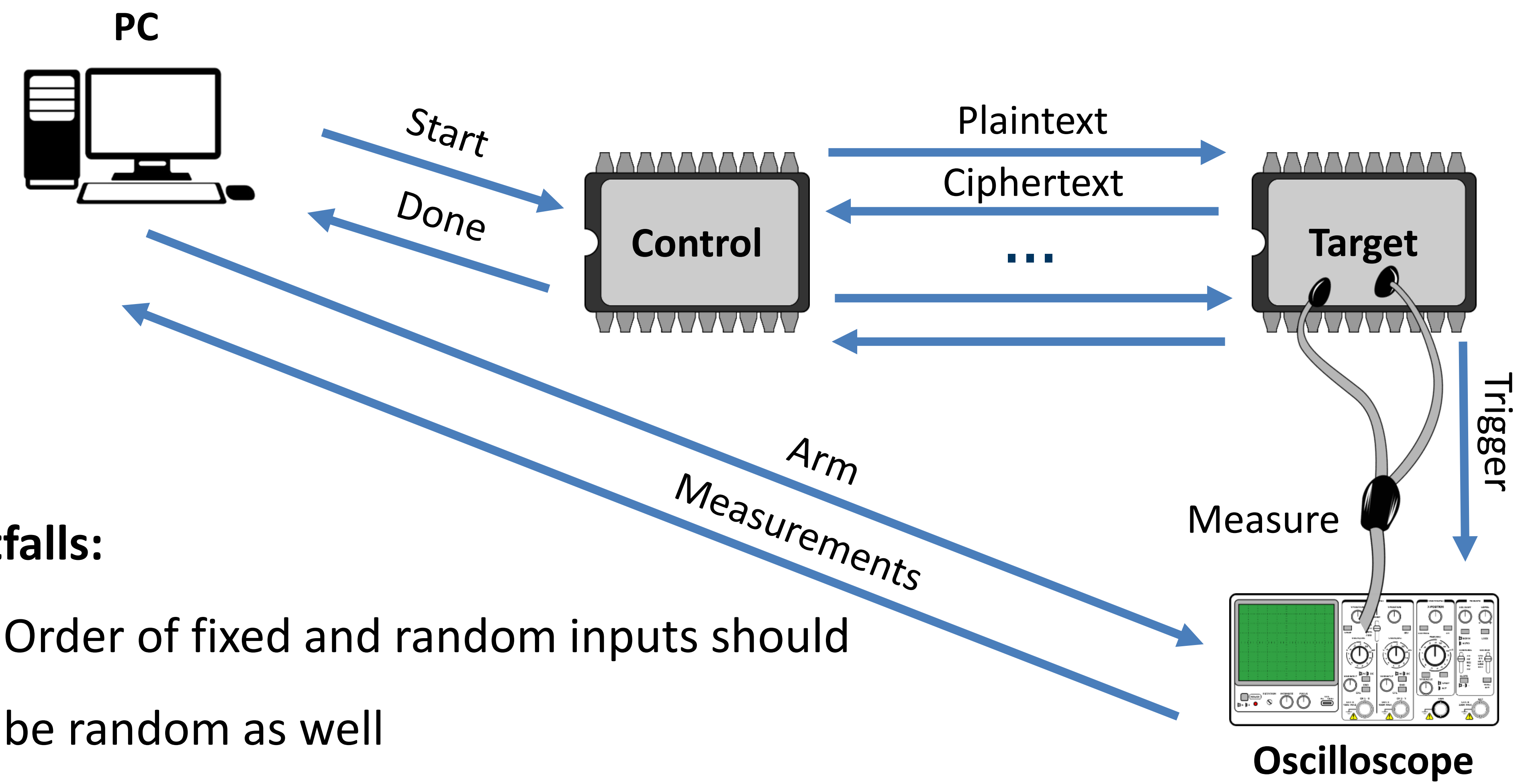
In general: $x' = (x - \mu)^d$

In some cases: $x' = \left(\frac{x-\mu}{s}\right)^d$

Correct Measurement

- Setup
- Case Study: Microcontroller
- Case Study: FPGA

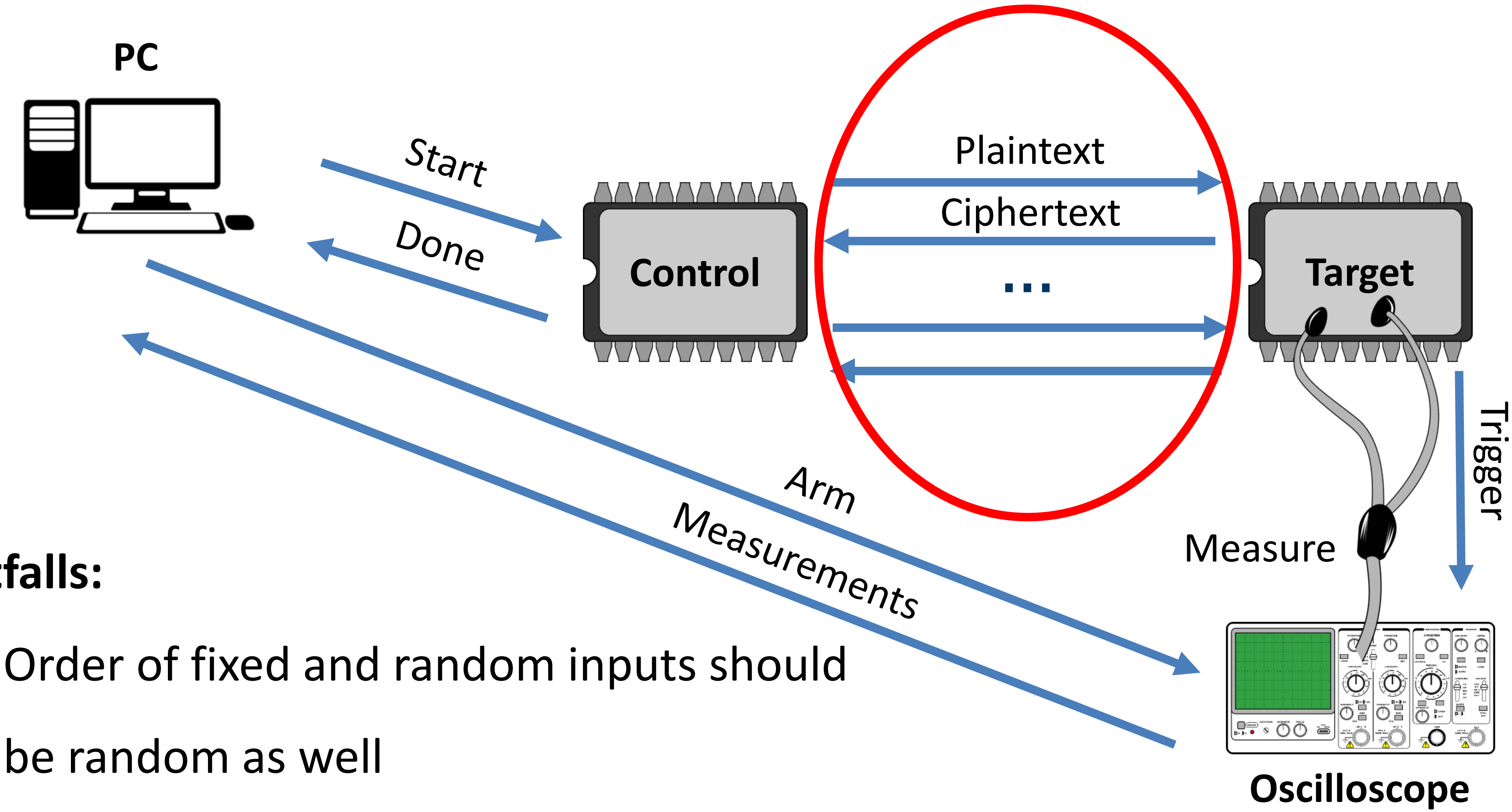
Correct Measurement Setup



Pitfalls:

- Order of fixed and random inputs should be random as well
- Communication between **Control** and **Target** should be masked (if possible)

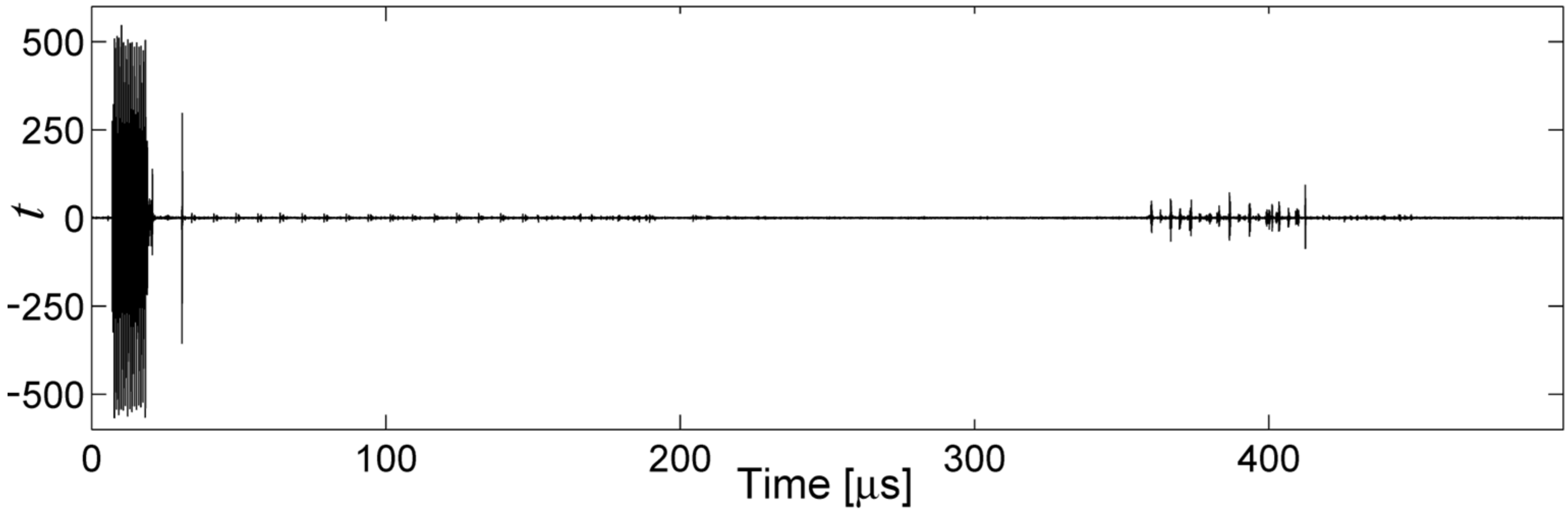
Correct Measurement Setup



Pitfalls:

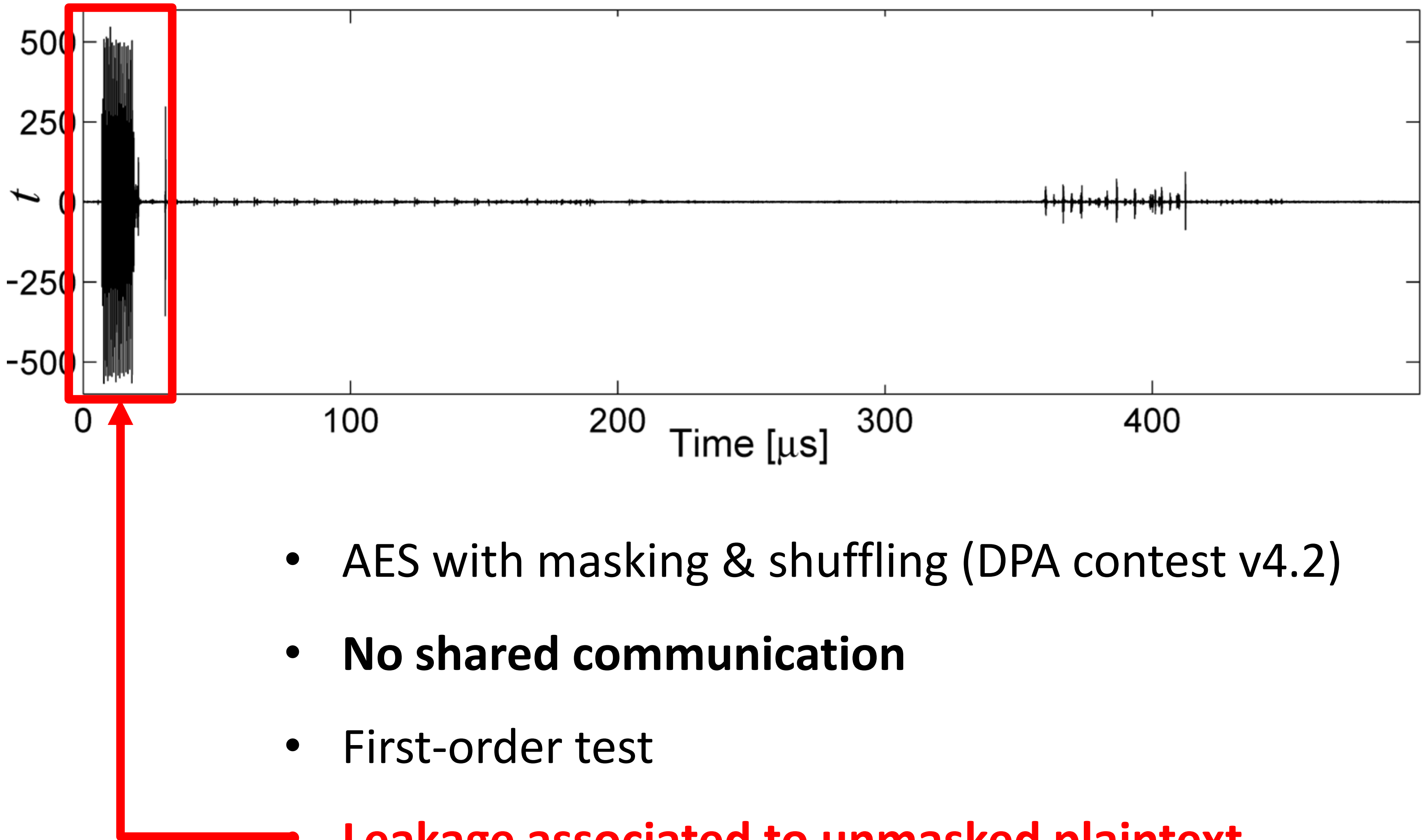
- Order of fixed and random inputs should be random as well
- Communication between **Control** and **Target** should be masked (if possible)

Correct Measurement CS: Microcontroller



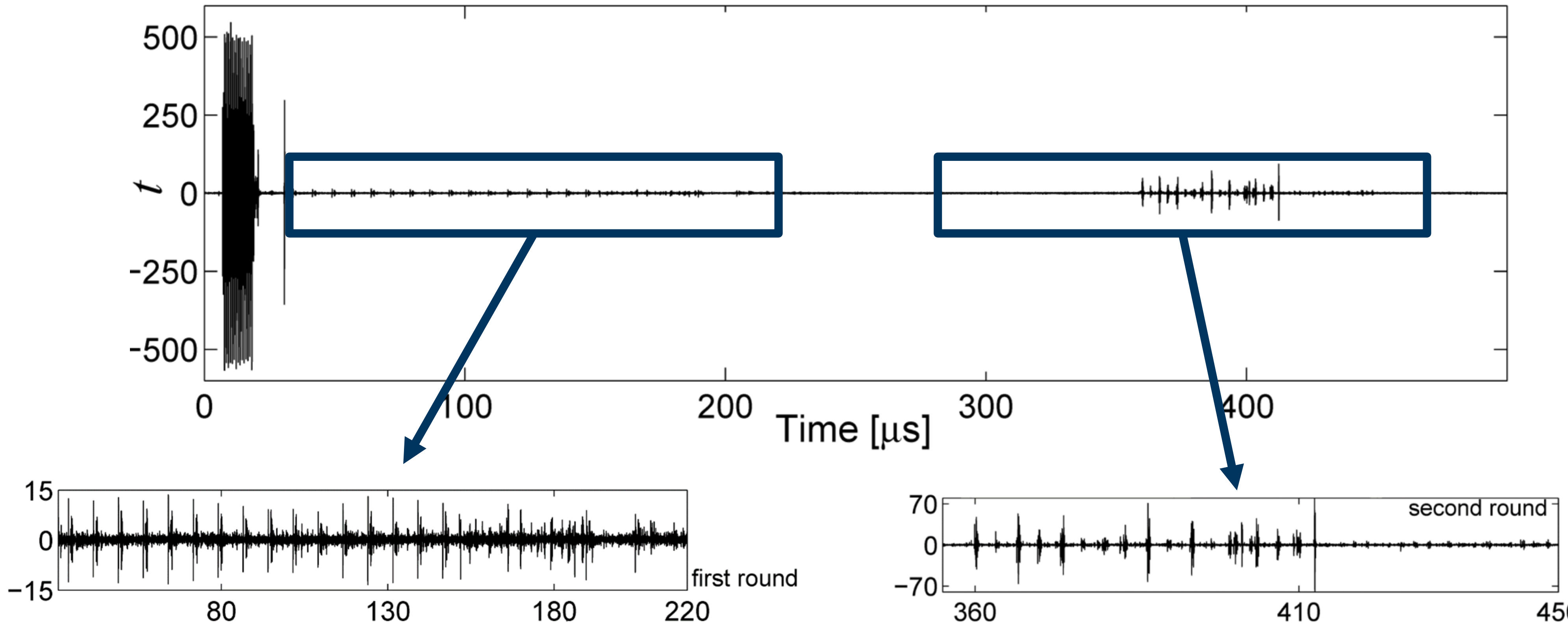
- AES with masking & shuffling (DPA contest v4.2)
- **No shared communication**
- First-order test

Correct Measurement CS: Microcontroller



- AES with masking & shuffling (DPA contest v4.2)
- **No shared communication**
- First-order test
- **Leakage associated to unmasked plaintext**

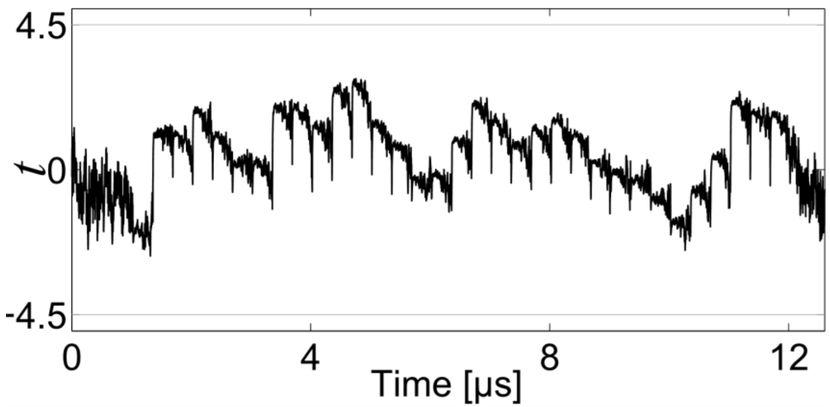
Correct Measurement CS: Microcontroller



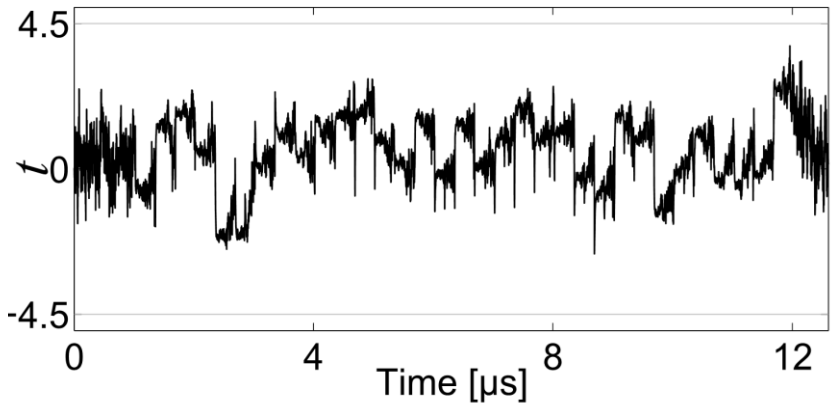
Detectable first order leakage

Correct Measurement CS: FPGA

First Order



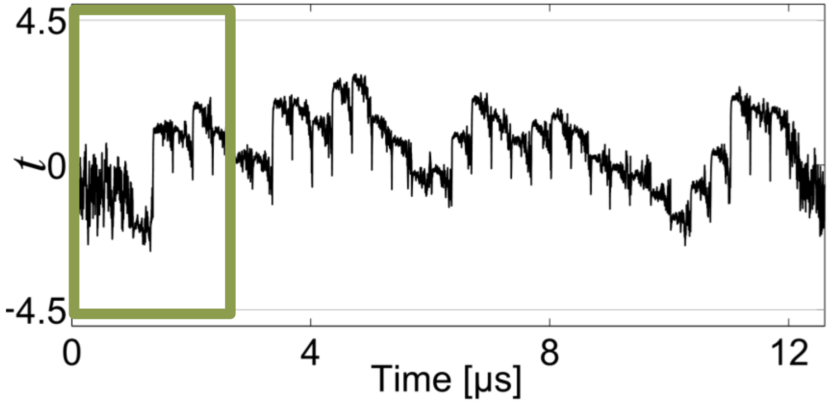
Second Order



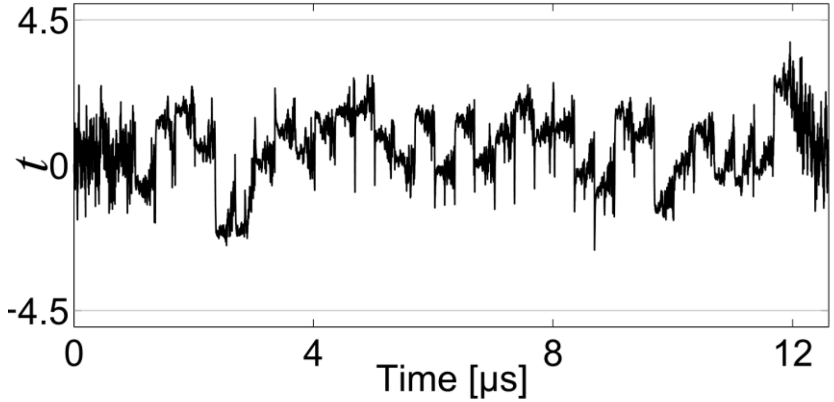
A note on the security of Higher-Order Threshold Implementations
Oscar Reparaz, ePrint Report 2015/001

Correct Measurement CS: FPGA

First Order

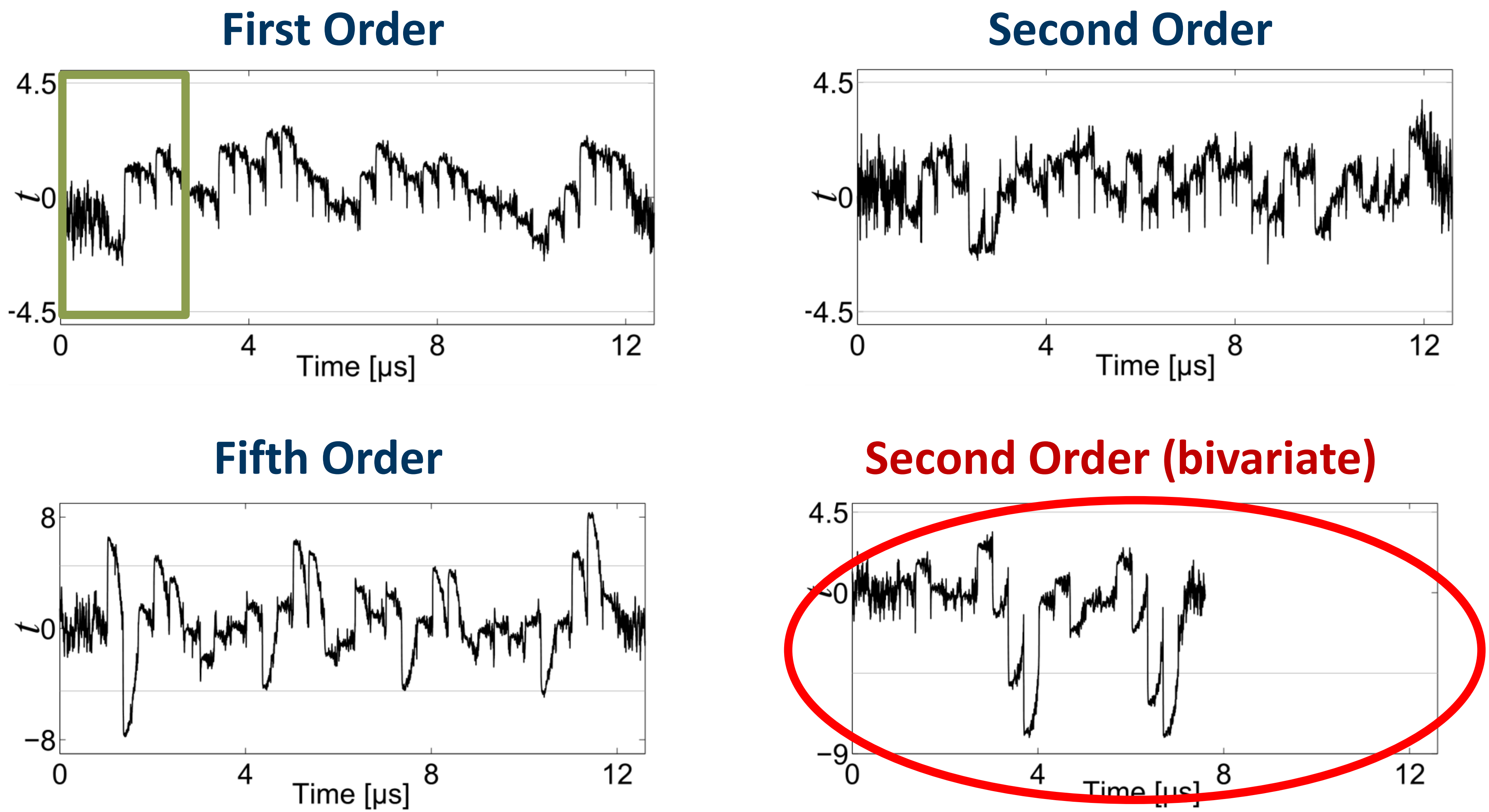


Second Order



A note on the security of Higher-Order Threshold Implementations
Oscar Reparaz, ePrint Report 2015/001

Correct Measurement CS: FPGA

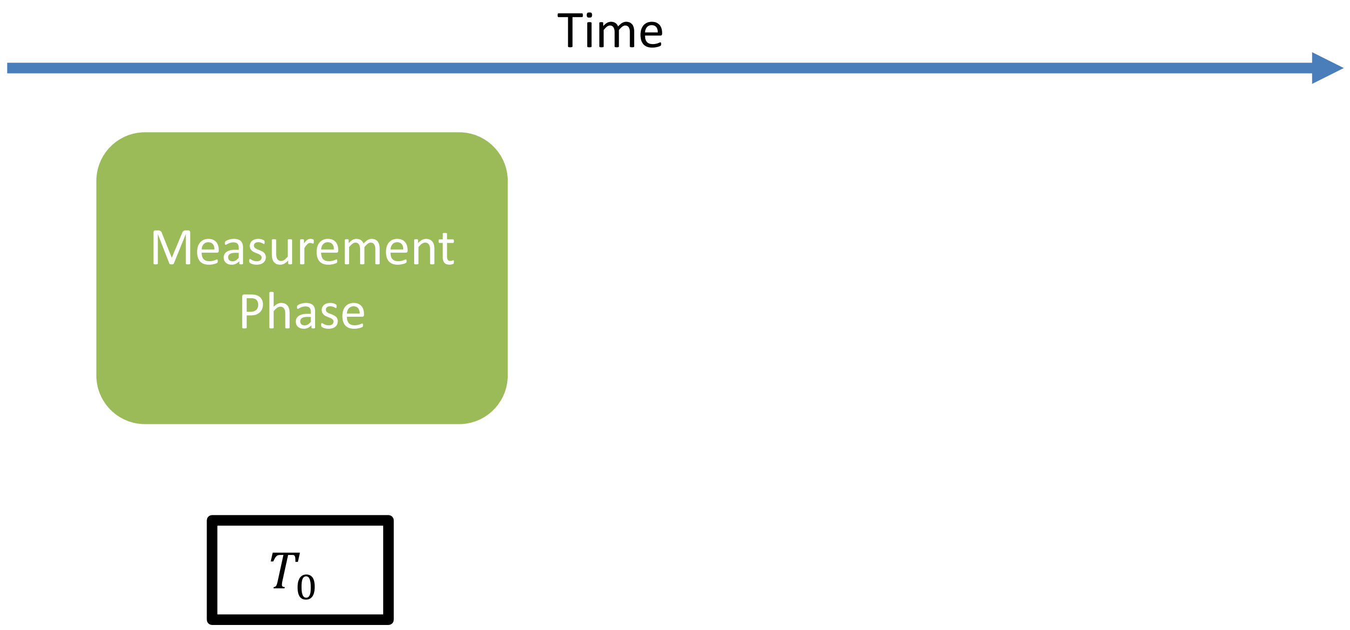


A note on the security of Higher-Order Threshold Implementations
Oscar Reparaz, ePrint Report 2015/001

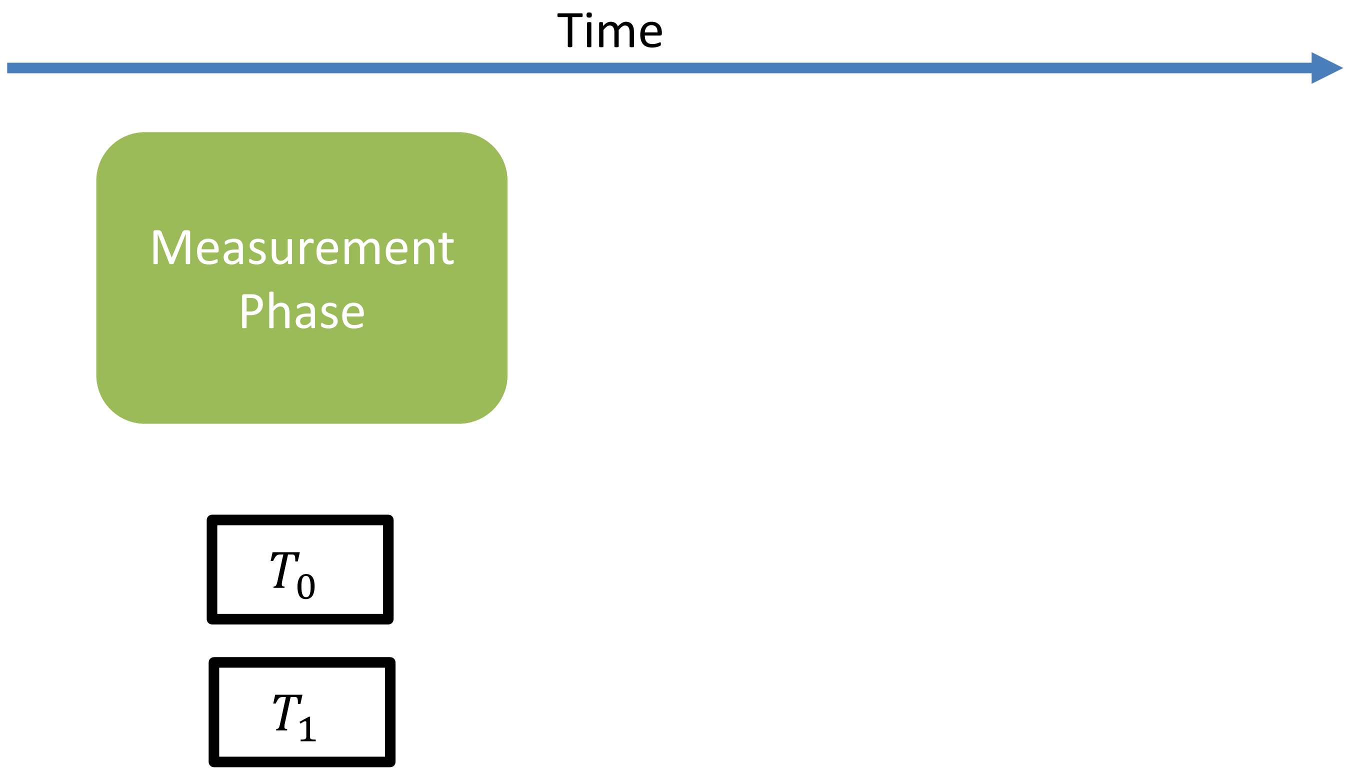
Efficient Computation

- Classical Approach
- Incremental
- Multivariate
- Parallelization

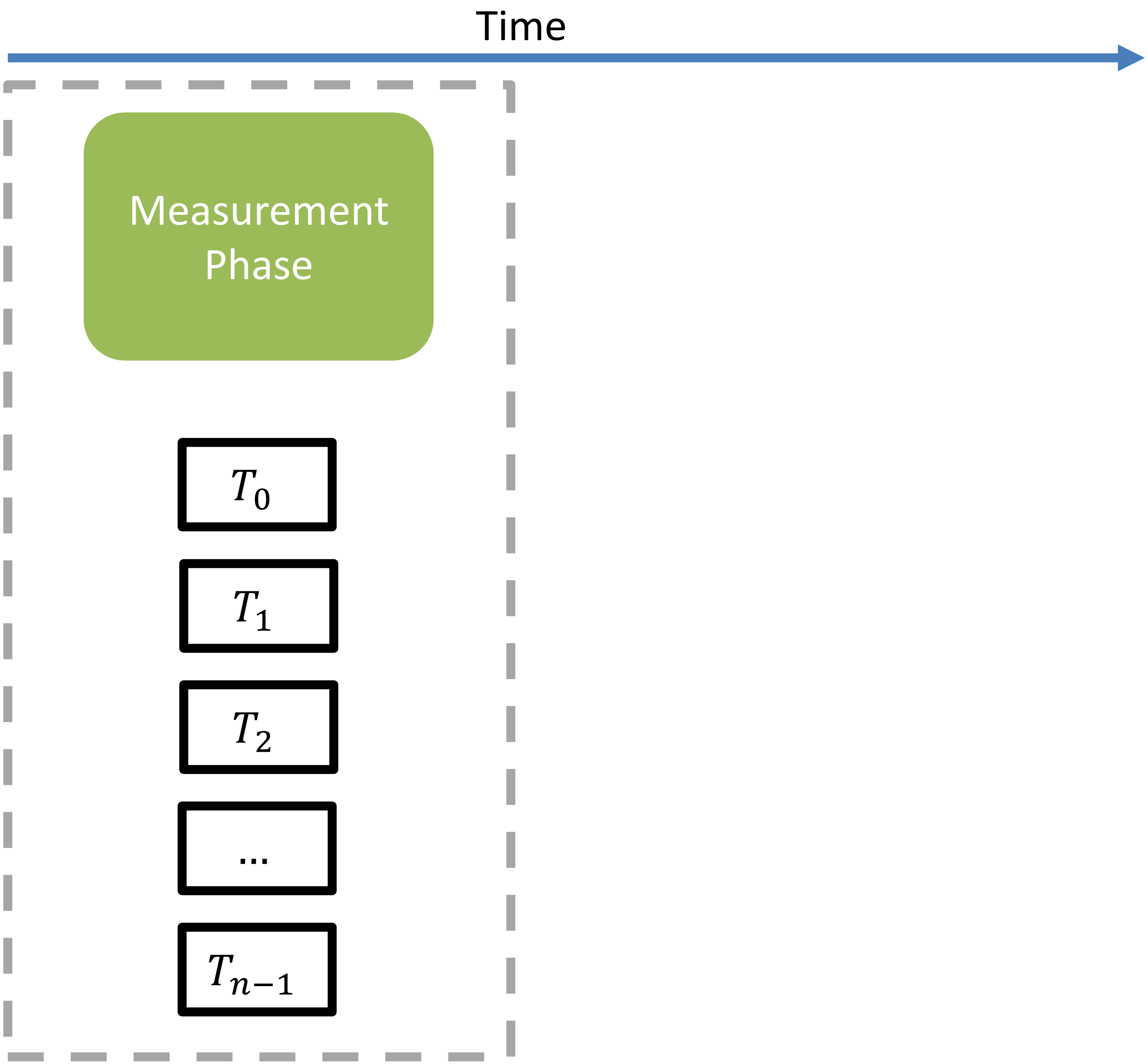
Efficient Computation Classical Approach



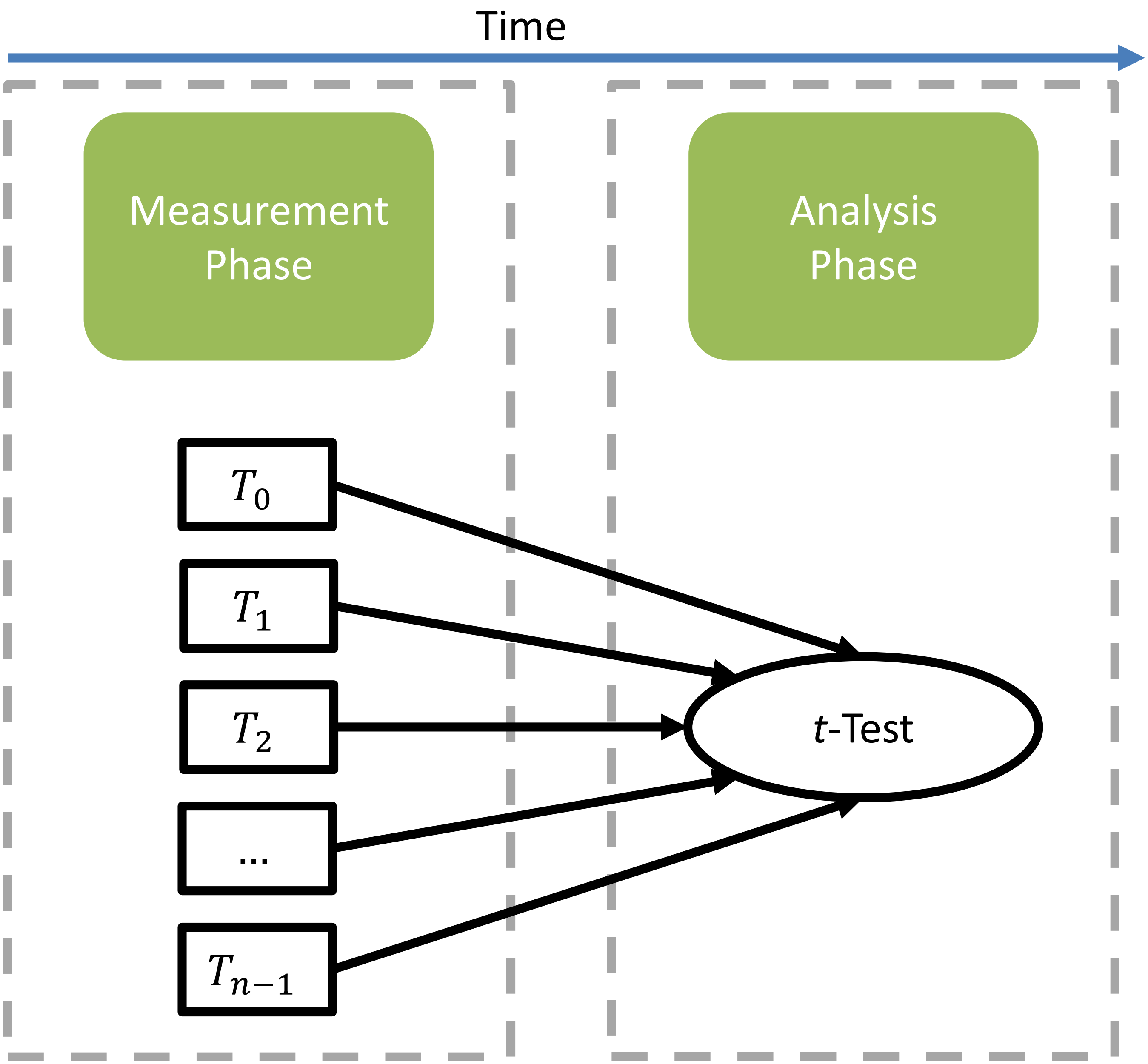
Efficient Computation Classical Approach



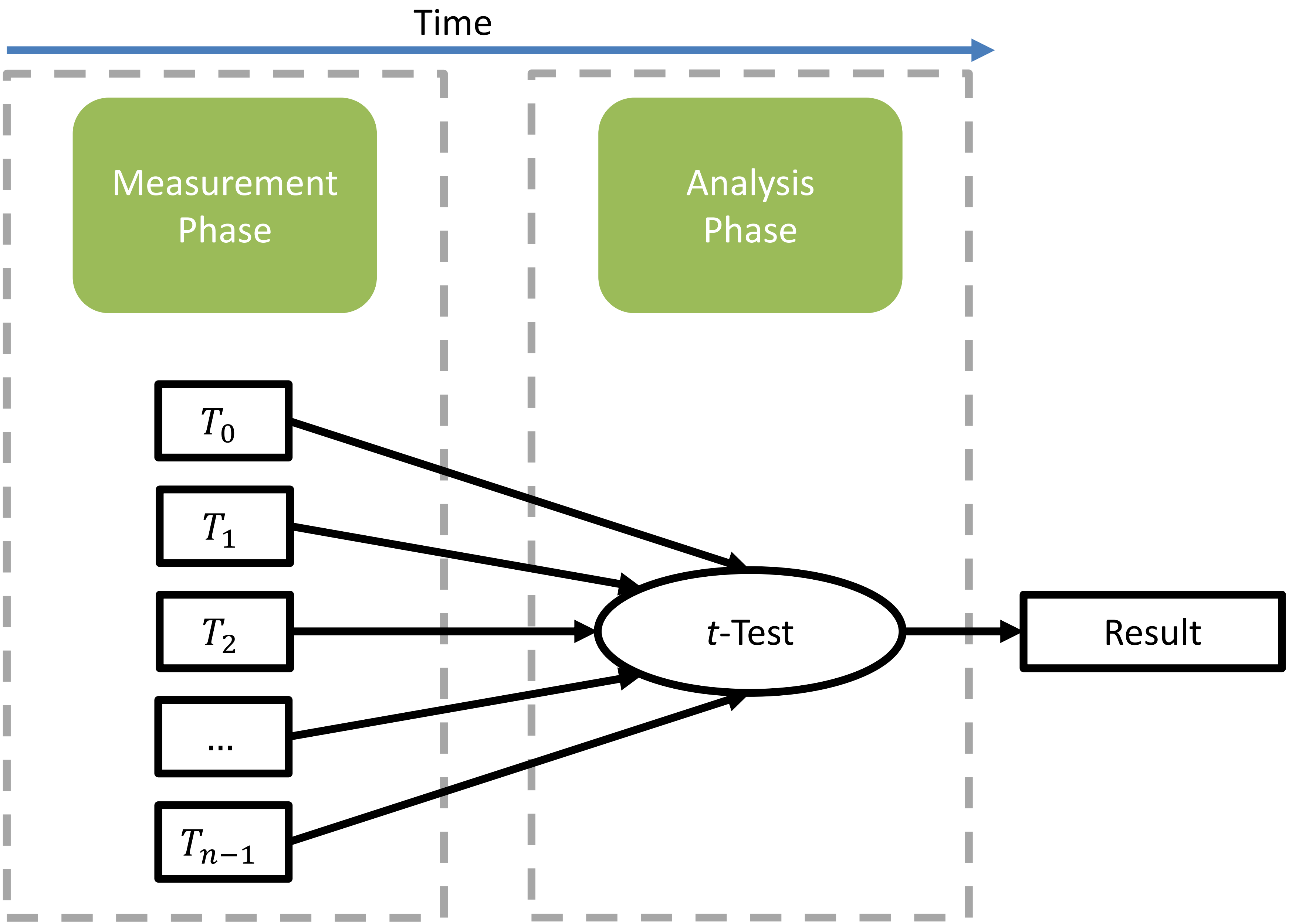
Efficient Computation Classical Approach



Efficient Computation Classical Approach



Efficient Computation Classical Approach



Efficient Computation Classical Approach

t-Test → $t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$

Requires estimation of: (μ_0, s_0^2) (μ_1, s_1^2)

Reminder:

- $\mu = E(T)$
- $s^2 = E\left((T - \mu)^2\right)$

Efficient Computation Classical Approach

t-Test → $t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$

Reminder:

- $\mu = E(T)$
- $s^2 = E\left((T - \mu)^2\right)$

Requires estimation of: (μ_0, s_0^2) (μ_1, s_1^2)

T_0

T_1

…

T_{n-1}

Efficient Computation Classical Approach

t-Test → $t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$

Requires estimation of: (μ_0, s_0^2) (μ_1, s_1^2)

Reminder:

- $\mu = E(T)$
- $s^2 = E\left((T - \mu)^2\right)$

Pass 1

T_0

T_1

…

T_{n-1}

$\mu = E(T)$

Efficient Computation Classical Approach

t-Test → $\mathrm{t} = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$

Reminder:

- $\mu = E(T)$
- $s^2 = E\left((T - \mu)^2\right)$

Requires estimation of: (μ_0, s_0^2) (μ_1, s_1^2)

T_0

T_1

...

T_{n-1}

Pass 1

$\mu = E(T)$

Pass 2

$s^2 = E\left((T - \mu)^2\right)$

Efficient Computation Classical Approach

t-Test → $t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$

Reminder:

- $\mu = E(T)$
- $s^2 = E\left((T - \mu)^2\right)$

Requires estimation of: (μ_0, s_0^2) (μ_1, s_1^2)

T_0, T_1, …, T_{n-1}

Pass 1	Pass 2	*Pass 3*
$\mu = E(T)$	$s^2 = E\left((T - \mu)^2\right)$	*Required for certain higher-order tests*

Efficient Computation Classical Approach

Problems:

1) Measurement phase need to be completed
2) All measurements need to be stored
3) Traces need to be loaded multiple times

Solution: ***Incremental Computation***

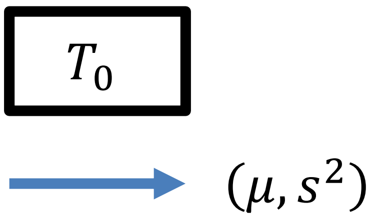
Efficient Computation Incremental

Idea: Update intermediate values for each new trace

T_0

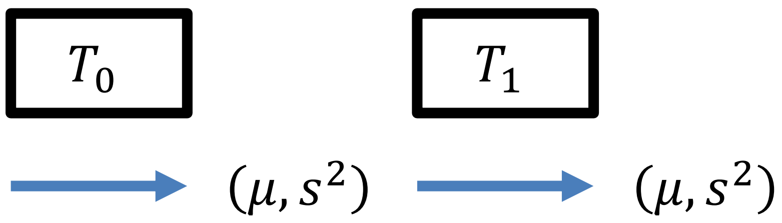
Efficient Computation Incremental

Idea: Update intermediate values for each new trace



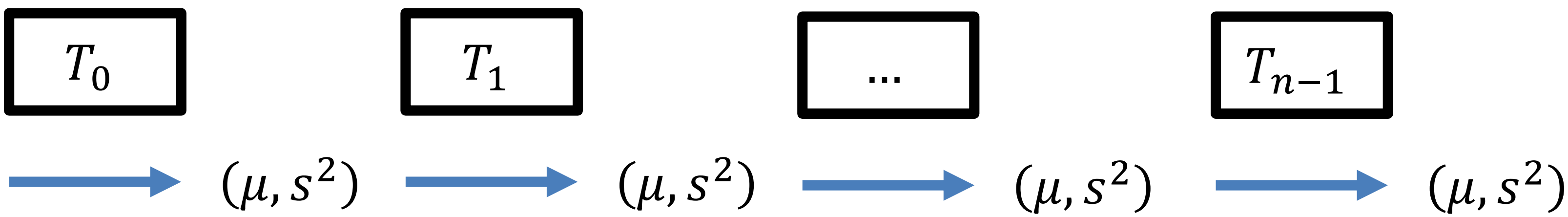
Efficient Computation Incremental

Idea: Update intermediate values for each new trace



Efficient Computation Incremental

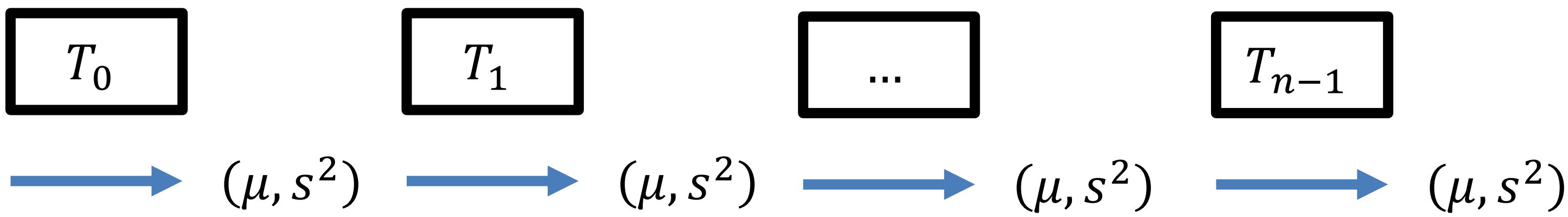
Idea: Update intermediate values for each new trace



Higher-order tests require the computation of additional values

Efficient Computation Incremental

Idea: Update intermediate values for each new trace



Higher-order tests require the computation of additional values

Advantages:

1) Can be run in parallel to measurement phase
2) Does not require that all measurements are stored
3) Loads each trace only once

Efficient Computation Incremental

Problem: Computation of intermediate values

Efficient Computation Incremental

Problem: Computation of intermediate values

Approach 1: Use raw moments

d[th]-order raw moment: $M_d = E\left(T^d\right)$

Given: M_1 M_2

Compute: $\mu = M_1$ $s^2 = M_2 - (M_1)^2$

Efficient Computation Incremental

Problem: Computation of intermediate values

Approach 1: Use raw moments

d^th^-order raw moment: $M_d = E\left(T^d\right)$

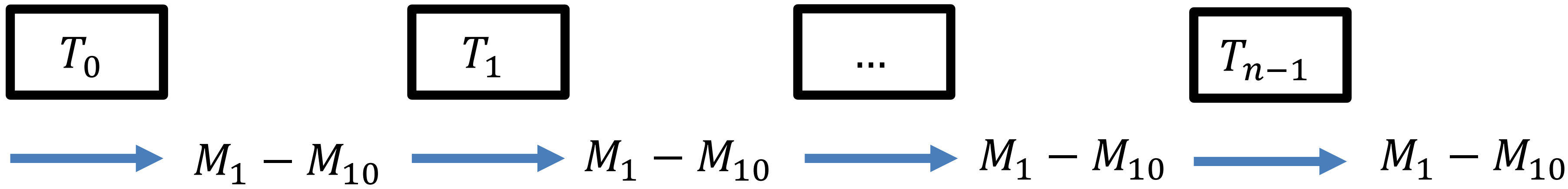
Given: M_1 M_2

Compute: $\mu = M_1$ $s^2 = M_2 - (M_1)^2$

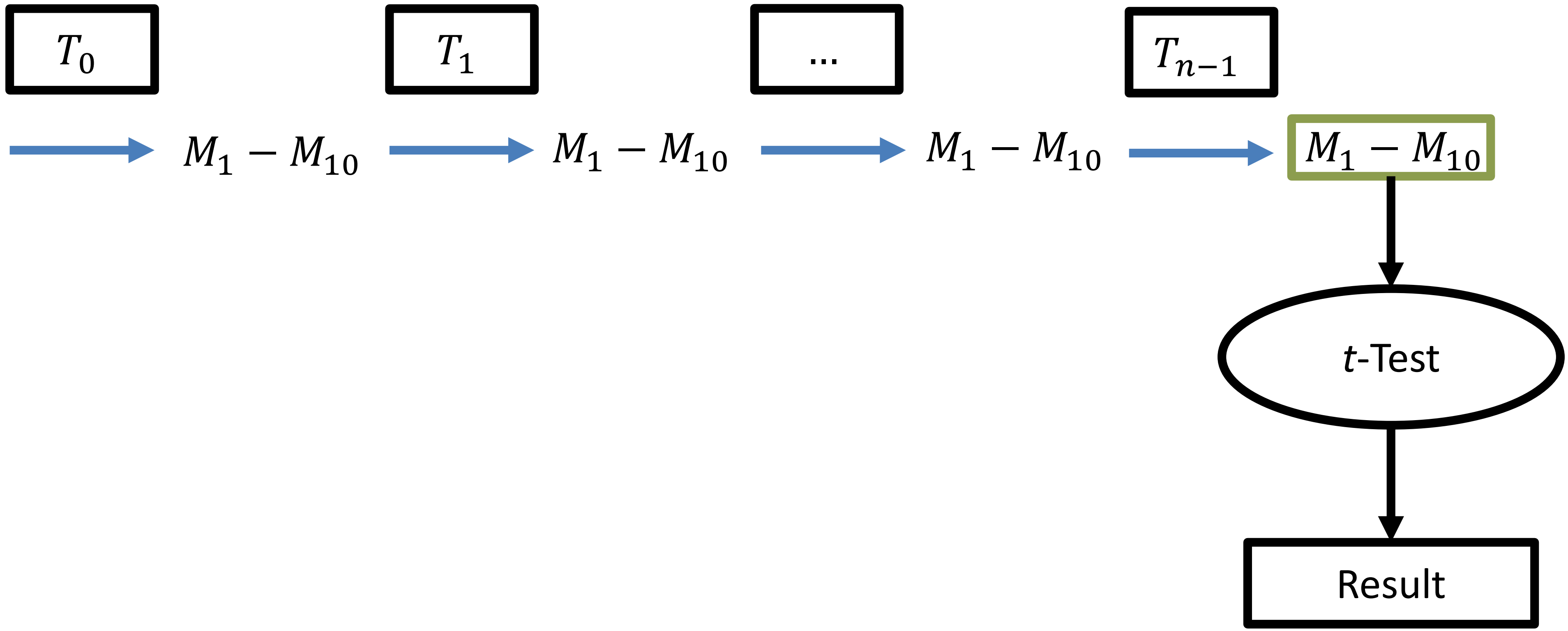
Higher-order test require additional moments

Example: Univariate 1st-5th order tests require $M_1 - M_{10}$

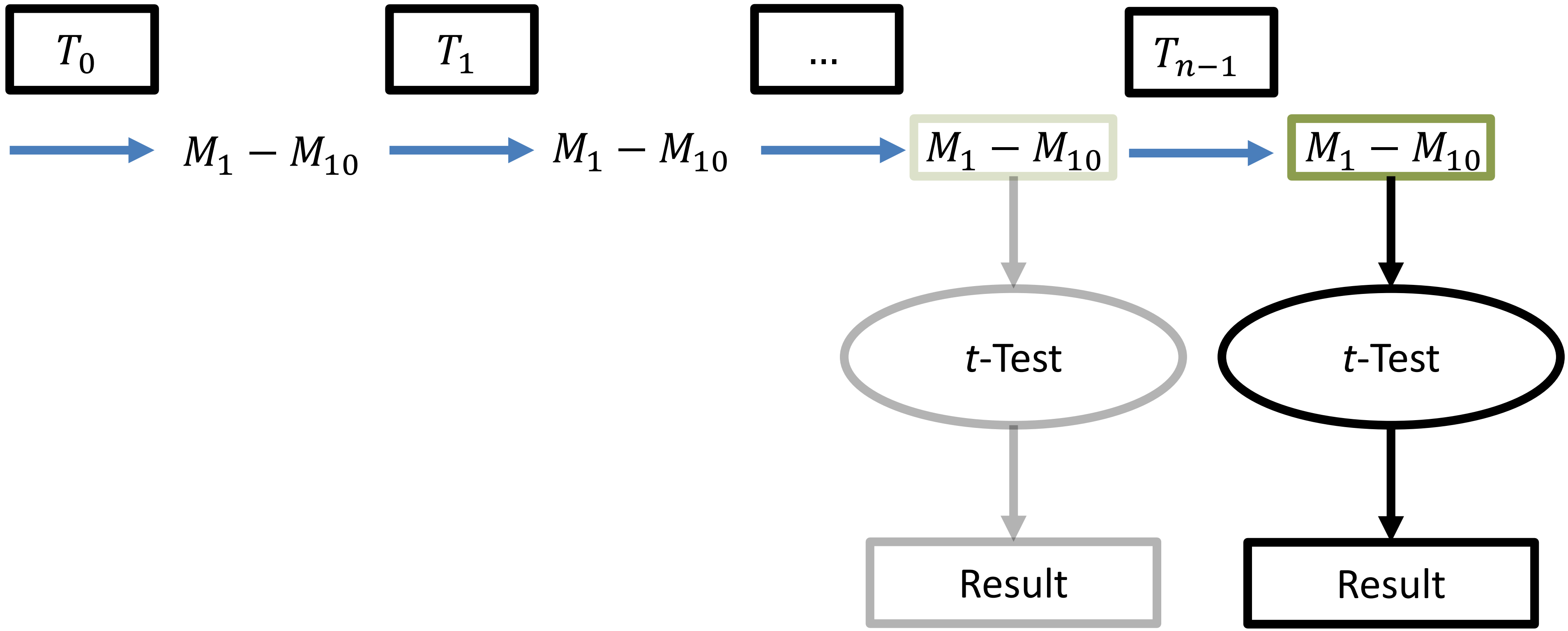
Efficient Computation Incremental



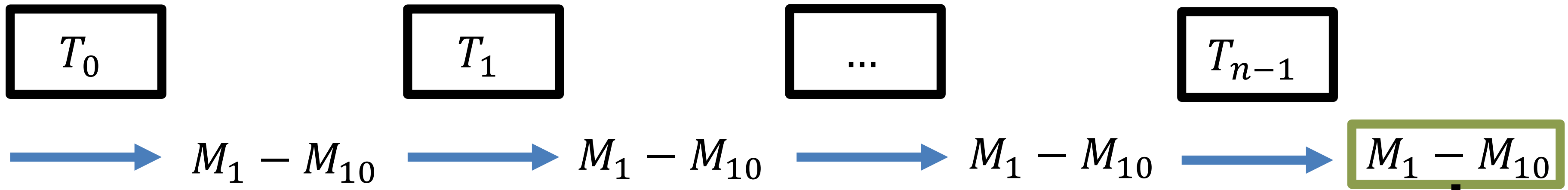
Efficient Computation Incremental



Efficient Computation Incremental



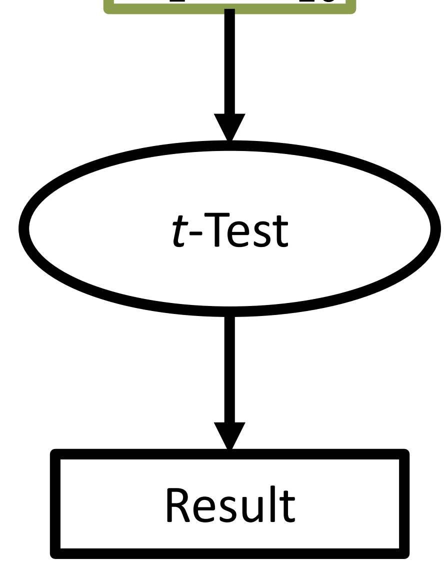
Efficient Computation Incremental



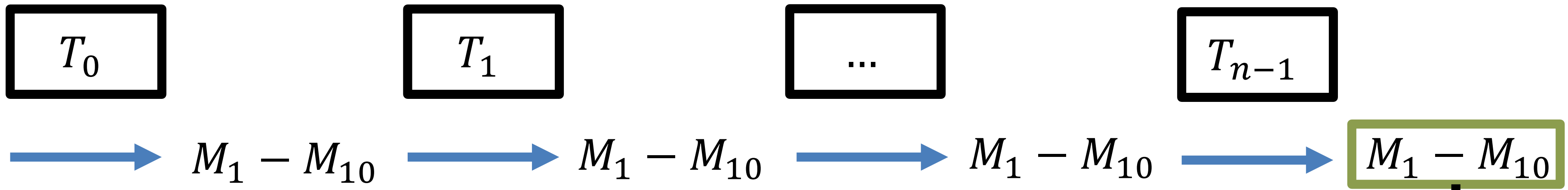
Easy to find update formulas for:

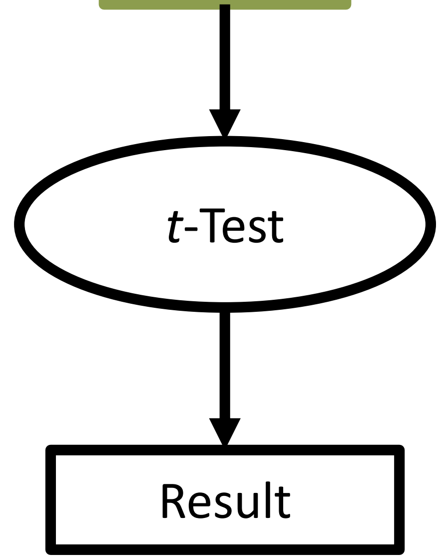
$$M_d = \sum_{i=0}^{n-1} \frac{(T_i)^d}{n}$$

Problem: Numerical unstable for large number of traces



Efficient Computation Incremental





Easy to find update formulas for:

$$M_d = \sum_{i=0}^{n-1} \frac{(T_i)^d}{n}$$

Problem: Numerical unstable for large number of traces

Example: Computation of variance based on simulations (100M traces) with $\mathcal{N}(100,25)$

Method	Order 1	Order 2	Order 3	Order 4	Order 5
3-Pass	25.08399	1258.18874	15.00039	96.08342	947.25523
Raw	25.08399	1258.14132	14.49282	-1160.83799	-1939218.83401

Efficient Computation Incremental

Approach 2: Use *central* moments (and M_1)

d^th-order central moment: $CM_d = E\left((T-\mu)^d\right)$

Given: M_1 CM_2

Compute: $\mu = M_1$ $s^2 = CM_2$

Efficient Computation Incremental

Approach 2: Use *central* moments (and M_1)

d^{th}-order central moment: $CM_d = E\left((T-\mu)^d\right)$

Given: M_1 CM_2

Compute: $\mu = M_1$ $s^2 = CM_2$

Not that easy to find update formulas for:

$$CM_d = \sum_{i=0}^{n-1} \frac{(T_i - \mu)^d}{n}$$

Multivariate tests require adjusted formulas

Efficient Computation Incremental

Incremental formulas for tests at arbitrary orders can be found in the paper.

Comparison to the raw moments approach:

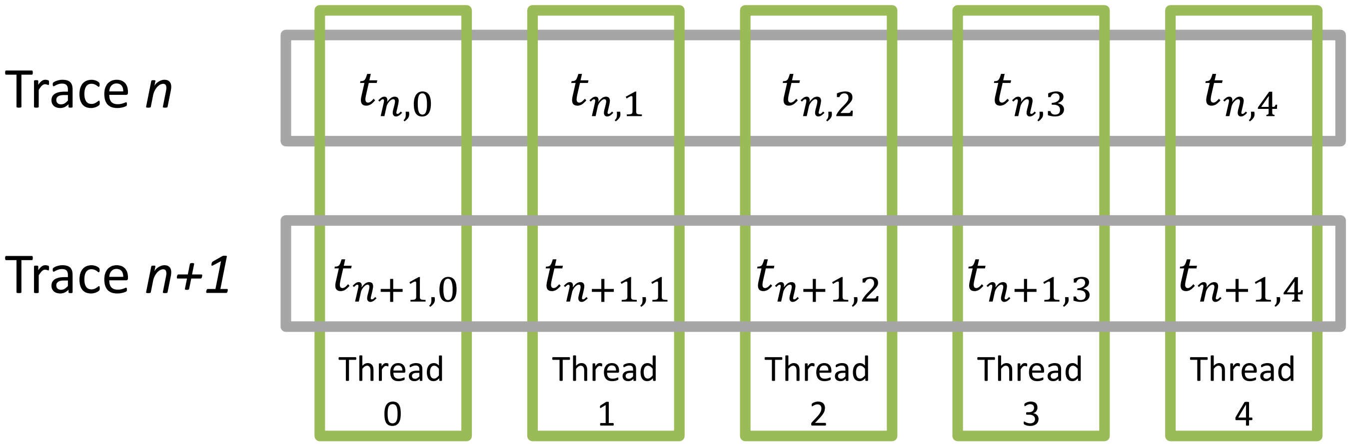
- Slightly higher computational effort
- Less numerical problems, higher accuracy

Method	Order 1	Order 2	Order 3	Order 4	Order 5
3-Pass	25.08399	1258.18874	15.00039	96.08342	947.25523
Raw	25.08399	1258.14132	14.49282	-1160.83799	-1939218.83401
Ours	25.08399	1258.18874	15.00039	96.08342	947.25523

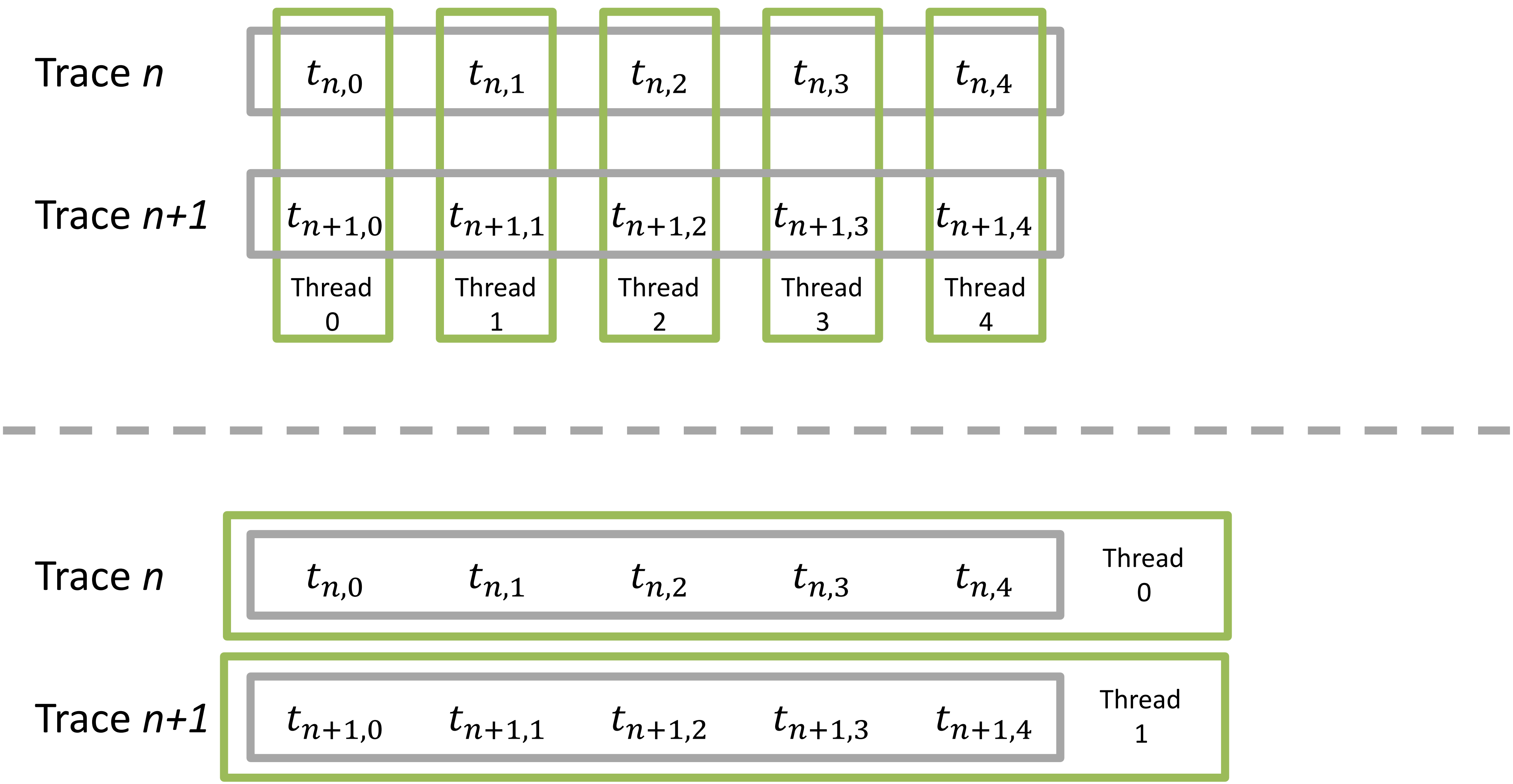
Efficient Computation Parallelization

Trace *n*	$t_{n,0}$	$t_{n,1}$	$t_{n,2}$	$t_{n,3}$	$t_{n,4}$
Trace *n+1*	$t_{n+1,0}$	$t_{n+1,1}$	$t_{n+1,2}$	$t_{n+1,3}$	$t_{n+1,4}$

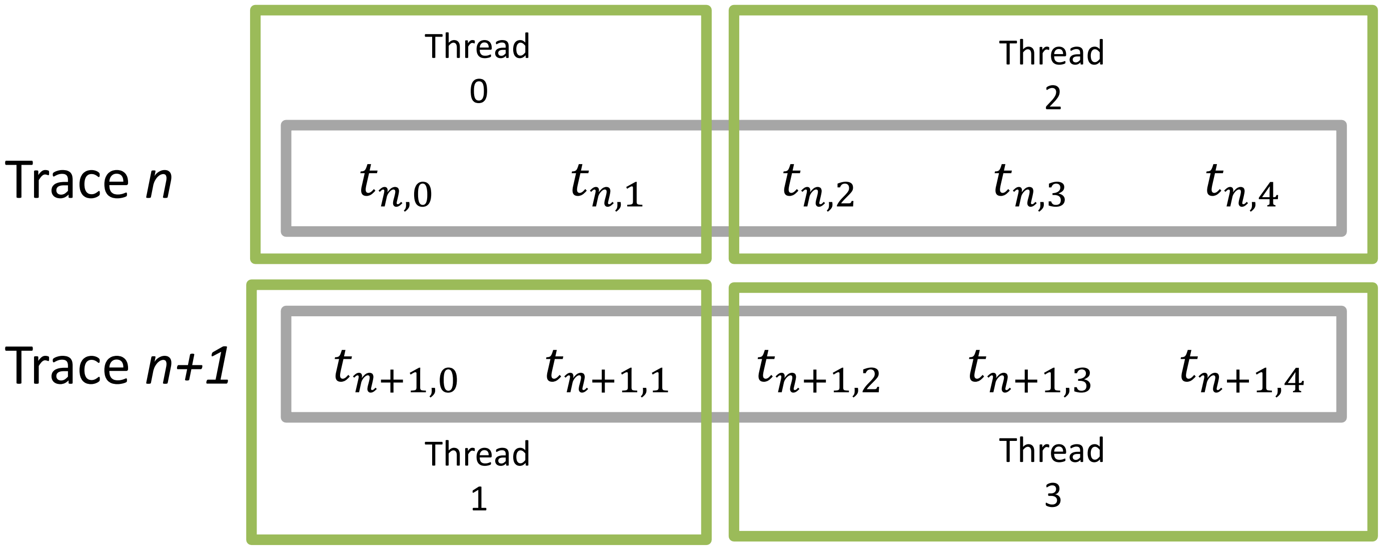
Efficient Computation Parallelization



Efficient Computation Parallelization



Efficient Computation Parallelization



Example:

- 1st-5th order *t*-test
- 100,000,000 traces (each with 3,000 sample points)
- 9h on 2 x *Intel Xeon X5670* CPUs @ 2.93 GHz (24 hyper-threading cores)

Conclusion

- Recommendations
- Summary
- Future Work

Conclusion Recommendations

Fixed vs. random:

- DUT with *masking* countermeasure
- With masked communication

Semi-fixed vs. random:

- DUT with *hiding* countermeasure
- Without masked communication

Specific t-test:

- DUT with *no* countermeasures
- Failed in former non-specific tests
- Identify suitable intermediate values for key recovery

Conclusion Summary

- Testing based on the *t*-test is simple and fast
- Has become popular in recent years

Things to consider:

  - Correct measurement phase is critical
  - Analysis phase can be strongly optimized
  - Higher-order testing easily possible

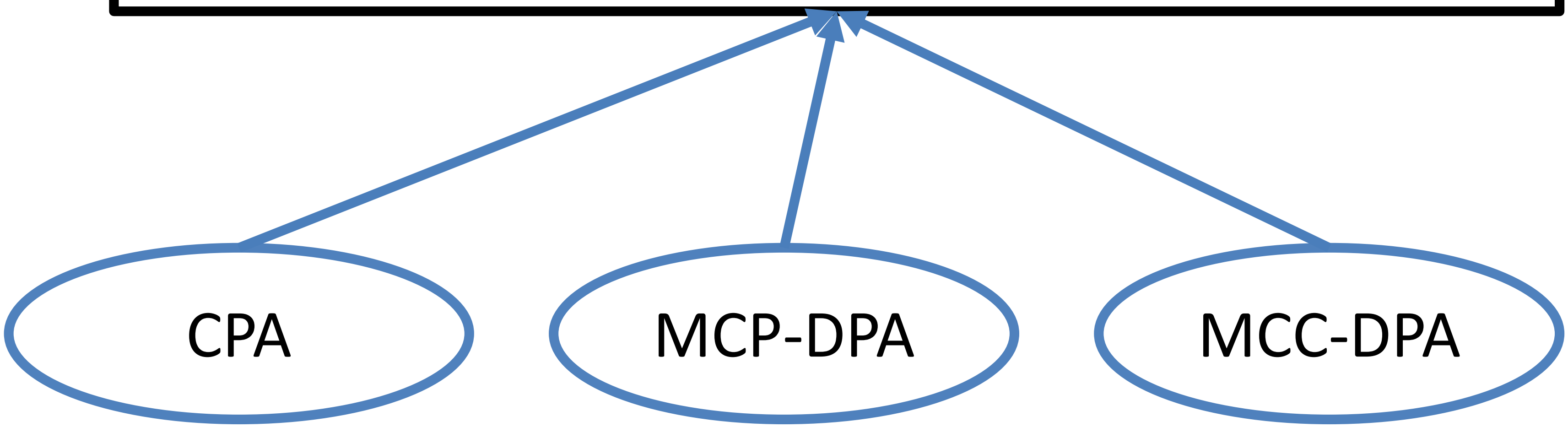
Additional important aspects:

  - Alignment and signal processing is necessary
  - Finding of points of interest

Conclusion Future Work

- Incremental computing for other attacks/evaluation techniques

Robust and One-Pass Parallel Computation of Correlation-Based Attacks at Arbitrary Order
Tobias Schneider, Amir Moradi, Tim Güneysu, ePrint Report 2015/571



Thanks for Listening!

Any Questions?