

Applied Cryptology

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
(csdsp@bristol.ac.uk)

September 5, 2025

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
 - ▶ the associated notes page may be pre-populated with extra, written explanation of material covered in lecture(s), plus
 - ▶ anything with a “grey’ed out” header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

► **Agenda:** explore **implementation attacks** via

1. an “in theory”, i.e., concept-oriented perspective,
 - 1.1 explanation,
 - 1.2 justification,
 - 1.3 formalisation.
- and
2. an “in practice”, i.e., example-oriented perspective,
 - 2.1 attacks,
 - 2.2 countermeasures.

► **Caveat!**

~ 2 hours \Rightarrow introductory, and (very) selective (versus definitive) coverage.

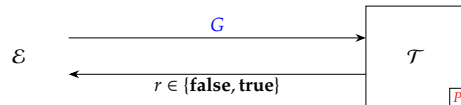
Notes:

Part 1.1: in theory (1)

Explanation

► **Scenario:**

- given the following interaction between an **attacker** \mathcal{E} and a **target** \mathcal{T}



- and noting that

- the password P has $|P|$ characters in it,
- each character in G and P is assumed to be from a known alphabet

$$A = \{'a', 'b', \dots, 'z'\}$$

such that $|A| = 26$,

- how can \mathcal{E} mount a successful attack, i.e., input a guess G matching P ?

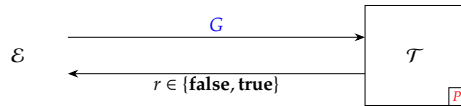
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

Attack ($P = \text{"pencil"}$)



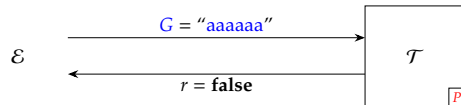
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

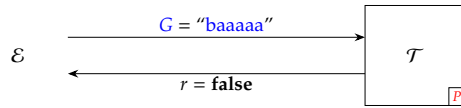
Attack ($P = \text{"pencil"}$)



Notes:

- **Idea: brute-force attack** (i.e., try every G).

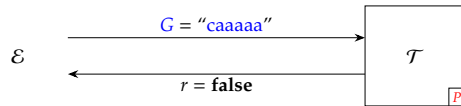
Attack ($P = \text{"pencil"}$)



Notes:

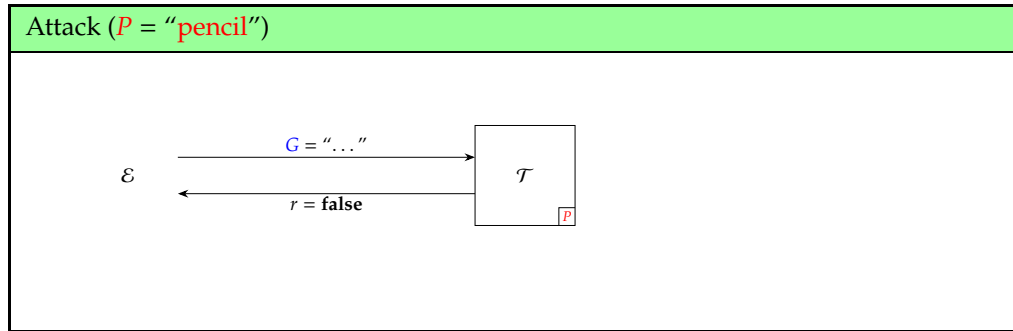
- **Idea: brute-force attack** (i.e., try every G).

Attack ($P = \text{"pencil"}$)



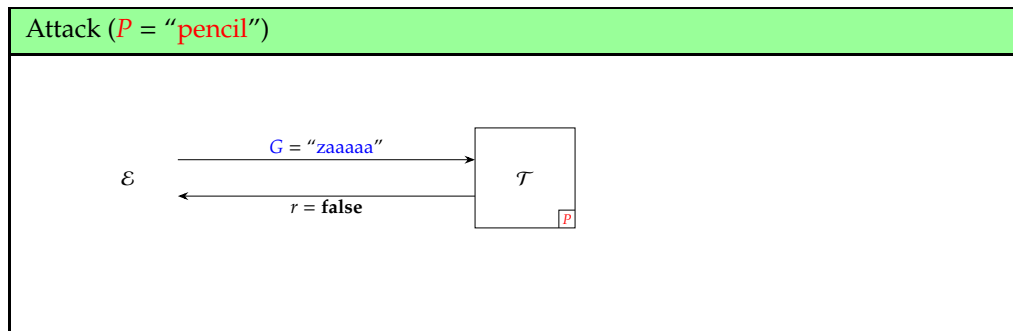
Notes:

- **Idea: brute-force attack** (i.e., try every G).



Notes:

- **Idea: brute-force attack** (i.e., try every G).



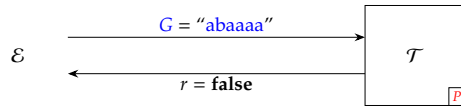
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

Attack ($P = \text{"pencil"}$)



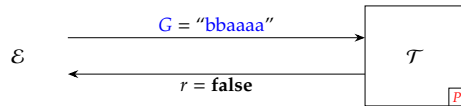
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

Attack ($P = \text{"pencil"}$)

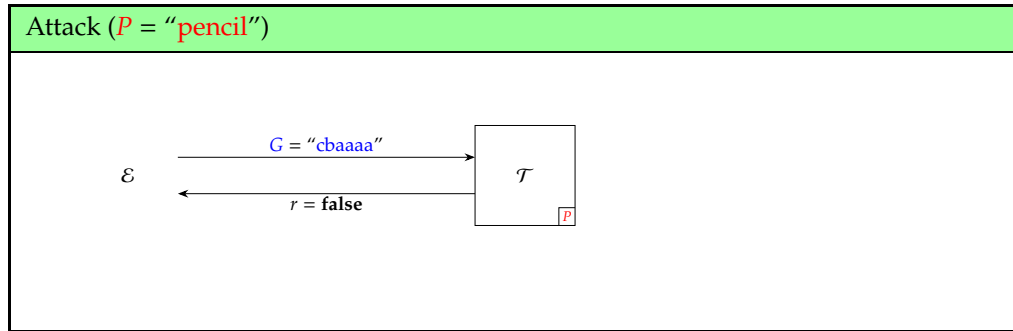


Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

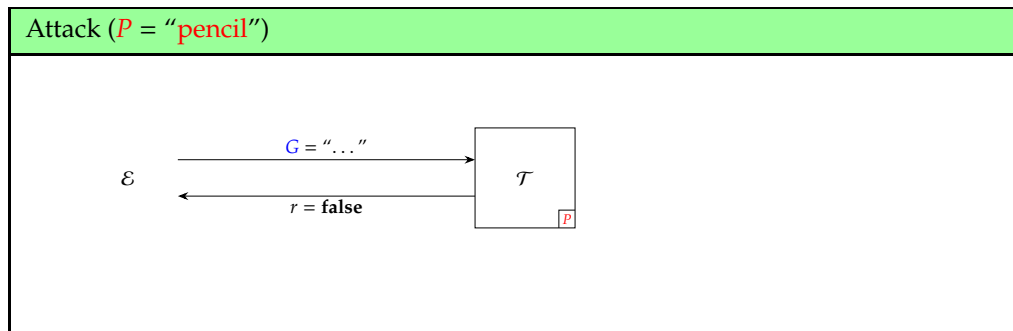


Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).



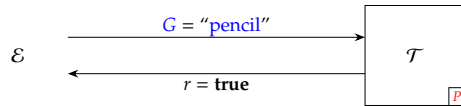
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: brute-force attack** (i.e., try every G).

Attack ($P = \text{"pencil"}$)



\therefore if we play by the rules then

+ve: we always guess a $G = P$

–ve: we need quite a lot of guesses, e.g., for a 6-character lower-case password we'd make

$$26^6 = 308915776$$

in the worst-case

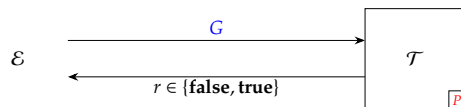
Notes:

Part 1.1: in theory (2)

Explanation

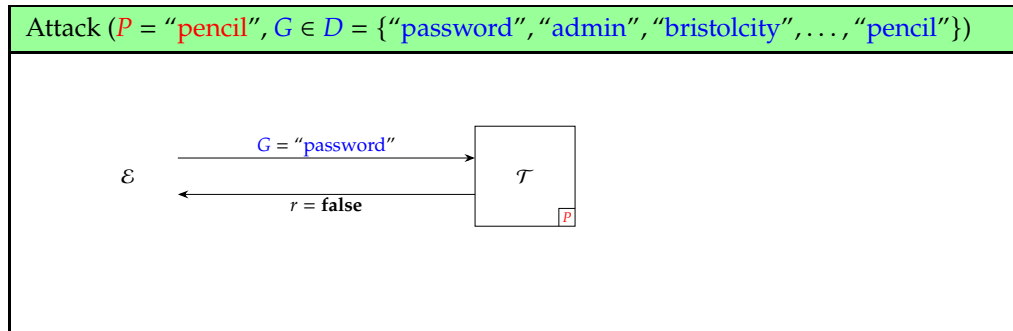
- **Idea: dictionary attack** (i.e., try common G).

Attack ($P = \text{"pencil"}$, $G \in D = \{\text{"password"}, \text{"admin"}, \text{"bristolcity"}, \dots, \text{"pencil"}\}$)



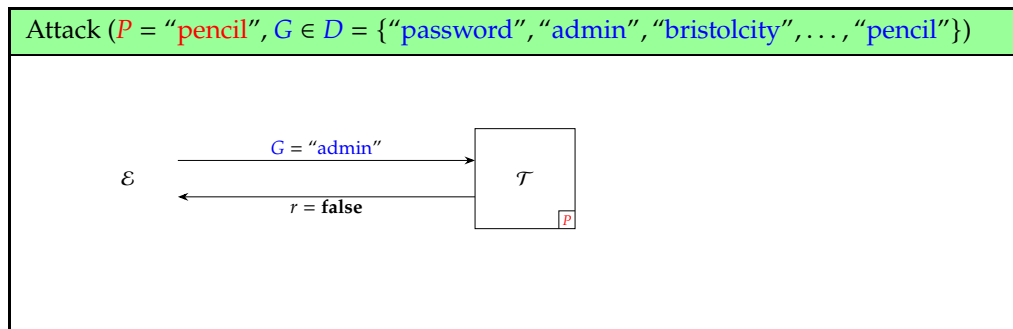
Notes:

- **Idea: dictionary attack** (i.e., try common G).



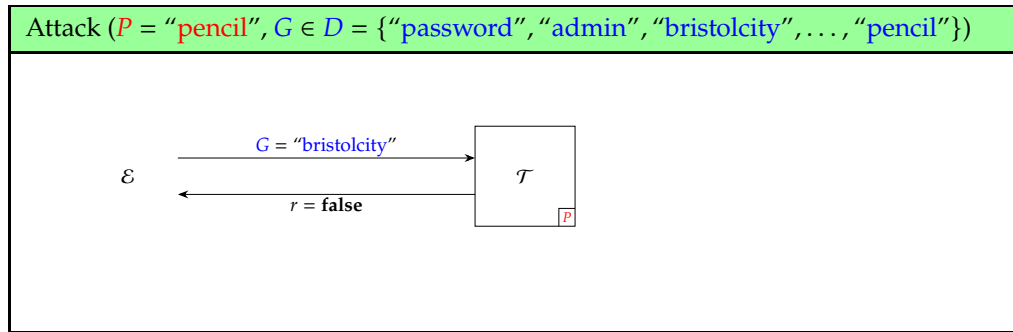
Notes:

- **Idea: dictionary attack** (i.e., try common G).



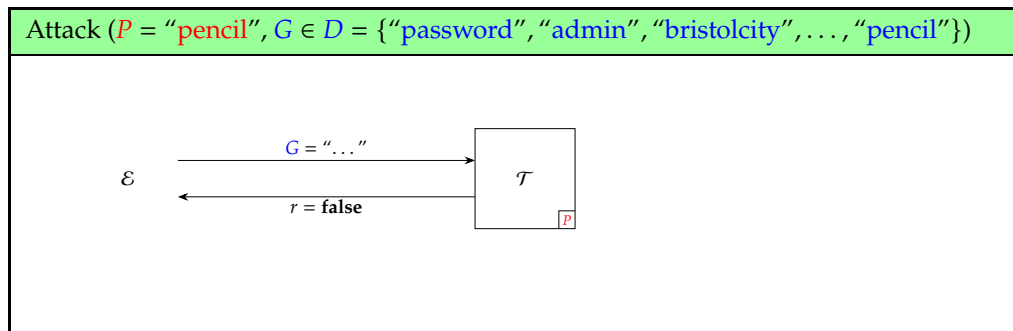
Notes:

- **Idea: dictionary attack** (i.e., try common G).



Notes:

- **Idea: dictionary attack** (i.e., try common G).



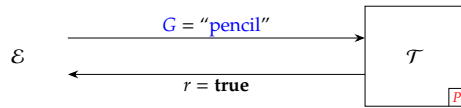
Notes:

Part 1.1: in theory (2)

Explanation

- **Idea: dictionary attack** (i.e., try common G).

Attack ($P = \text{"pencil"} , G \in D = \{\text{"password"}, \text{"admin"}, \text{"bristolcity"}, \dots, \text{"pencil"}\}$)



\therefore if we play by the rules then

–ve: if $P \notin D$, we won't guess a $G = P$

+ve: we need fewer guesses, i.e., $|D|$ in the worst-case

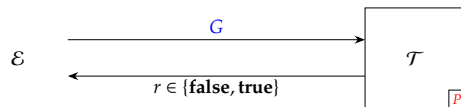
Notes:

Part 1.1: in theory (2)

Explanation

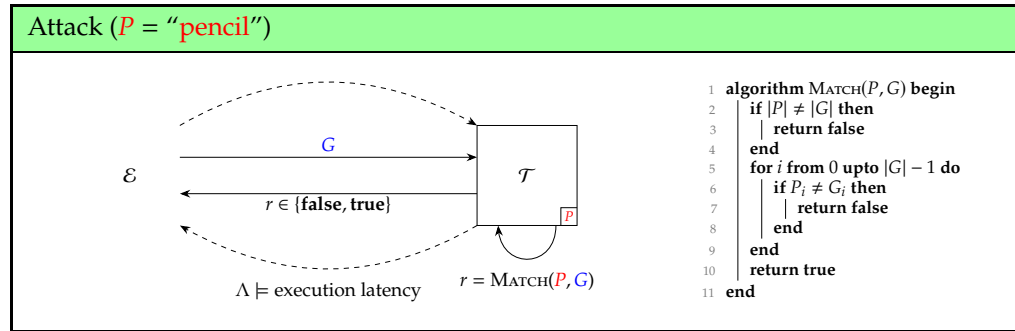
- **Idea: side-channel attack.**

Attack ($P = \text{"pencil"}$)



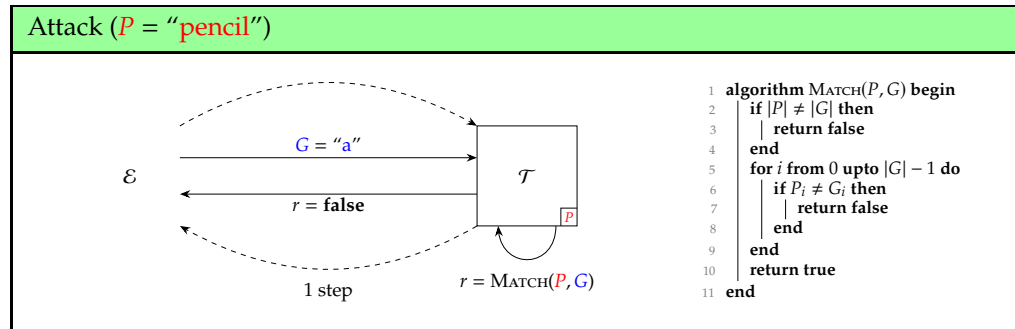
Notes:

► Idea: side-channel attack.



Notes:

► Idea: side-channel attack.

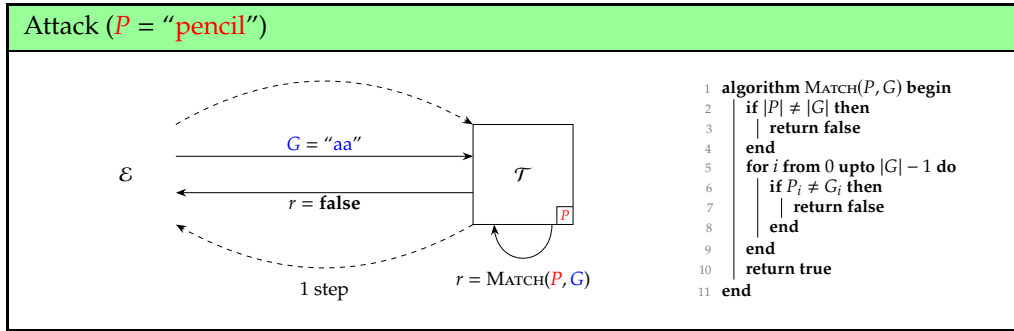


Notes:

Part 1.1: in theory (2)

Explanation

- ▶ Idea: side-channel attack.

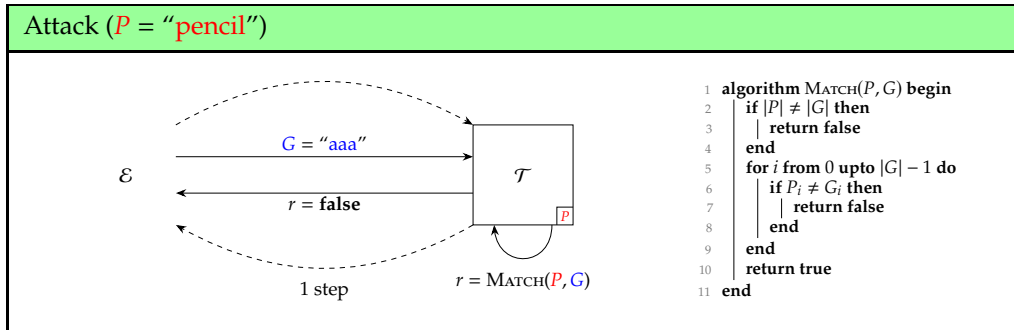


Notes:

Part 1.1: in theory (2)

Explanation

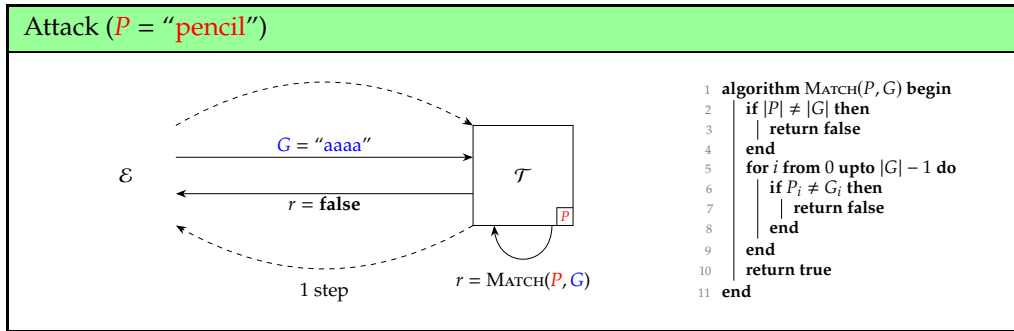
- ▶ **Idea: side-channel attack.**



Notes:

Part 1.1: in theory (2)

- **Idea: side-channel attack.**

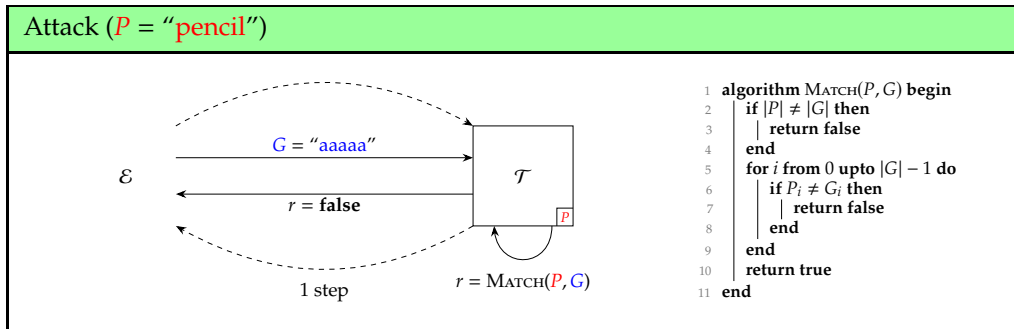


Notes:

Part 1.1: in theory (2)

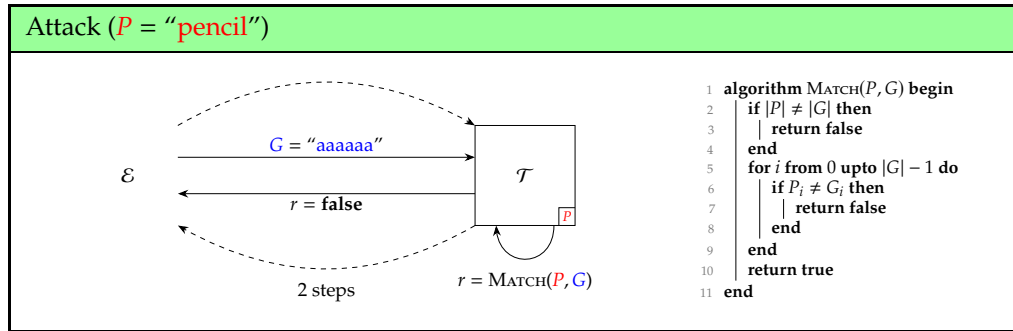
Explanation

- ▶ **Idea: side-channel attack.**



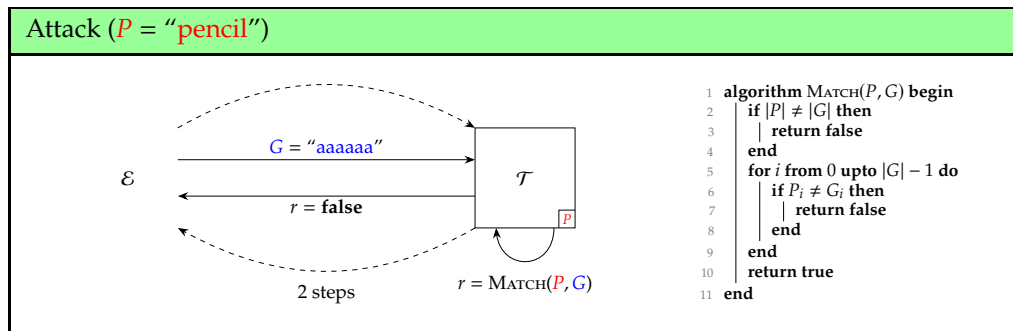
Notes:

► Idea: side-channel attack.



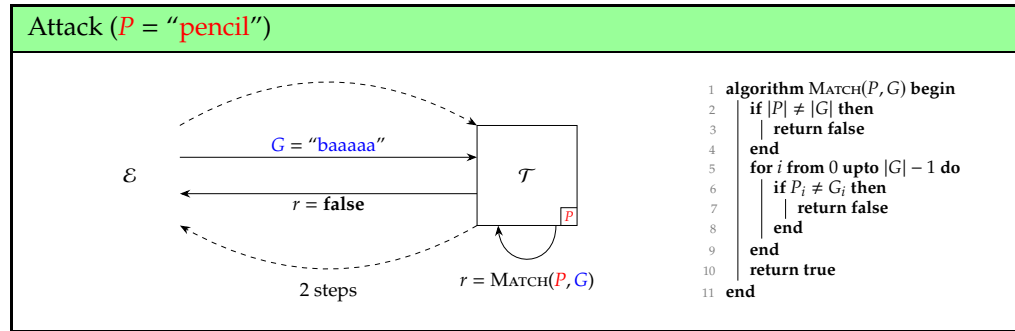
Notes:

► Idea: side-channel attack.



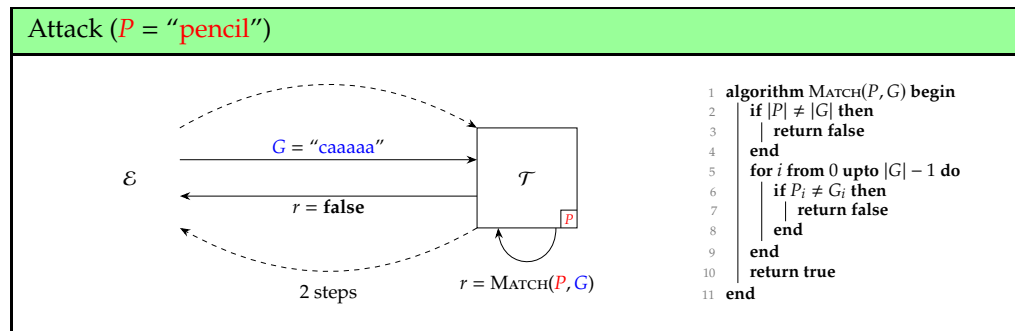
Notes:

► Idea: side-channel attack.



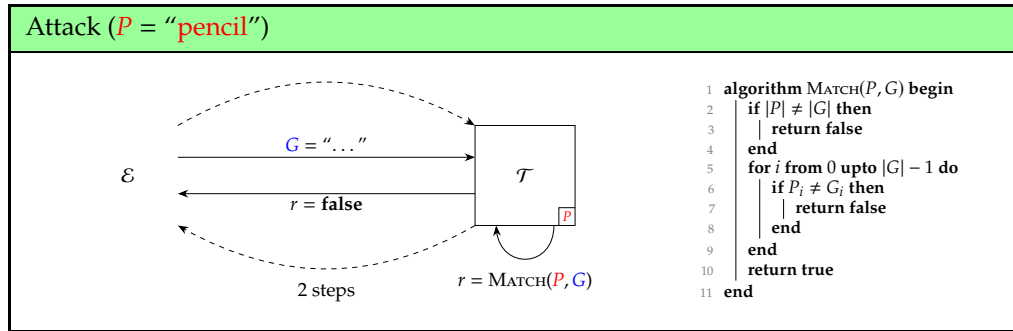
Notes:

► Idea: side-channel attack.



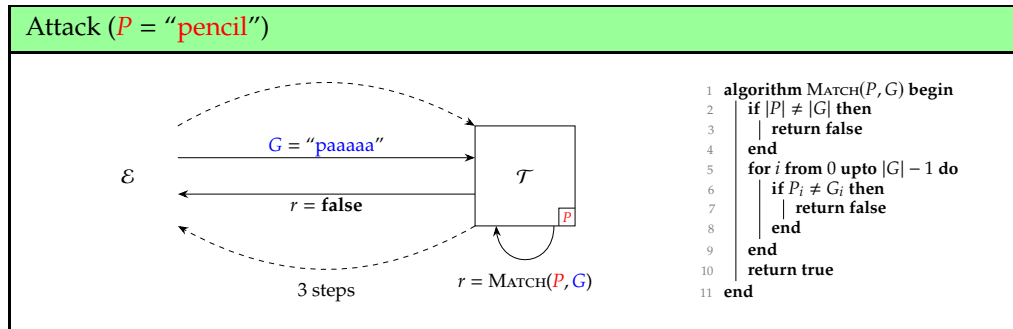
Notes:

► Idea: side-channel attack.



Notes:

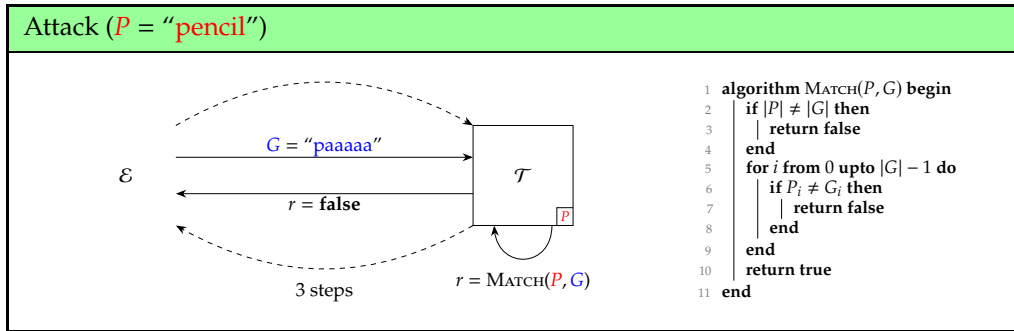
► Idea: side-channel attack.



Notes:

Part 1.1: in theory (2)

- **Idea: side-channel attack.**

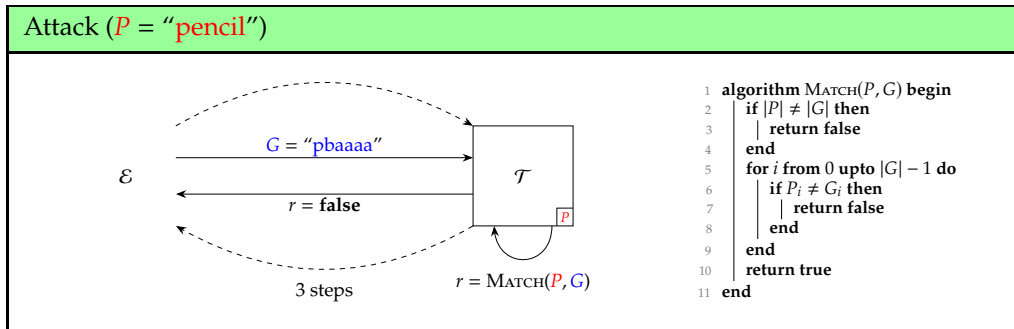


Notes:

Part 1.1: in theory (2)

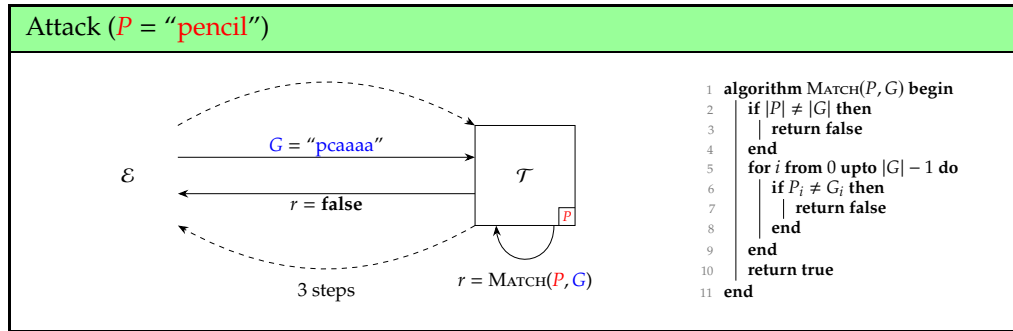
Explanation

- **Idea: side-channel attack.**



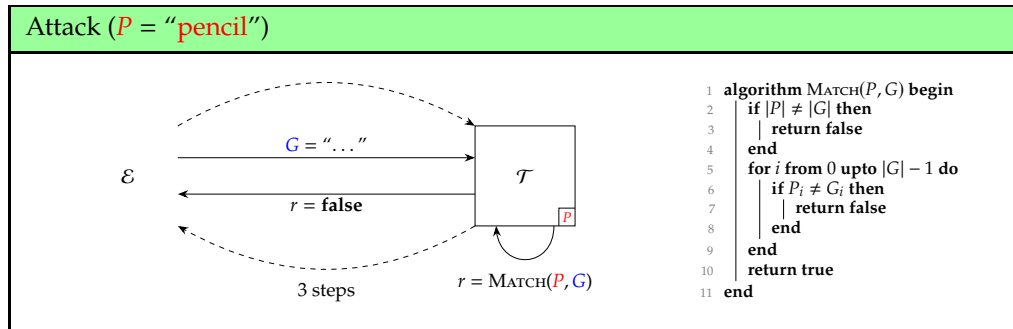
Notes:

► Idea: side-channel attack.



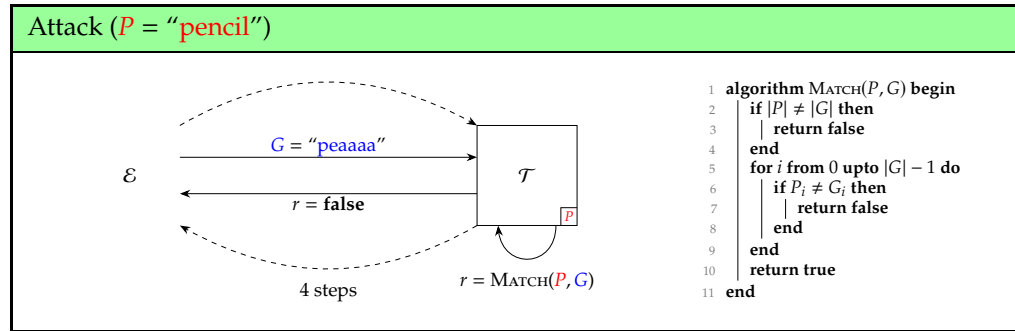
Notes:

► Idea: side-channel attack.



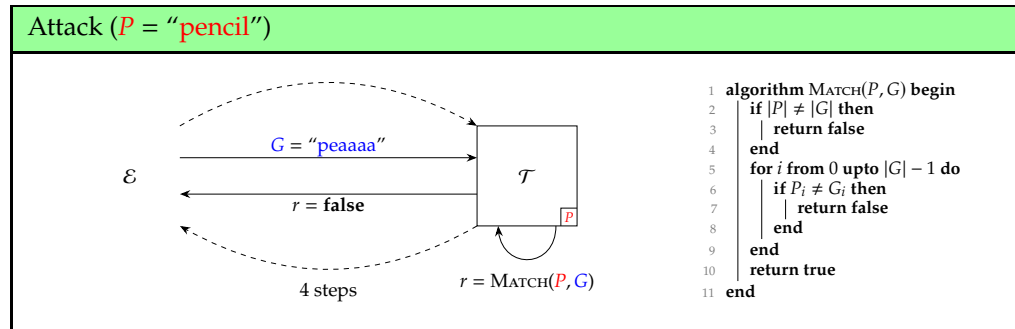
Notes:

► Idea: side-channel attack.



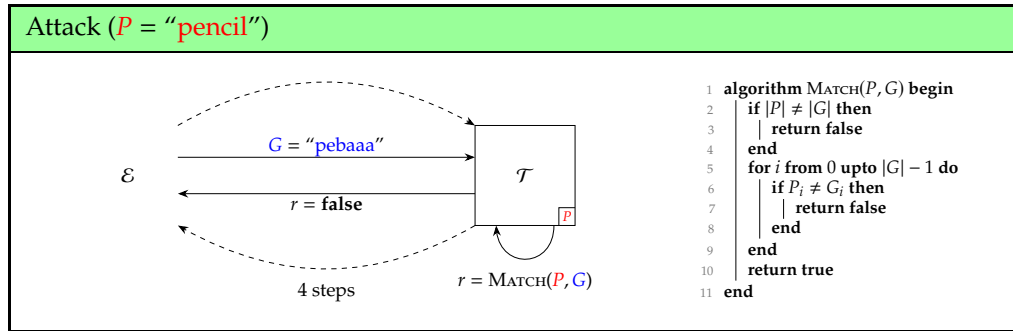
Notes:

► Idea: side-channel attack.



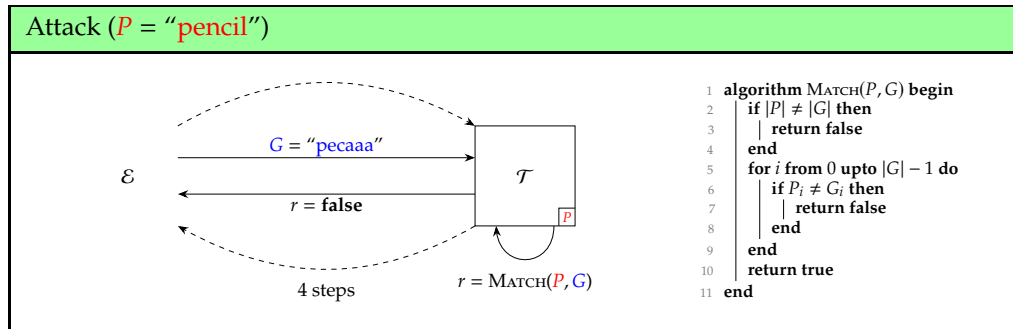
Notes:

► Idea: side-channel attack.



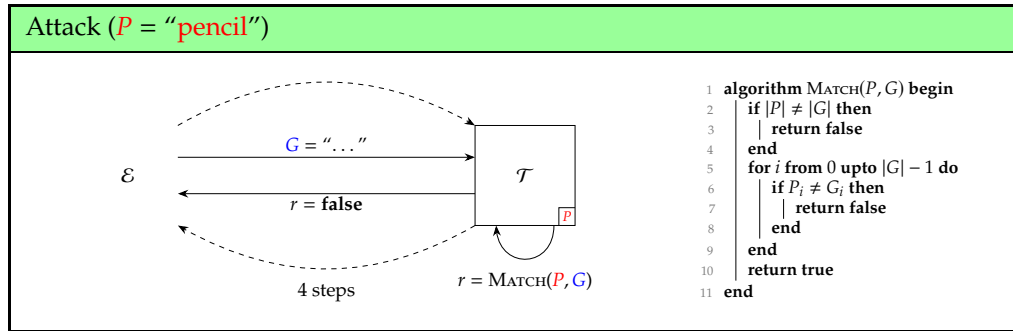
Notes:

► Idea: side-channel attack.



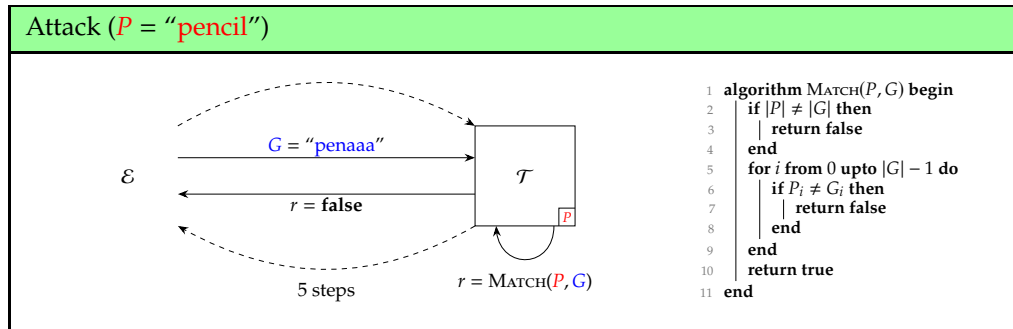
Notes:

► Idea: side-channel attack.



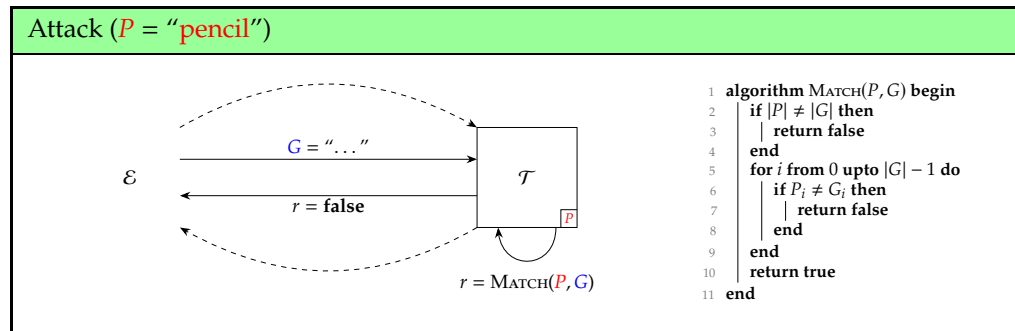
Notes:

► Idea: side-channel attack.



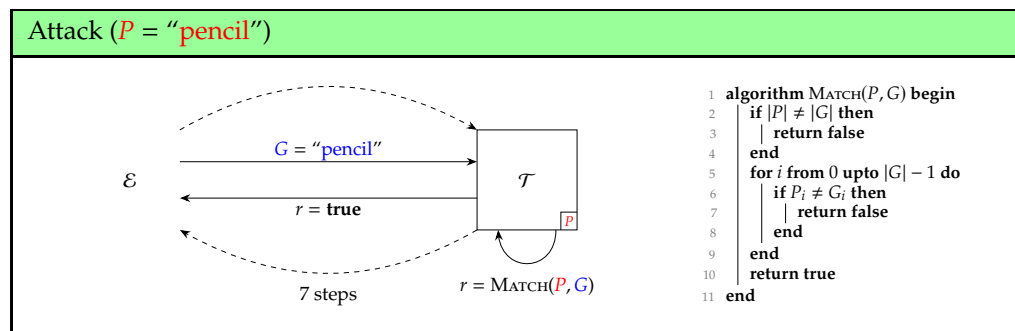
Notes:

► Idea: side-channel attack.



Notes:

► Idea: side-channel attack.



∴ if we *bend* the rules a little then

+ve: we always guess a $G = P$

+ve: we don't need too many guesses, e.g., for a 6-character lower-case password we'd make

$$26 \cdot 6 = 156$$

in the worst-case (plus the few extra to recover $|P|$)

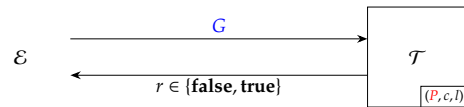
Notes:

Part 1.1: in theory (3)

Explanation

► Scenario:

- given the following interaction between an **attacker** \mathcal{E} and a **target** \mathcal{T}



- and noting that

- the Personal Identification Number (PIN) P has $|P| = 4$ digits in it,
- each digit in G and P is assumed to be from a known alphabet

$$A = \{0, 1, \dots, 9\}$$

such that $|A| = 10$,

- the counter c is incremented after each (successive) incorrect guess; when c exceeds a limit $l = 3$, the target becomes “locked”,
- how can \mathcal{E} mount a successful attack, i.e., input a guess G matching P ?

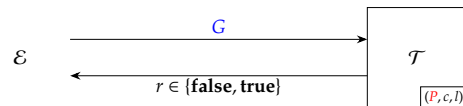
Notes:

Part 1.1: in theory (4)

Explanation

► Idea:

Attack ($P = 1234$)



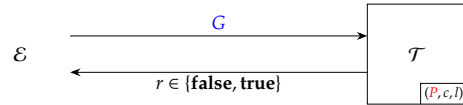
Notes:

Part 1.1: in theory (4)

Explanation

► Idea:

Attack ($P = 1234$)



∴ similar attacks as before apply, namely

1. brute-force attack:

+ve: $10^4 = 10000$ possible PINs is not many

–ve: the counter limits how viable this approach is

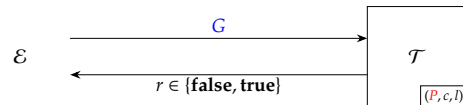
Notes:

Part 1.1: in theory (4)

Explanation

► Idea:

Attack ($P = 1234$)



∴ similar attacks as before apply, namely

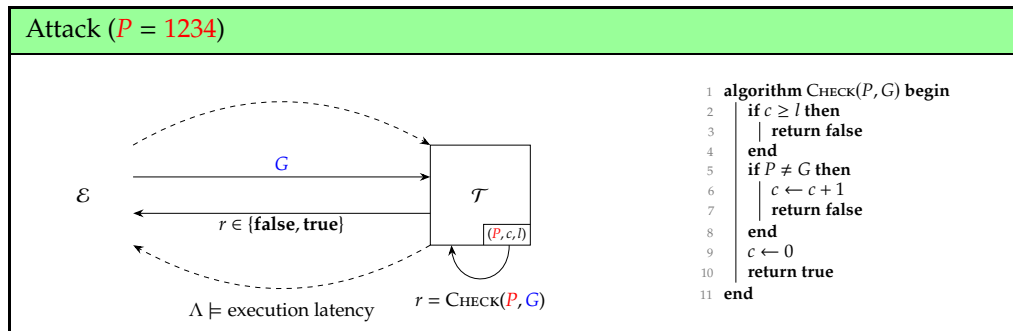
2. dictionary attack:

+ve: reasoning re. common passwords still applies to PINs (e.g., a birthday)

–ve: the counter limits how viable this approach is

Notes:

► Idea:



∴ similar attacks as before apply, namely

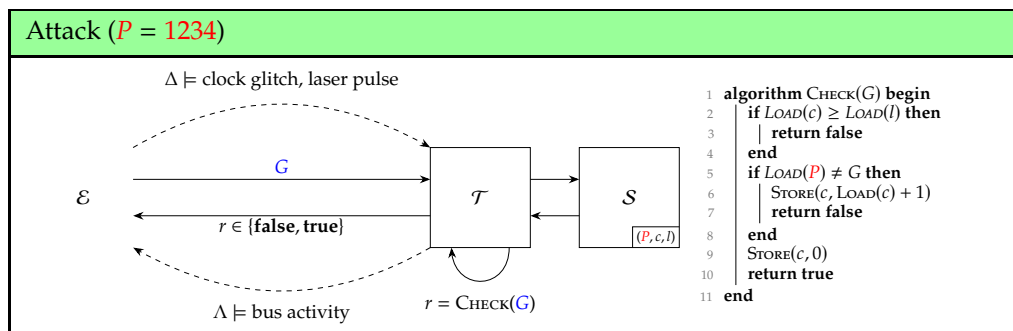
3. side-channel attack:

+ve: we can still measure execution time of CHECK

–ve: comparison of P and G no longer has data-dependent execution time

Notes:

► Idea:



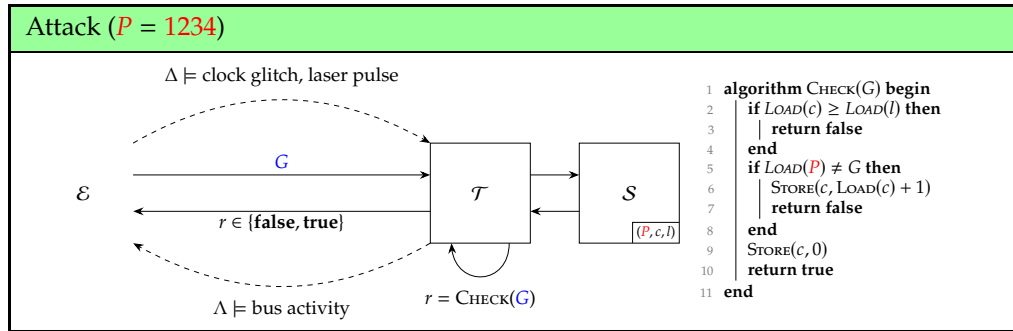
but consider some more implementation detail:

1. we might consider *different* indirect inputs *and* outputs,
2. use of an external, non-volatile storage (e.g., SIM card) implies that for $x \leftarrow y$ we have

$$\left. \begin{array}{l} x \text{ on LHS} \rightsquigarrow \text{store operation} \\ y \text{ on RHS} \rightsquigarrow \text{load operation} \end{array} \right\} \rightsquigarrow \text{STORE}(x, \text{LOAD}(y))$$

Notes:

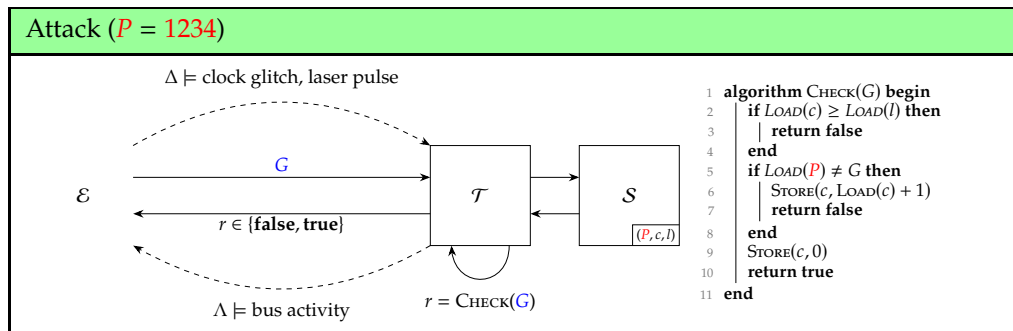
► Idea: fault induction attack.



∴ we could consider

1. disrupting *state*, e.g.
 - corrupt (or randomise) content stored by S ,
 - if l is an n -bit integer, all $2^n - l$ values of a random l' mean more guesses.

► Idea: fault induction attack.



∴ we could consider

2. disrupting *execution*, e.g.
 - control the power supply and probe the command bus,
 - when a command of the form $\text{STORE}(x, y)$ is detected, we know it relates to either

Line #6	:	we know $P \neq G$	↪	disconnect the power, and prevent update to c
Line #9	:	we know $P = G$	↪	do nothing

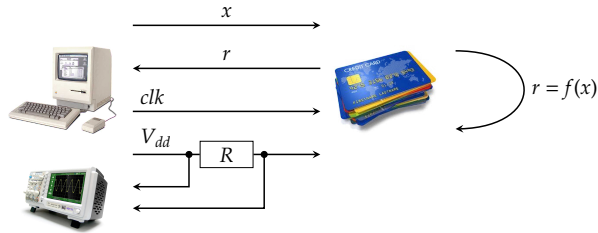
Notes:

Notes:

Part 1.2: in theory (1)

Justification: Λ = power consumption

- **Example:** consider a scenario



whereby

- Ohm's Law tells us that, i.e., $V = IR$, so
- we can acquire a power consumption trace

$$\Lambda = \langle \Lambda_0, \Lambda_1, \dots, \Lambda_{l-1} \rangle$$

i.e., an l -element sequence of instantaneous samples during execution of f .

Notes:

Part 1.2: in theory (1)

Justification: Λ = power consumption

- **Claim:** Λ may be
 - *computation*-dependent, i.e., depends on definition and implementation of f , and/or
 - *data*-dependent, i.e., depends on x .

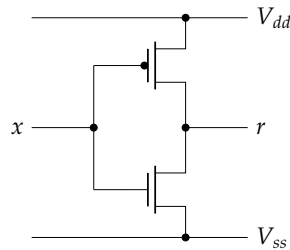
Notes:

Part 1.2: in theory (1)

Justification: Λ = power consumption

► Why?

- From a hardware perspective



power consumption will stem from

1. **static consumption**, and
2. **dynamic consumption**.

- Therefore, different switching behaviour \Rightarrow different power consumption, i.e.,

if $x = 0$, setting $x \leftarrow 0$	\Rightarrow static only	\Rightarrow low(er) power consumption
if $x = 0$, setting $x \leftarrow 1$	\Rightarrow static plus dynamic	\Rightarrow high(er) power consumption
if $x = 1$, setting $x \leftarrow 0$	\Rightarrow static plus dynamic	\Rightarrow high(er) power consumption
if $x = 1$, setting $x \leftarrow 1$	\Rightarrow static only	\Rightarrow low(er) power consumption

which is data-dependent, and not *necessarily* in a symmetric manner.

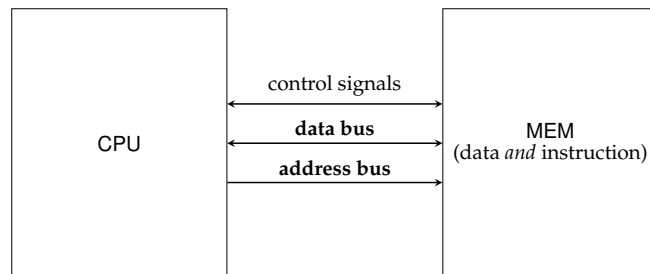
Notes:

Part 1.2: in theory (1)

Justification: Λ = power consumption

► Why?

- From a software perspective



power consumption will stem from

1. computation,
2. communication (i.e., use of buses), and
3. storage (e.g., registers, memory),
4. ...

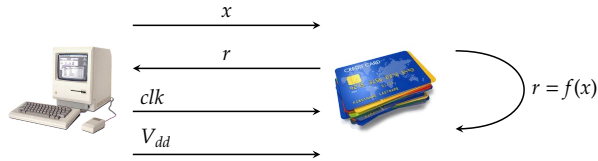
all of which are data-dependent.

Notes:

Part 1.2: in theory (2)

Justification: Λ = execution latency

- **Example:** consider a scenario



whereby

- we measure

Λ_x = time when x is transmitted

Λ_r = time when r is received

so that

- $\Lambda = \Lambda_r - \Lambda_x$ approximates the execution latency of f .

Notes:

Part 1.2: in theory (2)

Justification: Λ = execution latency

- **Claim:** Λ may be
 - *computation*-dependent, i.e., depends on definition and implementation of f , and/or
 - *data*-dependent, i.e., depends on x .

Notes:

Part 1.2: in theory (2)

Justification: Δ = execution latency

► **Why?** for example, in each of

1.
$$\left. \begin{array}{l} \dots \\ \text{if GPR}[x] = 0 \text{ then PC} \leftarrow \text{done} \\ \text{stmt} \\ \text{done} : \dots \end{array} \right\} \begin{array}{l} \text{a. GPR}[x] = 0 \text{ so stmt is not executed} \\ \text{b. GPR}[x] = 1 \text{ so stmt is executed} \end{array}$$
2.
$$\left. \begin{array}{l} \dots \\ \text{GPR}[r] \leftarrow \text{MEM}[\text{GPR}[x]] \\ \dots \end{array} \right\} \begin{array}{l} \text{a. MEM}[\text{GPR}[x]] \text{ is resident in cache} \\ \text{b. MEM}[\text{GPR}[x]] \text{ is not resident in cache} \end{array}$$
3.
$$\left. \begin{array}{l} \dots \\ \text{GPR}[r] \leftarrow \text{GPR}[x] \times \text{GPR}[y] \\ \dots \end{array} \right\} \begin{array}{l} \text{a. GPR}[x] \text{ has small magnitude} \\ \text{b. GPR}[x] \text{ has large magnitude} \end{array}$$

it *could* be the case that

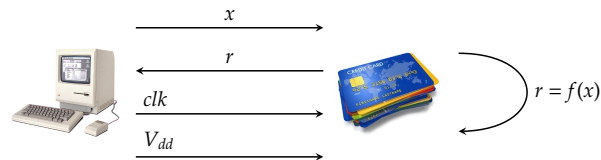
- a. \leadsto low(er) execution latency
- b. \leadsto high(er) execution latency

Notes:

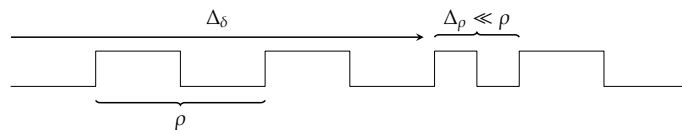
Part 1.2: in theory (3)

Justification: Δ = clock glitch

► **Example:** consider a scenario



whereby a controlled “glitch”, i.e.,



such that

- ρ is the clock period,
- Δ_p is the period of the glitch,
- Δ_δ is the offset of the glitch.

can be caused in the clock signal clk .

Notes:

Part 1.2: in theory (3)

Justification: Δ = clock glitch

► Claim: given

```
...  
if GPR[x] = 0 then PC ← done  
stmt  
done : ...
```

Δ might allow one to skip the branch instruction, i.e., always execute `stmt`.

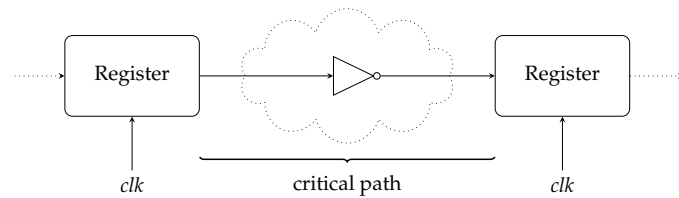
Notes:

Part 1.2: in theory (3)

Justification: Δ = clock glitch

► Why?

► recall that



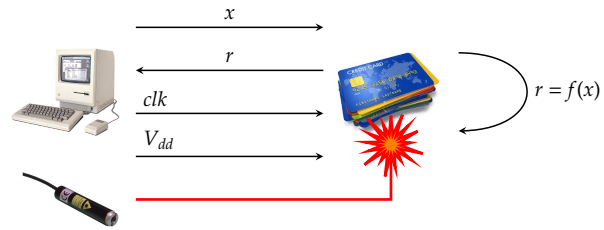
- where, if ρ is close to the critical path, the glitch is likely shorter,
- therefore, it is plausible such a glitch can prevent complete execution of an instruction, e.g.,
 - GPR[x] = 0 is not computed in time,
 - PC is not updated in time,
 - ...
- meaning that instruction is skipped.

Notes:

Part 1.2: in theory (4)

Justification: Δ = laser pulse

► **Example:** consider a scenario



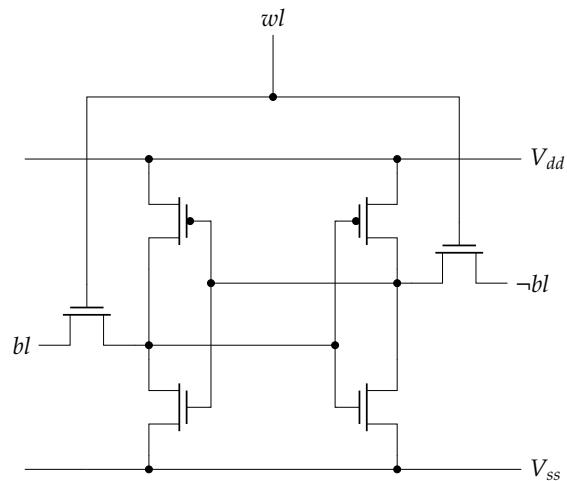
whereby a focused laser pulse can be aimed at the target device.

Notes:

Part 1.2: in theory (4)

Justification: Δ = laser pulse

► **Claim:** Δ might allow one to toggle the state of



i.e., an SRAM-based memory cell (within some larger device).

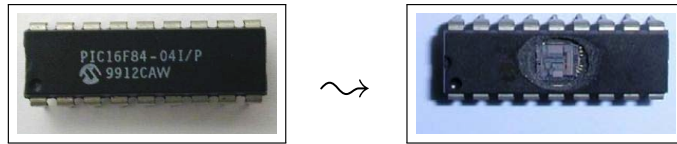
Notes:

Part 1.2: in theory (4)

Justification: Δ = laser pulse

► Why?

- after decapsulation



- at least the top layer of the device is exposed,
- the laser pulse can ionise regions of semi-conductor material,
- doing so can be used to activate a transistor,
- if the bottom-left transistor can be activated (for some short period), this will toggle Q .

<https://www.cl.cam.ac.uk/~sps32/ches02-optofault.pdf>

© Daniel Page (University of Bristol)
Applied Cryptology

University of
BRISTOL

git # b282dbb9 @ 2025-09-03

Part 1.3: in theory (1)

Formalisation: attacks

Definition

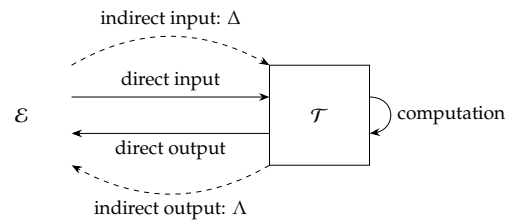
A **cryptanalytic attack** focuses on exploiting a vulnerability in the abstract, on-paper specification of a target. In contrast, an **implementation attack** focuses on exploiting a vulnerability in the concrete, in-practice implementation of a target by 1) actively influencing and/or 2) passively observing behaviour by it.

Notes:

Notes:

Definition

Within the following scenario



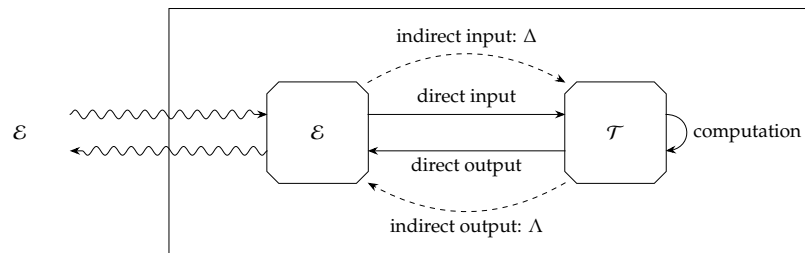
\mathcal{E} is said to

observe \mathcal{T} via $\Lambda \rightsquigarrow$ **side-channel attack**
influence \mathcal{T} via $\Delta \rightsquigarrow$ **fault induction attack**

Notes:

Definition

Within the following scenario



\mathcal{E} is said to

observe \mathcal{T} via $\Lambda \rightsquigarrow$ **side-channel attack**
influence \mathcal{T} via $\Delta \rightsquigarrow$ **fault induction attack**

Notes:

Definition

\mathcal{E} wants to realise some sort of **attack goal**, e.g.,

1. recovery of state from the target
2. manipulation of state in the target
3. manipulation of behaviour by the target

measured relative to both efficacy *and* efficiency.

Notes:

Definition

\mathcal{E} employs an **attack strategy**, which might be (generically) characterised as, e.g.,

1. profiled versus non-profiled
2. adaptive versus non-adaptive
3. differential versus non-differential

which also captures features of standard cryptanalysis, including known plaintext, chosen plaintext, etc.

Definition

\mathcal{E} operates an **attack process**: *typically* this involves

1. an offline pre-interaction phase : characterise, calibrate, pre-compute, etc.
2. an online interaction phase : use input to acquire output
3. an offline post-interaction phase : use input and output to realise goal

Notes:

Definition

\mathcal{E} employs an **attack mechanism**, which can be (generically) characterised as, e.g.,

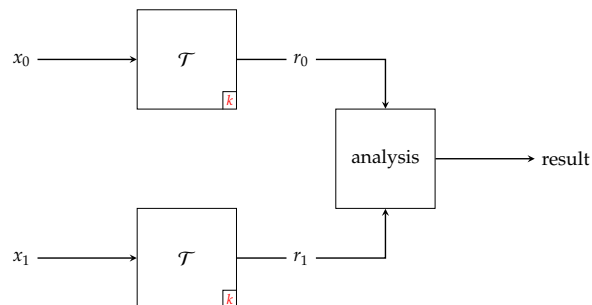
1. software versus hardware
2. generic versus specific
3. local versus remote
4. contact-based versus contact-less
5. invasive versus non-invasive
6. destructive versus non-destructive
7. synchronous versus non-synchronous
8. deterministic versus non-deterministic

Notes:

Part 1.3: in theory (3) Formalisation: attacks

► Note that:

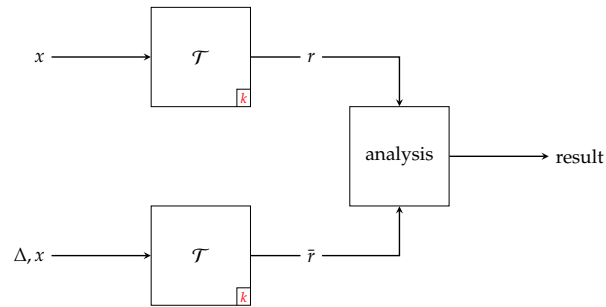
- a differential cryptanalytic attack [5]



(roughly) analyses how an input difference affects the output difference.

Notes:

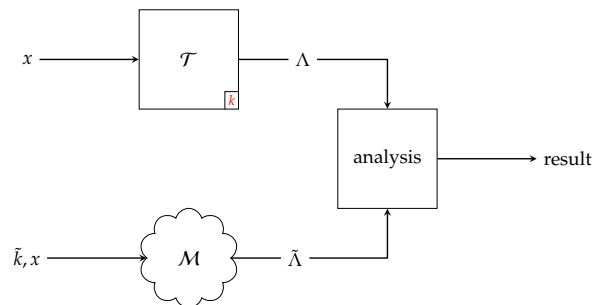
- Note that:
 - a differential fault induction attack



(typically) analyses how a fault affects the output difference.

Notes:

- Note that:
 - a differential side-channel attack



is (typically) such that

- \mathcal{M} is a **model** (or simulation) of \mathcal{T} ,
- \tilde{k} is a **hypothesis** about (part of) k ,
- $\tilde{\Lambda}$ is the **hypothetical leakage** (cf. the *actual* leakage Λ),

and so

non-differential \Rightarrow 1 interaction \simeq analysis within single Λ
 differential \Rightarrow n interactions \simeq analysis between many Λ

Notes:

Part 1.3: in theory (4)

Formalisation: attacks

Definition

The information leaked via some side-channel is modelled as $\mathcal{M}(\cdot) = \mathcal{M}_d(\cdot) + \mathcal{M}_n$, i.e., as the sum of 1) data-dependent **signal** (of interest) and 2) **noise** components.

Definition

Let V denote a set of values some (intermediate) variable can take, and L denote a set of leakage values.

- ▶ A **value**-based leakage model is such that $\mathcal{M}_d : V \rightarrow L$, meaning the leakage value depends on the current value of some variable.
- ▶ A **transition**-based leakage model is such that $\mathcal{M}_d : V \times V \rightarrow L$, meaning the leakage value depends on the previous and current value of some variable (i.e., the transition from the former to the latter).

Notes:

Part 1.3: in theory (4)

Formalisation: attacks

Definition

The information leaked via some side-channel is modelled as $\mathcal{M}(\cdot) = \mathcal{M}_d(\cdot) + \mathcal{M}_n$, i.e., as the sum of 1) data-dependent **signal** (of interest) and 2) **noise** components.

Definition

Let V denote a set of values some (intermediate) variable can take, and L denote a set of leakage values.

- ▶ A **value**-based leakage model is such that $\mathcal{M}_d : V \rightarrow L$, meaning the leakage value depends on the current value of some variable.
- ▶ A **transition**-based leakage model is such that $\mathcal{M}_d : V \times V \rightarrow L$, meaning the leakage value depends on the previous and current value of some variable (i.e., the transition from the former to the latter).

▶ Example:

1. Hamming weight \Rightarrow value-based leakage model
2. Hamming distance \Rightarrow transition-based leakage model

Notes:

Part 1.3: in theory (5)

Formalisation: attacks

Definition

A **fault model** is an abstraction of the fault induction mechanism, i.e., it separates fault *induction* from fault *exploitation*. it captures features such as

1. timing \Rightarrow precise control, imprecise control, no control
2. location \Rightarrow precise control, imprecise control, no control
3. duration \Rightarrow transient, permanent, destructive
4. plurality \Rightarrow single fault; multiple, i.e., n faults
5. granularity \Rightarrow 1 bit, n bits, variable
6. effect \Rightarrow set-to-0/1, stuck-at-0/1, flip, randomise, variable
7. implication \Rightarrow input data, computation on data, storage of data, execution of instructions

Notes:

Part 1.3: in theory (6)

Formalisation: countermeasures

Definition

\mathcal{T} might employ a **countermeasure strategy**, which can be (generically) characterised as, e.g.,

1. implicit versus explicit
2. detection versus prevention

and typically forms a layered approach, i.e., a suite of countermeasures versus a single “silver-bullet” or panacea.

Notes:

Definition

\mathcal{T} might design an *abstract* **countermeasure mechanism**, within (at least) the following *levels*

1. **protocol**,
2. **specification**,
3. **implementation**, i.e.,
 - ▶ software, and/or
 - ▶ hardware.

Definition

\mathcal{T} might implement a *concrete* **countermeasure mechanism**, which can be (generically) characterised as, e.g.,

1. software versus hardware
2. generic versus specific
3. selective versus non-selective
4. proactive versus reactive

Notes:

Definition

Countermeasures against implementation attacks based on information leakage often fall into the following *classes*:

1. **hiding** \approx decrease SNR, or
2. **masking** \approx randomised redundant representation.

Notes:

Definition

Among a large design space of countermeasures, instances that focus on hiding (typically) fall into the following sub-classes:

- 1. increase noise, e.g., make Λ random:
 - a. **spatial displacement**, i.e., *where* the operation is computed,
 - b. **temporal displacement**, i.e., *when* the operation is computed, which can be further divided into
 - ▶ padding (or skewing), and
 - ▶ reordering (or shuffling),
 - c. **diversified computation**, i.e., *how* the operation is computed,
 - d. **obfuscated computation**, e.g., *whether* the operation computed is real or fake (or a dummy).
- 2. decrease signal, e.g., make Λ constant:
 - a. **data-oblivious** (or “**constant-time**”) computation of the operation.

Notes:

Definition

Among a large design space of countermeasures, instances that focus on masking (typically) fall into the following sub-classes:

- 1. **Boolean masking** (or **additive masking**):

$$x \mapsto \hat{x} = \langle \hat{x}[0], \hat{x}[1], \dots, \hat{x}[d] \rangle$$

such that

$$x = \hat{x}[0] \oplus \hat{x}[1] \oplus \dots \oplus \hat{x}[d],$$

and

- 2. **arithmetic masking** (or **multiplicative masking**):

$$x \mapsto \hat{x} = \langle \hat{x}[0], \hat{x}[1], \dots, \hat{x}[d] \rangle$$

such that

$$x = \hat{x}[0] + \hat{x}[1] + \dots + \hat{x}[d] \pmod{2^w}.$$

Notes:

Part 1.3: in theory (10)

Formalisation: countermeasures

Definition

Countermeasures against implementation attacks based on fault induction often fall into the following *classes*:

1. **induction-oriented**, e.g.,
 - ▶ shielding,
 - ▶ sensing,
 - ▶ hiding,and
2. **exploitation-oriented**, e.g.,
 - ▶ duplication,
 - ▶ infection,
 - ▶ checksum.

Notes:

Part 1.3: in theory (11)

Formalisation: countermeasures

Definition

Among a large design space of countermeasures, instances that focus on exploitation are (typically) parameterised by

1. **type of duplication**, e.g.,
 - ▶ temporal duplication: n computations of $f(x)$ in 1 location,
 - ▶ spatial duplication: 1 computation of $f(x)$ in n locations,
2. **degree of duplication**,
3. **type of check**, e.g.,
 - ▶ direct check: $f(x) \stackrel{?}{=} f(x)$,
 - ▶ linearity check: $f(-x) \stackrel{?}{=} -f(x)$,
 - ▶ inversion check: $f^{-1}(f(x)) \stackrel{?}{=} x$,
4. **frequency of check**, and
5. **type of action**, e.g.,
 - ▶ preventative action: $f(x) \neq f(x) \rightsquigarrow \perp$,
 - ▶ infective action: $f(x) \neq f(x) \rightsquigarrow \$$,

and yield an outcome with an associated **detection probability**.

Notes:

Conclusions

- ▶ **Take away points:** implementation attacks
 1. are a potent threat, forming part of a complex attack landscape,
 2. extend well beyond cryptographic targets, posing a more general (cyber-)security challenge,
 3. present significant challenges, e.g., per
 - ▶ “attacks only get better” principle,
 - ▶ “no free lunch” principle,
 - ▶ need to consider multiple layers of abstraction,such that “raising the bar” is of use if not ideal,
 4. demand care re. evaluation and/or certification (e.g., FIPS 140-3 [9]) requirements.

Notes:

Additional Reading

- ▶ S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
- ▶ P.C. Kocher et al. “Introduction to differential power analysis”. In: *Journal of Cryptographic Engineering* (JCEN) 1.1 (2011), pp. 5–27.
- ▶ M. Joye and M. Tunstall, eds. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.
- ▶ H. Bar-El et al. “The Sorcerer’s Apprentice Guide to Fault Attacks”. In: *Proceedings of the IEEE* 94.2 (2006), pp. 370–382.
- ▶ A. Barenghi et al. “Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures”. In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076.
- ▶ D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede. “Hardware Designer’s Guide to Fault Attacks”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.12 (2013), pp. 2295–2306.
- ▶ B. Yuce, P. Schaumont, and M. Witteman. “Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation”. In: *Journal of Hardware and Systems Security* 2.2 (2018), pp. 111–130.

Notes:

References

[1] M. Joye and M. Tunstall, eds. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012 (see p. 175).

[2] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007 (see p. 175).

[3] H. Bar-El et al. “The Sorcerer’s Apprentice Guide to Fault Attacks”. In: *Proceedings of the IEEE* 94.2 (2006), pp. 370–382 (see p. 175).

[4] A. Barenghi et al. “Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures”. In: *Proceedings of the IEEE* 100.11 (2012), pp. 3056–3076 (see p. 175).

[5] E. Biham and A. Shamir. “Differential Cryptanalysis of DES-like Cryptosystems”. In: *Advances in Cryptology (CRYPTO)*. LNCS 537. Springer-Verlag, 1990, pp. 2–21 (see p. 147).

[6] D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede. “Hardware Designer’s Guide to Fault Attacks”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.12 (2013), pp. 2295–2306 (see p. 175).

[7] P.C. Kocher et al. “Introduction to differential power analysis”. In: *Journal of Cryptographic Engineering (JCEN)* 1.1 (2011), pp. 5–27 (see p. 175).

[8] B. Yuce, P. Schaumont, and M. Witteman. “Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation”. In: *Journal of Hardware and Systems Security* 2.2 (2018), pp. 111–130 (see p. 175).

[9] *Security Requirements For Cryptographic Modules*. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 140-3. 2001. URL: <https://doi.org/10.6028/NIST.FIPS.140-3> (see p. 173).

Notes: