Proving Crypto Implementations Secure

Interplay between crypto and program verification

Manuel Barbosa
HASLab – INESC TEC
Faculty of Science – University of Porto
mbb@dcc.fc.up.pt
http://www.dcc.fc.up.pt/~mbb

June 2018

Part 1: Interplay between crypto and verification

Goals of this talk:

- Motivate provable security for implementations
- Framework for implementation security models
- Focus timing attacks
- Argue that in this case stronger is better
- Interesting argument comes from program verification

What is provable security?

Precisely define a *security model*:

- What *functionality* must the system provide?
- What qualifies as a *break* for the system?
- What *class of attackers* should it protect against?

To claim that a system is secure one must:

- State the assumptions upfront:
  - security properties of low-level components
  - these should be *widely used* and *well studied*
- Prove that
  $\forall$ attackers, assumptions $\Rightarrow$ no break
- Or, equivalently,
  $\forall$ attackers, break $\Rightarrow$ assumption false

Which security model?

All security models are abstractions

They result from a compromise:

- More detail $\rightarrow$ less likely to ignore relevant attacks
- Less detail $\rightarrow$ proofs become feasible

Cryptographers have been developing security models for crypto primitives for a long time

These models assume abstract I/O interface given to attacker.

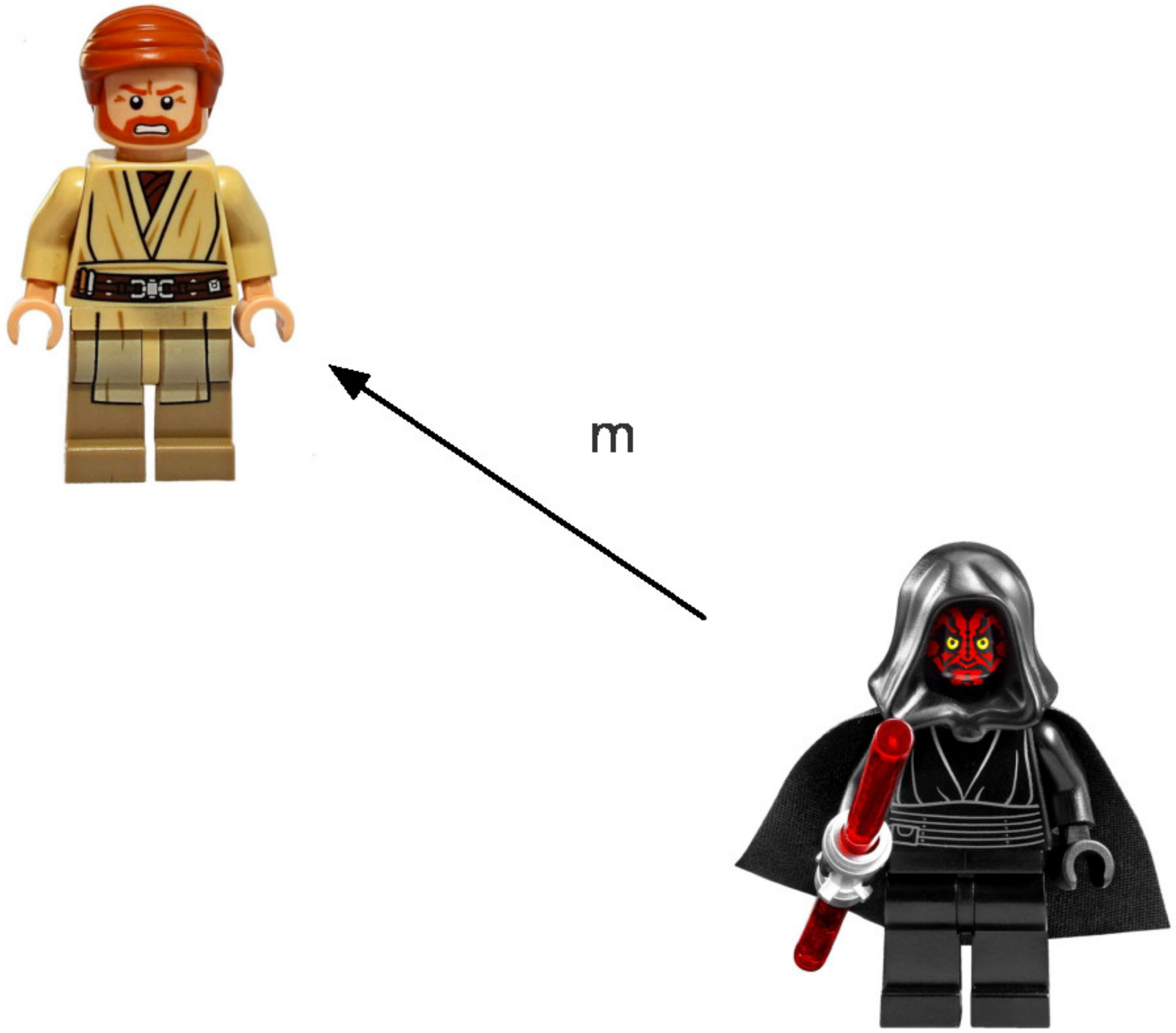
(Lower level abstractions: real-world crypto, e.g. [BDPS'12])

A simple example follows.

SM symmetric encryption security

$\mathsf{k} \leftarrow_{\$} \mathsf{Gen}(1^{\lambda})$

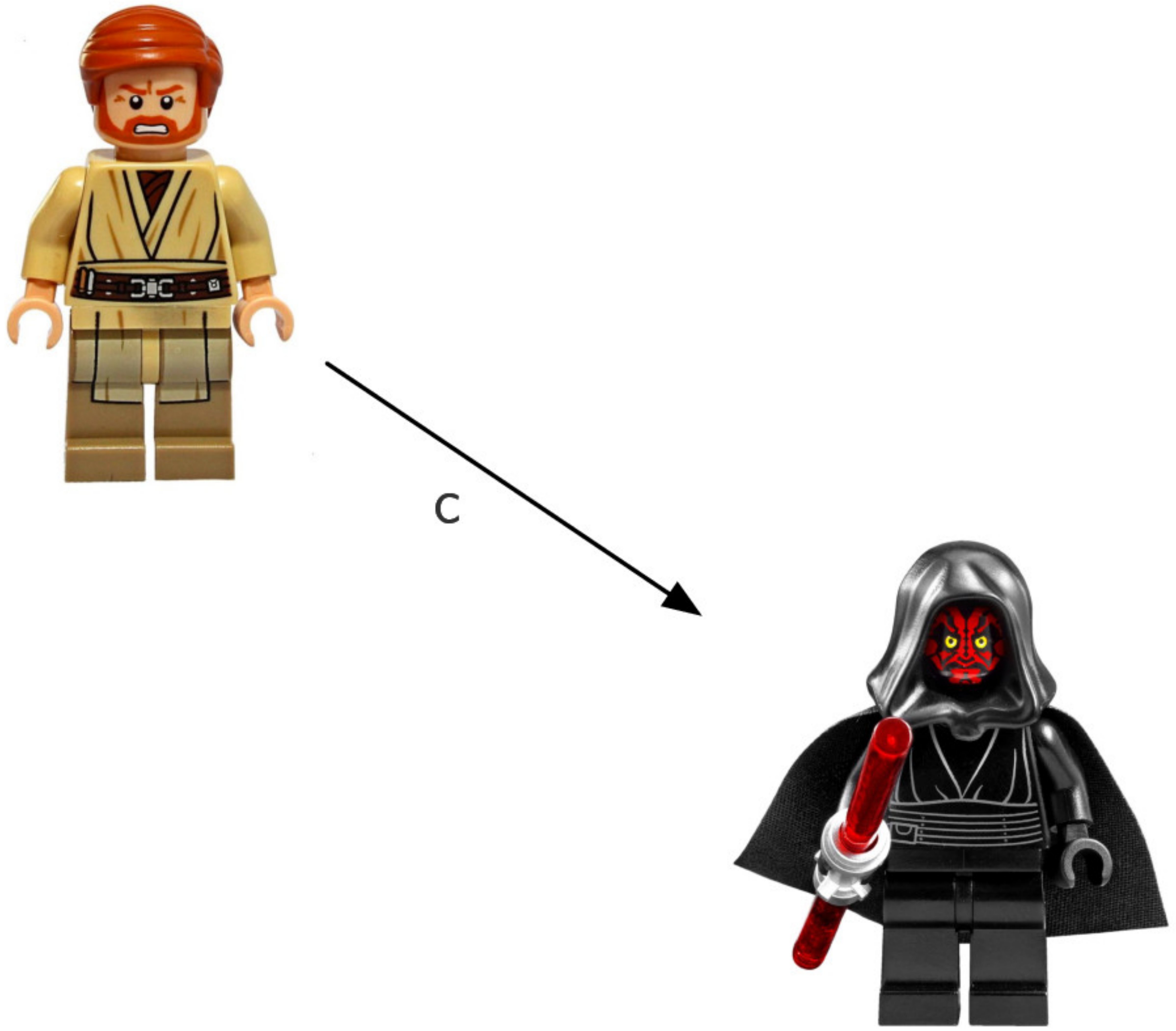
SM symmetric encryption security



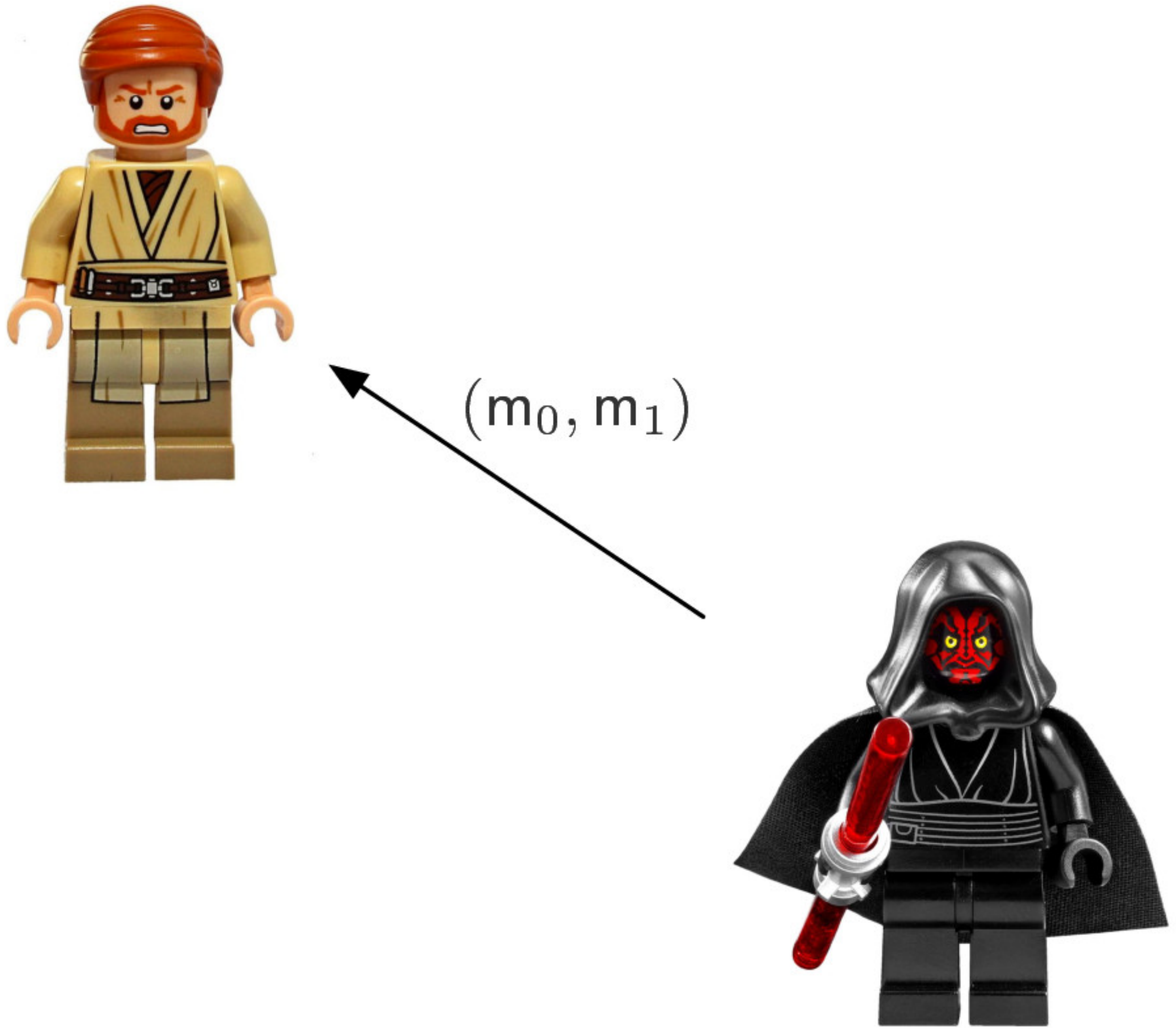
SM symmetric encryption security

$c \xleftarrow{\$} \mathsf{Enc}(k, m)$

SM symmetric encryption security



SM symmetric encryption security

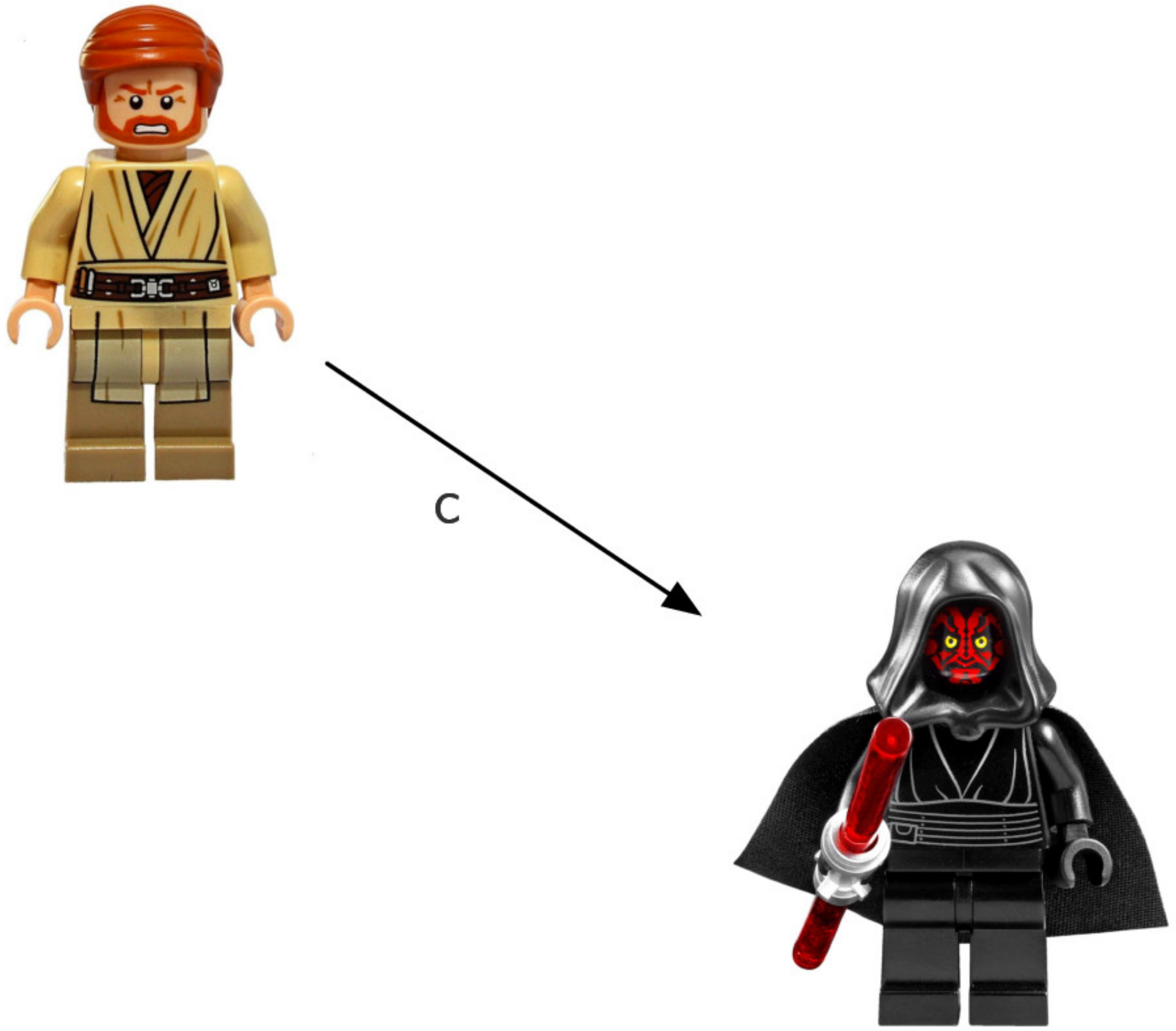


SM symmetric encryption security

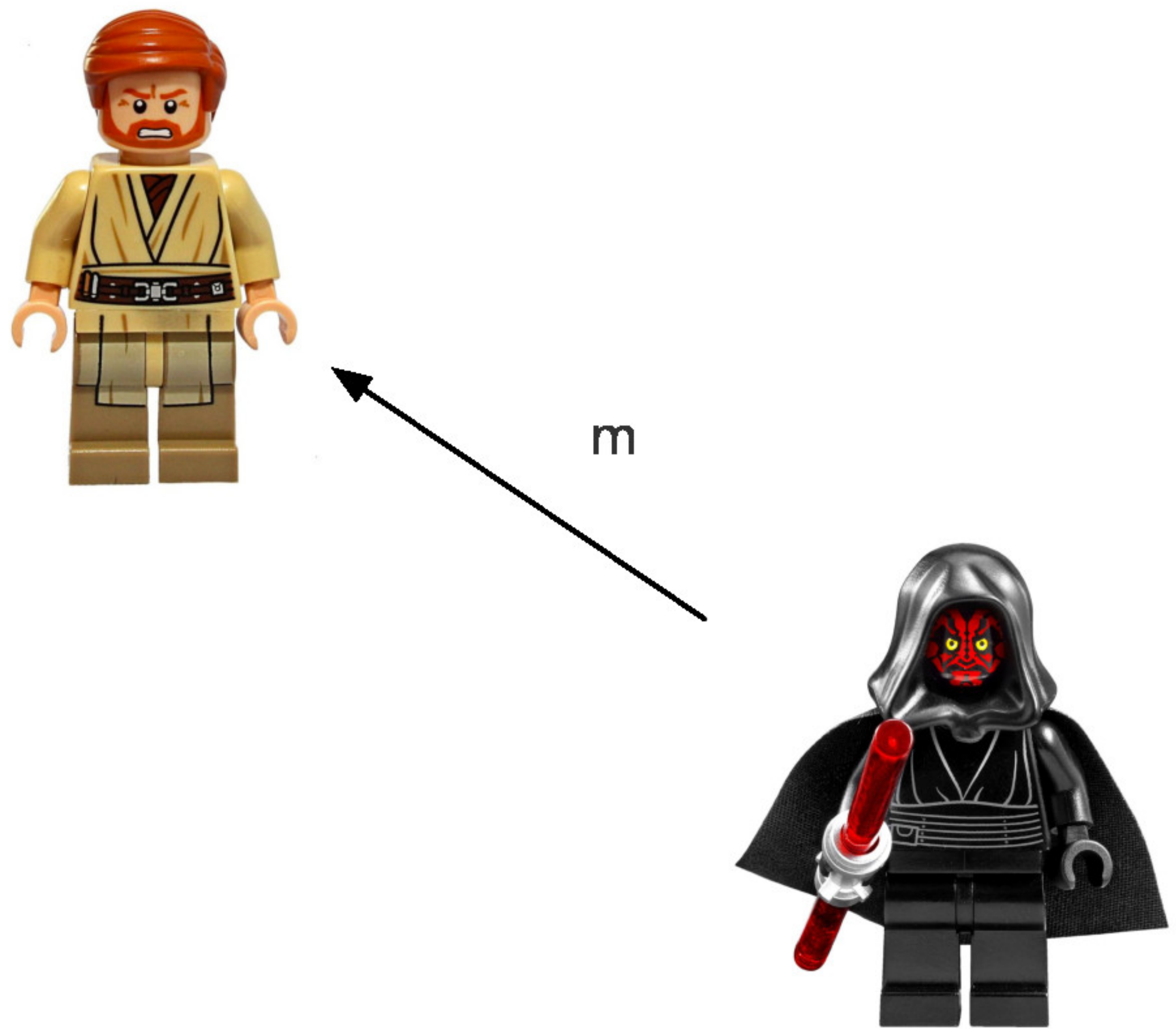
$b \leftarrow_{\$}$ ●

$c \leftarrow_{\$} \mathsf{Enc}(k, m_b)$

SM symmetric encryption security



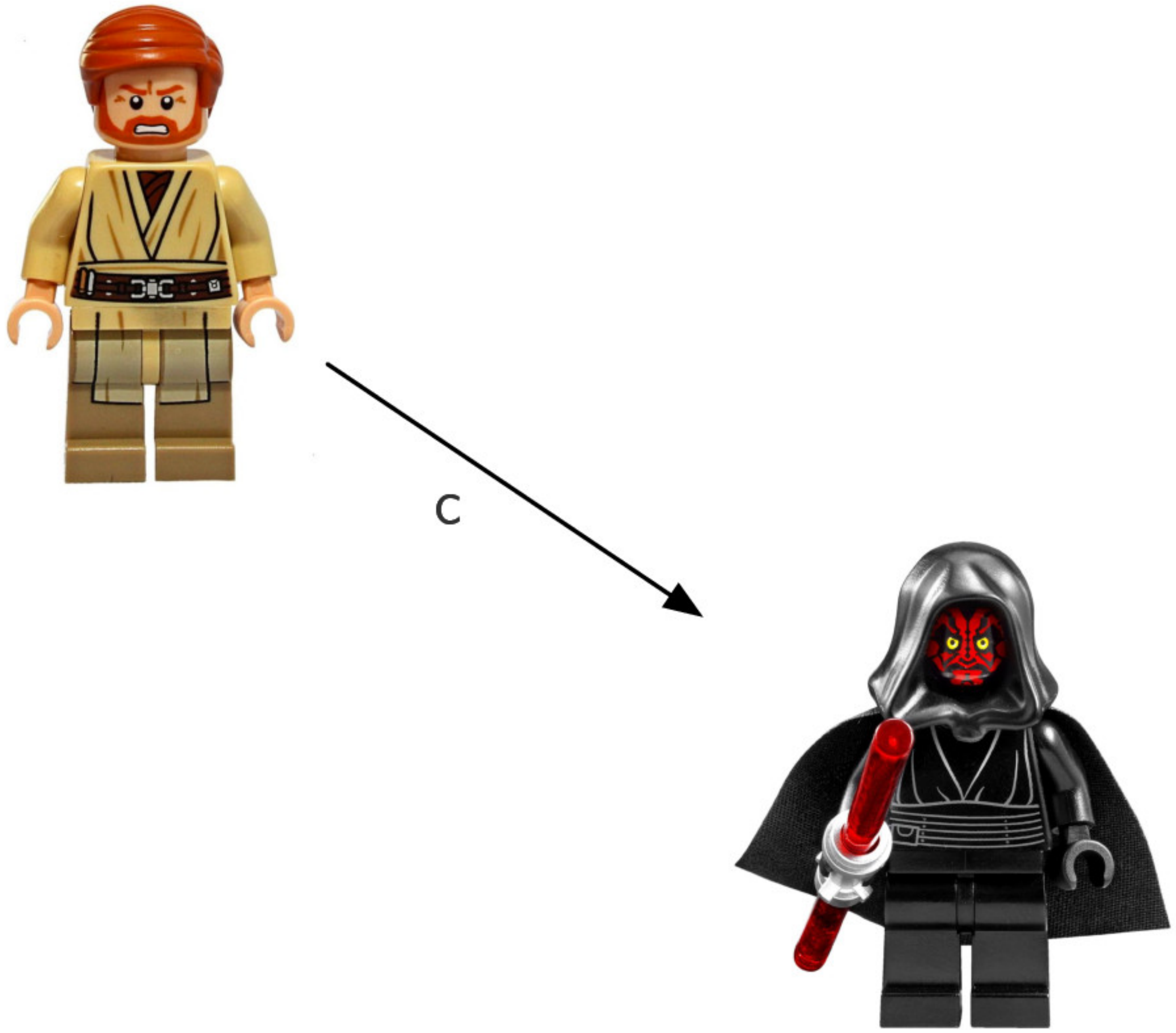
SM symmetric encryption security



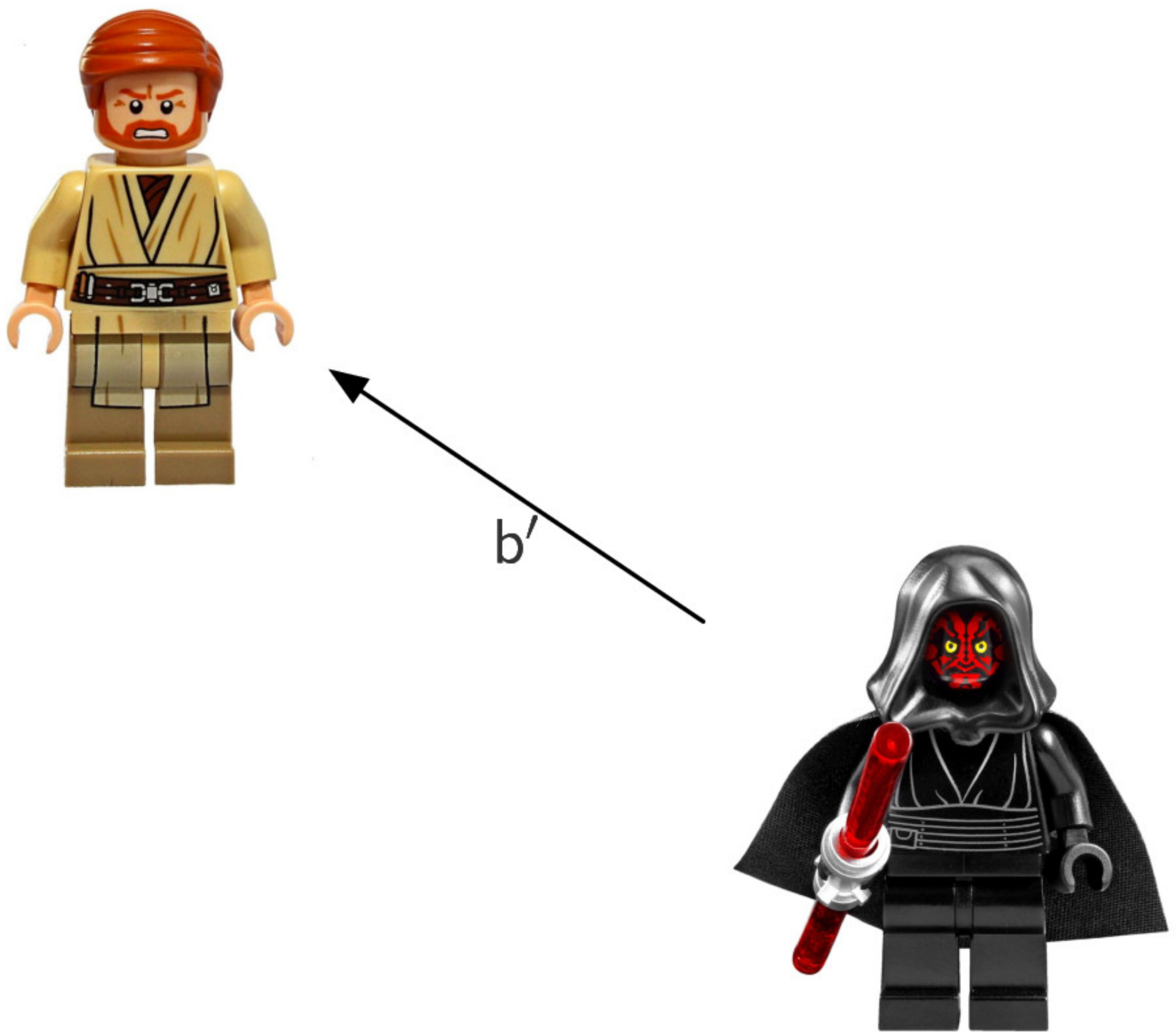
SM symmetric encryption security

$c \leftarrow_{\$} \mathsf{Enc}(k, m)$

SM symmetric encryption security



SM symmetric encryption security



SM symmetric encryption security

$b' \overset{?}{=} b$

Provable implementation security

Most algorithms (widely) deployed today are provably secure.

Still we frequently hear of crypto implementations being broken

There is no theory to extend the proof guarantees to practice

How could we do that?

Let us look at some examples.

Example #1: Bleichenbacher's attack

In 1998 SSLv3 and PKCS#1 v1.5 were very widely used

- Session key transported using RSA with random padding
- Bleichenbacher's attack was published [CRYPTO'98]

- Some SSL servers would signal padding errors in decryption
- These revealed if exponentiation result ended with `0x0002`
- I.e., revealed if exponentiation result was in known range

- Vulnerability allowed decrypting arbitrary messages
- Using adaptive chosen-ciphertext attack

«The SSL documentation does not clearly specify the error conditions and corresponding alerts» [B98]

Example #1: Discussion

Why wasn't the SSLv3 spec precise about this?

1. The attack was not a part of the security model?

«Attackers are assumed to have the ability to capture, modify, delete, replay, and otherwise tamper with messages sent over the communication channel.» [SSLv3 spec]

2. It was, but error messages left as "implementation details"

[JK'02,KPW'13] gave security rationale for SSLv3 handshake with careful error handling

Example #1: Lesson learned

The I/O behaviour of crypto algorithms must be fully accounted for in provable implementation security

Functional correctness of implementation with respect to standard/spec is necessary

This is the case for

- Return $\perp$
- Encoding/decoding
- Unspecified inputs/cases

All that the adversary can «capture,modify, delete,replay» must be in the model

Example #2: MEE-CBC and timing

Are sound security analysis and functional correctness sufficient?

- TLSv1.0 uses an authenticated encryption scheme called MEE-CBC.
- This is a MAC-then-Pad-Then-Encrypt construction
- MEE-CBC is known to be vulnerable to padding oracle attacks using error messages [V'02]

- In TLSv1.0 error messages for MEE-CBC decryption failures are sent over an encrypted channel
- So security analysis took this possibility into consideration and all is OK
- Not quite ...

Example #2: Discussion

Back to defining what a reasonable attacker is:

- Canvel et al. [CRYPTO'03] break TLSv1.0
- Padding oracle attack can recover encrypted passwords

- Attack strategy is "not reasonable":
  - Measure the time that a server takes to abort: checks padding only or padding and MAC.
  - If short time, guess it was padding error
  - If long time, guess it was not a padding error
- Put differently:
  - Security analysis removes padding oracle assuming hidden error messages,
  - Simple timing attack brings padding oracle back

Example #2: Lesson learned

Real-world attackers can observe more than the I/O values of crypto algorithms

This is the case for

- Timing
- Power
- Electromagnatic radiation
- Sound emmitted by fan (!)
- ...

All that the adversary can somehow measure and use as a *side-channel*.

Model must account for side-channel leakage

Interlude: How to deal with side-channels formally?

- Most provable security abstracts side-channels away
- Attacker is given I/O access to crypto algorithms

- For the vast majority of implementations models are vague and/or justified heuristically.

- Exception is *leakage resilient* (LR) crypto
- Ambitious, but hard to connect to practice [FXS'16]
- We deal with simpler setting specific to timing.

First attempt for timing: coarse measure attacker

- The CME assumption states that adversary can only detect large variations in execution time
- After Example #2, MEE-CBC implementations ensured:
  - MAC computation was always carried out
  - If padding error, MAC a dummy message
  - So padding errors had "small" impact on execution time

- For example, CME assumption was rigorously stated in Go TLS code [imperialviolet.org]:
- *«Attacker can align record so that a correct padding will cause one less hash block to be calculated.»*

- Intuition: execution time is essentially constant for CME adversaries, so there can be no timing attack

Example #3: Again MEE-CBC timing

What can go wrong?

- AlFardan and Paterson [S&P'13] come up with CME-unreasonable attacker
- Attack strategy is very close to 2003 attack:
  - Measure time until MEE-CBC decryption error
  - If long time, guess it was padding error
  - If short time, guess it was not a padding error

- How could this happen?
  - Lucky 13 timing attack brings padding oracle back.
  - CME assumption was false

Example #3: Lesson learned

Assumptions on the power of the adversary must be validated

For assumptions to be validated they need to be rigorously stated

If assumptions cannot be validated, then one should assume worst-case scenario

I.e., consider adversaries that can detect “very small” timing variations

Example #3: Discussion

What is the worst-case scenario?

- Program-Counter model: adversary can observe which atomic computations have been carried out.
- What is an *atomic computation*?
  1. processor arithmetic instructions
  2. processor external memory/cache memory access
  3. call to AES/SHA-256 operation in library/hardware

- What if large atomic blocks always take the same time to execute?
- Can we just take those?

Example #4: Still MEE-CBC timing

Suppose a call to a library executes in "constant time".

How can we use this assumption?

- Implementation guarantees that number of calls to crypto library depends only on ciphertext length
- Can we prove there can be no padding oracle attack?

- Intuitively, yes:
  - adversary knows which calls will be made in advance
  - measuring these calls can provide no more information
- In theory, assuming PC-CME model, yes:
  - We will see shortly how to formalize this intuition
- In practice it depends . . .

Example #4: Still MEE-CBC timing

AWS S2N uses random delays and large atomic block assumption

What can go wrong?

- Albrecht and Paterson [EC'16] come up with another-unreasonable attacker
- Lucky microseconds is again a padding oracle attack

- How could this happen?
    - "Atomic" ops had timing variations for concrete inputs
    - Padding values were leaked by triggering these cases
    - Randomized delays were not sufficient to make large atomic block assumption "look" true

Example #4: Lesson learned

Assumptions on the atomicity of operations must be validated

How can we validate these assumptions?

We should follow good practices in crypto

- Fix a small number of simple assumptions
- The community studies these assumptions
- Assumptions are good until broken

Security models: implementation security

Implementation model must capture the execution of code

Include computational machine $\mathcal{M}$:

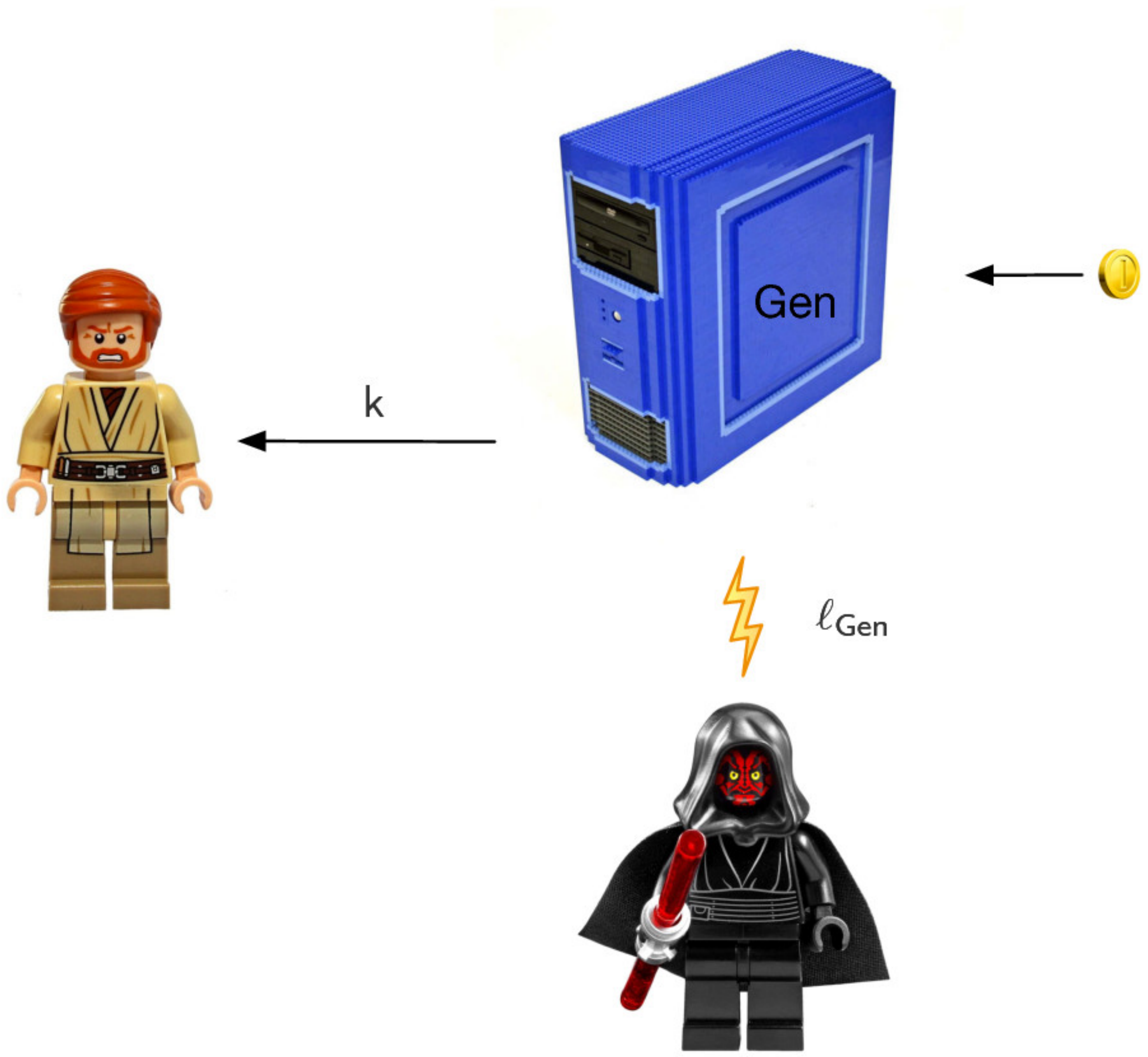
- If side channel attacks are part of threat model
  - $\mathcal{M}$ specifies what leaks when implementation is run
- If attacker can physically influence platform (e.g. faults)
  - $\mathcal{M}$ must provide additional oracles to attacker

Take any specification security model $\mathcal{G}_S$.

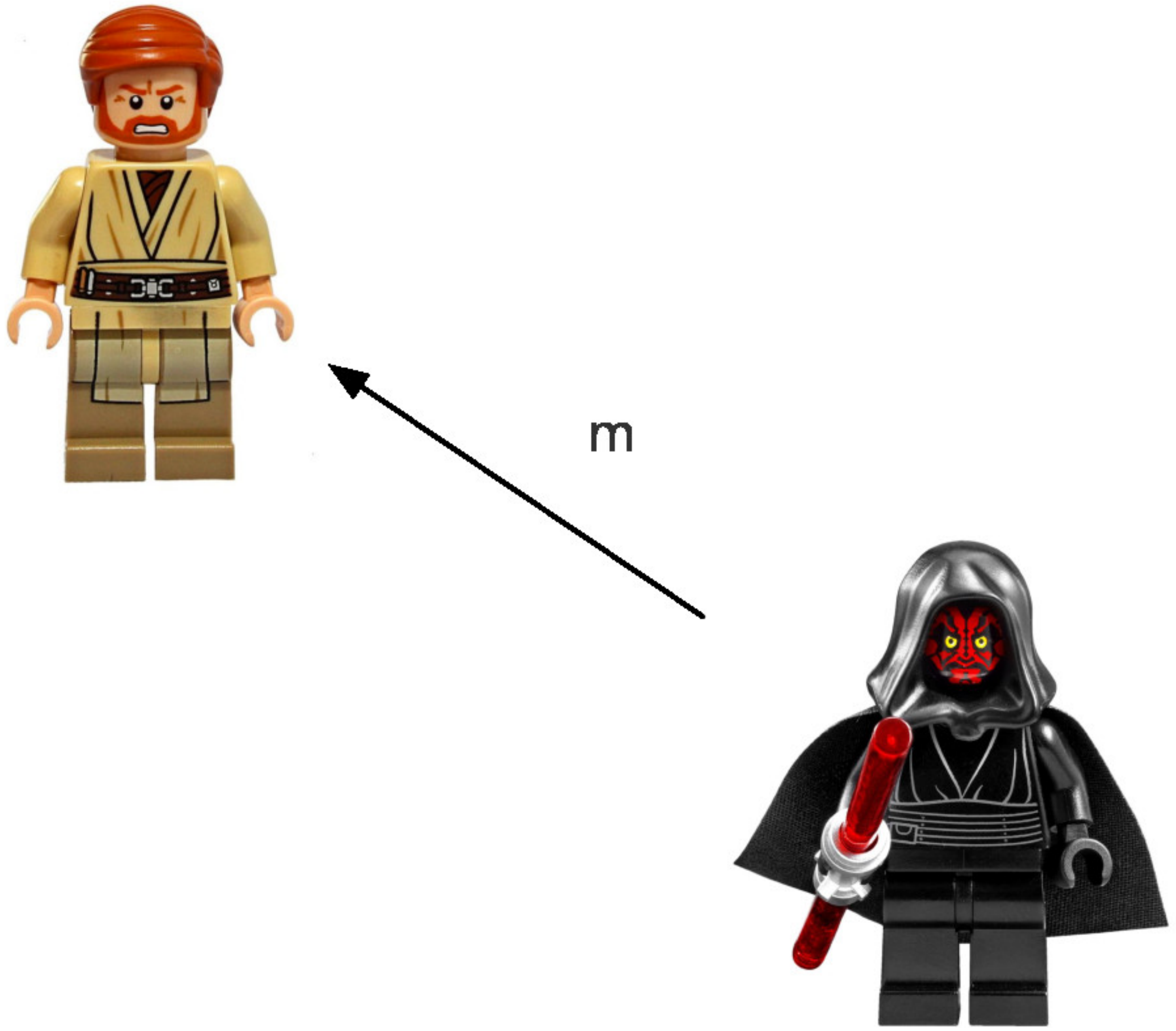
Then $\mathcal{M}$ **induces an implementation security model** $\mathcal{G}_I$:

- Whenever $\mathcal{G}_S$ runs specification
  - Then $\mathcal{G}_I$ runs implementation in $\mathcal{M}$
  - If there is leakage this is given to attacker with the result
- Additional oracles present in $\mathcal{M}$ offered to attacker

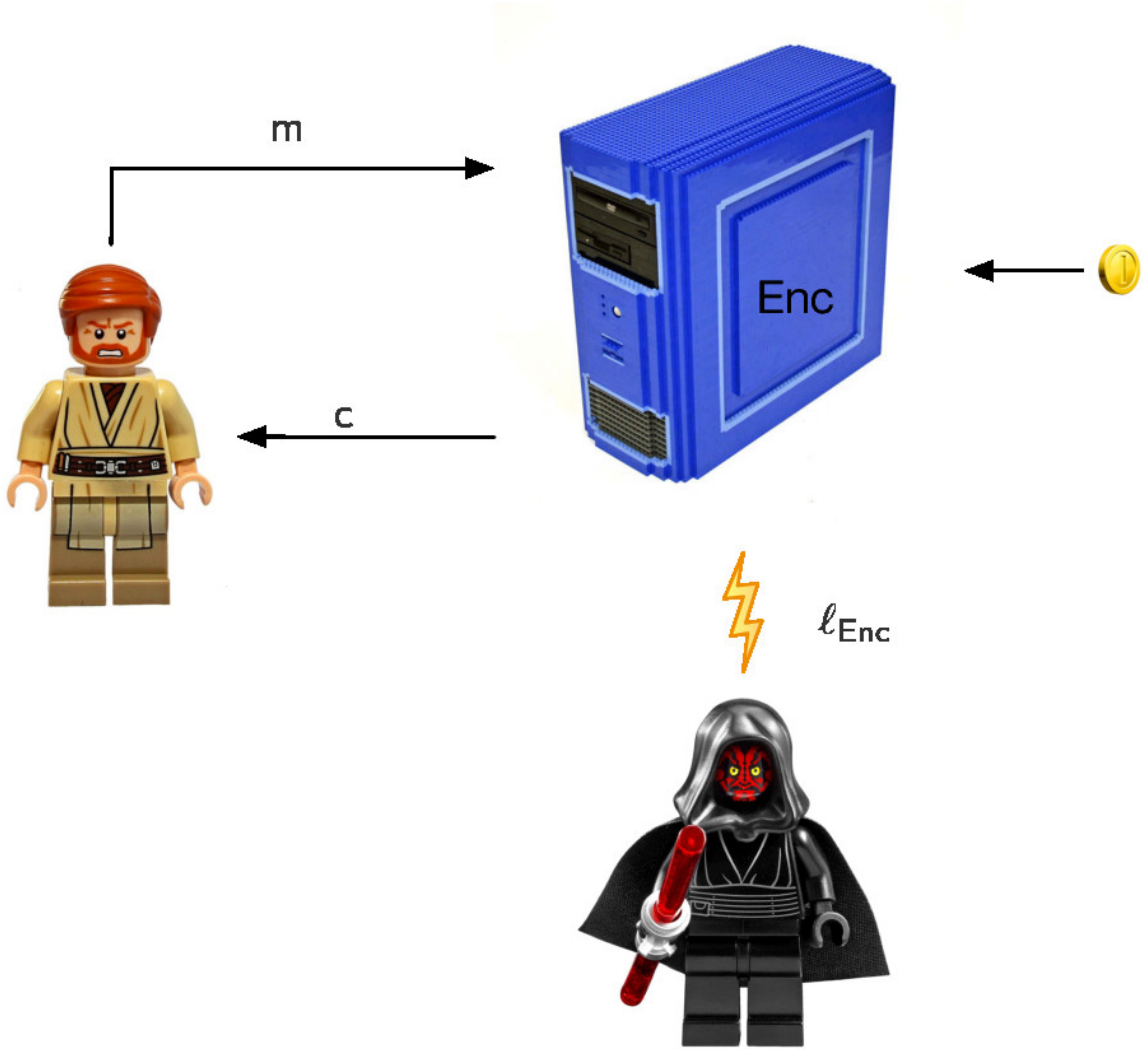
IM symmetric encryption security



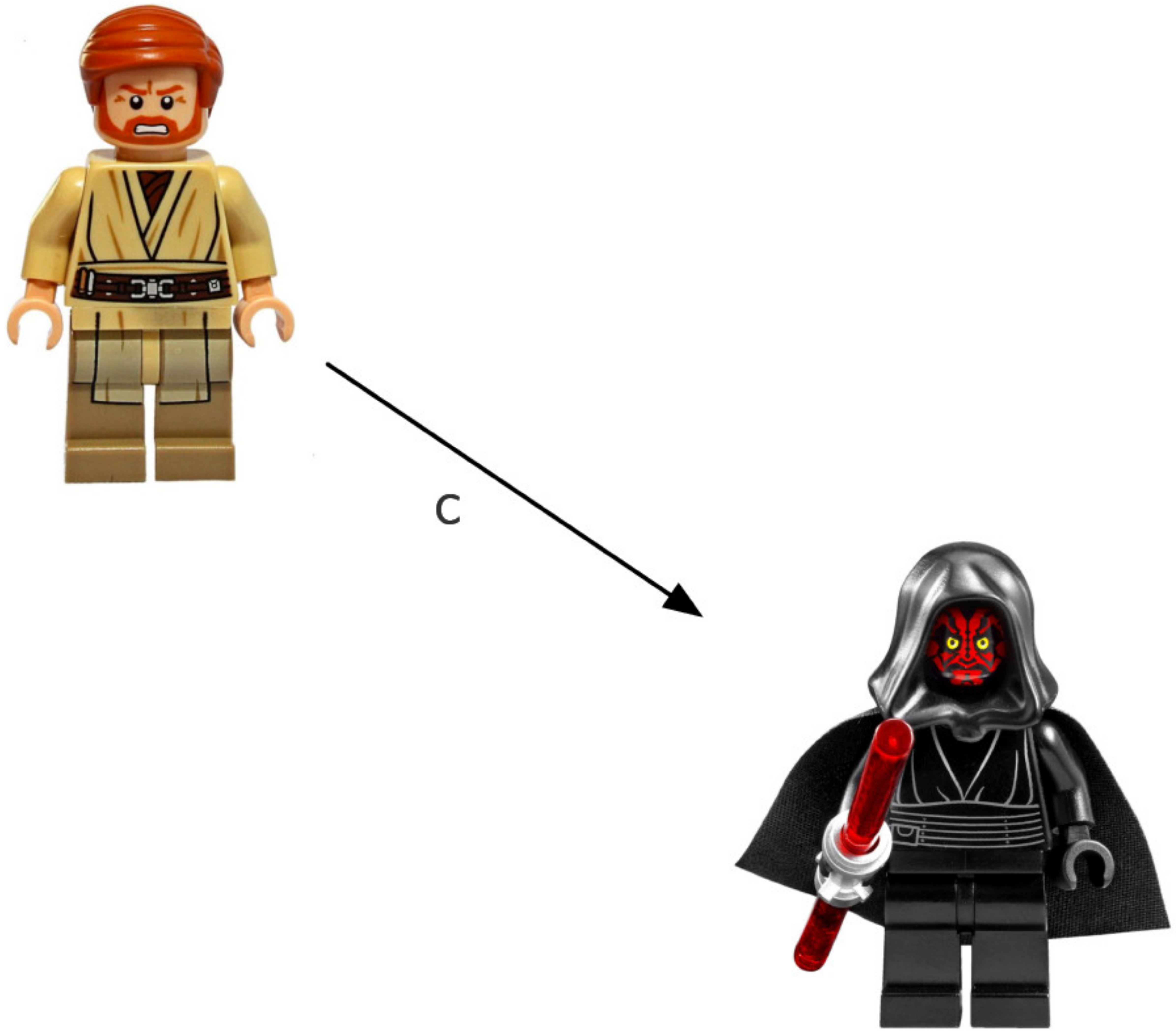
IM symmetric encryption security



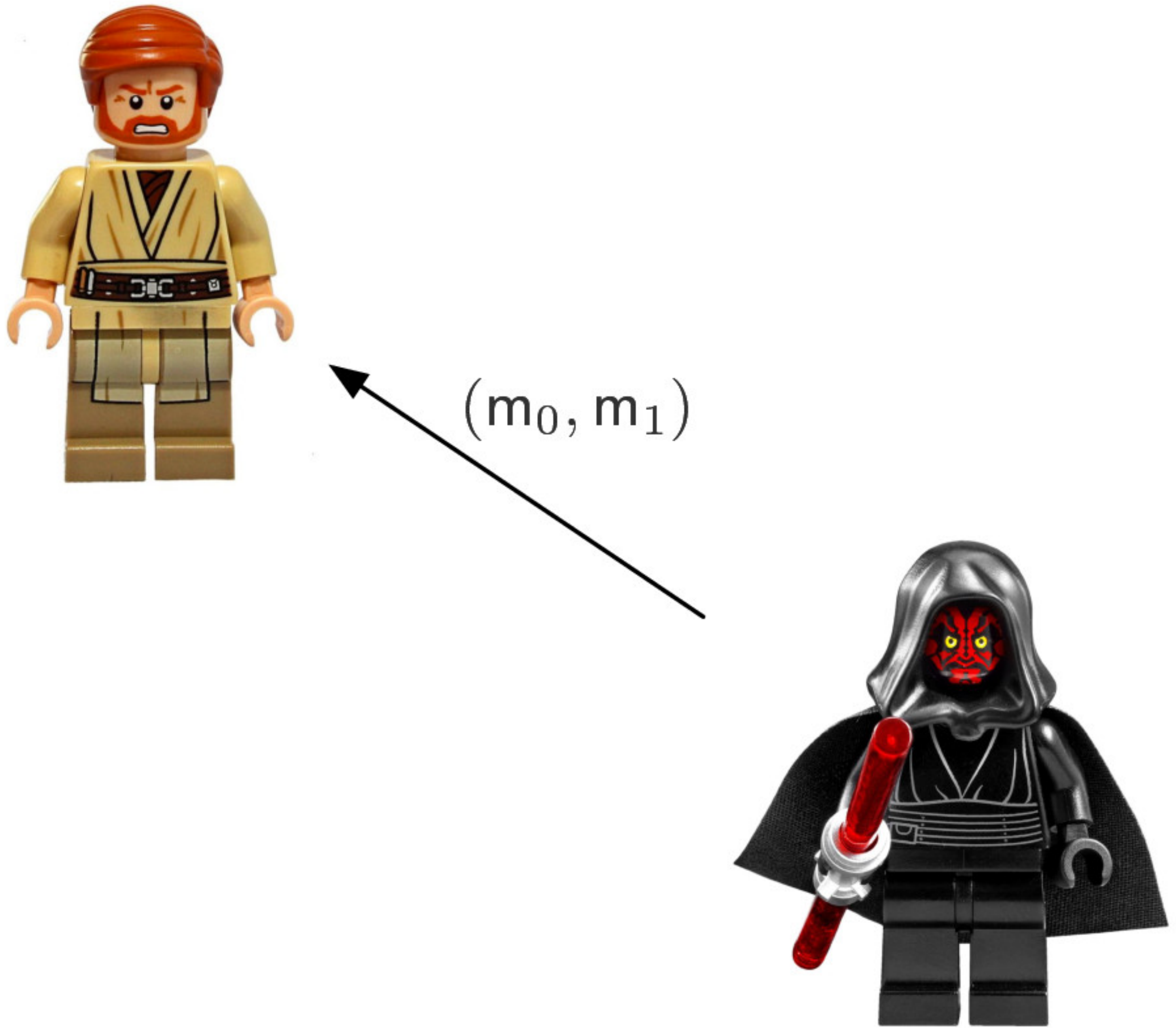
IM symmetric encryption security



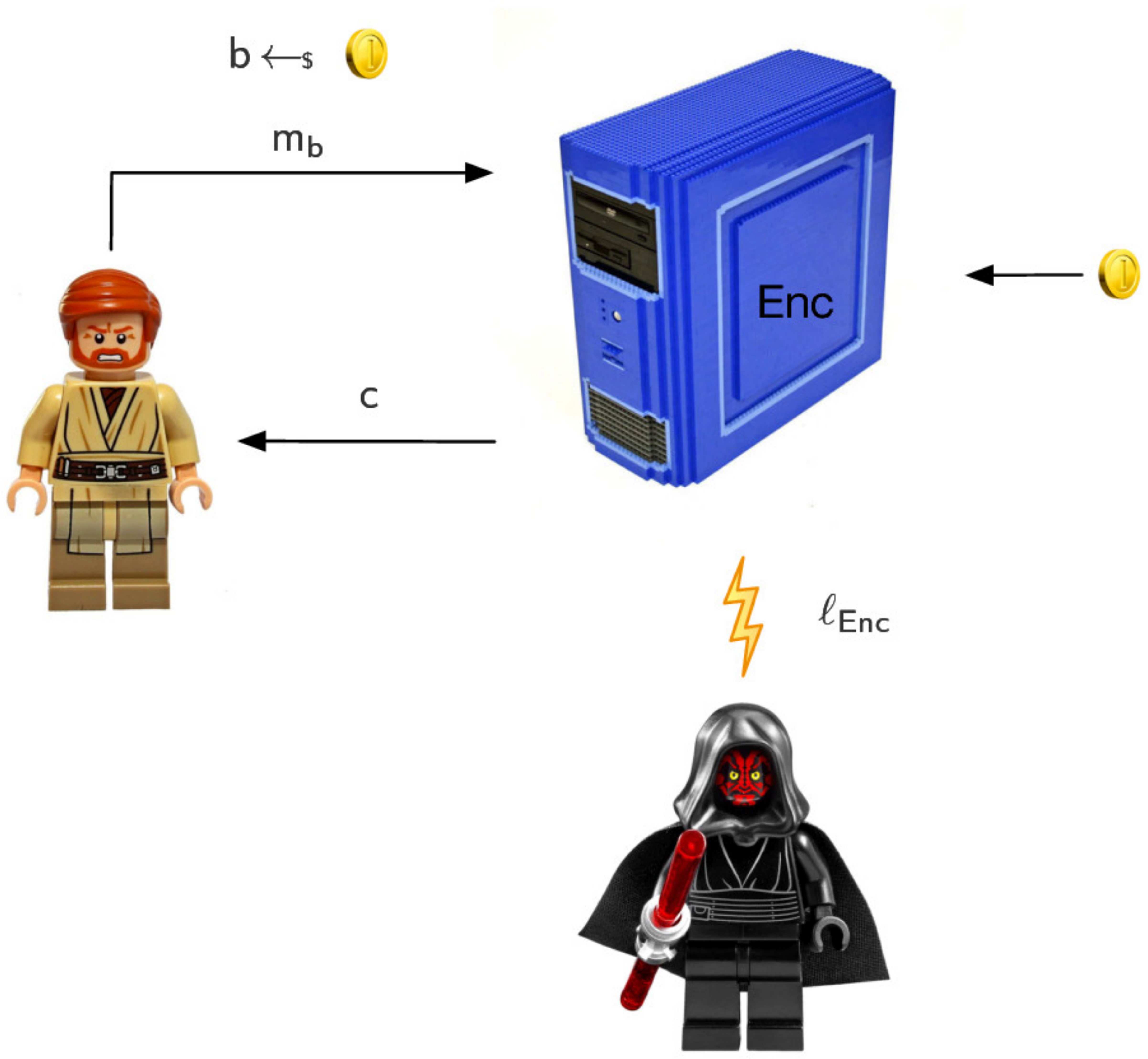
IM symmetric encryption security



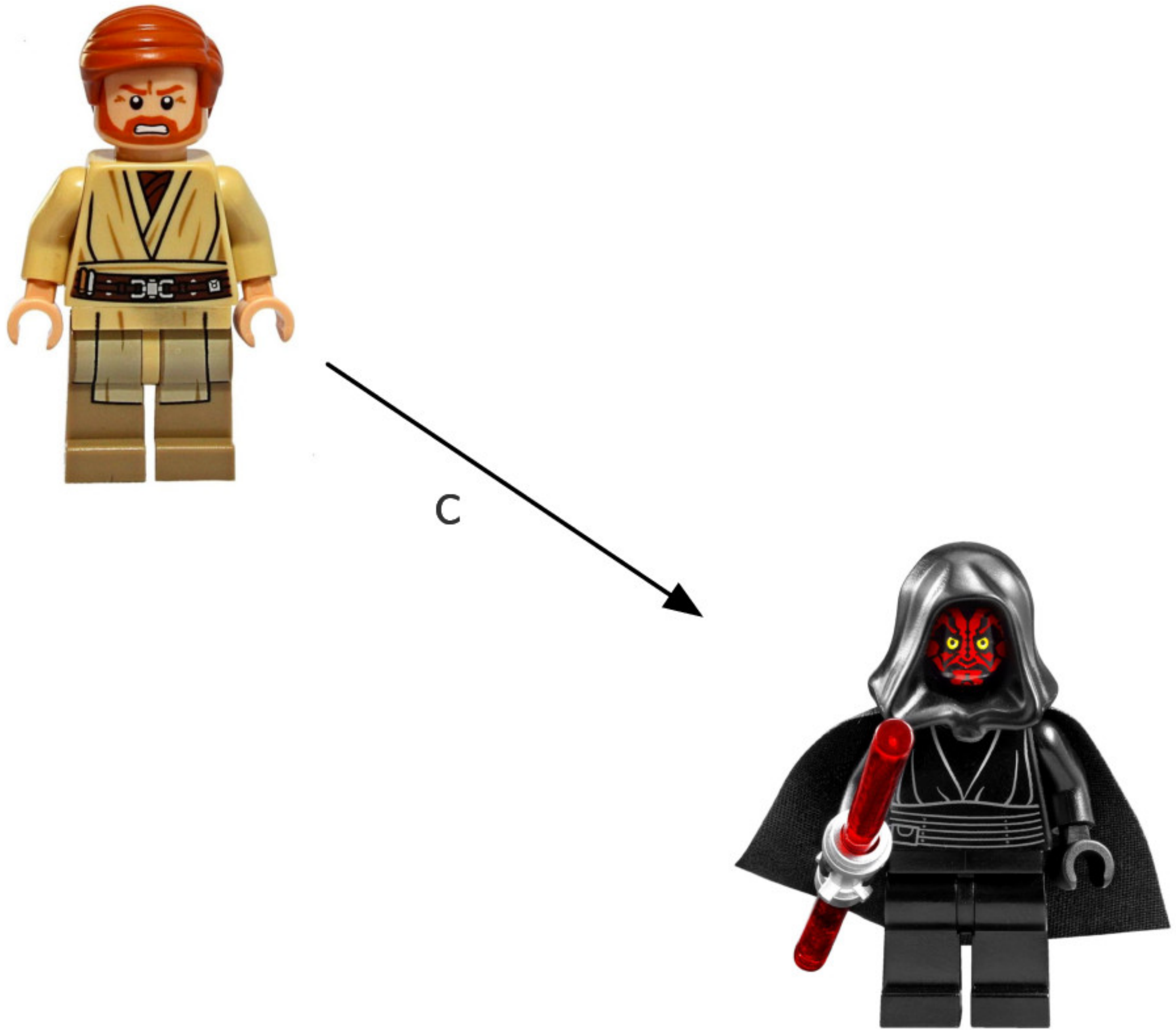
IM symmetric encryption security



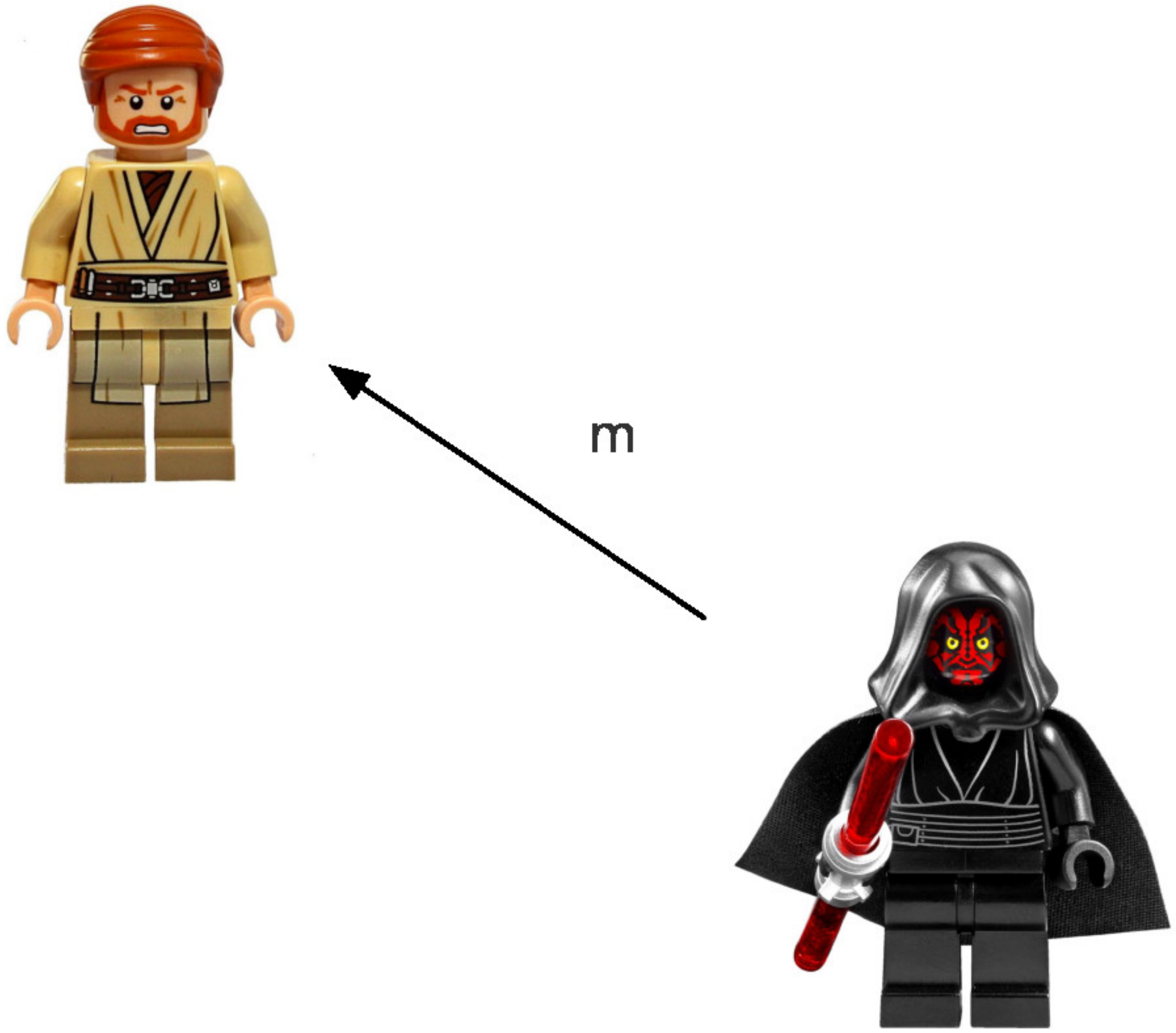
IM symmetric encryption security



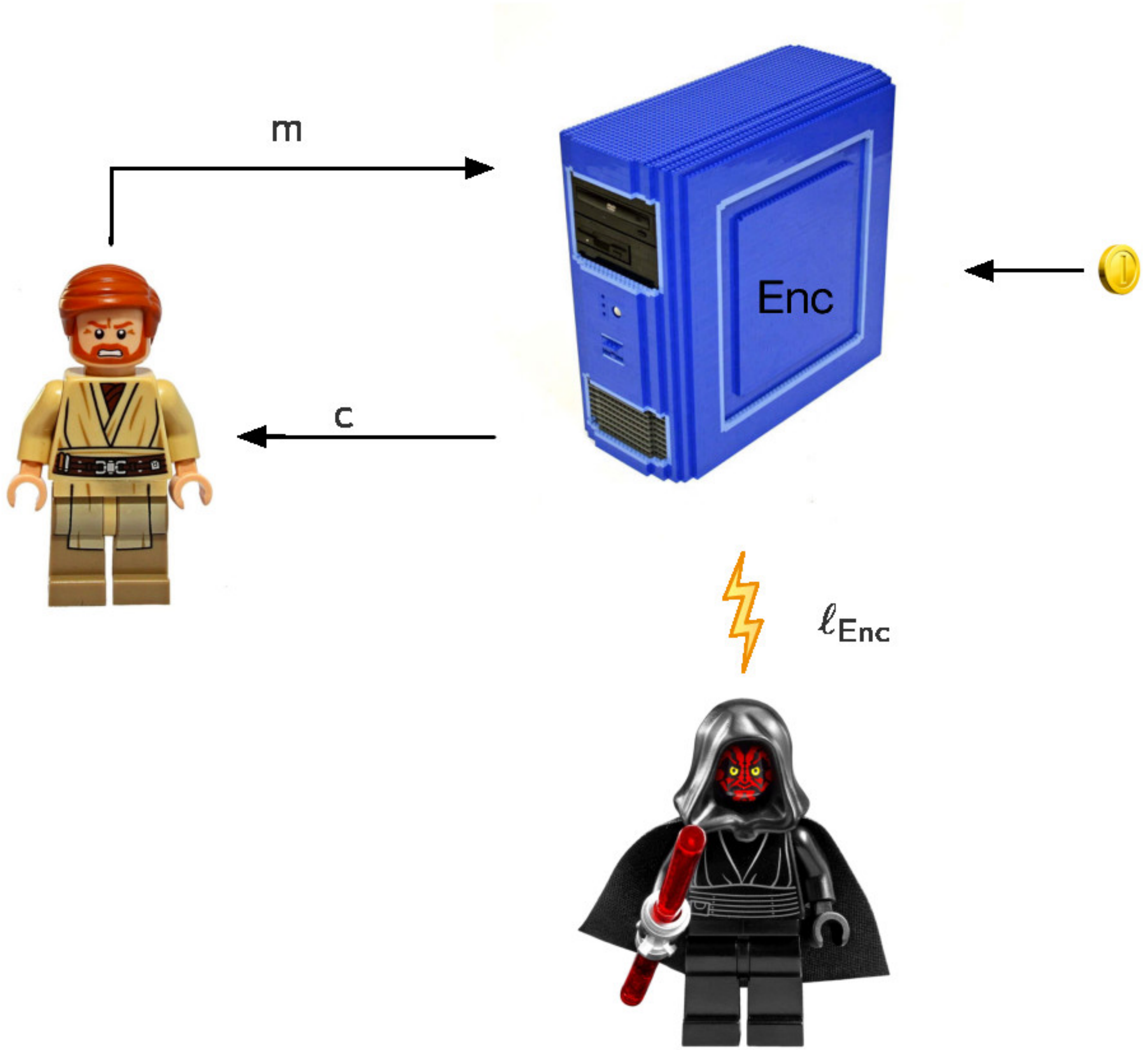
IM symmetric encryption security



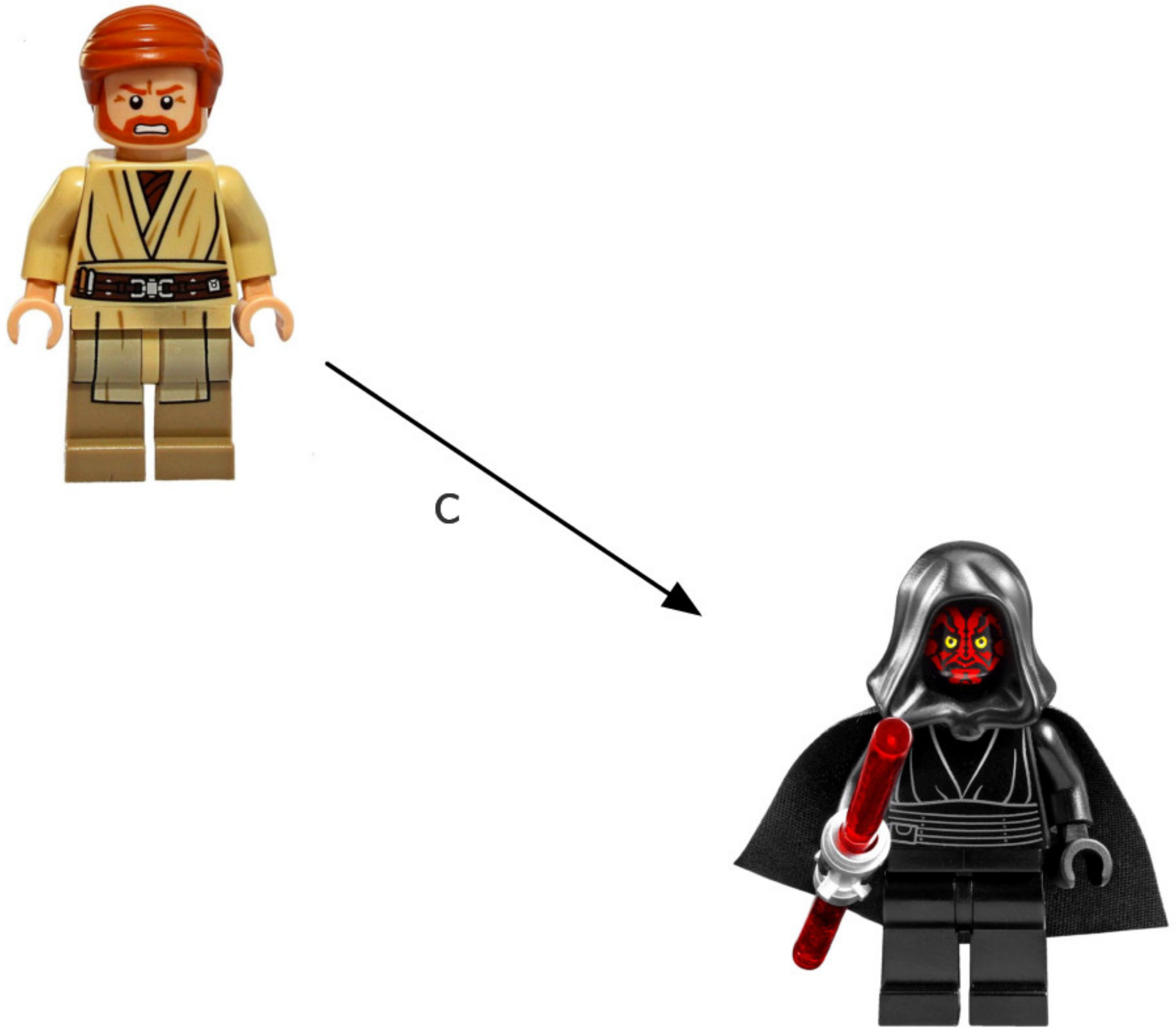
IM symmetric encryption security



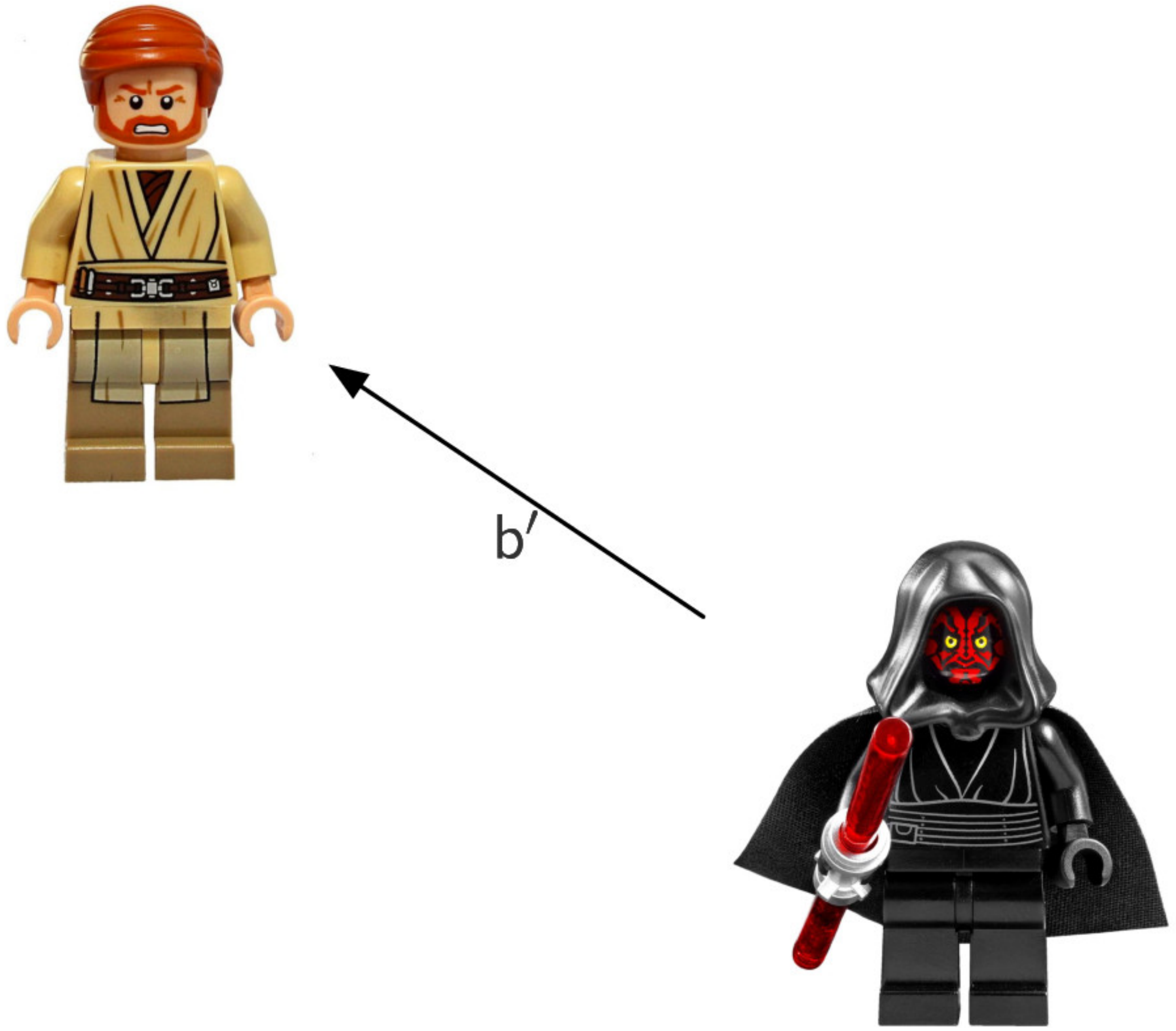
IM symmetric encryption security



IM symmetric encryption security



IM symmetric encryption security



IM symmetric encryption security

$$b' \overset{?}{=} b$$

What about the models we curently have?

Should we abandon the models we have?

- No!
- The complexity of such proofs needs to be tamed!
- We should assume the starting point of implementation security is a proof in a **Specification Model** (SM)

Our goal:

- Introduce **Implementation Model** (IM)
- Develop tools and techniques to ensure $A \wedge B \Rightarrow C$
  - A: Specification S is provably secure in SM
  - B: Implementation P is “valid” with respect to S
  - C: Implementation P is secure in the IM

Security models: Timing attacks

Rest of this talk: **timing attacks**.

What should $\mathcal{M}$ be?

- Utopia: exact model of the physical computer
- **Sufficient for experts in implementation security**:
  - control-flow should not depend on secrets
  - memory addresses should not depend on secrets
  - some specific operations should be avoided as they leak part of the input (e.g., shifts)

This is often known as the **constant-time** discipline.

(Not actually constant, just independent of secrets.)

Security models: Timing attacks

What security model does “constant-time” correspond to?

- Allow adversary to launch worst-case timing attack

More precisely, one assumes $\mathcal{M}$:

- reveals the code and its initial public state
- reveals which instructions were executed (PC trace)
- reveals accessed memory addresses (memory trace)
- reveals inputs for problematic operations (e.g., shifts)

Is this really the worst-case leakage for timing?

Security assumptions: Validating the model

We are dealing with machine code, so no detail is lost there.

We can validate the atomicity of operations for each platform:

- ensure processor instructions are constant-time
- memory access times only depend on addresses
- restrict the instructions you use if needed

We should take care about our attacker:

- We said nothing about randomness or code isolation
- We assume only instructions executed by our program leak
- We assume implementations have access to ideal randomness
- Could extend the model to capture these (should we?)

Important: for this setting we get automated proofs!

3. Security proofs

Our goal is to prove $A \wedge B \Rightarrow C$

- A: Specification S is secure in SM
- B: Implementation P is "valid" with respect to S
- C: Implementation P is secure in the IM

We know what A and C means:

- Cryptographers take care of A
- We want C: security as in A plus no timing attacks
- We don't want to write another proof!
- **That's why we need the B side condition**

Security proofs via program verification

What does B mean?

- Implementation P is “valid” wrt specification S

PL community has been thinking about B for a while

- This is called **program verification**

If we can make B side condition more precise then . . .

PL *theory* and *tools* can be used to complete our proofs

Meta theorem (informal)

Let us try to prove $A \Rightarrow C$ by reduction

- Take successful attacker $\mathcal{A}$ against implementation P
- Construct an attacker $\mathcal{B}^{\mathcal{A}}$ that breaks specification S

We want to show that

$$\mathcal{A} \text{ wins in } \mathcal{G}_I \Rightarrow B^{\mathcal{A}} \text{ wins in } \mathcal{G}_S$$

This implies $\mathcal{B}$ has large advantage if $\mathcal{A}$ does.

Implementation break implies specification break.

Specification security carries over to implementation.

Meta theorem: here focus on restricted version

1. P represents values just like S
2. $\mathcal{M}$ models constant time leakage
   - When running P on input x we get $(P(x), CT(P, x))$
   - $CT(P, x)$ is list of instructions/addresses used by $P(x)$
3. $\mathcal{M}$ has access to ideal randomness
   - S and P consume exactly same random coins

Meta theorem: proof technique

Pattern in reduction for each oracle call:

- $\mathcal{B}$ tries to simulate IM for $\mathcal{A}$
- Suppose $\mathcal{A}$ queries message m to IM encryption oracle
- $\mathcal{B}$ can get $c_S = S_{\mathsf{Enc}}(m, sk; r)$ from SM encryption oracle

- $\mathcal{A}$ is expecting IM oracle answer created by $\mathcal{M}$
  - $c_I = P_{\mathsf{Enc}}(m, sk; r)$
  - $\ell = CT(P_{\mathsf{Enc}}(m, sk; r))$

- What can $\mathcal{B}$ do?
  - Just give $c_I = c_S$ to $\mathcal{A}$?
  - Make up plausible ℓ (without knowing sk or r)?

Meta theorem: proof technique

The following conditions are sufficient for proof to go through:

- **Functional Correctness:** $\forall x.\ P(x) = S(x)$
- **Leakage Simulation**: $\exists f . \forall x . CT(P, x) = f(pub(x))$

Conditions permit using PL theory to complete the proof

Functional correctness is a classical PL problem

But what is Leakage Simulation in PL terms?

Meta theorem: PL theory completes the proof

Another classical PL notion is called non-interference:

- $\forall\ x\ x'.\ pub(x) = pub(x') \Rightarrow pub(P(x)) = pub(P(x'))$

Can automatically verify this for $pub(P(x)) = CT(P, x)$.

Non interference implies leakage simulation:

- Suppose P is non-interferent
- Simulator $f(pub(x))$ can be defined as follows:
  - Given x_p pick any x conditioning on $pub(x) = x_p$
  - Return $CT(P, x)$
  - OK because all pub-equivalent inputs give same leakage

Back to the reduction:

- S_{Enc} must be non-interferent when $|m|$ is public
- $\mathcal{B}$ picks any (m', sk', r') with |m| = |m'| and runs $P(m', sk'; r')$ to compute the list of instructions/addresses.

Meta theorem: PL tools complete the proof

Meta theorem generalizes to large class of security games and various types of leakage

In the particular case of timing attacks in CT-model, PL side-conditions can be proven automatically!

Next class: how to generate P from S such that

1. P is functionally correct
   - This is the classical problem of **certified compilation**
2. P is non-interferent
   - Constant-time leakage brought up a new PL problem
   - Contrarily to general non-interference it can be checked automatically

The End

Thank you for your attention.

Questions?