Fujisaki-Okamoto - a recipe for post-quantum public key encryption

CrySP Speaker Series on Privacy

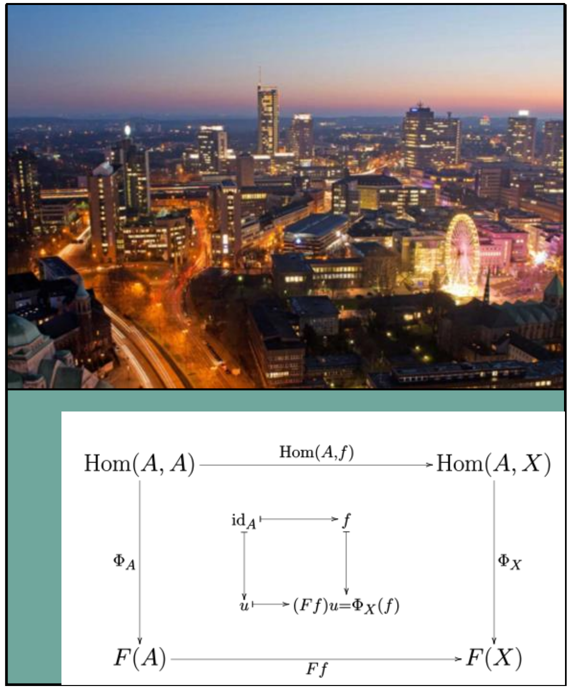
Kathrin Hövelmanns

April 3rd, 2024, University of Waterloo

EINDHOVEN UNIVERSITY OF TECHNOLOGY

About me

Studied Math in Essen, GER…

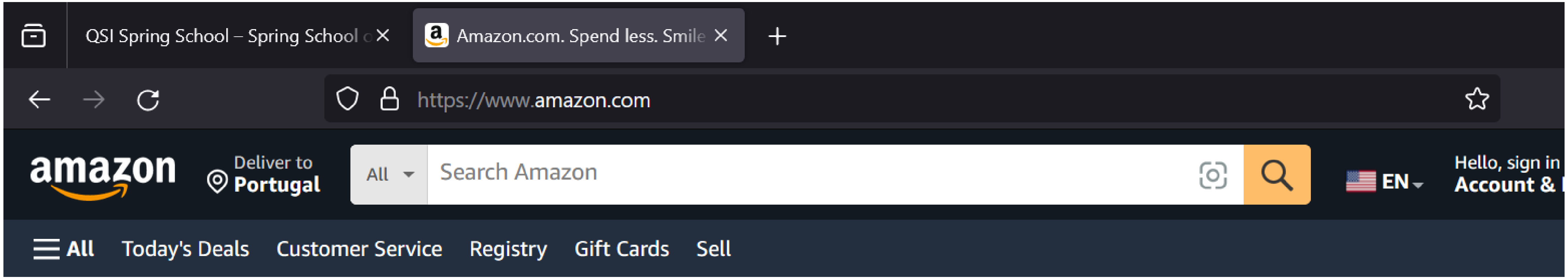


… PhD on Crypto (Dec '20) in Bochum, still GER …

KYBER

… postdoc, then faculty (Sep '22) at TUEe, NL

Did you use any cryptography today?



Amazon uses https → https invokes TLS → TLS uses crypto

TLS is everywhere:

shopping, banking, Netflix, gmail, Facebook, ...

Quantum computers vs crypto

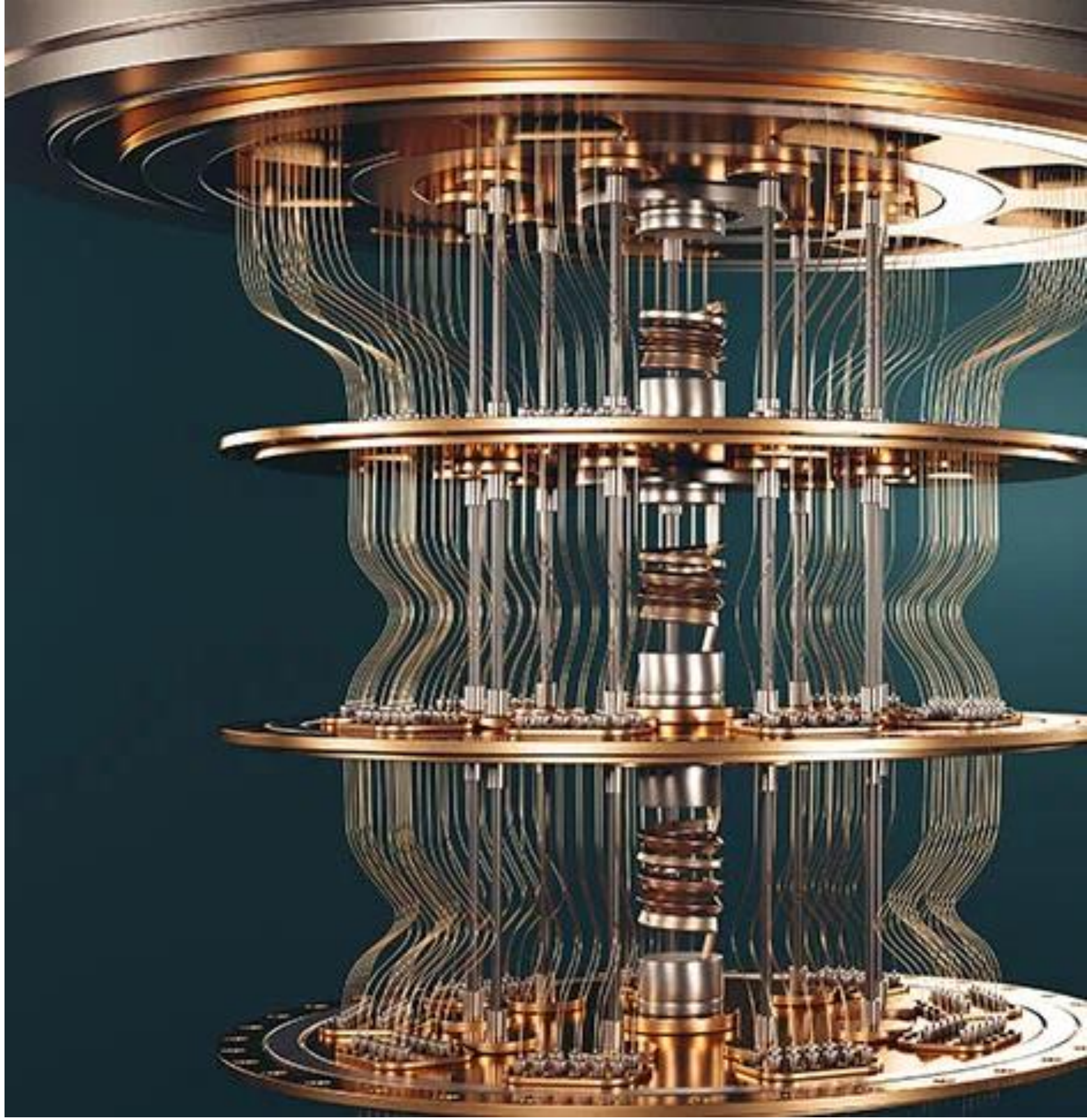
? Why care about solutions today?

! Major investments (est.: $35.5 billion*)

Quantum Delta Nederland · Google · IBM · intel · Microsoft · QUANTUM FLAGSHIP

‘Store now, exploit later’

‘The standards are coming anyways’ ☺



* World Economic Forum, Insight report, September ‘22

Secret-key crypto: quantum impact does not seem to be catastrophic -

but **how to share secret keys ad hoc**?

Public-key encryption (PKE)

Alice

Bob

Image source: xkcd.com

Public-key encryption (PKE)

Image source: xkcd.com

Public-key encryption (PKE)

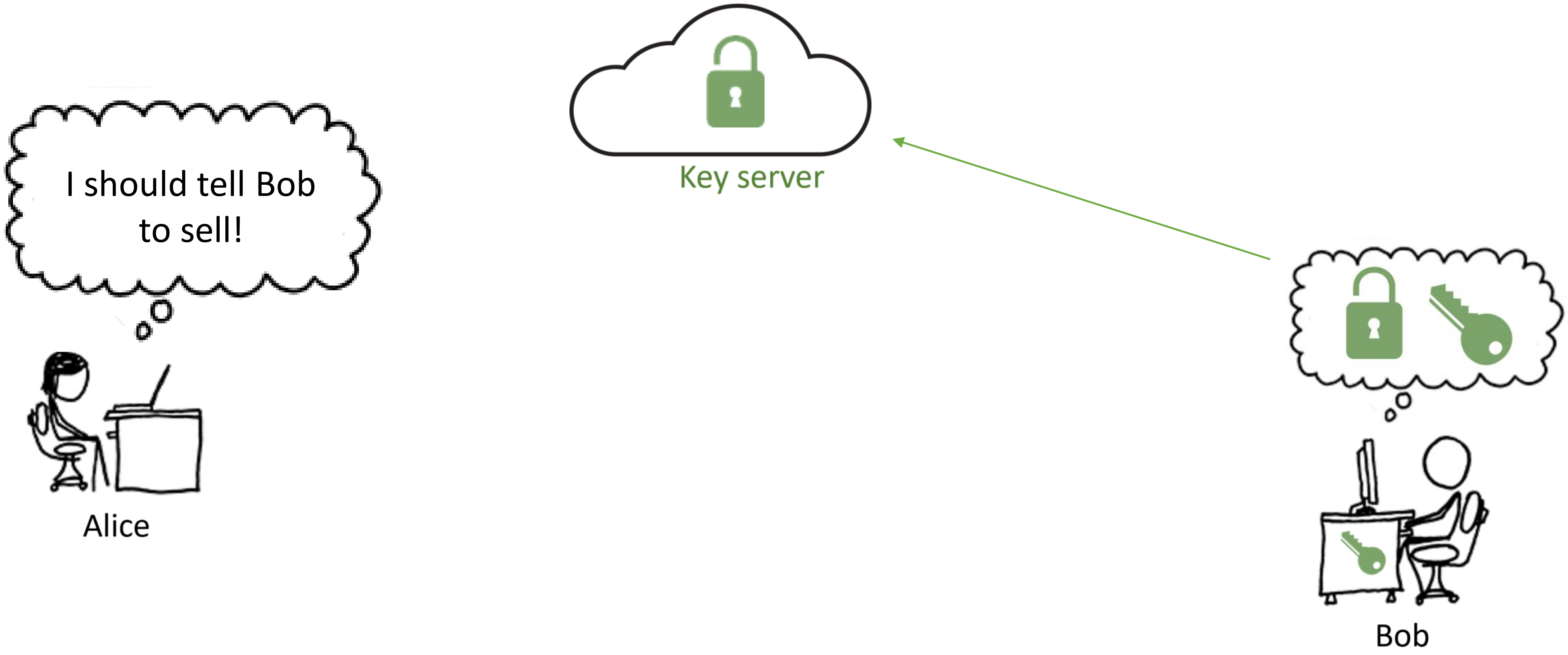


Image source: xkcd.com

Public-key encryption (PKE)

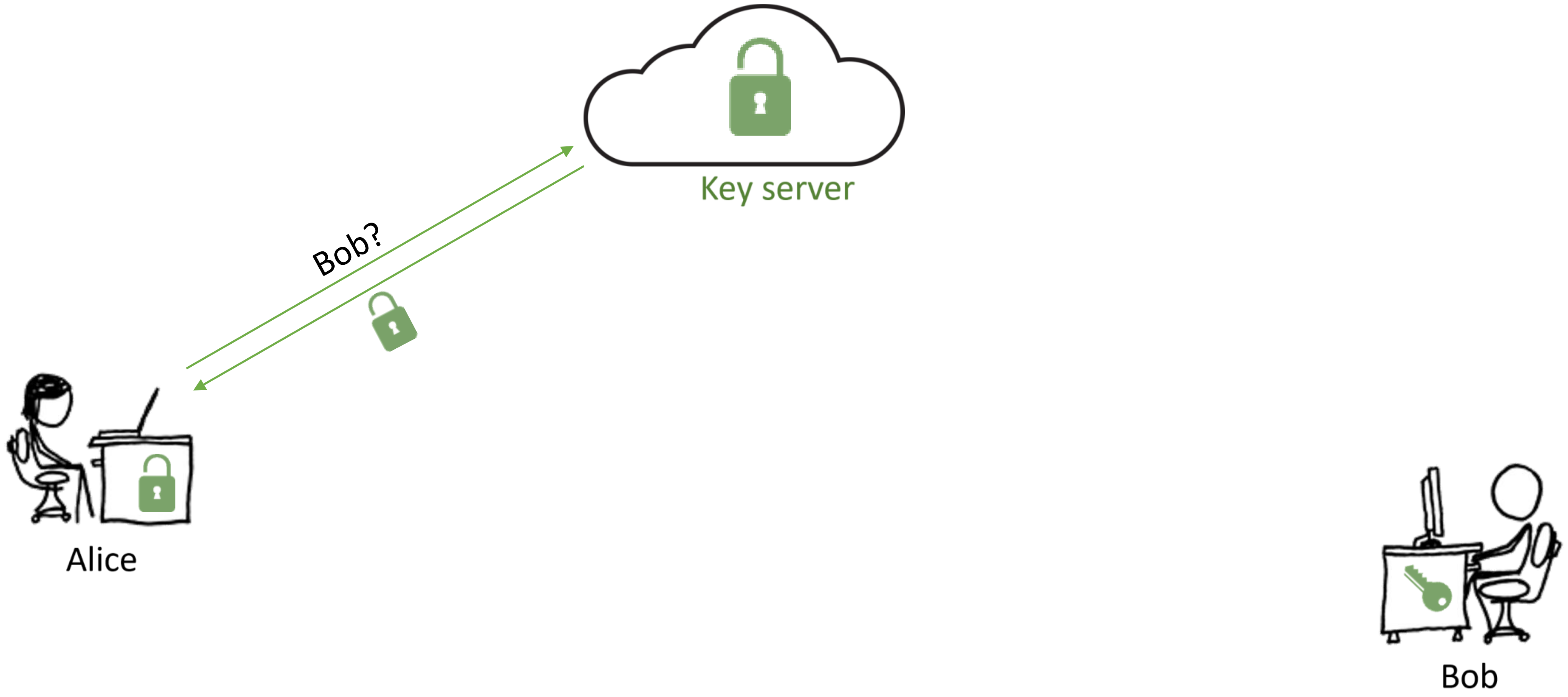


Image source: xkcd.com

Public-key encryption (PKE)

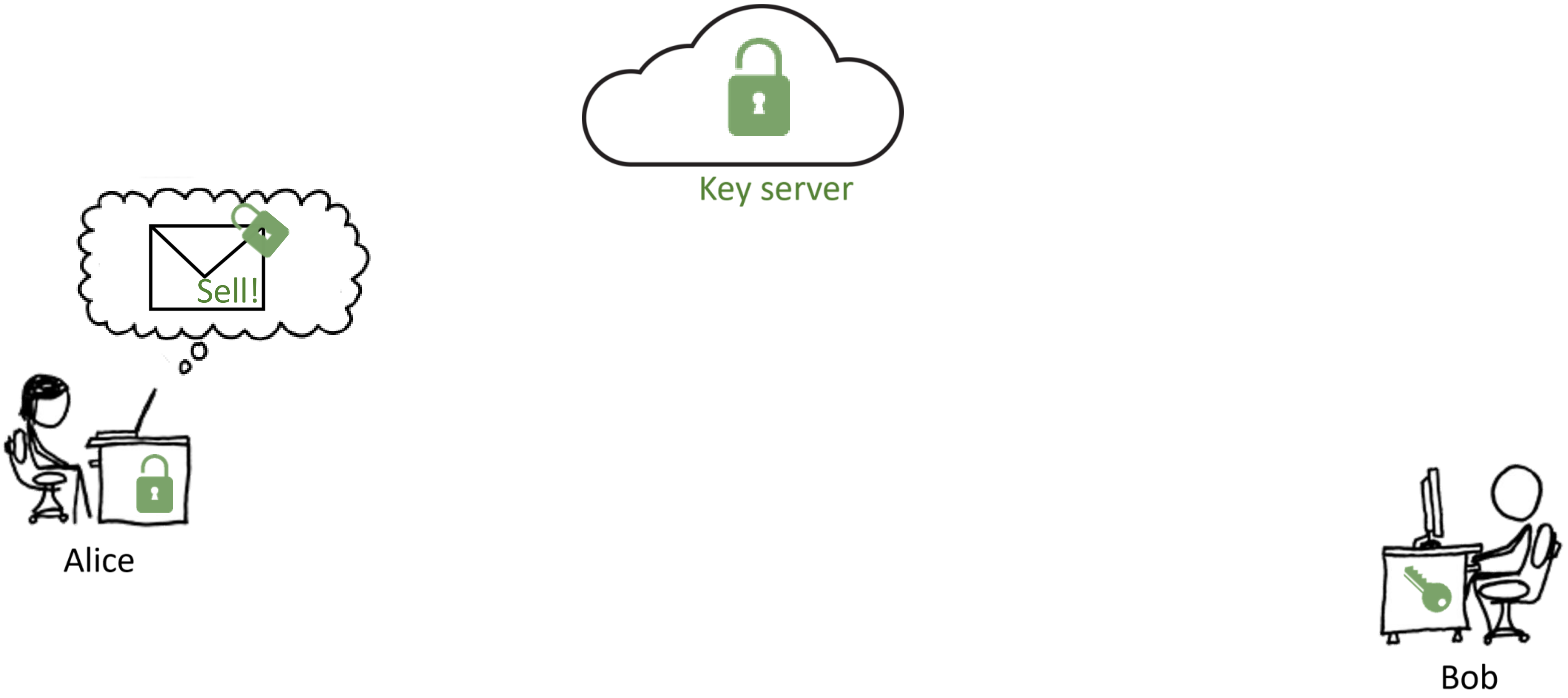


Image source: xkcd.com

Public-key encryption (PKE)

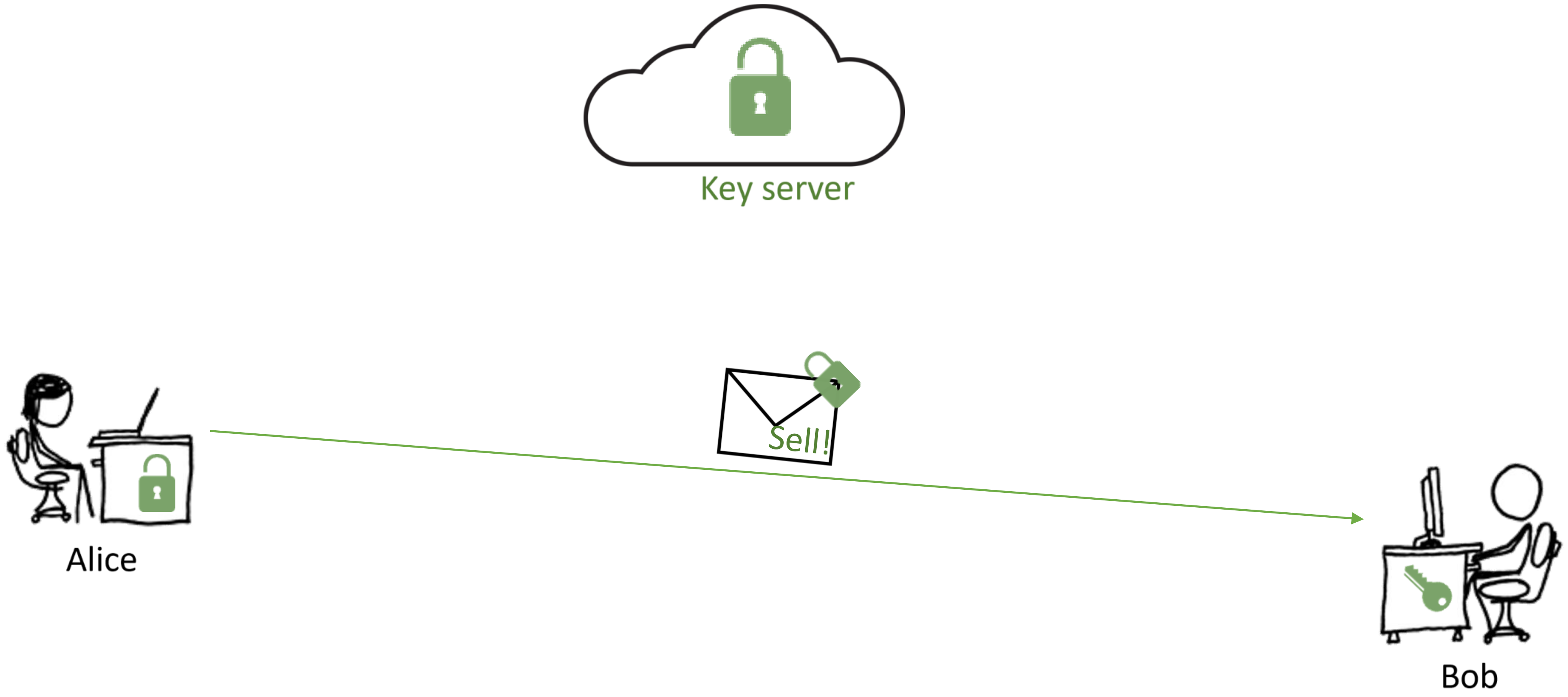


Image source: xkcd.com

Public-key encryption (PKE)

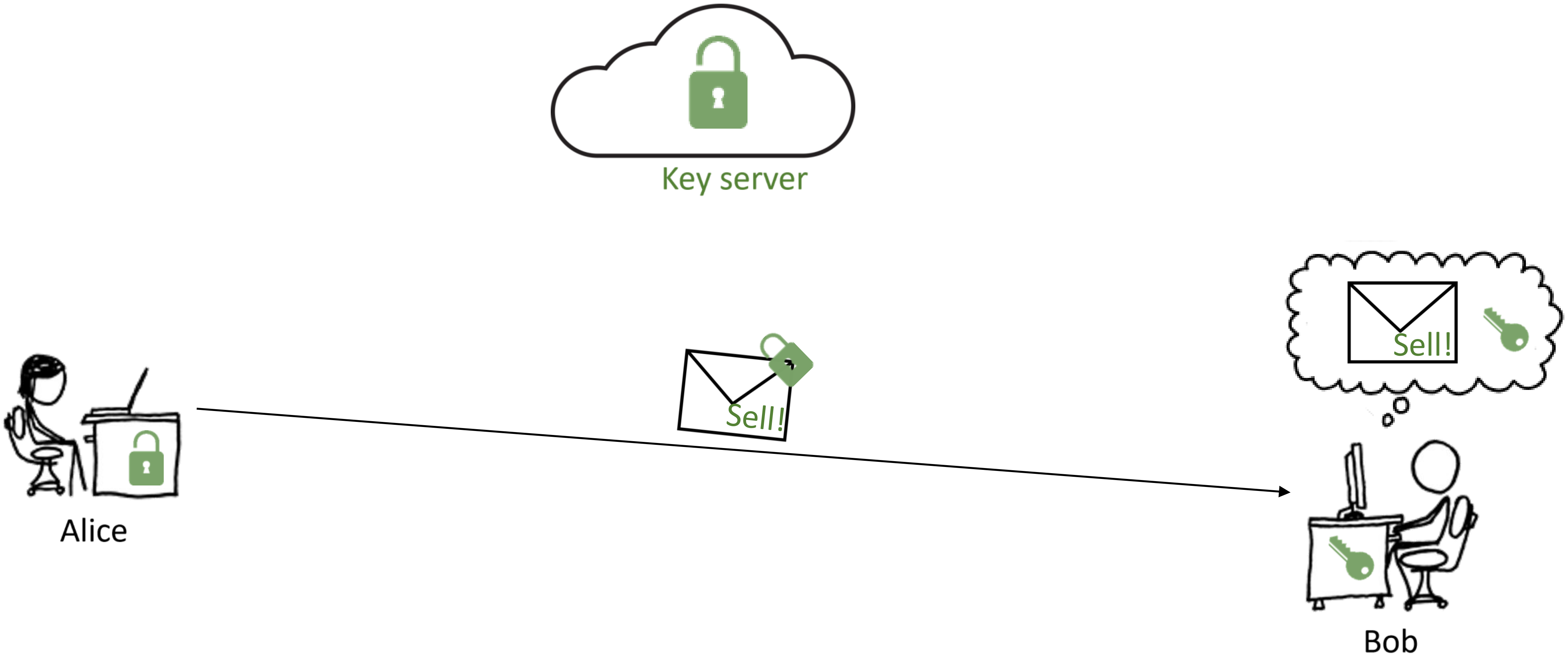


Image source: xkcd.com

Public-key encryption (PKE)

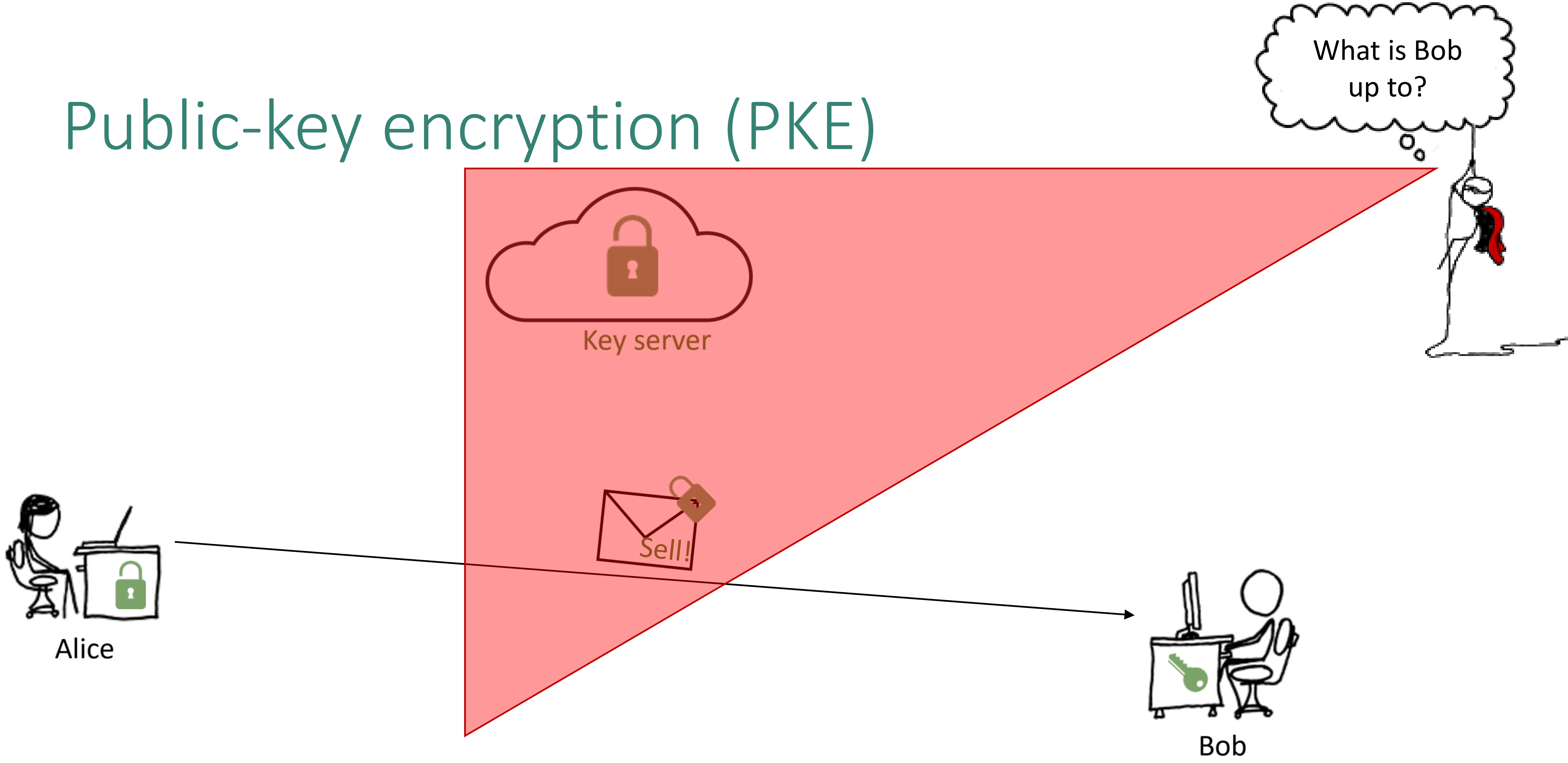
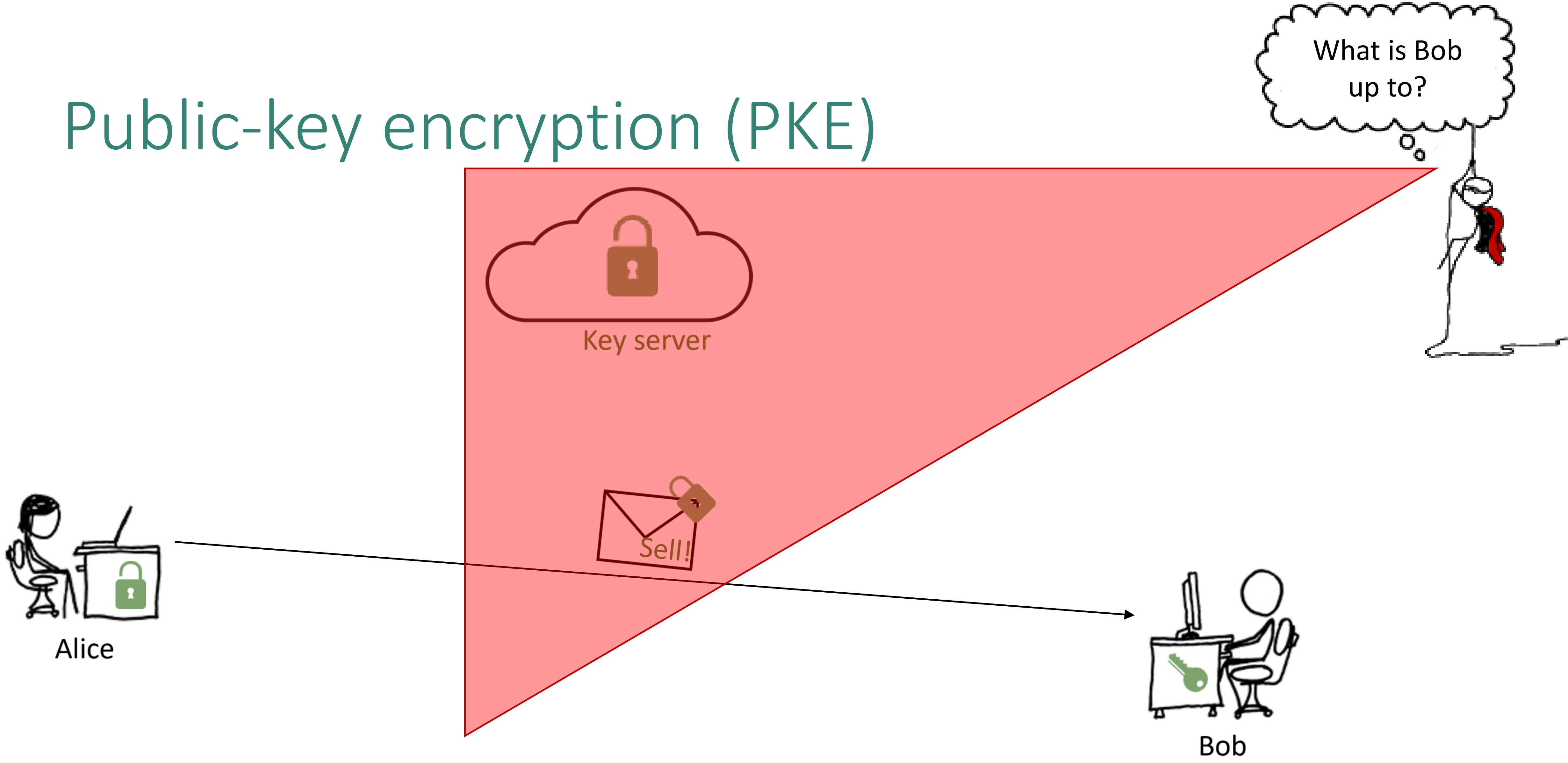


Image source: xkcd.com

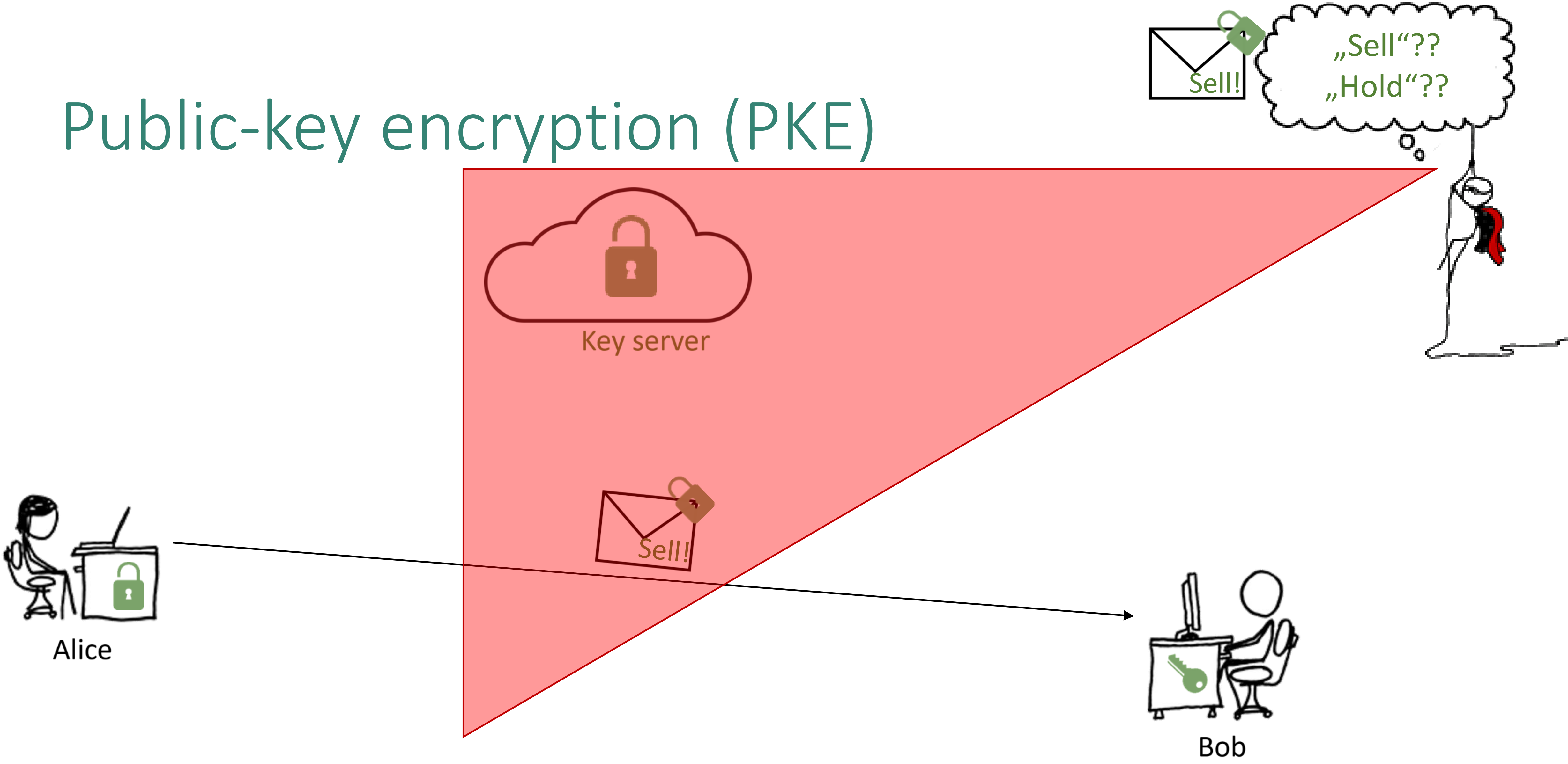
Public-key encryption (PKE)



Obvious goal: without the secret key, encryptions should be hard to invert.

Image source: xkcd.com

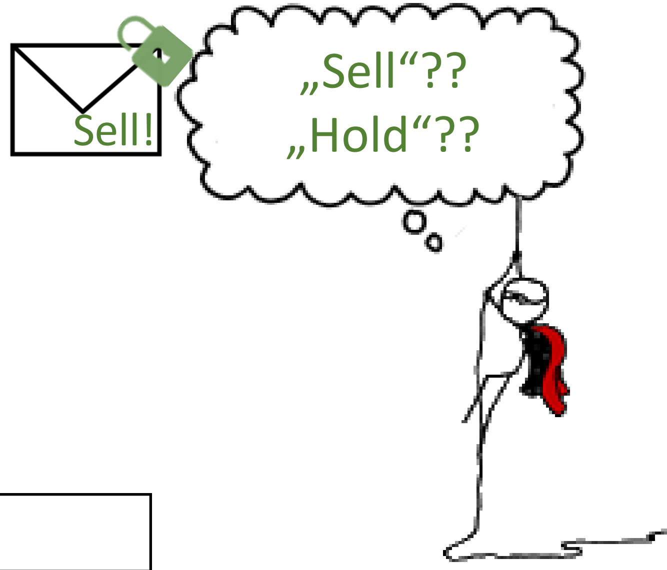
Public-key encryption (PKE)



Obvious goal 2: encryptions should not leak significant info about their plaintexts.

Image source: xkcd.com

IND-CPA security game



INDistinguishability under **C**hosen-**P**laintext **A**ttacks

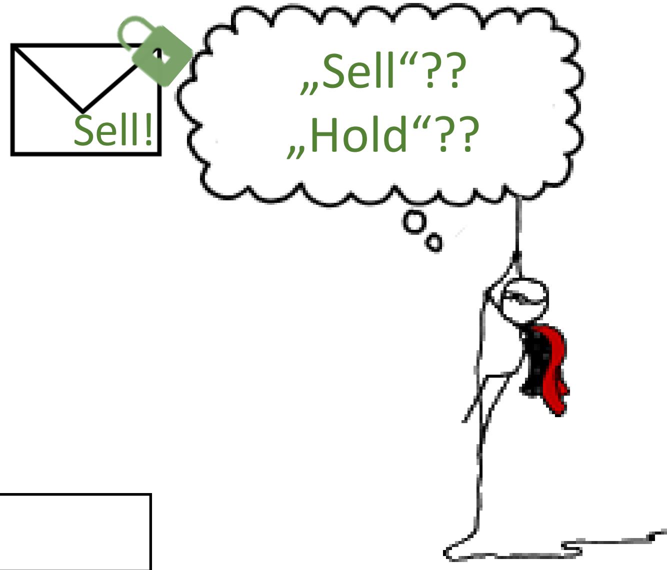
Left game	Right game
Adversary gets public key Adversary chooses two messages m_1 and m_2 Adversary gets encryption of:	
m_1	m_2
Adversary guesses which game it's playing	

Question: Can we have IND-CPA security if encryption is deterministic*?

* = encrypting a message always gives the same result

Image source: xkcd.com

IND-CPA security game



INDistinguishability under **C**hosen-**P**laintext **A**ttacks

Left game	**Right game**
Adversary gets public key Adversary chooses two messages m_1 and m_2 Adversary gets encryption of:	
m_1	m_2
Adversary guesses which game it's playing	

Question: Can we have IND-CPA security if encryption is deterministic*?

No. (But encryption could still be hard to invert.)

* = encrypting a message always gives the same result

Image source: xkcd.com

Chosen-ciphertext attacks

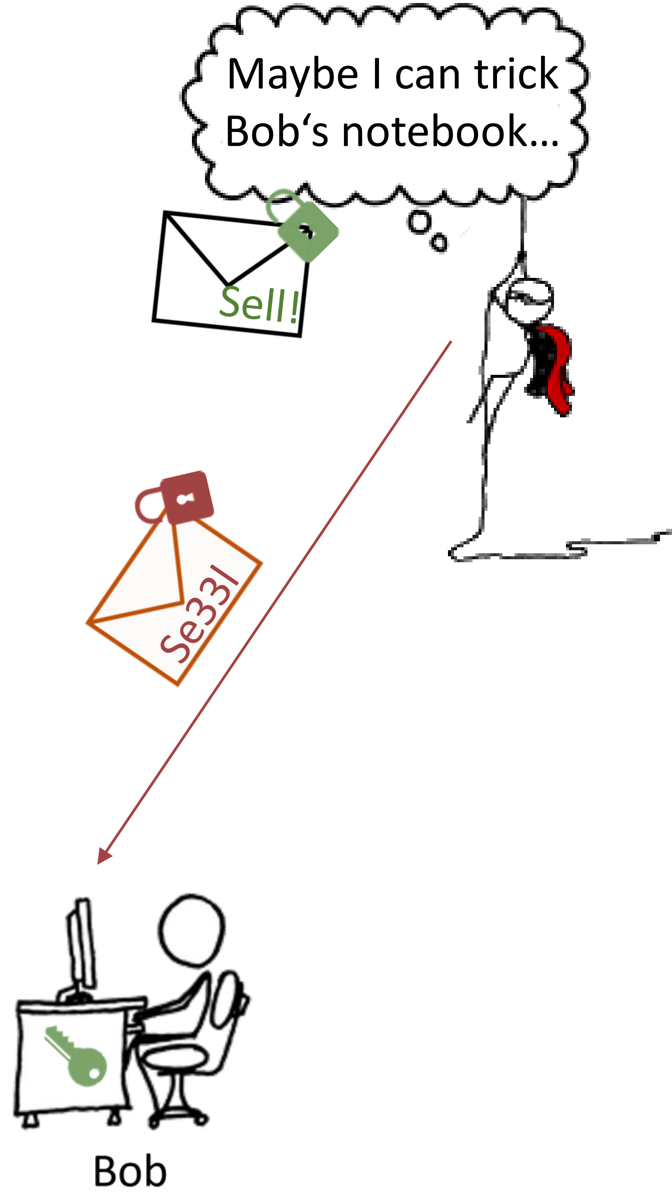


Image source: xkcd.com

Chosen-ciphertext attacks

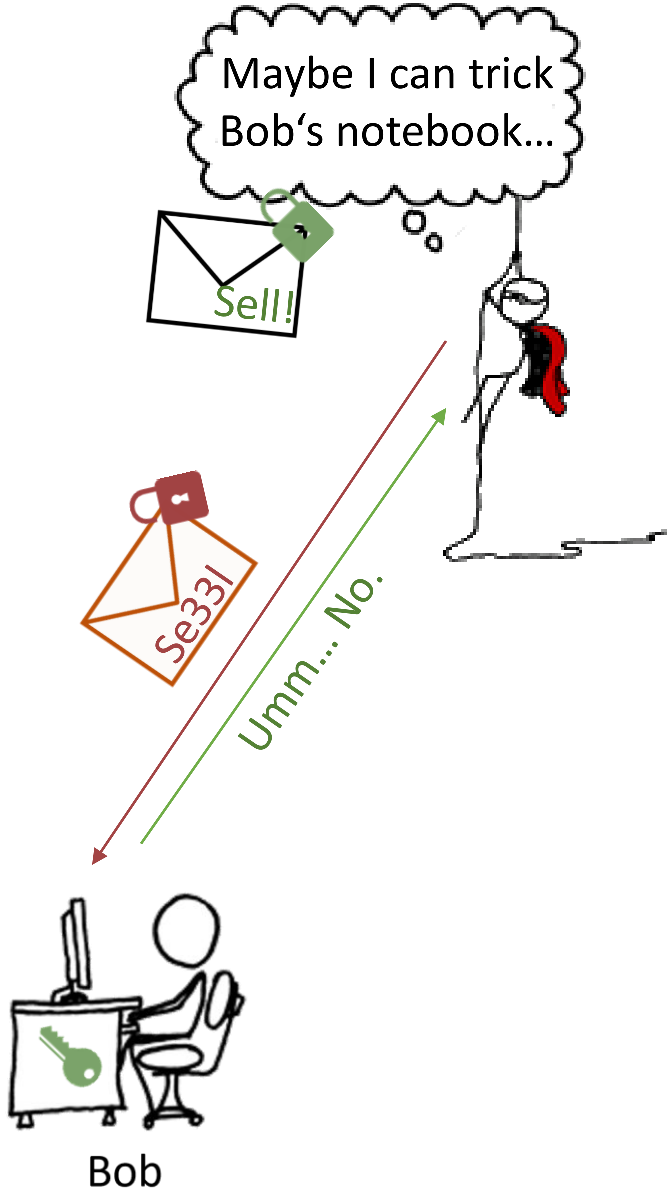


Image source: xkcd.com

Chosen-ciphertext attacks

Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1

Daniel Bleichenbacher

Bell Laboratories
700 Mountain Ave.
Murray Hill, NJ 07974
E-mail: bleichen@research.bell-labs.com

[Bleichenbacher 98]

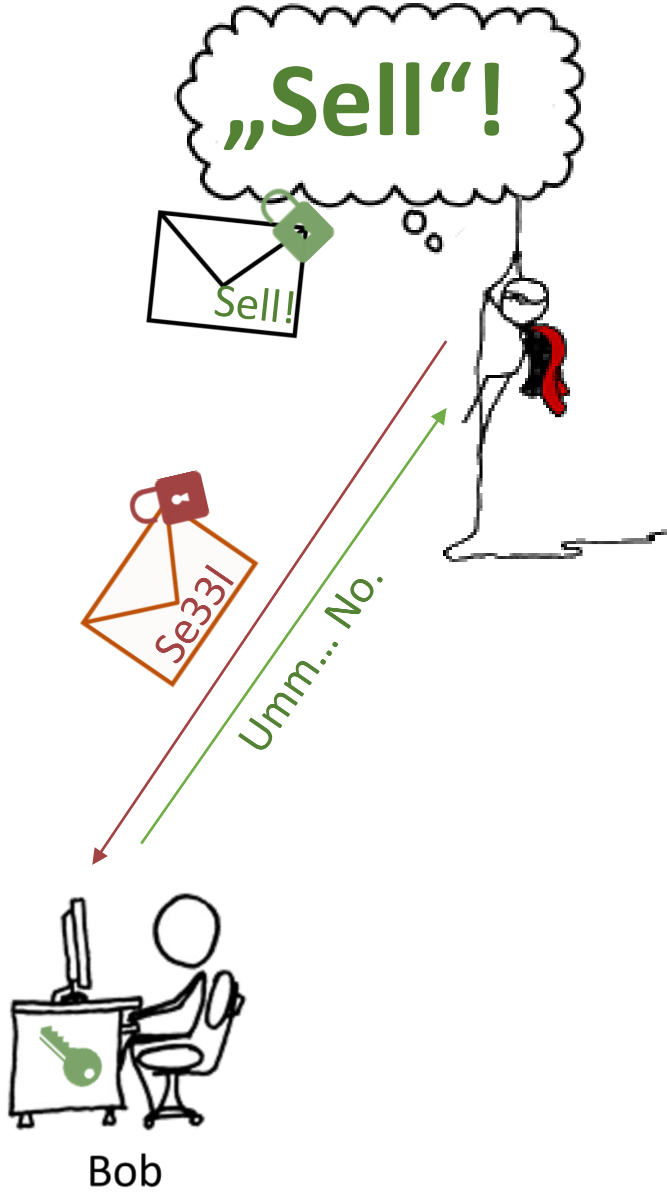
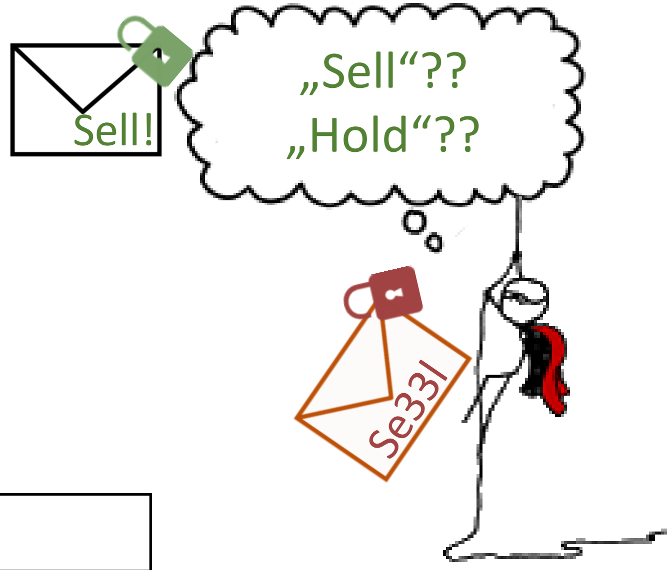


Image source: xkcd.com

IND-CCA security game



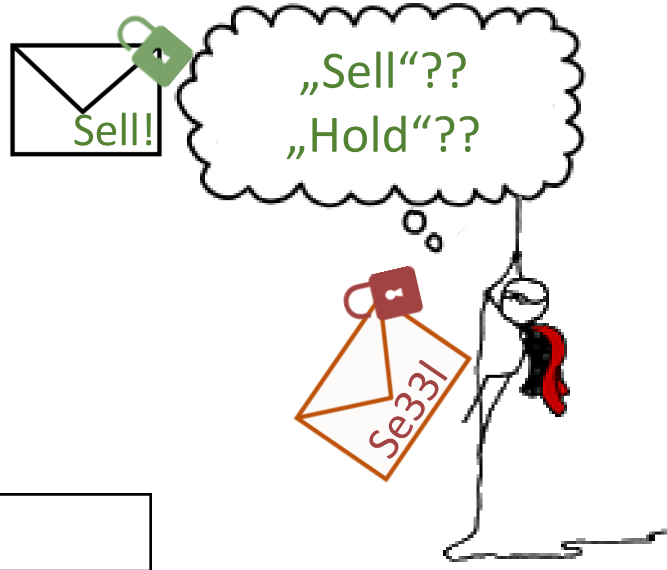
INDistinguishability under **C**hosen-<u>**C**iphertext</u> **A**ttacks.

Left game	**Right game**
Adversary gets public key Adversary chooses two messages m_1 and m_2 Adversary gets encryption of:	
m_1	m_2
Adversary guesses which game it's playing	

Difference to IND-CPA: Adversary can additionally request decryptions for any ciphertext is chooses…

Wait, can't this always be won?

Image source: xkcd.com



IND-CCA security game

INDistinguishability under **C**hosen-**C**iphertext **A**ttacks.

Left game	Right game
Adversary gets public key Adversary chooses two messages m_1 and m_2 Adversary gets encryption of:	
m_1	m_2
Adversary guesses which game it’s playing	

Difference to IND-CPA: Adversary can additionally request decryptions for any ciphertext is chooses… except the provided encryption of m_1/m_2

Image source: xkcd.com

Back to sharing symmetric keys

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

(Why not just use PKE to send encrypted messages? Efficiency.)

Such a method is called a **K**ey **E**ncapsulation **M**echanism (KEM).

→ KEMs are what NIST is looking for!

Image source: xkcd.com

Key Encapsulation Mechanisms (KEMs)

A KEM consists of 3 Algorithms:

1. **KeyGen**: Outputs a public/secret key pair pk, sk (like in public-key encryption)
2. **Encapsulate**(pk): Use pk to create K_{sym} and ciphertext c that ‘encrypts’ K_{sym}
3. **Decapsulate**(sk, c): Use sk to recreate (‘decrypt’) K_{sym} from c

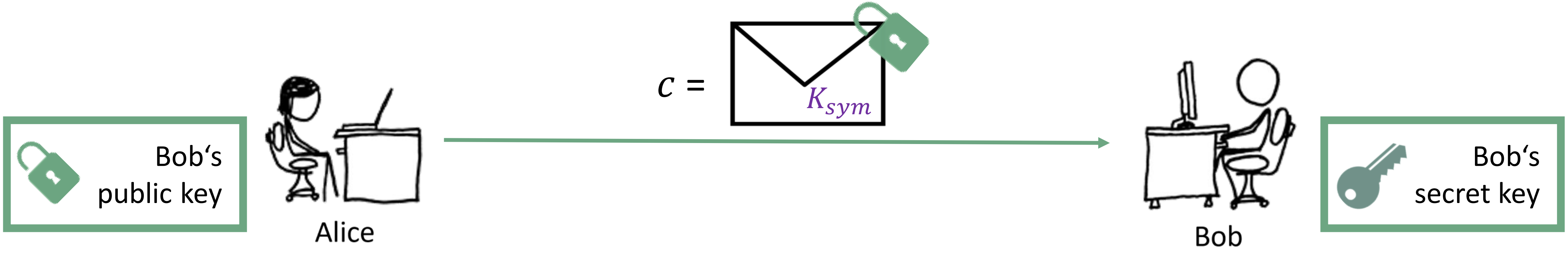
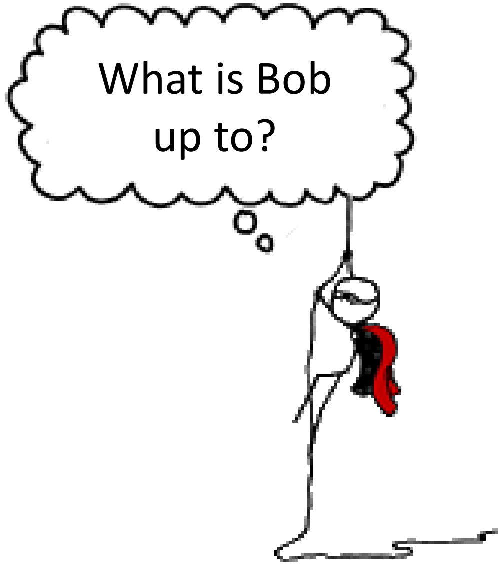


Image source: xkcd.com



KEMs: Security definition

A ciphertext c shouldn‘t leak substantial information about K_{sym}.

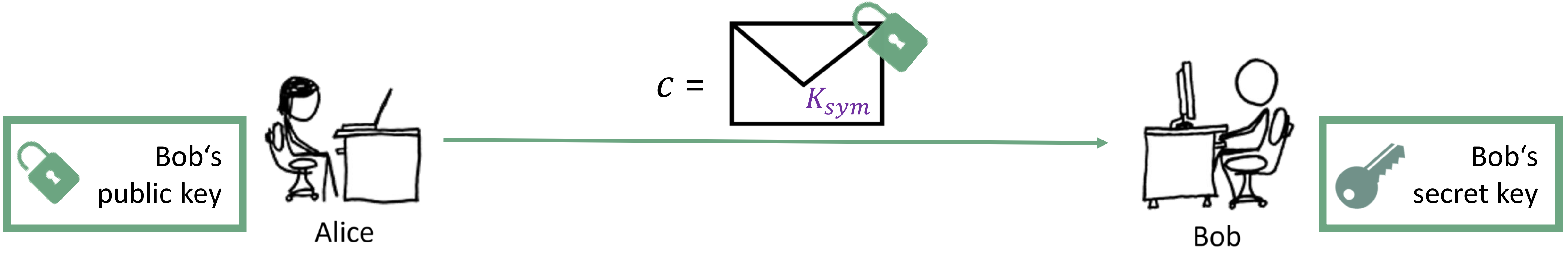
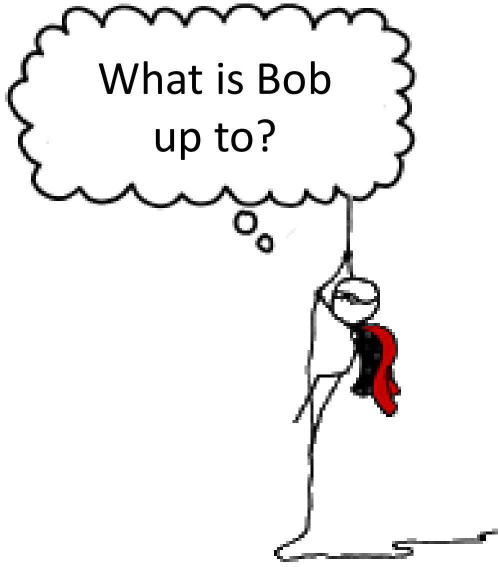


Image source: xkcd.com

Indistinguishability games for KEMs

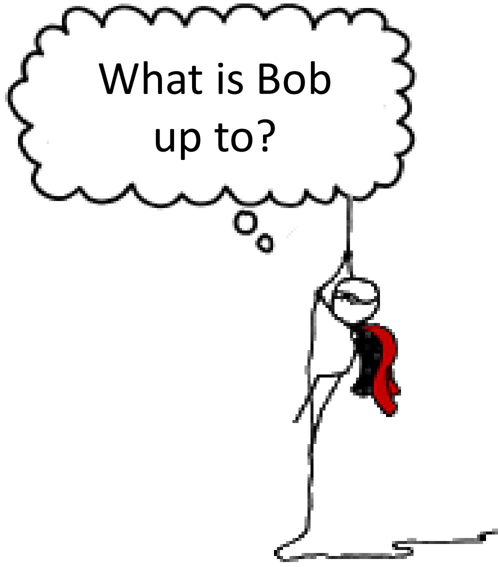


IND-CPA-KEM security: **IND**istinguishability for KEMs.

Left game	**Right game**
Adversary gets public key Adversary gets ciphertext c that 'encrypts' a symmetric key K_{sym}, together with	
The K_{sym} that belongs to c	A uniformly random K_{sym}
Adversary guesses which game it's playing	

Image source: xkcd.com

Indistinguishability games for KEMs



IND-CCA-KEM security: **IND**istinguishability for KEMs under **C**hosen-Ciphertext **A**ttacks.

Left game	**Right game**
Adversary gets public key Adversary gets ciphertext c that 'encrypts' a symmetric key K_{sym}, together with	
The K_{sym} that belongs to c	A uniformly random K_{sym}
Adversary guesses which game it's playing	

Difference to IND-CPA: Adversary can additionally request decryptions for any ciphertext is chooses… except the provided 'challenge' ciphertext c.

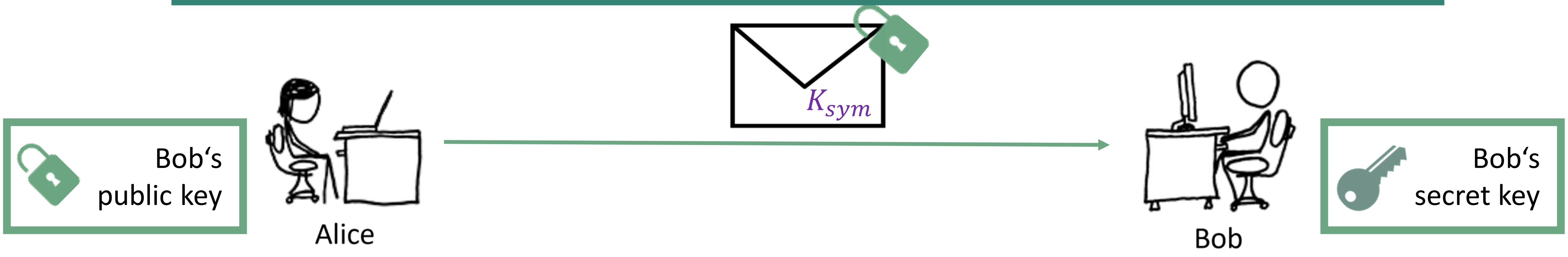
Image source: xkcd.com

KEMs in the NIST standardization process

Shared approach: PKE from hardness assumption + Fujisaki-Okamoto 'recipe'

Fujisaki-Okamoto (FO): 'generic' encryption-to-key-encapsulation recipe

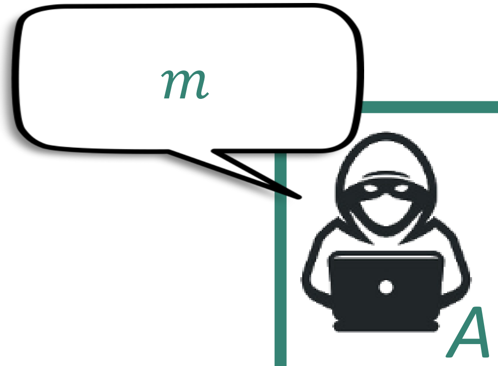
- KYBER = moduleLWE encryption, plugged into FO
- FrodoKEM = LWE encryption, plugged into FO



Fujisaki-Okamoto KEMs: initial idea

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

You may use a public-key encryption scheme that is one-way secure.



Breaking one-wayness:

A must invert 

.

Bob's public key

Alice

Bob

Bob's secret key

Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Fujisaki-Okamoto KEMs: initial idea

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

You may use a public-key encryption scheme that is one-way secure.

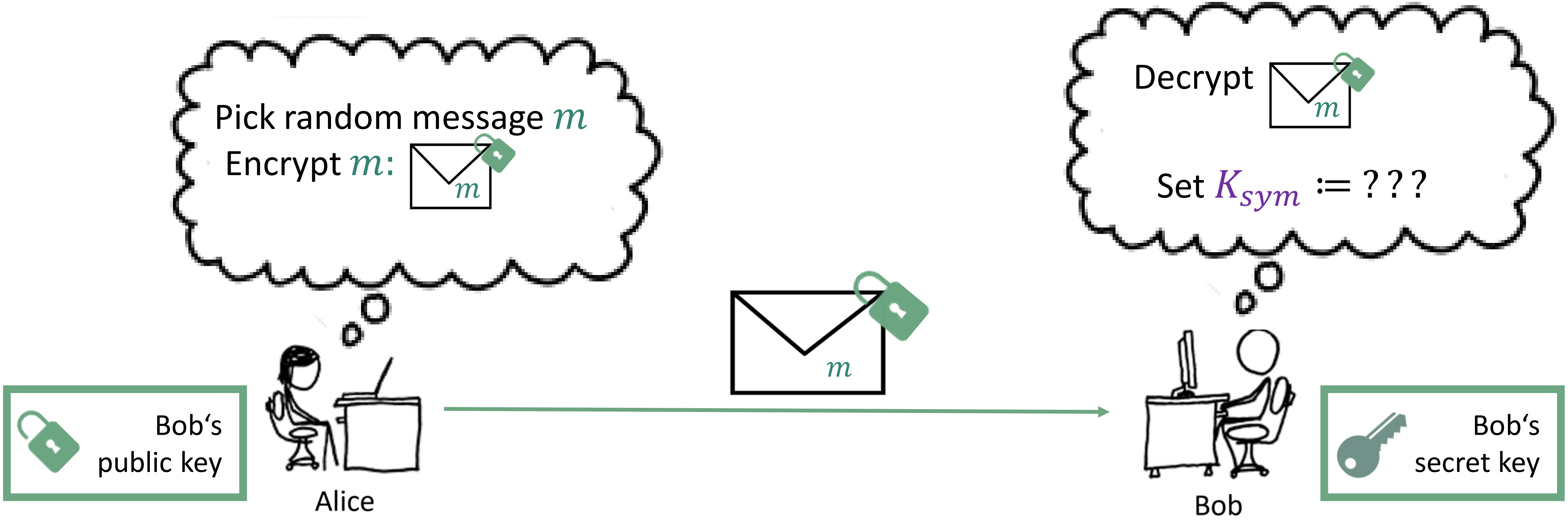


Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Fujisaki-Okamoto KEMs: initial idea

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

You may use a public-key encryption scheme that is one-way secure.

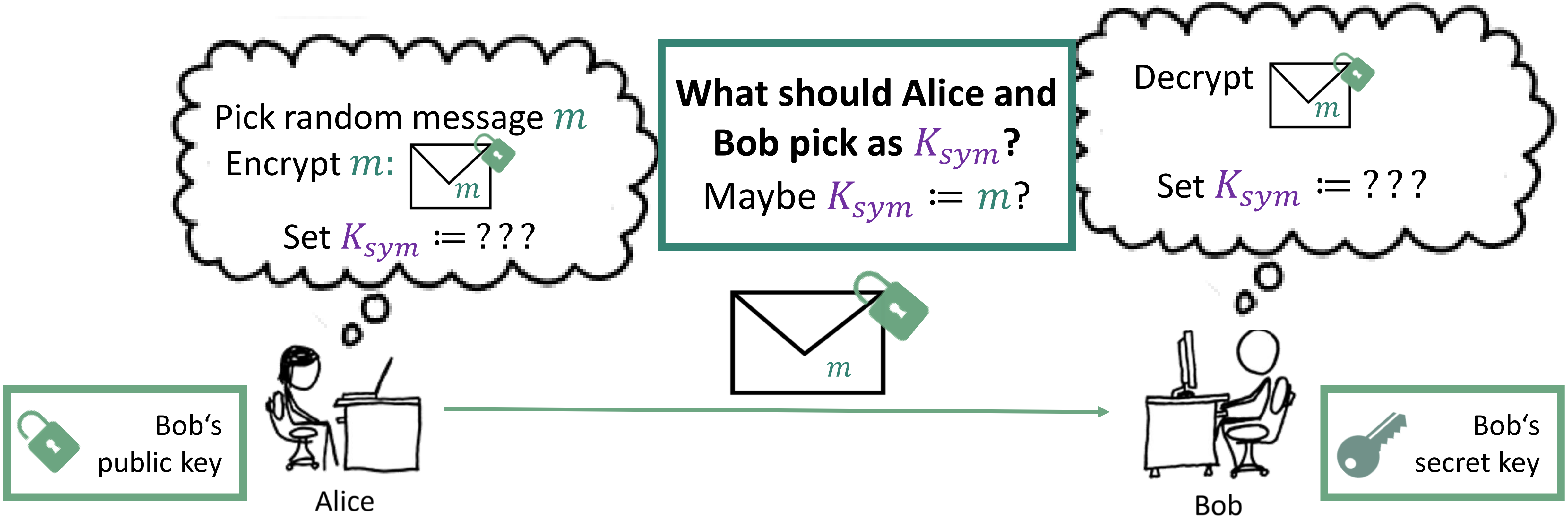


Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Fujisaki-Okamoto KEMs: initial idea

Goal: Find a public-key method to securely establish s… s K_{sym}.

You may use a public-key encryption scheme that is o…

Pick random message m

Encrypt m: m

Set $K_{sym} := ???$

What should Alice and Bob pick as K_{sym}?

Maybe $K_{sym} := m$?

De…

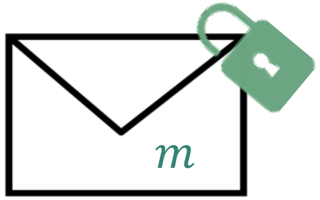
S…

‘real’ / ’random’

Breaking the KEM:

$\boldsymbol{A}$

Seeing an encryption of m, $\boldsymbol{A}$’s task is to tell $K_{sym} = m$ apart from random.



Bob‘s public key

Alice

Bob

Bob‘s secret key

Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Fujisaki-Okamoto KEMs: initial idea

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

You may use a public-key encryption scheme and a hash function.

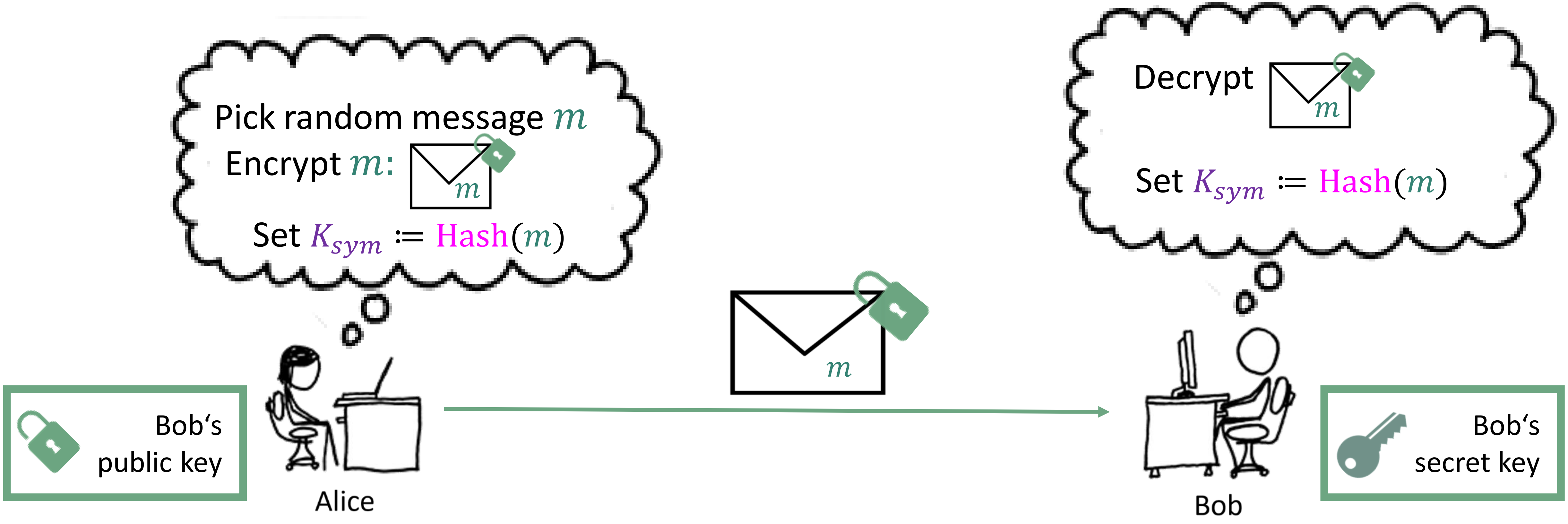


Image source: xkcd.com

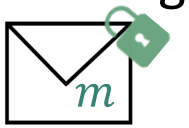
[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Fujisaki-Okamoto KE

Goal: Find a public-key method to

You may use a public-key encrypti

Pick random message m

Encrypt m: 

Set $K_{sym} := \mathrm{Hash}(m)$

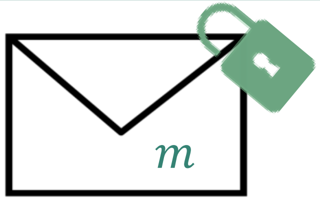
Q: Is this secure?

Breaking the KEM:

Seeing an encryption of m, A's task is to tell $K_{sym} = \mathrm{Hash}(m)$ apart from random.

Goal:

Show that A has 0 chance breaking the KEM without inverting encryption.



Bob's public key

Alice

Bob

Bob's secret key

Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Interlude: the *Provable Security* paradigm

- Cryptographic design
- Security model ('game')
- **Security 'proof'**

'real' /
'random'

Breaking the KEM:

$\boldsymbol{A}$

Seeing an encryption of m, $\boldsymbol{A}$'s task is to tell $K_{sym} =$ Hash(m) apart from random.

Why formal models? To avoid ambiguity.

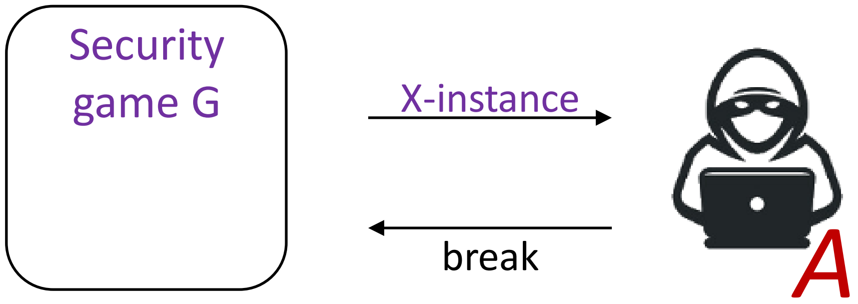
Security ‘proofs’

Intuition: ‘If it’s hard to solve problem P, then design X is secure’

e.g. inverting encryptions e.g. Fujisaki-Okamoto KEM

Proof approach:

- Imagine (black-box) attacker *A*, breaking X according to security game G (e.g., distinguishing KEM output keys from random)



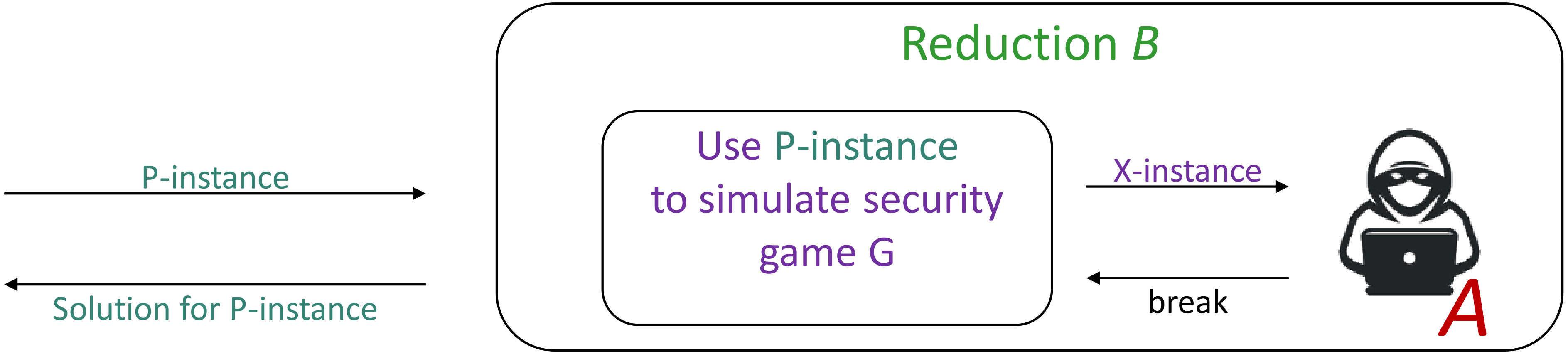
Security ‘proofs’

Intuition: ‘If it’s hard to solve problem P, then design X is secure’

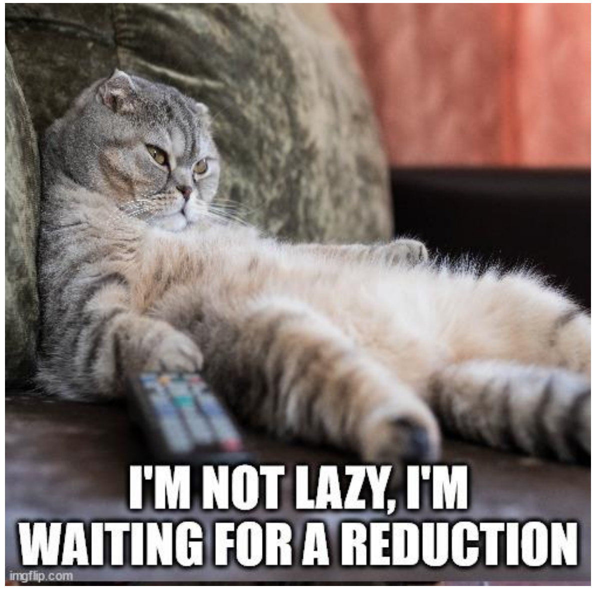
e.g. inverting encryptions e.g. Fujisaki-Okamoto KEM

Proof approach:

- Imagine (black-box) attacker *A*, breaking X according to security game G (e.g., distinguishing KEM output keys from random)
- Construct reduction *B* that uses *A* to solve problem P



Security 'proofs'

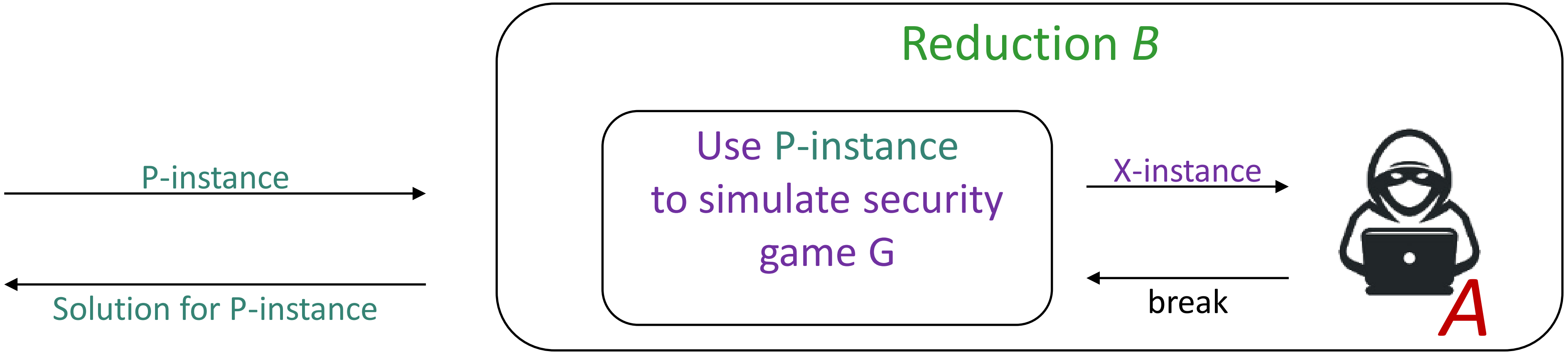


Intuition: 'If it's hard to solve problem P, then design X is secure'

e.g. inverting encryptions — e.g. Fujisaki-Okamoto KEM

Proof approach:

- Imagine (black-box) attacker *A*, breaking X according to security game G (e.g., distinguishing KEM output keys from random)
- Construct reduction *B* that uses *A* to solve problem P

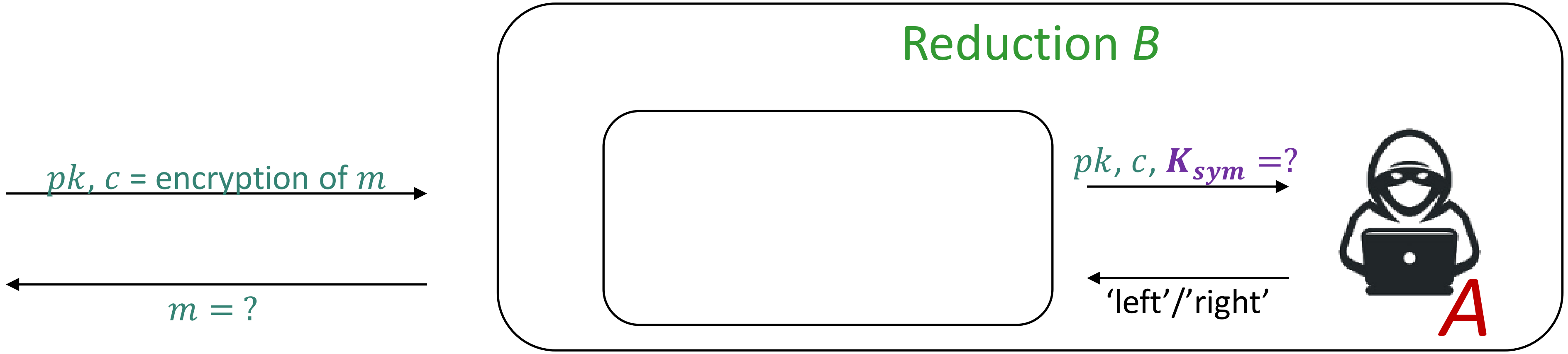


Security ‘proofs’: FO example

Intuition: ‘If it’s hard to invert encryptions, then the Fujisaki-Okamoto KEM is secure’

Proof approach:

- Imagine (black-box) attacker *A*, distinguishing KEM output keys from random
- Construct reduction *B* that uses *A* to invert encryptions



Security ‘proofs’: FO example

Intuition: ‘If it’s hard to invert encryptions, then the Fujisaki-Okamoto KEM is secure’

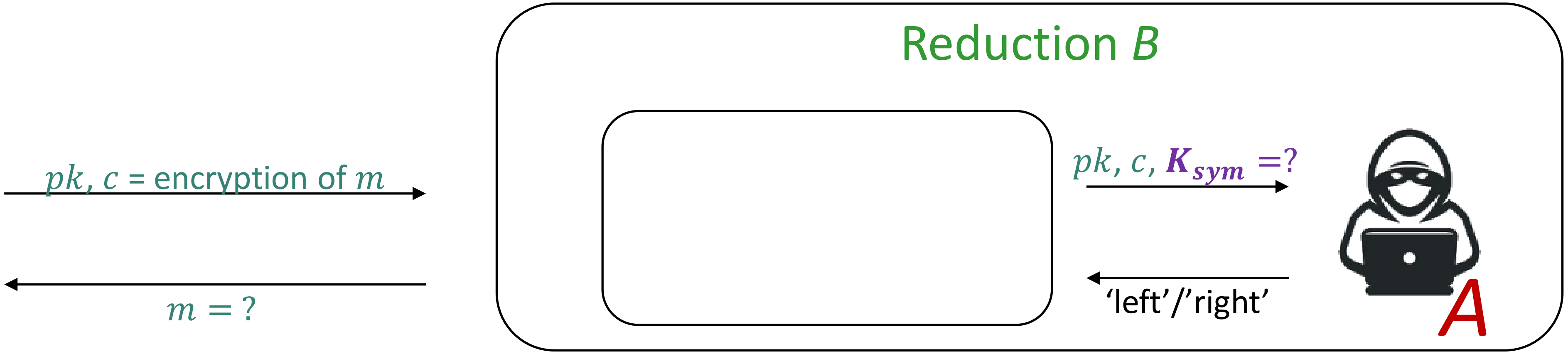
Proof approach:

- Imagine (black-box) attacker A, distinguishing KEM outp
- Construct reduction B that uses A to invert encryptions

Qs:

How can B simulate K_{sym}?

How does B invert c?



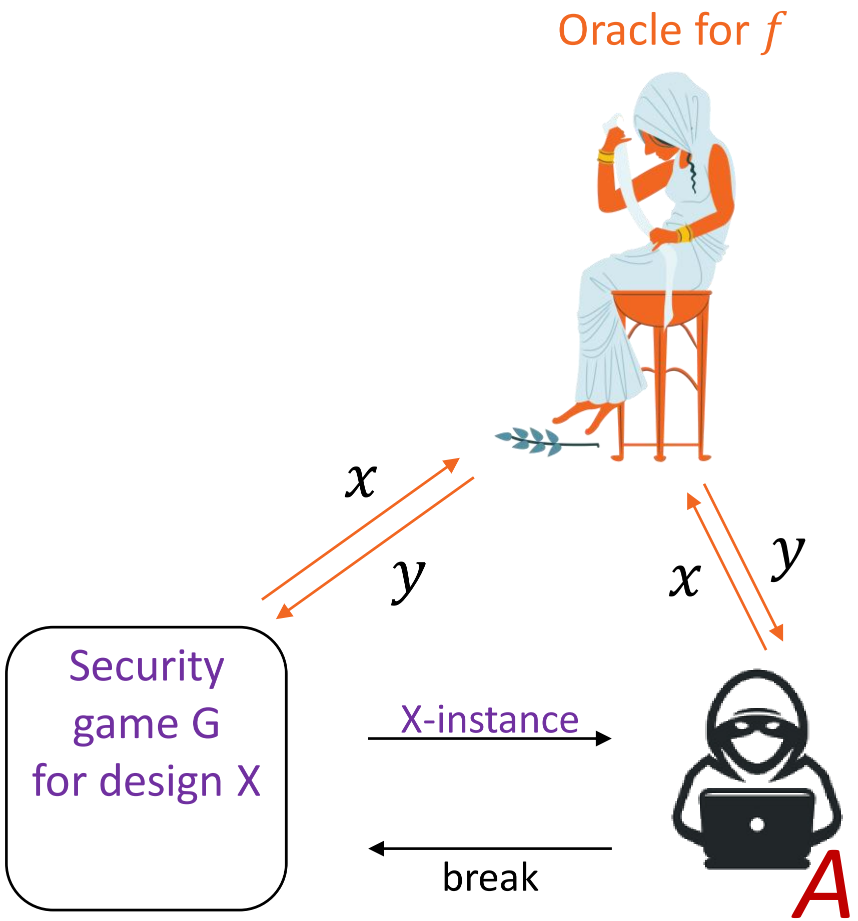
We'll use the random oracle model (ROM)

Heuristic: Replace hash function

$$\text{Hash}: \{0,1\}^n \to \{0,1\}^m$$

with 'oracle box' for truly random

$$f: \{0,1\}^n \to \{0,1\}^m$$



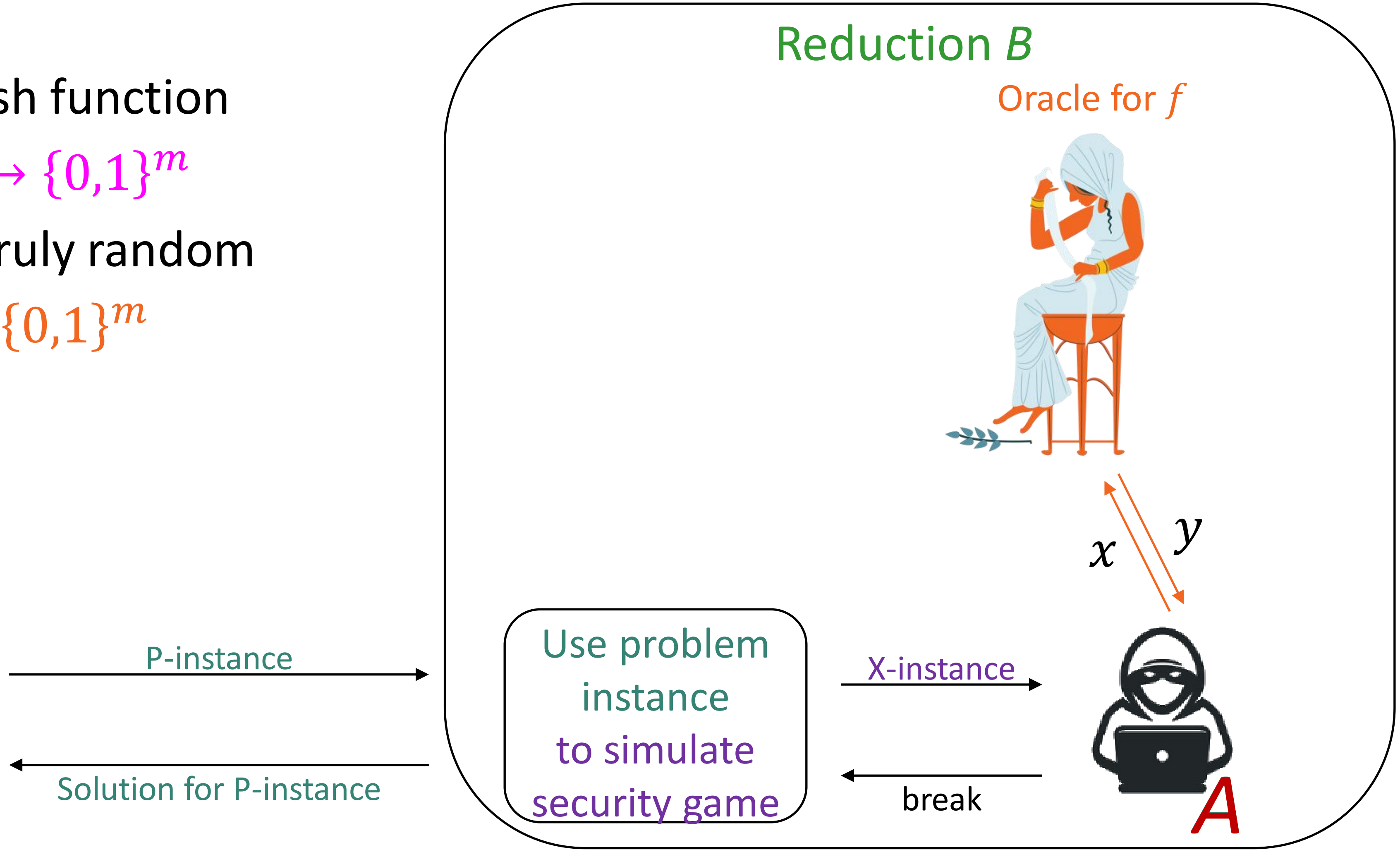
We'll use the random oracle model (ROM)

Heuristic: Replace hash function

$$\text{Hash}: \{0,1\}^n \to \{0,1\}^m$$

with 'oracle box' for truly random

$$f: \{0,1\}^n \to \{0,1\}^m$$

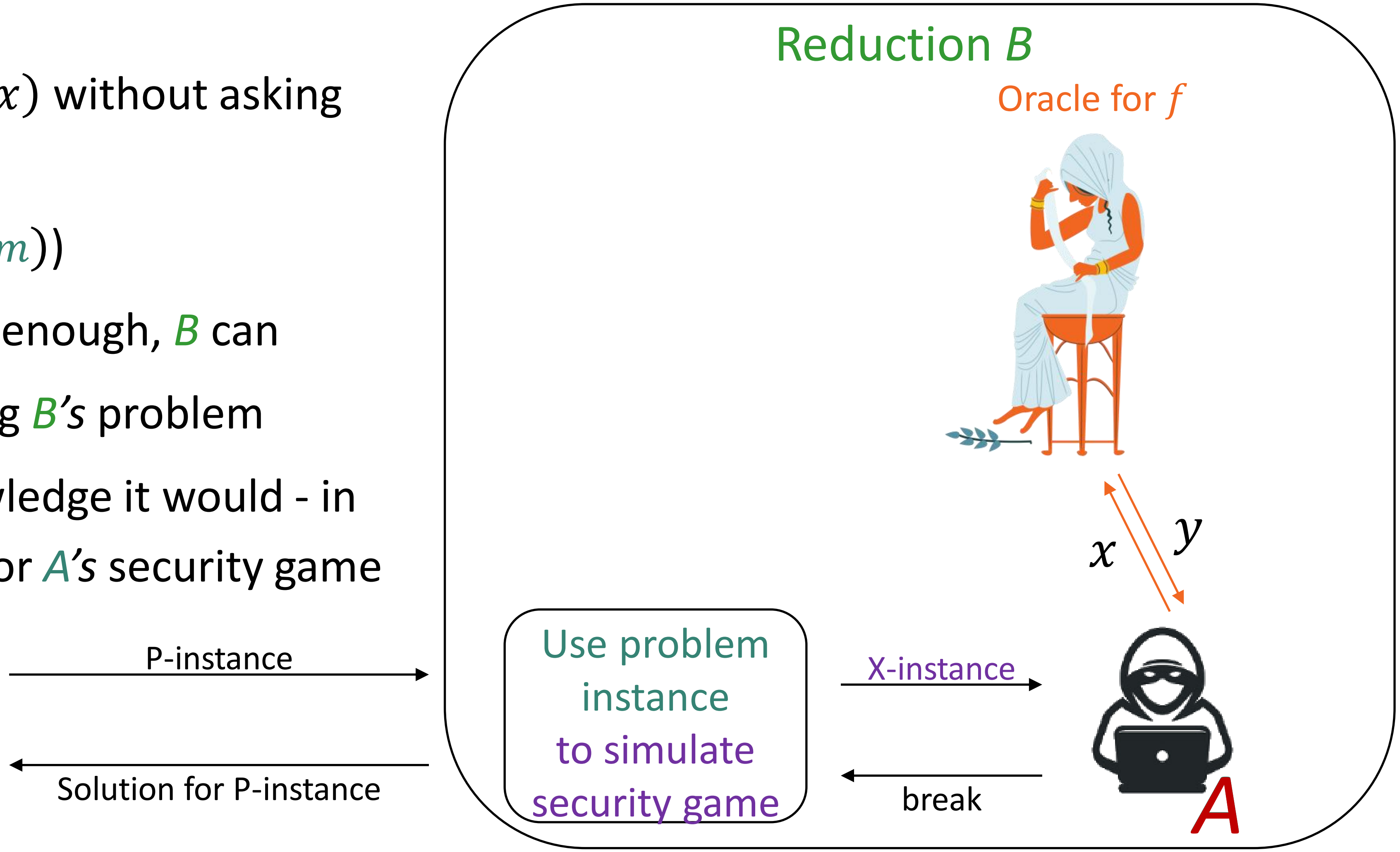


Perks of the random oracle model

- **Unpredictability** of $f(x)$ without asking oracle for $f(x)$

  (e.g., $\boldsymbol{K_{sym}} := f(m)$)

- Picking the ys smartly enough, *B* can
  a) trick *A* into solving *B*'s problem
  b) feign secret knowledge it would - in principle - need for *A*'s security game



Security argument for FO, using the ROM

Intuition: ‘If it’s hard to invert encryptions, then the Fujisaki-Okamoto KEM is secure’

Qs:

How can B simulate K_{sym}?

How does B invert c?

Reduction B

Oracle for f

m

k

pk, c

$m = ?$

Use PKE instance to simulate KEM game

$pk, c, K_{sym} = ?$

break

A

Security argument for FO, using the ROM

Intuition: 'If it's hard to invert encryptions, then the Fujisaki-Okamoto KEM is secure'

Qs:
✓ **How can *B* simulate K_{sym}?**
How does *B* invert c?

- **Unpredictability** → *A* has 0 chance telling $K_{sym} := f(m)$ from $\$$ (random) unless it queries oracle f on the plaintext m that belongs to c

Reduction *B*

Oracle for f

m k

pk, c →

← $m = ?$

Use PKE instance to simulate KEM game

$pk, c, K_{sym} = \$$ →

← break

A

$\boxed{\Pr[A \text{ breaks } K_{sym}] \leq \Pr[A \text{ queries } f \text{ on } m]}$

Security argument for FO, using the ROM

Intuition: 'If it's hard to invert encryptions, then the Fujisaki-Okamoto KEM is secure'

Qs:
✓ **How can B simulate K_{sym}?**
✓ **How does B invert c?**

- **Unpredictability** → A has 0 chance telling $K_{sym} \coloneqq f(m)$ from \$ (random) unless it queries oracle f on the plaintext m that belongs to c
- But then B sees m (it still needs to guess in which of the q many RO queries though)

pk, c

m

Reduction B

Oracle for f

m k

Use PKE instance to simulate KEM game

$pk, c, K_{sym} = \$$

break

A

$$\Pr[A \text{ breaks } K_{sym}] \le \Pr[A \text{ queries } f \text{ on } m]$$
$$\le q \cdot \Pr[B \text{ can invert } c]$$

The ROM heuristic seems weird.

☹ No theoretical justification

Counterexamples: (convoluted) designs that are

- secure in the ROM, but
- insecure when instantiating RO with any hash function

☺ Good track record for ‘natural’ schemes

Helps identify design bugs

Attacks on ‘ROM-secure’ schemes would be kind of surprising

Recap: initial idea

Goal: Find a public-key method to securely establish symmetric keys K_{sym}.

You may use a public-key encryption scheme and a hash function.

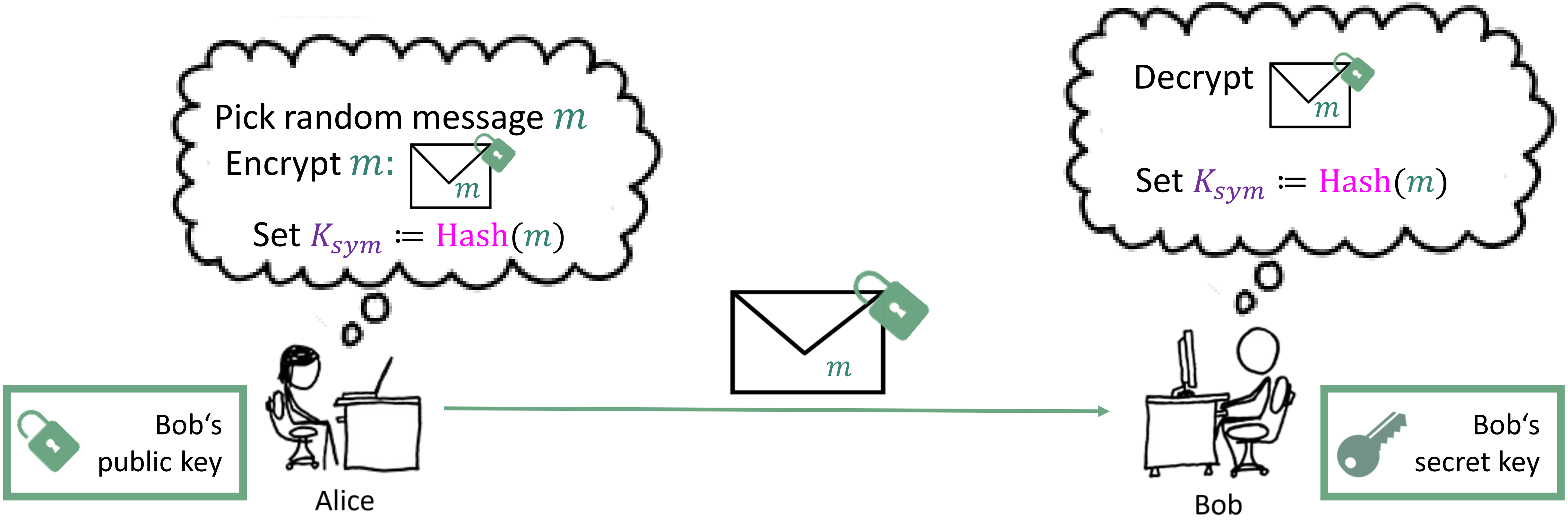


Image source: xkcd.com

[Hofheinz Hövelmanns Kiltz 17]: A modular analysis of the Fujisaki-Okamoto Transform.

Security against chosen-ciphertext attacks

Goal: Find a way to establish symmetric keys K_{sym} **with chosen-ciphertext security.**

→ attacker allowed to **request decapsulation for any ciphertext.**

High-level idea: alter how the KEM en-/decapsulates:

Altered decapsulation will

- detect dishonest ciphertexts
- punish those by rejecting to return a meaningful key.

→ hard for attacker to request useful decapsulations

‘Full’ FO

Goal: Make decryptions useless for *A*!

Pick random message m

Encrypt m, **deterministically**

Set $K_{sym} := \mathrm{Hash}(m)$

Instead of randomly sampling encryption randomness r:

Use $r = \mathrm{Hash}'(m)$

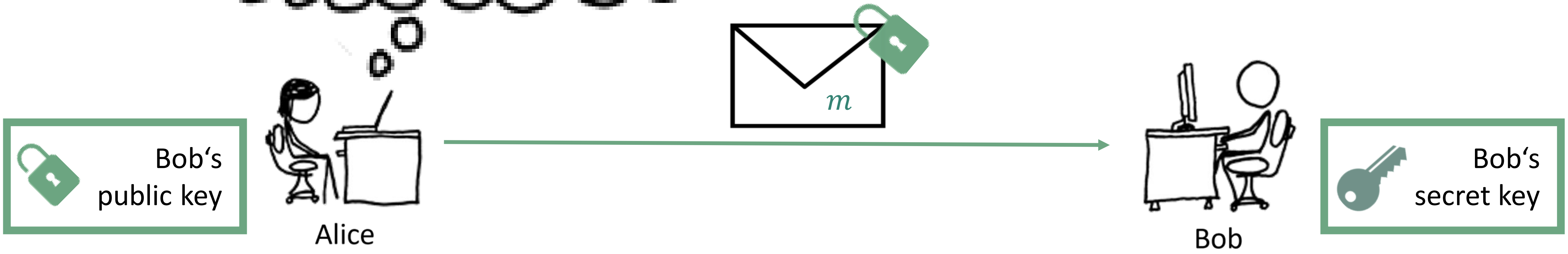


Image source: xkcd.com

Using Hash$'(m)$ as randomness

'Full' FO

Goal: Make decryptions useless for A!

Pick random message m

Encrypt m, **deterministically**

Set $K_{sym} \coloneqq \mathrm{Hash}(m)$

Decrypt m

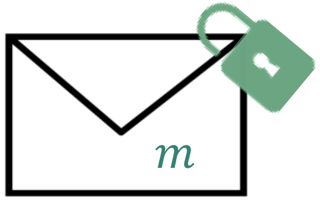
Only if m encrypts to m :

Set $K_{sym} \coloneqq \mathrm{Hash}(m)$

Otherwise, reject!

Bob's public key

Alice



Bob

Bob's secret key

Image source: xkcd.com

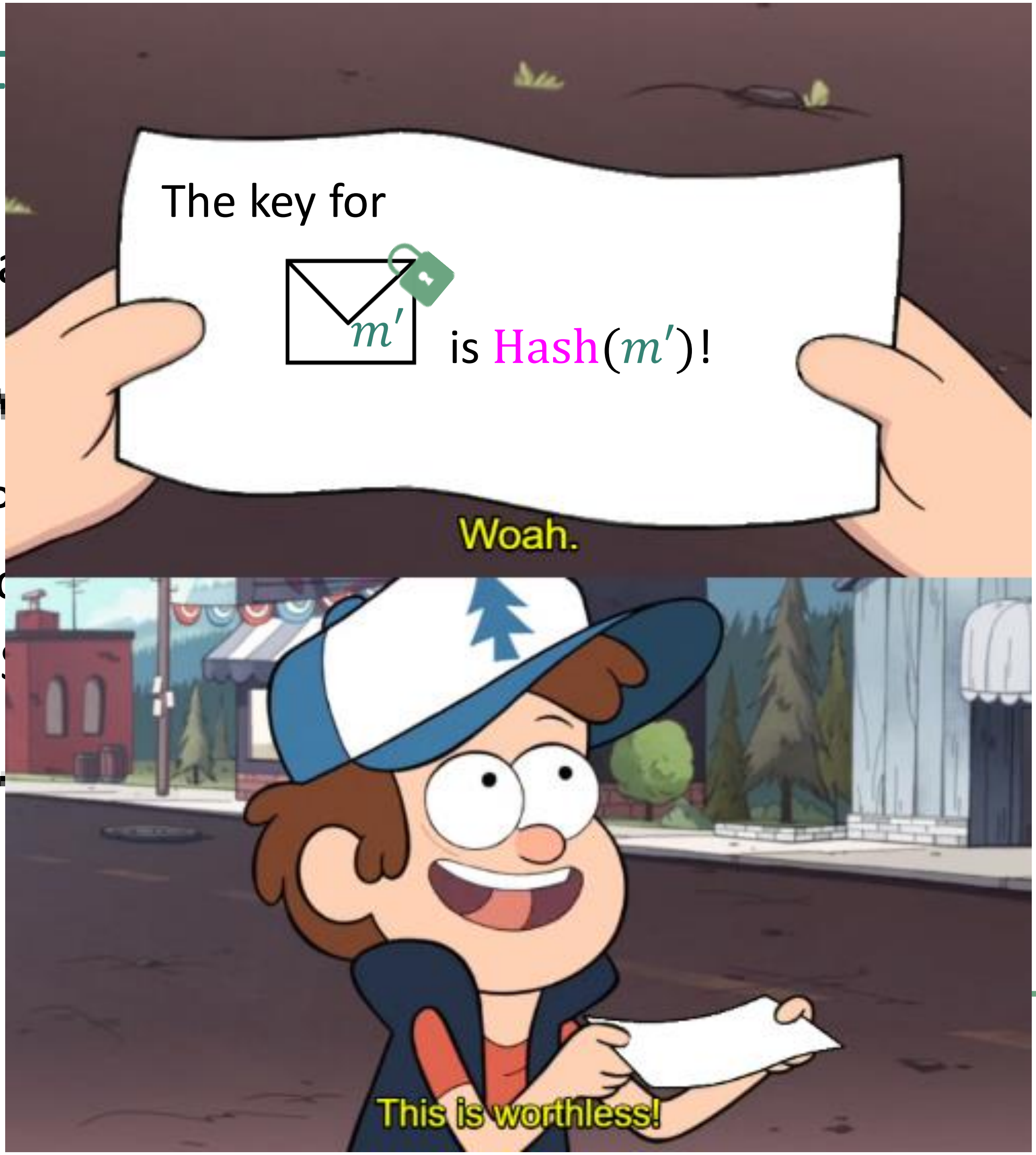
‘Full’ F

Using $\mathrm{Hash}'(m)$ as randomness

Goal: Ma

P

Enc



Decrypt m

Only if m encrypts to m:

Set $K_{sym} := \mathrm{Hash}(m)$

Otherwise, reject!

Bob‘s public key

Bob

Bob‘s secret key

Image source: xkcd.com

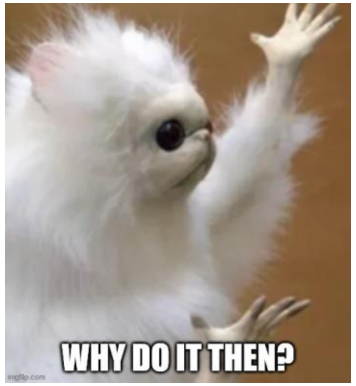
‘Full’ FO

Goal: Make decryptions useless for A!

Pick random message m

This creates side-channel vulnerabilities.

(smarter ‘works-always’ solutions are elusive so far ☹)

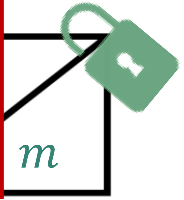


Decrypt m

Only if m encrypts to m :

Set $K_{sym} := \mathrm{Hash}(m)$

Otherwise, reject!



m

Bob‘s public key

Alice

Bob

Bob‘s secret key

Image source: xkcd.com

‘Full’ FO

Goal: Make decryptions useless for A!

Pick random message m

Not as critical, but still subject to debate:

Which rejection method is better?

- returning explicit failure symbol $\perp$?
- returning pseudorandom key?

Decrypt m

Only if m encrypts to m :

Set $K_{sym} := \mathrm{Hash}(m)$

Otherwise, reject!

m

Bob‘s public key

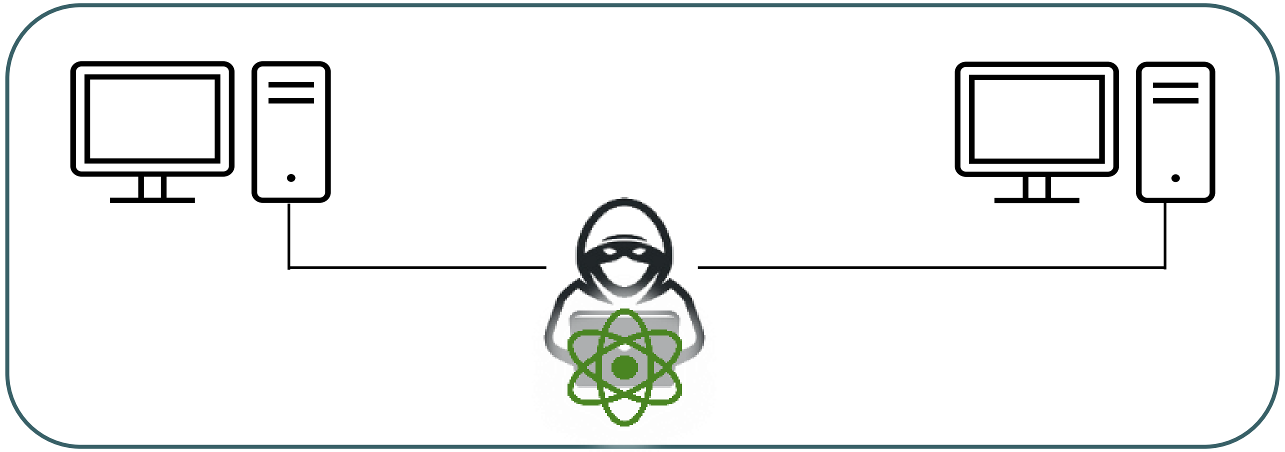
Alice

Bob

Bob‘s secret key

Image source: xkcd.com

Adapting security proofs to quantum attackers

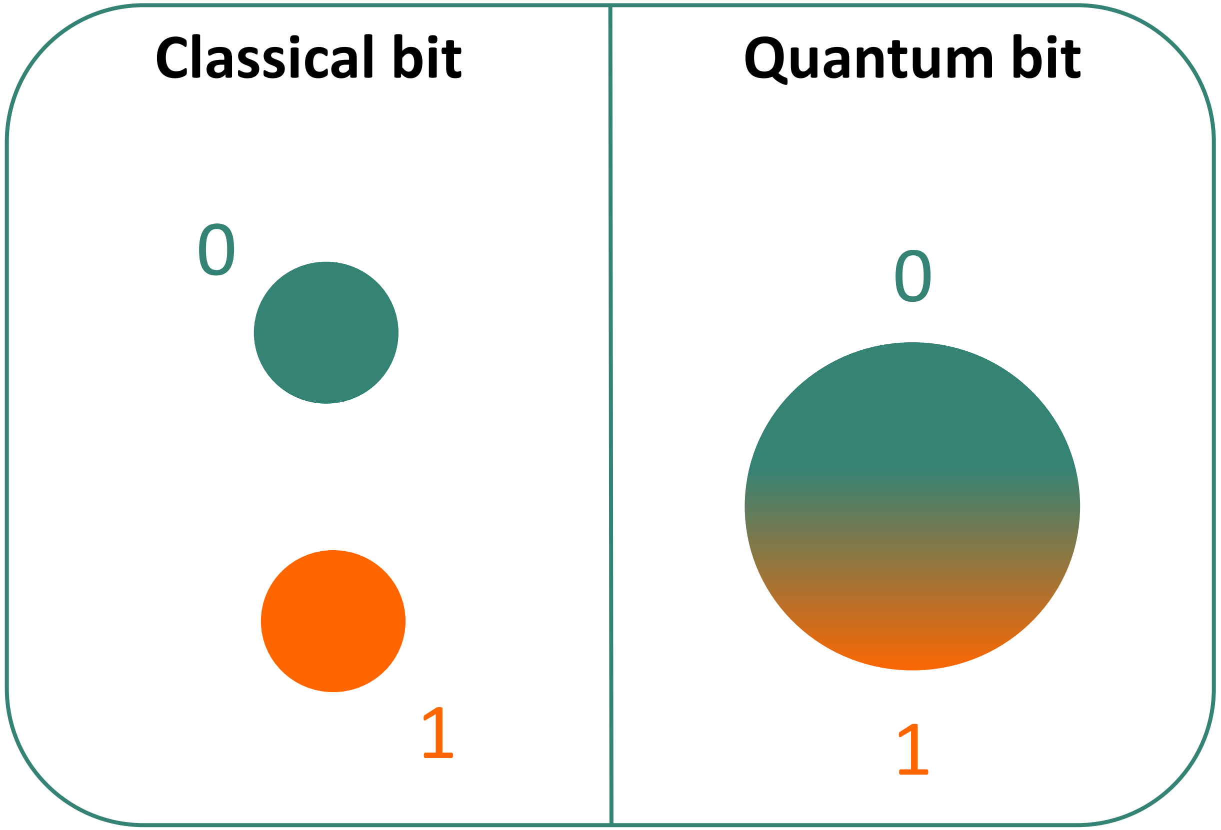


- ‘Online’ functionality (decryption, signing, ...) stays classical
- ‘Offline’ functionality computable by quantum attacker

Random oracle model: Hash functions can be computed offline

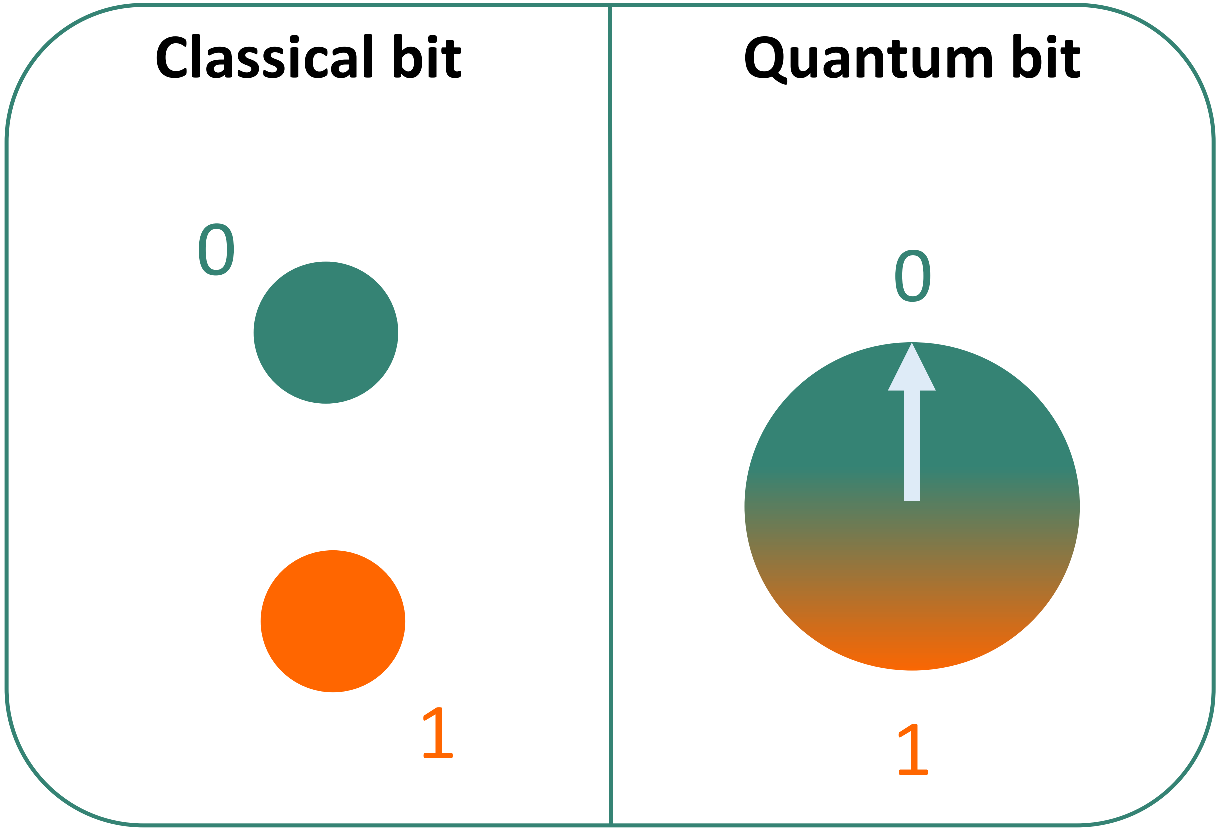
→ **Quantum access to random oracles**!

Quantum bits



Quantum bits

Notation:
- $|0\rangle$ for ‚truly 0‘

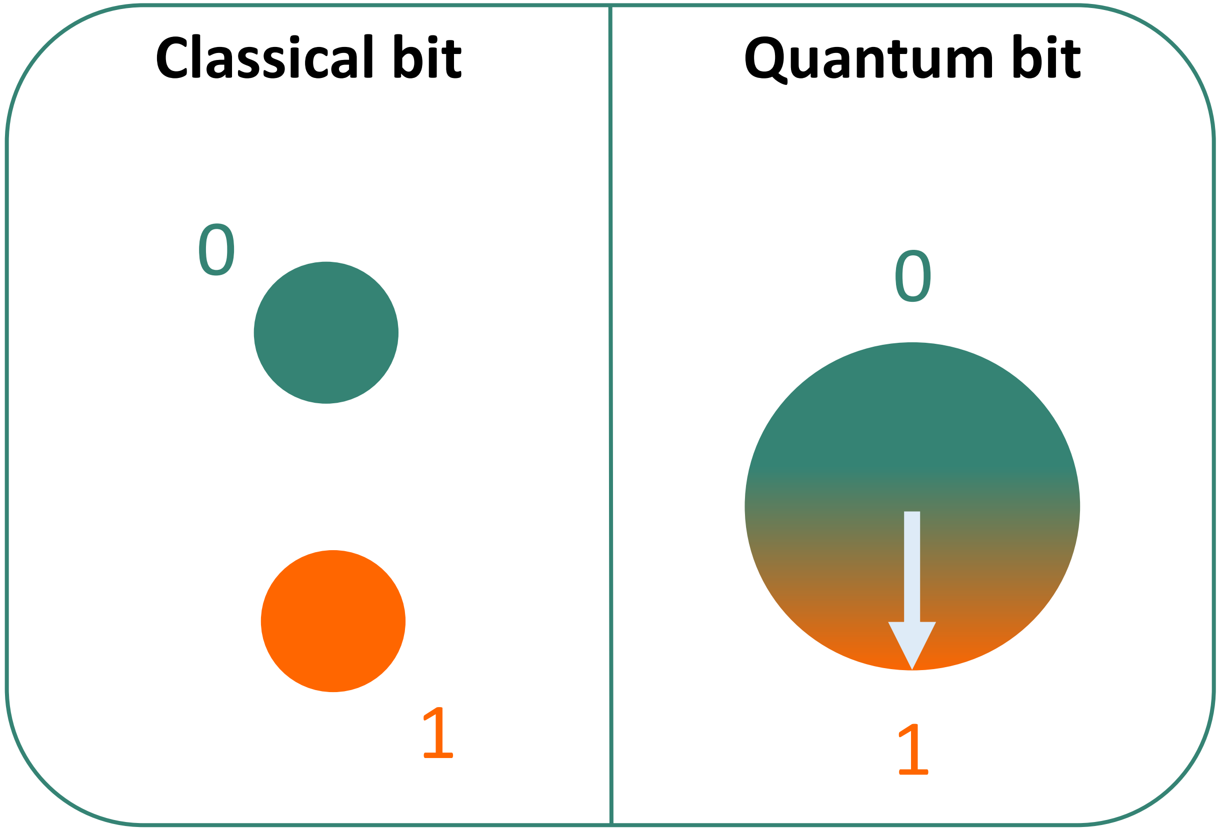


Quantum bits

Notation:

- $|0\rangle$ for ‚truly 0‘
- $|1\rangle$ for ‚truly 1‘

} **‘base states’**



Quantum bits

Notation:

- $|0\rangle$ for ‚truly 0‘
- $|1\rangle$ for ‚truly 1‘

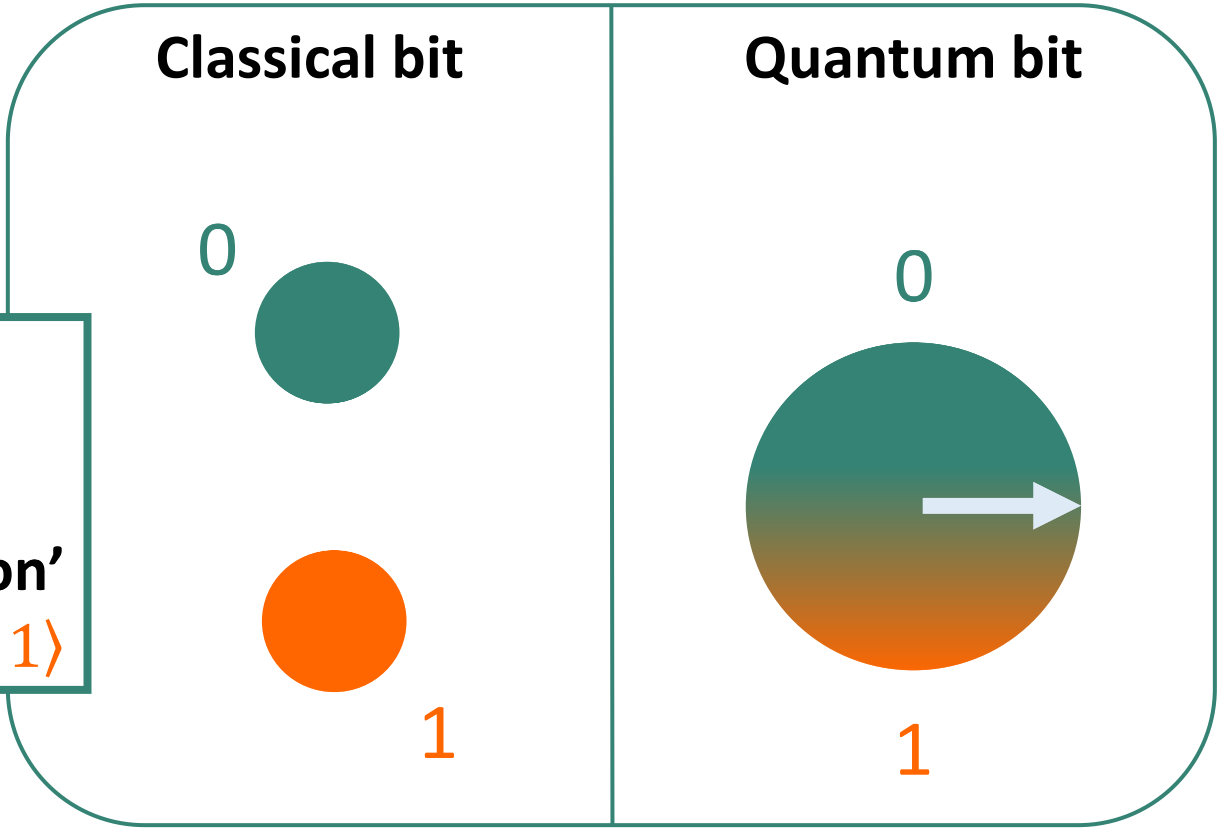
} **‘base states’**

‘Non- base states’:

- $\alpha_0|0\rangle + \alpha_1|1\rangle$ for complex numbers α_0, α_1
- Requirement on (α_0, α_1):

$$||\alpha_0||^2 + ||\alpha_1||^2 = 1$$

‘Superposition’ of $|0\rangle$ and $|1\rangle$

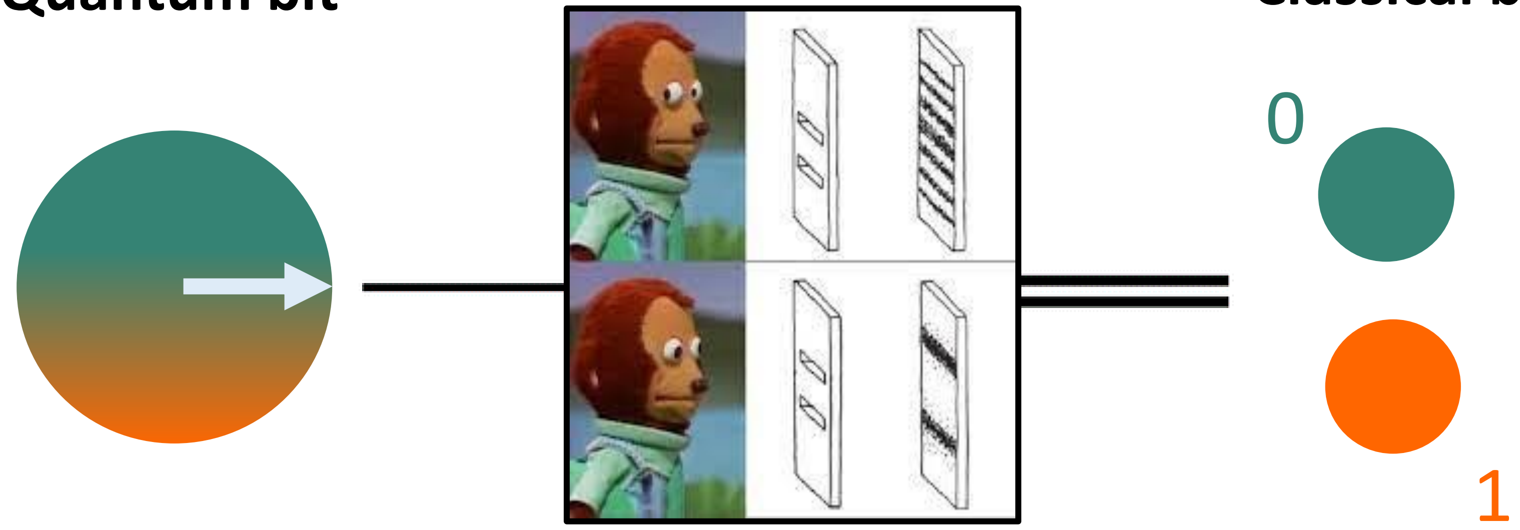


Example: uniform (‚half-half‘) state

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

Measuring quantum bits

Quantum bit **Classical bit**



What happens?

$\alpha_0|0\rangle + \alpha_1|1\rangle$ 'collapses' to $\begin{cases} 0 & \text{with probability } ||\alpha_0||^2 \\ 1 & \text{with probability } ||\alpha_1||^2 \end{cases}$

Quantum bitstrings (qubit strings)

Same principle: Put all possible bitstrings of length ℓ into superposition

E.g., for length 2:

- qubit strings are of the form $\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \ \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$
- similar **requirement on 'probability coefficients'** $\alpha_{00}, \cdots, \alpha_{11}$:

$$||\alpha_{00}||^2 + ||\alpha_{01}||^2 + ||\alpha_{10}||^2 + \ ||\alpha_{11}||^2 \ = 1$$

Measuring:

$\alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \ \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$ collapses to '00' with prob. $||\alpha_{00}||^2$etc.

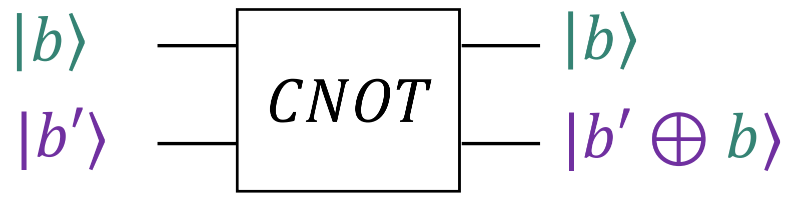
Computing on quantum states

Fact: Any quantum computation can be described by a ‘nicely-invertible’ map U.

Example: a map for strings of length 2

$$\left.\begin{array}{l} |00\rangle \to |00\rangle \\ |01\rangle \to |01\rangle \\ |10\rangle \to |11\rangle \\ |11\rangle \to |10\rangle \end{array}\right\} |b, b'\rangle \to |b, b' \oplus b\rangle$$

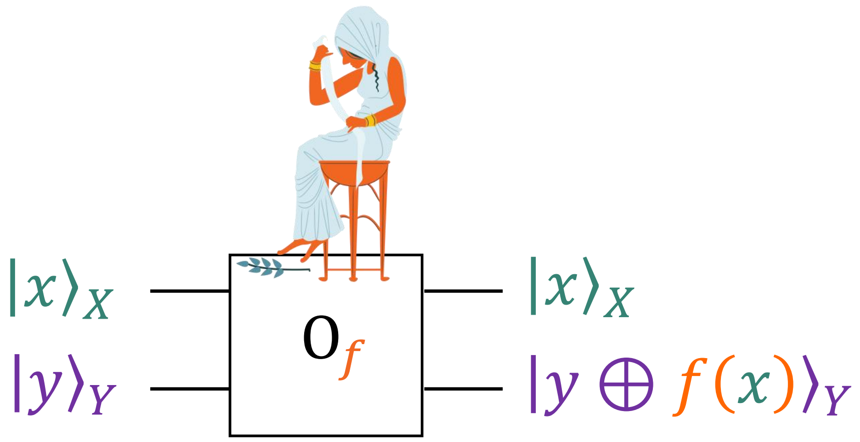
Gate description:



Random oracles: How to describe them in a ‘nicely-invertible’ way?

Quantum-accessible random oracles (QROs)

Model the QRO as oracle box O_f for random function $f : X \to Y$ as follows:



So for any classical input value x,

$$|x\rangle|0 \cdots 0\rangle \to |x\rangle|f(x)\rangle.$$

(O_f simply carries over the probability coefficients)

[Boneh Dagdelen Fischlin Lehmann Schaffner Zhandry 11]

What about our Random Oracle proof?

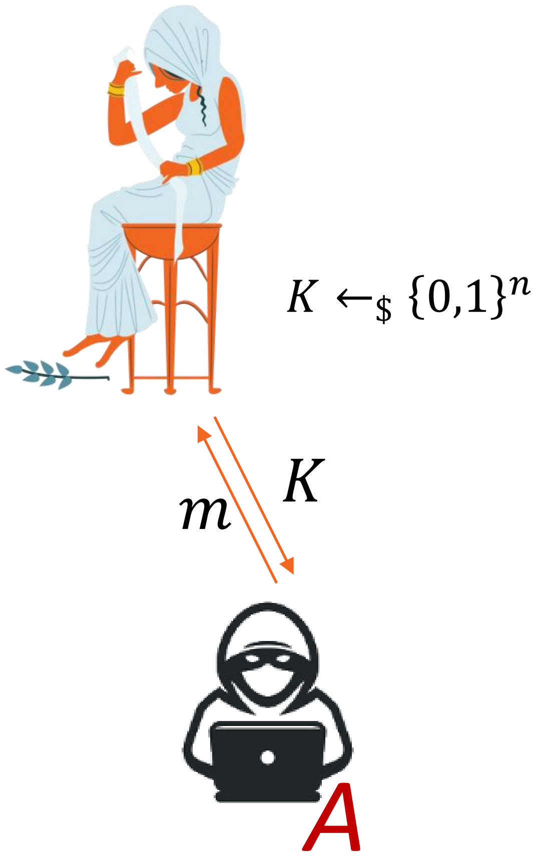
- ‘See how *A* ticks’?

  e.g., seeing plaintext m belonging to  in *A*’s queries

m now ’hides’ in superpositions (linear combinations)

$$\alpha_m |m\rangle |y_m\rangle + \alpha_{not\ m} |not\ m\rangle |y_{not\ m}\rangle$$

How to extract m from the queries? By measuring them?



(classical) RO

What about our Random Oracle proof?

- ‘See how *A* ticks’?

  e.g., seeing plaintext m belonging to 

 in *A*’s queries

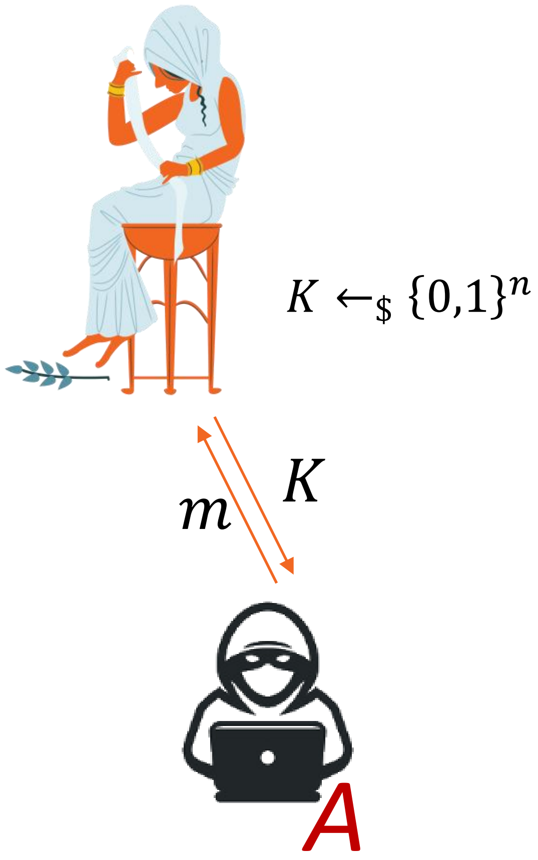
m now ’hides’ in superpositions (linear combinations)

$$\alpha_m |m\rangle |y_m\rangle + \alpha_{not\ m} |not\ m\rangle |y_{not\ m}\rangle$$

How to extract m from the queries? By measuring them?

Wouldn‘t that change (‘collapse’) them and thereby *A*’s behavior?

Can we still extract interesting queries, without derailing *A* too much?



(classical) RO

What about our Random Oracle proof?

- ‘See how *A* ticks’?

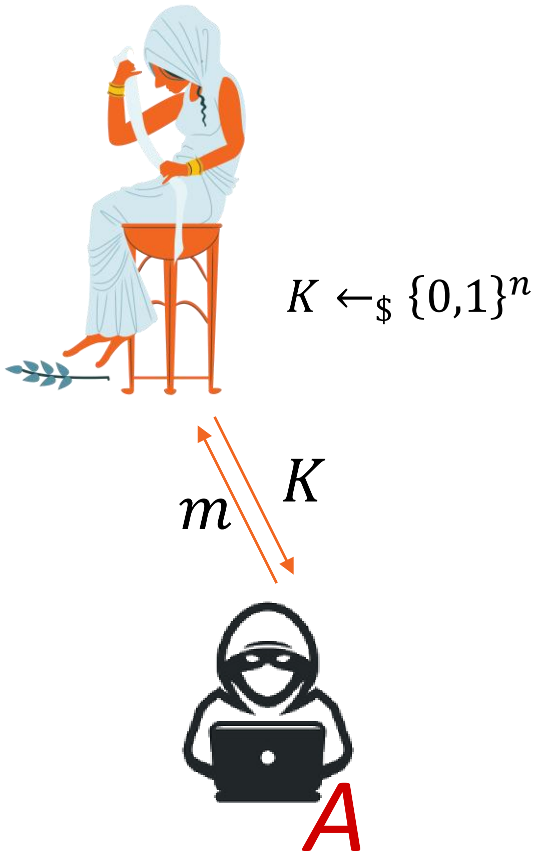
  e.g., seeing plaintext m belonging to  in *A*’s queries

‘Random-until-queried’ formalised via quantum query extractor

[Unruh 14 + follow-ups]

Caveat: loss in security parameters (minimal loss still tbd)

→ proofs so far only **apply to less efficient schemes**



(classical) RO

[Unruh 14]: Revocable quantum timed-release encryption.

[Ambainis Hamburg Unruh 18]: Quantum security proofs using semi-classical oracles.

[Bindel Hamburg Hövelmanns Hülsing Persichetti 19]: Tighter proofs of CCA security in the QROM.

[Kuchta Sakzad Stehlé Steinfeld Sun 20]: Measure-rewind-measure: Tighter QROM proofs for one-way to hiding and CCA security.

CCA means dealing with decryption failures

Many post-quantum (e.g. LWE-based) schemes occasionally exhibit decryption errors:

$$Decrypt\big(Encrypt(m)\big) \neq m$$

Failure secret-key-dependent

→ leakage on secret key [D’Anvers 18 + follow-ups]

Original solution ([HHK17]): Assume worst-case bound ε on failure probability

→ hard for attacker to find failing ciphertexts in the first place.

[D’Anvers Vercauteren Verbauwhede 18]: On the impact of decryption failures on the security of LWE/LWR based schemes

[Bindel Schanck 20]: Decryption failure is more likely after success

[D’Anvers Rossi Virdia 20]: (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes

CCA means dealing with decryption failures

Many post-quantum (e.g. LWE-based) schemes occasionally exhibit decryption errors:

$$Decrypt(Encrypt(m)) \neq m$$

Failure secret-key-dependent

→ leakage on secret key [D'Anvers 18 + follow-ups]

Original solution ([HHK17]): Assume worst-case bound ε on failure probability

→ hard for attacker to find failing ciphertexts in the first place.

Irritating facts:	**Applicability issue:**
HHK17 gives Grover-like search term: $q^2 \cdot \varepsilon$	Concrete ε – estimations ⚡ security proofs
ε is a somewhat 'unnatural' bound	

ε - estimations vs security proofs

$\varepsilon \triangleq$ success probability in

Necessary?

Correctness game

$c \leftarrow \mathsf{Enc}(pk, m)$
return $[\![\mathsf{Dec}(sk, c) = m]\!]$

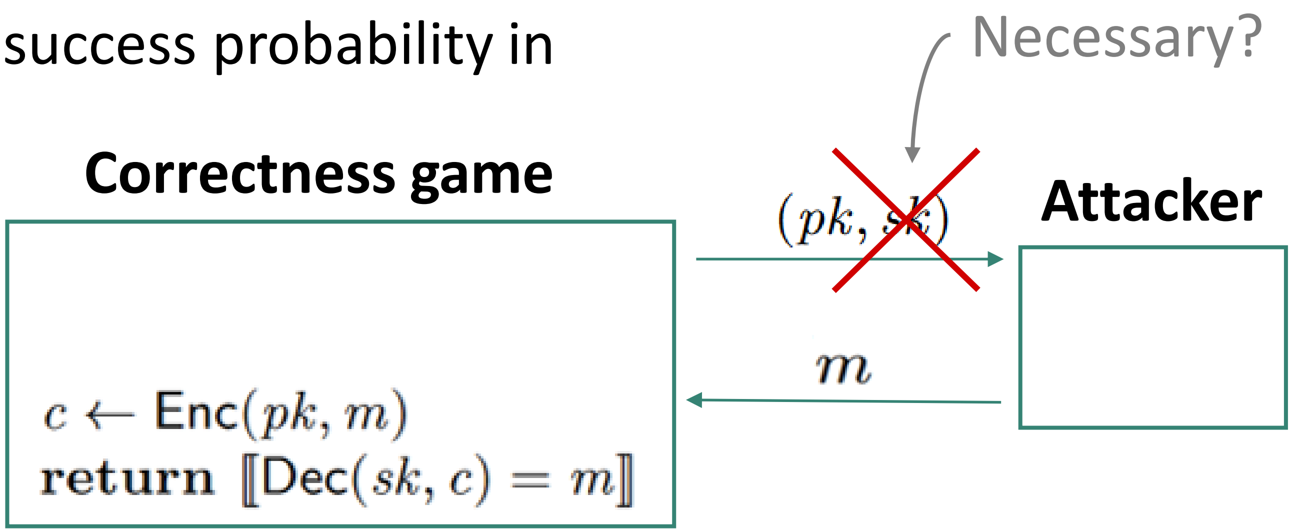
(pk, sk)

m

Attacker

ε - estimations vs security proofs

$\varepsilon \triangleq$ success probability in



ε-estimator scripts:

estimate $\triangleq$ success probability in game **without sk**

observed by Manuel Barbosa while formally verifying Kyber

Applicability issue

Concrete ε – estimations security proofs

Improving the treatment of decryption failures

[HHM 22]: Assume more natural bound (**sk-less failure finding** → **estimator-script-compatible** ☺)

How?

- Classical ROM:
  1. helpful decryption query = adversary found failing plaintext (**without** knowing sk)
  2. analyse failure finding in more fine-grained way
- Quantum:
  1. more sophisticated ('extractable') QROM [DFMS21] allows 'almost-classical' reasoning for 1.
  2. search bounds for 2.
  3. prove 'random-until-queried' argument for extractable QROM

Additional advantage: proof technique agnostic to rejection type

→ **Aligns the two (previously unaligned) rejection methods in terms of QROM bounds**

[Hövelmanns Hülsing Majenz 22]: Failing Gracefully: Decryption Failures and the Fujisaki-Okamoto Transform

[Don Fehr Majenz Schaffner 21]: Online-extractability in the quantum random-oracle model.

Improving the treatment of decryption failures

[HHM 22]: Assume more natural bound (**sk-less failure finding** → **estimator-script-compatible** ☺)

How?

- Classical ROM:
  1. helpful decryption query = adversary found failing plaintext (**without** knowing sk)
  2. analyse failure finding in more fine-grained way
- Quantum:
  1. more sophisticated (‘extractable’) QROM [DFMS21] allows ‘almost-classical’ reasoning for 1.
  2. search bounds for 2.
  3. prove ‘random-until-queried’ argument for extractable QROM

Additional advantage: proof technique agnostic to rejection type

→ **Aligns the two (previously unaligned) rejection methods in terms of QROM bounds**

(maybe) disadvantage: new analysis tasks, designers might be fine with ε - heuristic.

Improving the treatment of decryption failures

[HHM 22]: Assume more natural bound (**sk-less failure finding** → **estimator-script-compatible** ☺)

Advantage: proof technique agnostic to rejection type

→ **Aligns the two (previously unaligned) rejection methods in terms of QROM bounds**

(maybe) disadvantage: new analysis tasks, designers might be fine with ε - heuristic.

[HM 23]: reconcile 'rejection method alignment' with ε – heuristic

[Hövelmanns Majenz 23]: A note on failing gracefully: Completing the picture for explicitly rejecting FO transforms using worst-case correctness

Cheaper security for NTRU-based schemes

It’s not good if attackers can easily trigger decryption failures.

(Efficient) NTRU-based schemes: failures not generally independent of plaintext at hand → leverage for the attacker!

High-level idea: Pre-transformations that

- detach decryption failure likelihood from the concrete plaintext (‘average-case- to worst-case-correctness’);
- without giving up efficiency.

→ hard for attacker to trigger decryption failures
→ more efficient NTRU-based designs.

[Duman Hövelmanns Kiltz Lyubashevsky Seiler Unruh 21]: A Thorough Treatment of Highly-Efficient NTRU Instantiations

Security against multi-user attacks

Limitation so far: in practice, many users will use this KEM

→ we want to ensure that collected info on Bob does not help with attacking Carol

High-level idea: Use domain separation to bind Bob's identity (a prefix $pref$ of the public key pk) to

- how we define validity of a ciphertext:

  use $\mathrm{Hash}'(\boldsymbol{pref}, m)$ as encryption randomness

- how the symmetric key is computed:

$$K_{sym} := \mathrm{Hash}(\boldsymbol{pref}, m)$$

→ hard for attacker to exploit information related to Bob to attack Carol.

[Duman Hövelmanns Kiltz Lyubashevsky Seiler 21]: Faster lattice-based KEMs via a generic Fujisaki-Okamoto transform using prefix hashing.

Thanks for listening!

Fujisaki-Okamoto = ‘PKE-to-KEM cooking recipe’:

- How to use public-key encryption to securely transmit symmetric keys.
- Underpins all NIST proposals for KEMs

K Y B E R FrodoKEM

ROM heuristic:

- Helps prevent design flaws.
- Post-quantum (**QROM**) tools for almost-classical reasoning are emerging, but
  - usually at a loss in efficiency.

Qs I’m interested about:

- FO alternatives
  - without re-encrypting?
  - without resorting to the ROM?
- Best way to ‘punish’ malicious ciphertexts? (implicit vs explicit reject)
- FO-KEM security in the real world (e.g., side-channels)
- How to plug FO-KEMs into bigger/more complex protocols
- QROM: improving tool efficiency

Proof technique: extractable QROM

Idea: ROM-like reduction via preimage extraction

QROM $O: X \to Y$ via compressed oracle (Zha19)

+ interface $\mathrm{Extract}_f$ for $f: X \times Y \to T$:

$\mathrm{Extract}_f(\mathrm{t})$:

Collapse oracle database such that

- for one x, $f(x, y) = t$ for all y that are in the database superposition for x

Return x

In FO proof:

$O = \mathrm{Hash}_{\mathrm{rand}}: M \to R$

$f = \mathrm{Encrypt}: M \times R \to C$

$\mathrm{Extract}_f(\mathrm{c})$ = 'preimage' *m*

'Surprising' ≜ PKE spreadness

$\mathrm{Extract}_f$ commutes nicely with O-operations for sufficiently surprising f.

Compressed oracle (Zha19)

- Oracle database initalised to $D := \bigotimes_{x \in query\ domain} |x, \perp >_{D_x}$
- Process queries $|x, y >$ by applying
  - F_{D_x} to output register of D_x

$$F_{D_x}|\psi > := \begin{cases} uniform\ superposition, & |\psi >= \perp \\ \perp, & |\psi > = uniform\ superposition \\ |\psi >, & |\psi > orthogonal\ to\ \perp, uniform \end{cases}$$

  - $\mathrm{CNOT}^{\otimes}_{D_x:Y}$ to D_x, query output register Y
  - F_{D_x} to output register of D_x