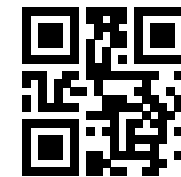


QKD Oracles for Authenticated Key Exchange

ia.cr/2025/1671 arXiv:2509.12478



Kathrin Hövelmanns¹, Daan Planken^{2,3}, Christian Schaffner^{2,3}, **Sebastian Verschoor**^{2,3}



¹ **TU/e** EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

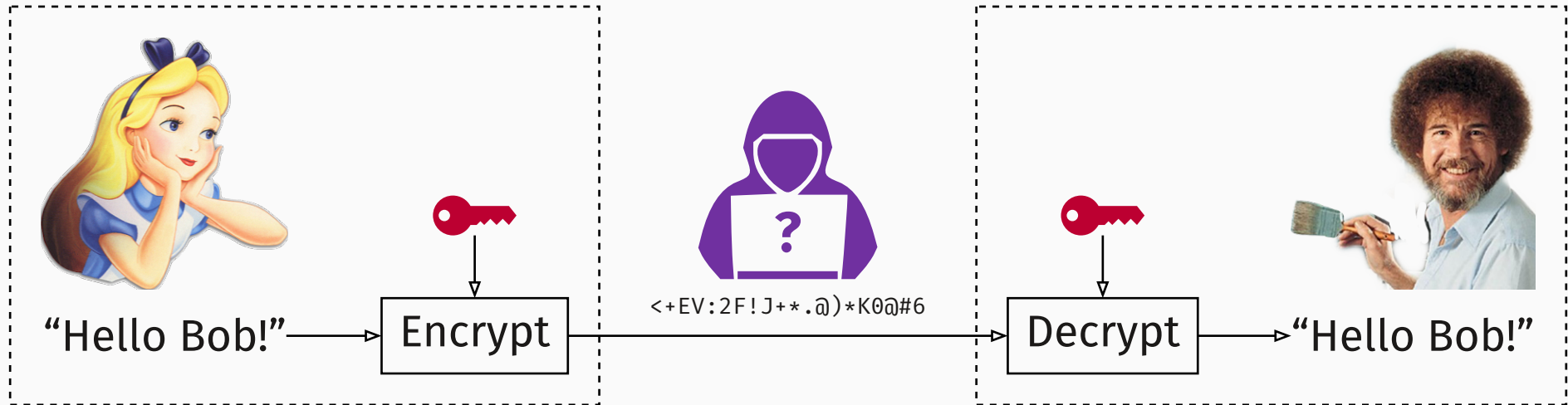
²  UNIVERSITY
OF AMSTERDAM

³ **QuSoft**
Research Center for Quantum Software

2025-10-16 Presentation for PLSec@VU

Authenticated Key Exchange

Encryption

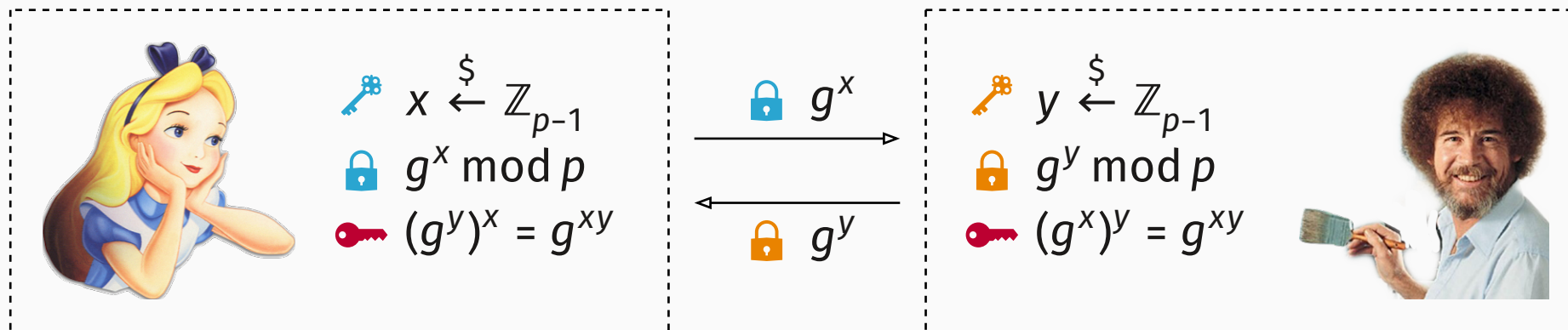


🔑 is called a **symmetric** key

Question: how do Alice and Bob get 🔑?

Answer: Authenticated Key Exchange (AKE)

Diffie-Hellman [DH76]



(, ) is called an **asymmetric** key pair

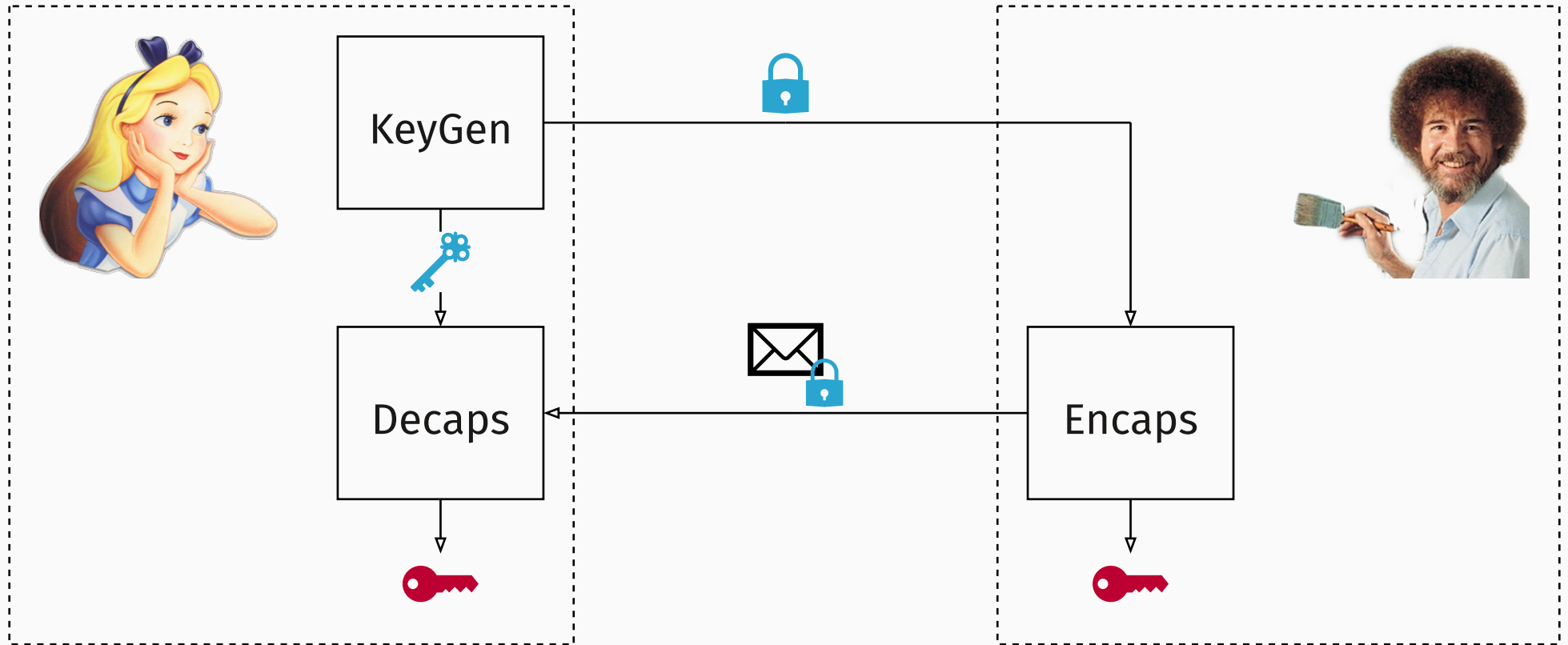
 is the **private key**

 is the **public key**

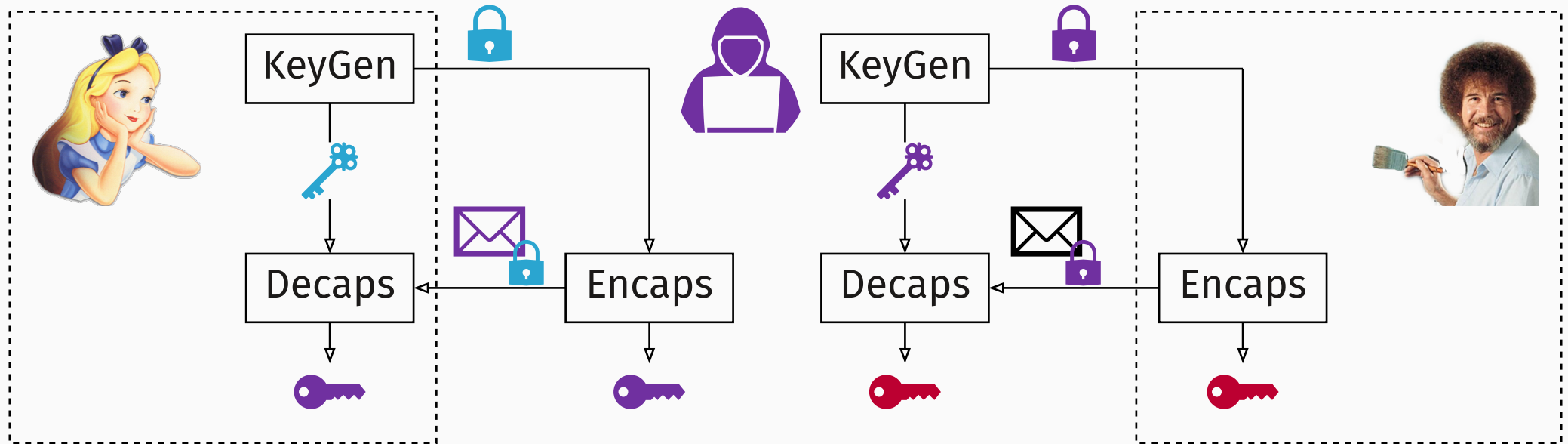
Discrete log problem: $g^x \bmod p \mapsto x$

- this must be (computationally) hard

Key Encapsulation Mechanism (KEM)



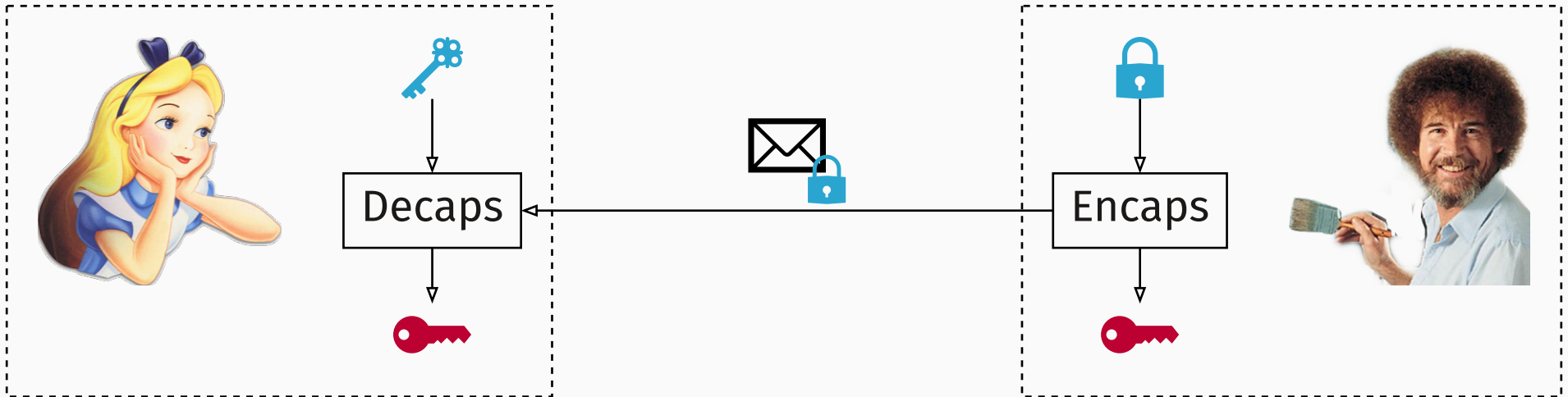
Mallory in the Middle (MitM)



Problem: no authentication

One-way authentication

If Bob already has :

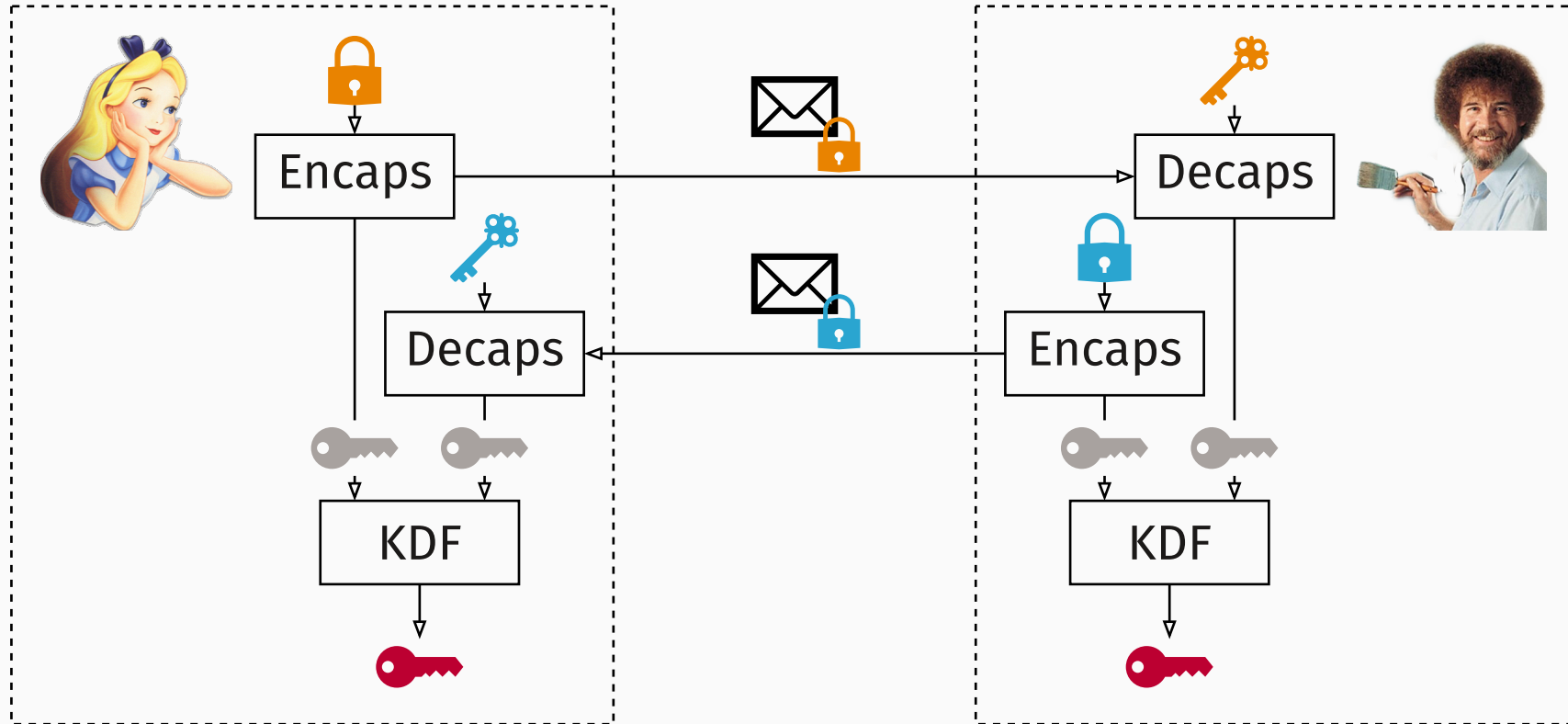


One-way authentication (sort of...)

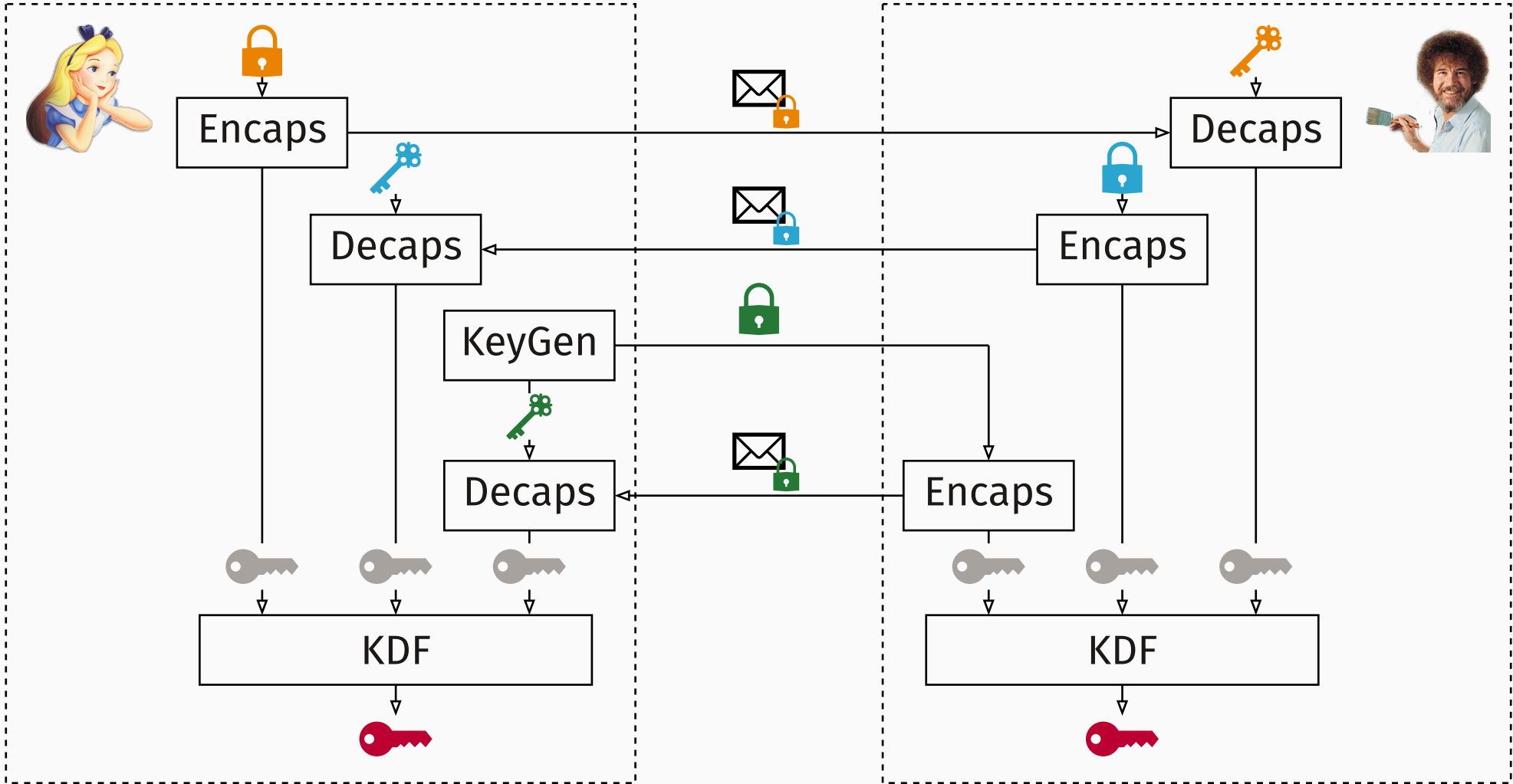
- Bob knows only Alice can open 
- Alice does not know who sent 

Mutual authentication

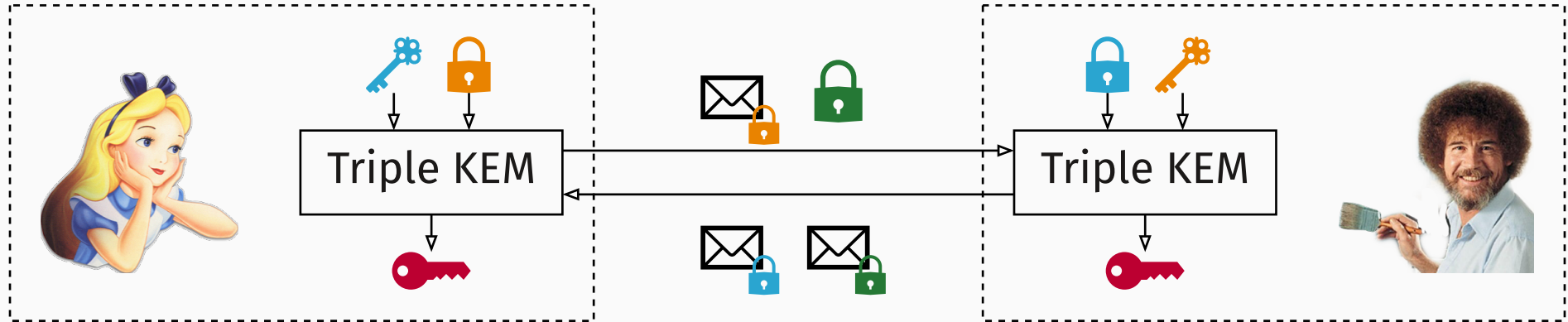
If Bob already has  and Alice already has .



Triple KEM



Triple KEM



- Static KEMs (●, ●) for **authentication**
- Ephemeral KEM (●) for **forward secrecy**

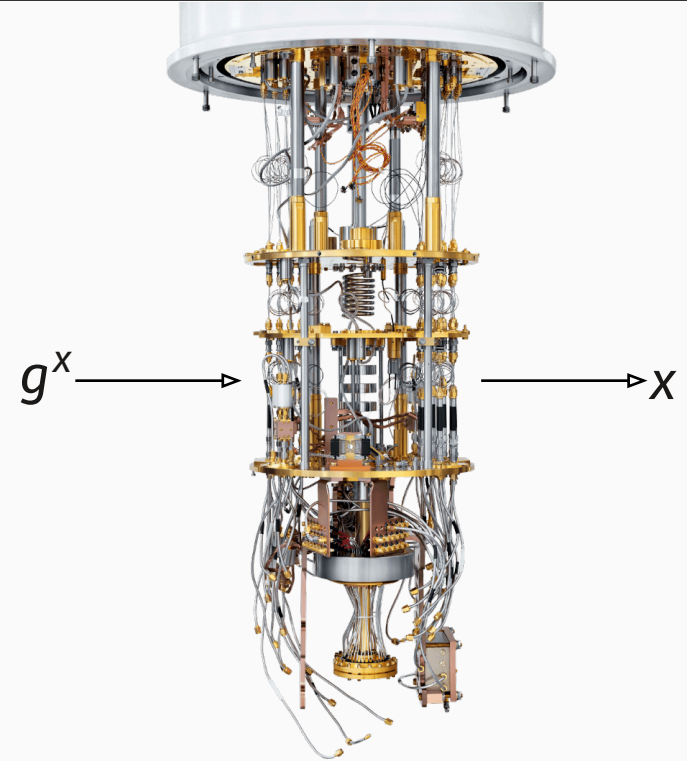
Alice and Bob can run this many times and get **independent keys**

Quantum Computing

Quantum Computing

Shor's quantum algorithm [Sho94]
solves discrete log.

Oops.



Post-Quantum Cryptography (PQC)

Replace discrete-log with

- lattices
- error correcting codes
- supersingular isogenies

We can build KEMs out of these → we can build AKE out of these

Are they secure?

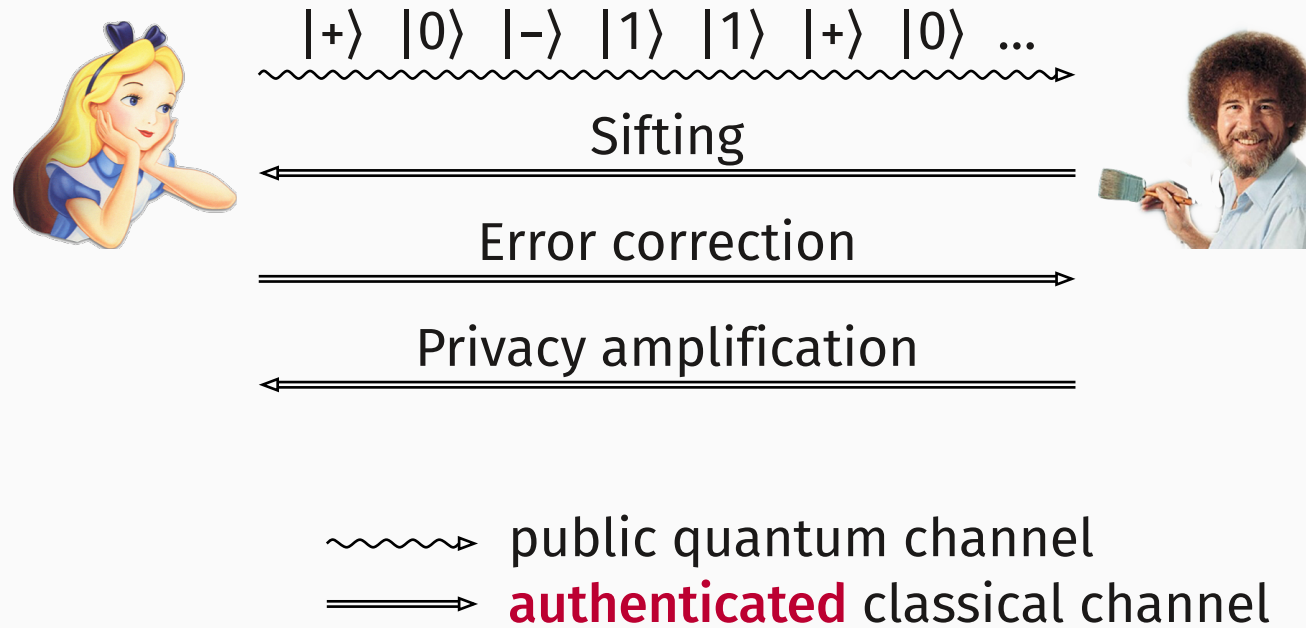
There are no (quantum) algorithms for breaking them **that we know of...**

Relative proofs are the best we can do:

- “**if** you break this scheme, **then** you can solve that *hard* problem”

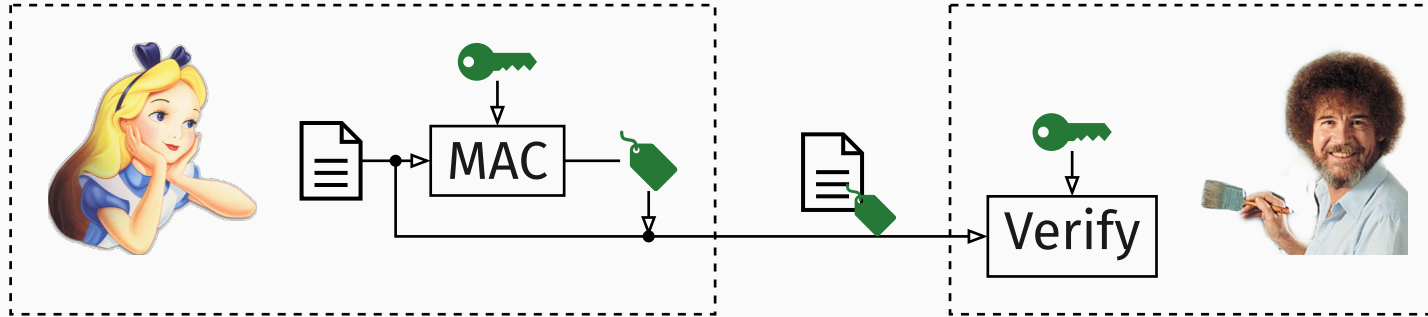
Quantum Key Distribution

Quantum Key Distribution [BB84]

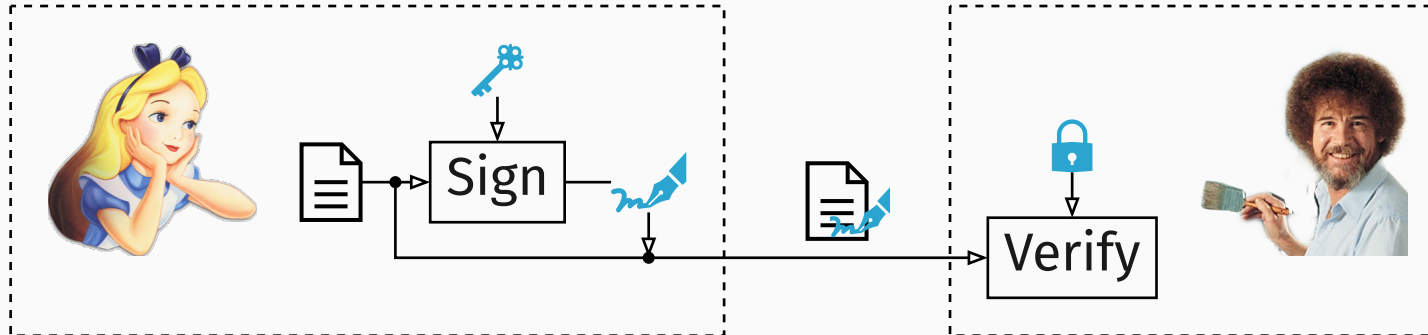


Message authentication

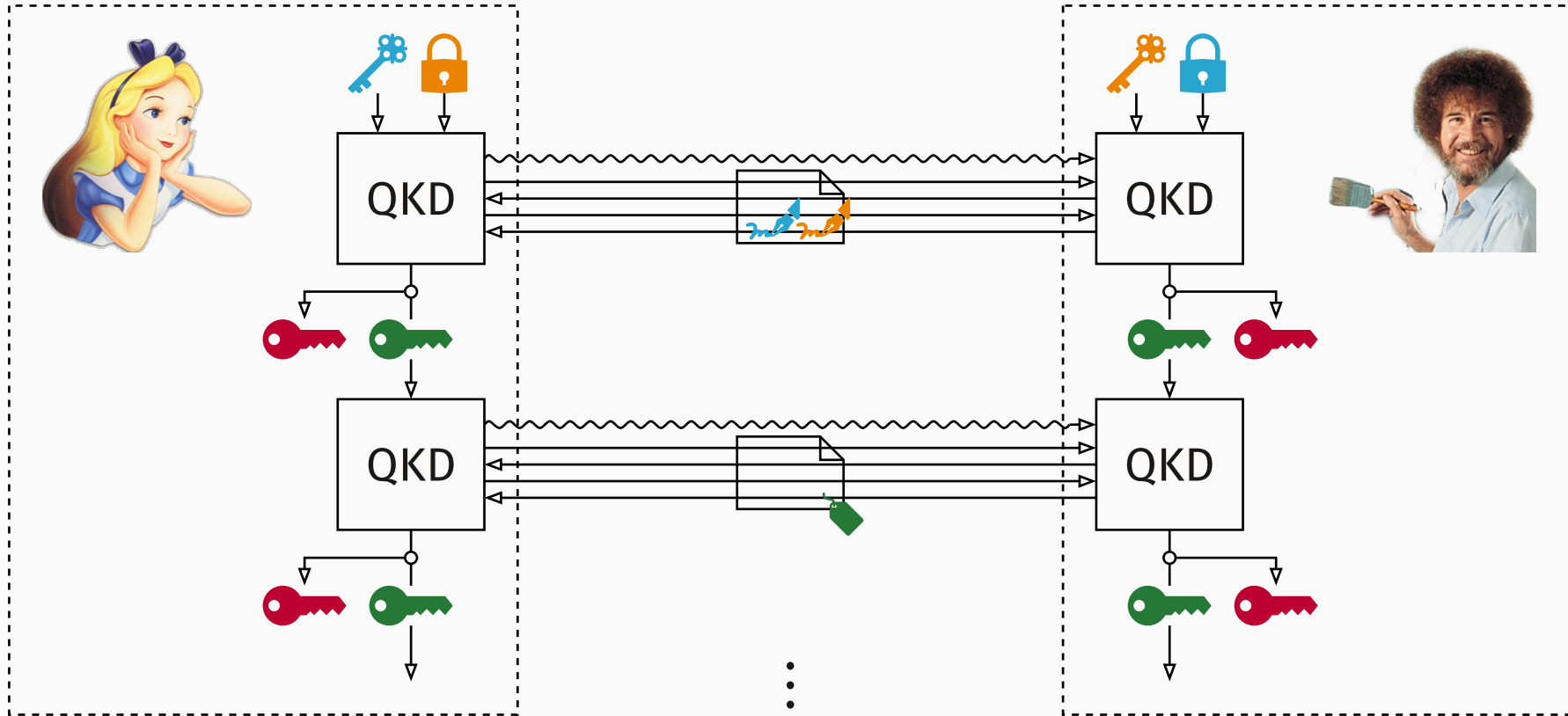
Message Authentication Code (MAC)



Digital Signatures



QKD: bootstrapping



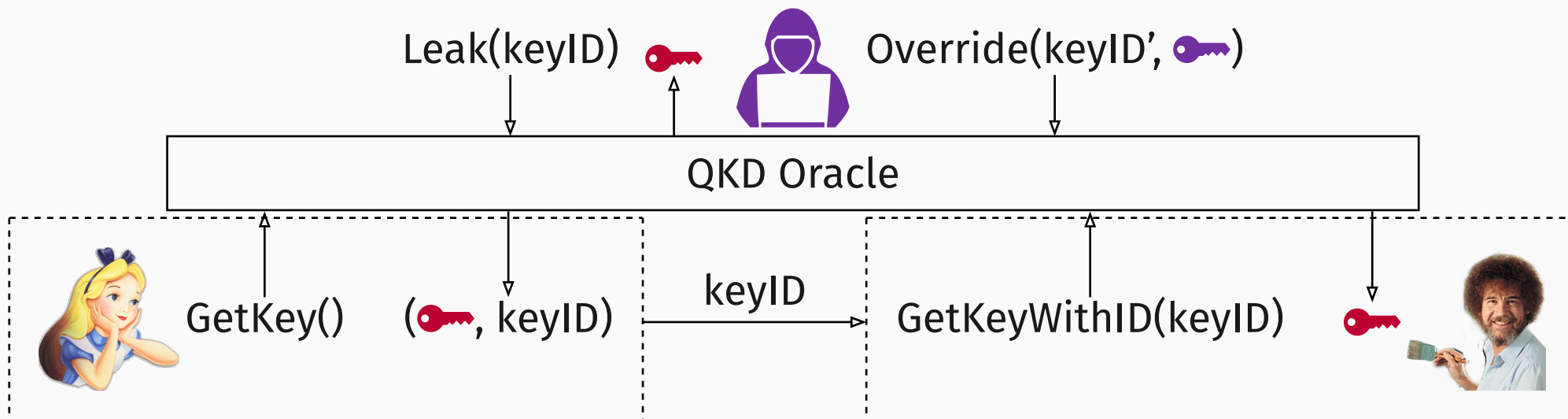
Confidentiality of QKD is **statistical**:

- security independent of the adversary's computation power
- also known as **information theoretical security**

Authentication	Security	QKD Security
 MAC	Statistical	Statistical
 ,  Signature	Computational	Everlasting secrecy

QKD interface: ETSI 014

QKD is slow, so we have to run it continuously in the background

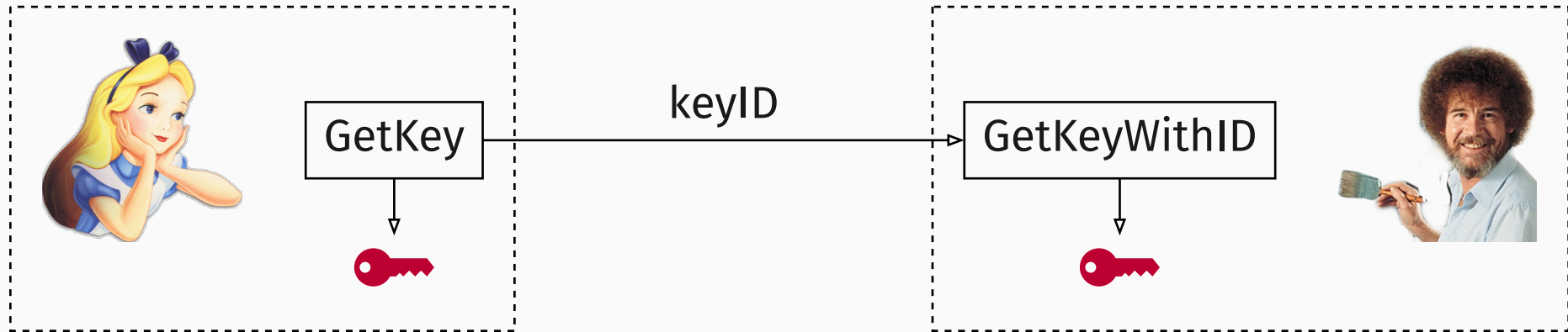


We abstract away the details: our QKD Oracle represents idealized QKD

QKD **implementations** might be broken

QKD interface

Simplified:



Combining PQC and QKD

Combining PQC and QKD

What if our computational cryptography gets broken?

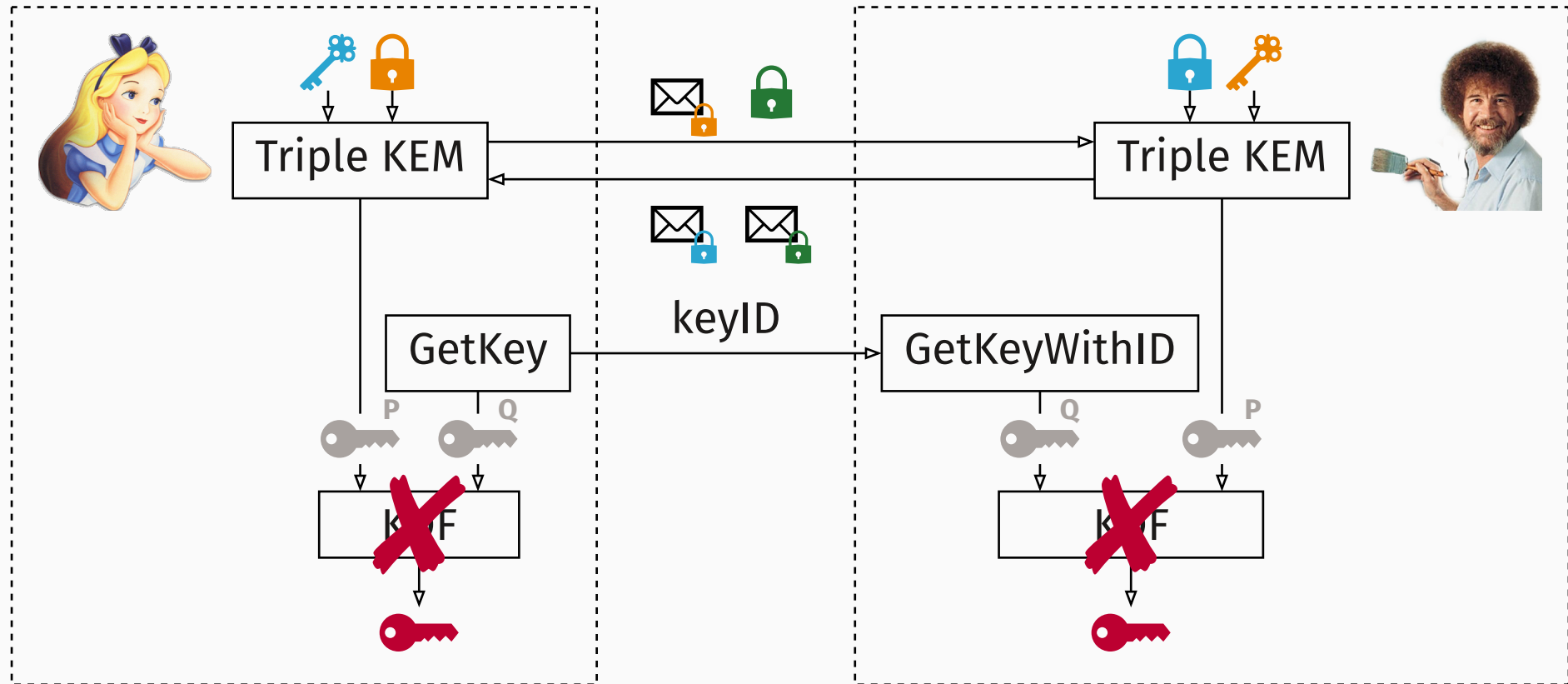


What if our QKD implementation is insecure?

Can we combine PQC and QKD keys...

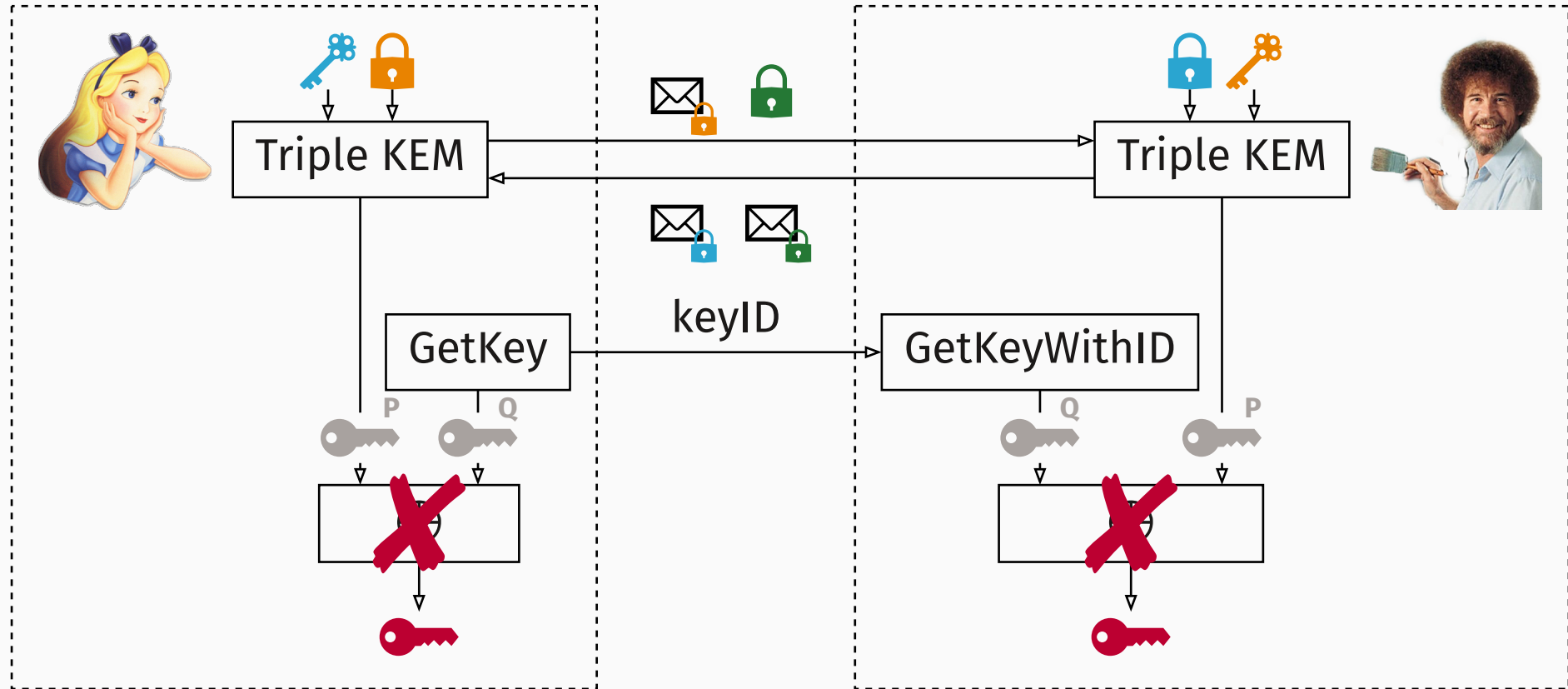
- ...such that we have security if one is broken?
- ...such that we maintain ITS if QKD is secure?

Combining (a first attempt)



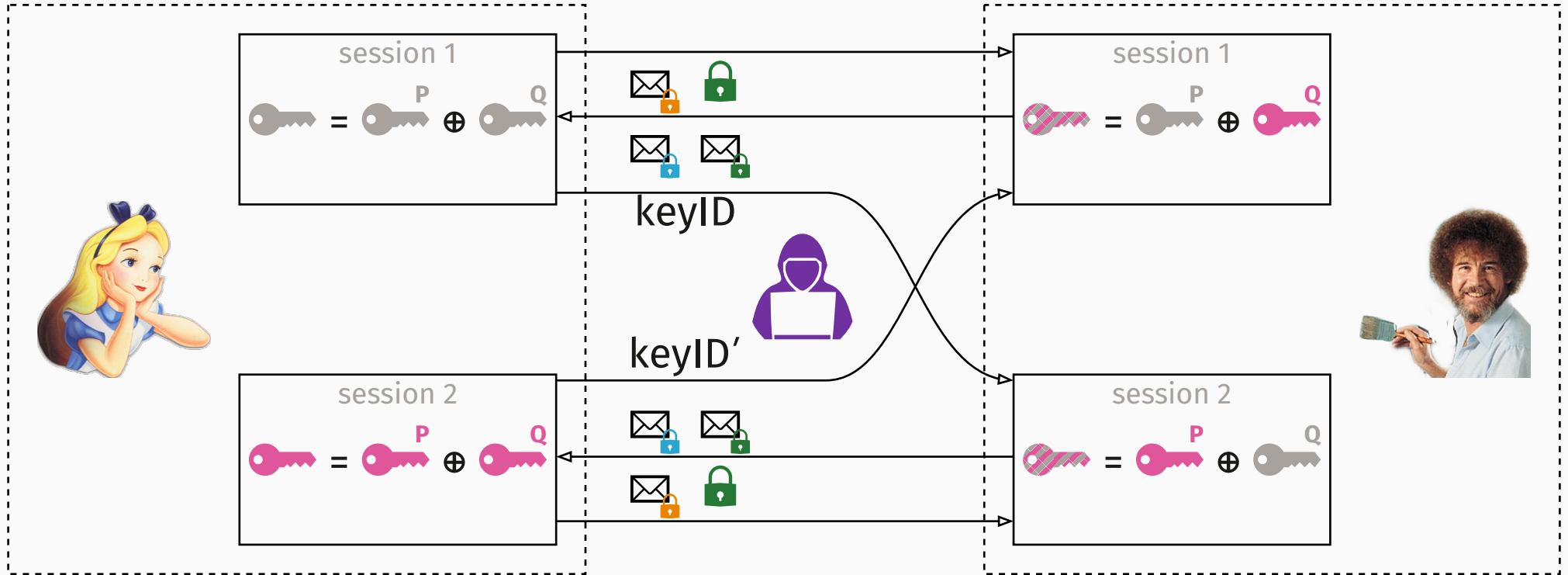
KDFs are only computationally secure: we might as well have no QKD at all*

Combining (a second attempt)



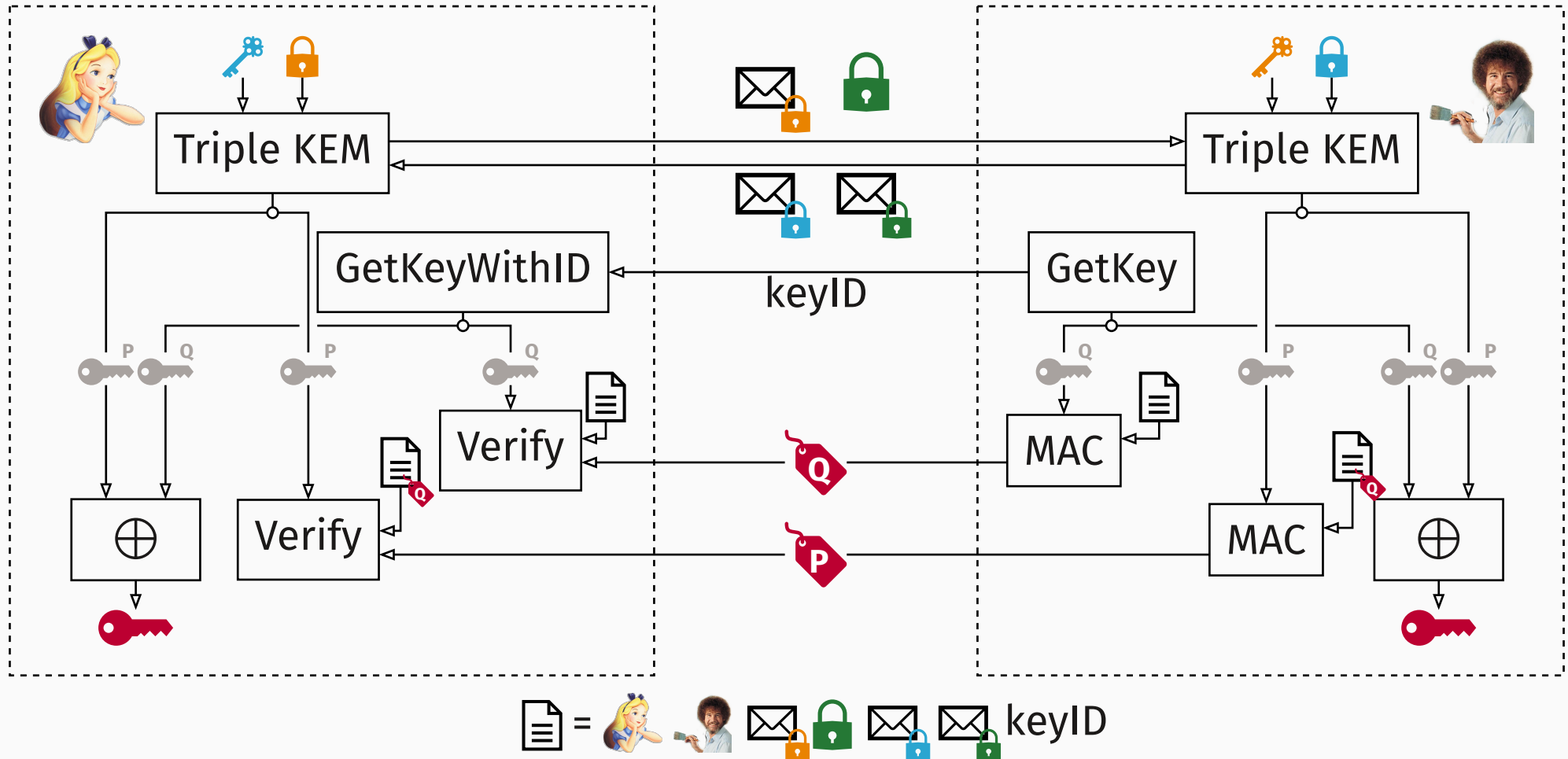
Does not create **independent** keys

Dependent key attack



Leaking a session key reveals secrets of **other** sessions

Combining (third time is the charm)



Proving security

Modelling AKE

Bellare-Rogaway [BR93]

- game between a **challenger** and the **adversary**:

The challenger:

- runs the protocol on behalf of all parties
- gives every message to the adversary

The adversary:

- can send any message to any party
- may reveal *some* secrets
- chooses a **test session**
 - challenger flips a fair coin: $b \stackrel{\$}{\leftarrow} \{0, 1\}$
 - on heads ($b = 0$): reveals real key 
 - on tails ($b = 1$): reveals a random key
- wins the game if they can guess b

Proving security

Game-hopping proof

- start with the **real game** (where $b = 0$)
- make small changes to the game
 - show that no adversary can detect this small change *with high probability*
 - either prove that the subsequent games are statistically close
 - or provide a **reduction**: if the adversary can detect the change, then we can use this adversary to solve a *hard* problem
- end at the **random game** (where $b = 1$)

Conclude that the adversary cannot distinguish the real game from the random game *with high probability*

- and so the protocol reveals *almost* no information about the real key

Security of our protocol

Main theorems (informal):

- If (only) PQC is secure, our protocol is computationally secure
 - by reduction to the components (Triple KEM + MAC)
 - proof in the QROM
- If (only) QKD is secure, our protocol is statistically secure
 - by statistical security of the components (QKD + MAC)
 - this does require correctness of the KEMs (which is statistical)

Conclusion

Conclusion

Combined cryptography is non-trivial

- Especially if we cannot rely on cryptographic hash functions

It is essential to include the QKD key ID

- In the protocol
 - In the model (for example using our oracle)
-

Ongoing work

- modeling relaying on QKD networks
- analysis of a closely related KEM combiner



[zeroknowledge.me/
talks/#2026-vu-plsec](https://zeroknowledge.me/talks/#2026-vu-plsec)

Thank you for your attention

- [DH76] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: 10.1109/TIT.1976.1055638.
- [BB84] C. H. Bennett and G. Brassard, “Quantum Cryptography: Public Key Distribution and Coin Tossing,” in *Proceedings of the IEEE International Conference on Computers, Systems and Signal Processing*, Bangalore, India: IEEE, 1984, pp. 175–179. doi: 10.1016/j.tcs.2014.05.025.
- [BR93] M. Bellare and P. Rogaway, “Entity Authentication and Key Distribution,” in *CRYPTO93*, D. R. Stinson, Ed., Berlin, Heidelberg: Springer, 1993, pp. 232–249. doi: 10.1007/3-540-48329-2_21.
- [Sho94] P. W. Shor, “Algorithms for Quantum Computation: Discrete Logarithms and Factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.
- [Bin+19] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila, “Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange,” in *Post-Quantum Cryptography*, Jintai Ding and Rainer Steinwandt, Eds., Cham: Springer International Publishing, Jul. 2019, pp. 206–226. doi: 10.1007/978-3-030-25510-7_12.

<pre> QkdInit() 01 $kID_{ctr} := 0$ 02 $qKey := []$ 03 $qStatus := []$ 04 $qSentSid := []$ 05 $qRecvSid := []$ QKD-KEY-HOLDERS(kID) 06 return ($qSentSid[kID], qRecvSid[kID]$) QKD-LEAK($kID$) 07 if $qStatus[kID] = \text{HONEST}$ 08 $qStatus[kID] := \text{REVEALED}$ 09 return $qKey[kID]$ QKD-OVERRIDE(kID, k) 10 $qKey[kID] := k$ 11 $qStatus[kID] := \text{CORRUPT}$ </pre>	<pre> QKD-GET-KEY(sID, ℓ) 12 $kID_{ctr} := kID_{ctr} + 1$ 13 $qSentSid[kID_{ctr}] := sID$ 14 if $qKey[kID_{ctr}] = \perp$ 15 $qKey[kID_{ctr}] \leftarrow_{\\$} \{0, 1\}^\ell$ 16 $qStatus[kID_{ctr}] := \text{HONEST}$ 17 $k := qKey[kID_{ctr}]$ 18 return (kID_{ctr}, k) QKD-GET-KEY-WITH-ID(sID, kID) 19 $sID_{qs} := qSentSid[kID]$ 20 if ($owner[sID_{qs}], peer[sID_{qs}] \neq$ $(peer[sID], owner[sID])$) 21 return $\perp$ 22 if $qKey[kID] = \perp$ 23 return $\perp$ 24 $qRecvSid[kID_{ctr}] += [sID]$ 25 $k := qKey[kID]$ 26 $qKey[kID] := \perp$ 27 return k </pre>
--	--

Fig.6: QKD oracle model. Honest sessions have access to the **QKD-GET-KEY**($sID, \cdot$) and **QKD-GET-KEY-WITH-ID**($sID, \cdot$) oracles (in purple), while the adversary may query **QKD-KEY-HOLDERS**, **QKD-LEAK**, and **QKD-OVERRIDE** (in red). The party calling **QKD-GET-KEY** specifies the key length ℓ . kID is a (predictable) key identifier. Besides the functionality, we add some bookkeeping values (in green) that are used in the proofs, but are not available to either the honest user or the adversary. Keys are removed in Line 26 from the oracle when they are requested by the receiver (the party calling **QKD-GET-KEY-WITH-ID**).

Formalizing

<p><u>Init(pk_j)</u></p> <pre> 01 $(c_b, k_b) \leftarrow \text{Encaps}_{\text{stat}}(pk_j)$ 02 $(pk_e, sk_e) \leftarrow \text{KeyGen}_{\text{eph}}()$ 03 $m_1 := (c_b, pk_e)$ 04 $s' := (k_b, sk_e, m_1)$ 05 return (m_1, s') </pre>	<p><u>SendM2$^{\text{QKD-GET-KEY-WITH-ID}(sID, \cdot)}(i, sk_i, j, s, m_2)$</u></p> <pre> 16 $(c_a, c_e, kID, \tau'_1, \tau'_2) := m_2$ // or abort 17 $(k_b, sk_e, m_1) := s$ // or abort 18 $k'_a := \text{Decaps}_{\text{stat}}(sk_i, c_a)$ 19 $k'_e := \text{Decaps}_{\text{eph}}(sk_e, c_e)$ 20 $k_{\text{pqc}} := \text{KDF}(k_b, k'_a, k'_e)$ 21 $k_{\text{qkd}} :=$ </pre>
<p><u>SendM1$^{\text{QKD-GET-KEY}(sID, \cdot)}(i, sk_i, j, pk_j, m_1)$</u></p> <pre> 06 $(c_b, pk_e) := m_1$ // or abort 07 $k'_b := \text{Decaps}_{\text{stat}}(sk_i, c_b)$ 08 $(c_a, k_a) \leftarrow \text{Encaps}_{\text{stat}}(pk_j)$ 09 $(c_e, k_e) \leftarrow \text{Encaps}_{\text{eph}}(pk_e)$ 10 $k_{\text{pqc}} := \text{KDF}(k'_b, k_a, k_e)$ 11 $(kID, k_{\text{qkd}}) \leftarrow \text{QKD-GET-KEY}(sID, \ell_{\text{qkd}})$ 12 $t := (m_1, (c_a, c_e, kID))$ 13 $(k_{\text{sess}}, \tau_1, \tau_2) := \text{Combine}(j, i, t, k_{\text{qkd}}, k_{\text{pqc}})$ 14 $m_2 := (c_a, c_e, kID, \tau_1, \tau_2)$ 15 return (m_2, k_{sess}) </pre>	<p>$\text{QKD-GET-KEY-WITH-ID}(sID, kID)$</p> <pre> 22 if $k_{\text{qkd}} = \perp$: abort 23 $t := (m_1, (c_a, c_e, kID))$ 24 $(k_{\text{sess}}, \tau_1, \tau_2) := \text{Combine}(i, j, t, k_{\text{qkd}}, k_{\text{pqc}})$ 25 if $\tau_1 \neq \tau'_1$ or $\tau_2 \neq \tau'_2$: abort 26 return k_{sess} </pre>
	<p><u>Combine($i_I, i_R, t, k_{\text{qkd}}, k_{\text{pqc}}$)</u></p> <pre> 27 $(k_{\text{qkd}, m} \parallel k_{\text{qkd}, s}) := k_{\text{qkd}}$ 28 $(k_{\text{pqc}, m} \parallel k_{\text{pqc}, s}) := k_{\text{pqc}}$ 29 $k_{\text{sess}} := k_{\text{qkd}, s} \oplus k_{\text{pqc}, s}$ 30 $\tau_1 := \text{MAC}_{k_{\text{qkd}, m}}^{(\text{qkd})}((t, i_I, i_R))$ 31 $\tau_2 := \text{MAC}_{k_{\text{pqc}, m}}^{(\text{pqc})}((t, \tau_1, i_I, i_R))$ 32 return $(k_{\text{sess}}, \tau_1, \tau_2)$ </pre>

Fig. 9: A formal implementation of our AKE protocol $\Pi = (\text{Init}, \text{SendM1}, \text{SendM2})$, using the QKD oracle ($\text{QKD-GET-KEY}, \text{QKD-GET-KEY-WITH-ID}$), in the case of end-to-end QKD.

Theorem 5.2 (PQC-based security). *Let Π be the protocol of Fig. 9, where KEM_{eph} is δ_{eph} -correct and has key length ℓ_{eph} , and KEM_{stat} is δ_{stat} -correct, has collision probabilities $\mu(Encaps_{stat})$ and $\mu(Secret_{stat})$, and has key length ℓ_{stat} . Let A be an IND-StAA-PQC adversary executing Π with n_{pt} parties that establishes n_s sessions that makes at most q_H calls to the key-derivation function KDF (modelled as a QROM). Then there exist IND-CPA adversary B_1 against KEM_{eph} , IND-CPA adversary B_2 against KEM_{stat} , IND-CCA adversary B_3 against KEM_{stat} , and OT-sEUF-CMA adversary B_4 against $MAC^{(pqc)}$ such that*

$$\begin{aligned} \text{Adv}^{\text{IND-StAA-PQC}}(\Pi, A) \leq & 2n_s^2 \left(\delta_{eph} + \text{Adv}_{KEM_{eph}}^{\text{IND-CPA}}(B_1) + 2q_H \frac{1}{\sqrt{2^{\ell_{eph}}}} \right) \\ & + 2n_s^2 n_{pt} \left(\delta_{stat} + \text{Adv}_{KEM_{stat}}^{\text{IND-CCA}}(B_2) + 2q_H \frac{1}{\sqrt{2^{\ell_{stat}}}} \right) + 2 \cdot n_s \cdot \mu(Encaps_{stat}) \\ & + 8n_s n_{pt} \left(\delta_{stat} + \text{Adv}_{KEM_{stat}}^{\text{IND-CCA}}(B_3) + n_s^2 \mu(Secret_{stat}) + n_s \frac{2q_H}{\sqrt{2^{\ell_{stat}}}} + n_s \text{Adv}_{MAC^{(pqc)}}^{\text{OT-sEUF-CMA}}(B_4) \right) \end{aligned}$$

where the runtime of B_1, B_2, B_3, B_4 is about that of A .

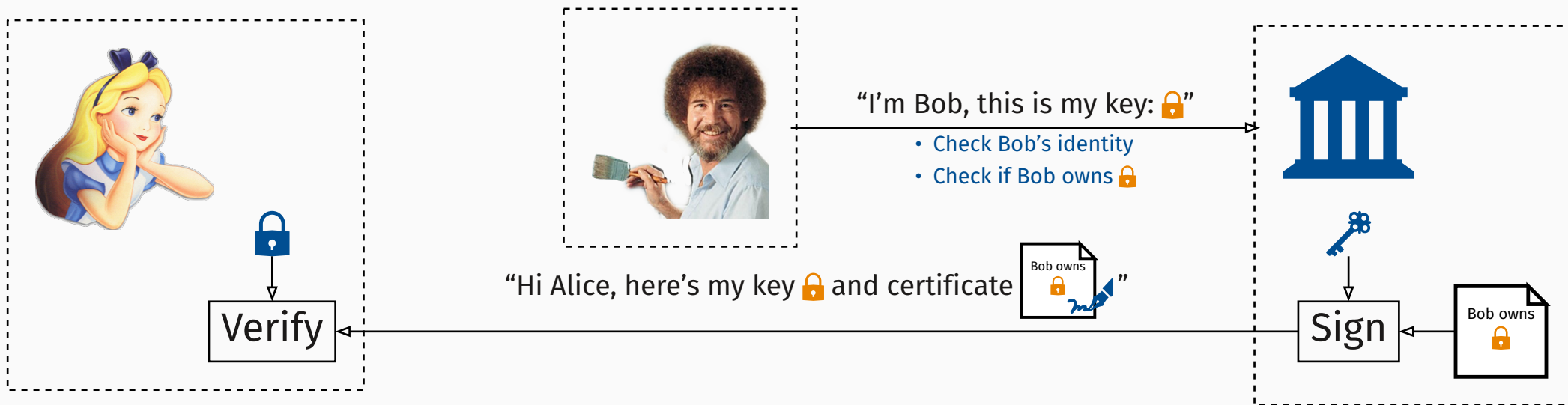
Theorem 5.5 (QKD-based security). *Let Π be the protocol of Fig. 9, where KEM_{eph} is δ_{eph} -correct and KEM_{stat} is δ_{stat} -correct. Let A be an IND-AA-QKD adversary executing Π that establishes n_s sessions. Then there exist an OT-sEUF-CMA adversary B_7 against $MAC^{(qkd)}$ such that*

$$\text{Adv}^{\text{IND-AA-QKD}}(\Pi, A) \leq 2n_s \left(n_s(2\delta_{stat} + \delta_{eph}) + \text{Adv}_{MAC^{(qkd)}}^{\text{OT-sEUF-CMA}}(B_7) \right)$$

where the runtime of B_7 is potentially unbounded.

Certificates

- A trusted third party 🏛️ with a well-known public key 🔒



- Alice trusts 🏛️, so she accepts that 🔒 belongs to Bob

Modelling AKE

The test session has

- owner (e.g. Alice)
- peer (e.g. Bob)
- matching session: the session with the exact same transcript (if it exists)

The adversary may reveal:

- long term keys 
 - except the one of the peer
- session keys 
 - except those of the test session and matching session
- QKD keys
 - except those used in the test session
- session state

KEM Combiners

Can we combine KEM_1 and KEM_2

- such that the combination is secure even if one is broken?

Relevant for combining pre- and post-quantum KEMs

```
Encaps((pk1, pk2)):
  (c1, k1)  $\xleftarrow{\$}$  Encaps1(pk1)
  (c2, k2)  $\xleftarrow{\$}$  Encaps2(pk2)
  return (c1 || c2, k1 ⊕ k2)
```

Is this IND-CCA secure?

Mix-and-Match attack [Bin+19]

```
ADec(·)((pk1, pk2), (c1, c2), kb):  
  (c'1, k'1)  $\xleftarrow{\$}$  Encaps1(pk1)  
  (c'2, k'2)  $\xleftarrow{\$}$  Encaps2(pk2)  
  k''1  $\leftarrow$  Dec(c'1 || c2)           // c'1 || c2 ≠ c1 || c2  
  k''2  $\leftarrow$  Dec(c1 || c'2)           // c1 || c'2 ≠ c1 || c2  
  return kb = k'1 ⊕ k''1 ⊕ k'2 ⊕ k''2
```

XOR-then-MAC (XtM) Combiner [Bin+19]

$$s_1 \parallel m_1 \leftarrow k_1$$

$$\tau_1 \leftarrow \text{MAC}(m_1, c_1 \parallel c_2)$$

$$s_2 \parallel m_2 \leftarrow k_2$$

$$\tau_2 \leftarrow \text{MAC}(m_2, c_1 \parallel c_2)$$

and then

$$c \leftarrow c_1 \parallel c_2 \parallel \tau_1 \parallel \tau_2$$

$$k \leftarrow s_1 \oplus s_2$$

(Upcoming work) the existing security theorem for XtM is flawed

- but it can be fixed (and formally verified)