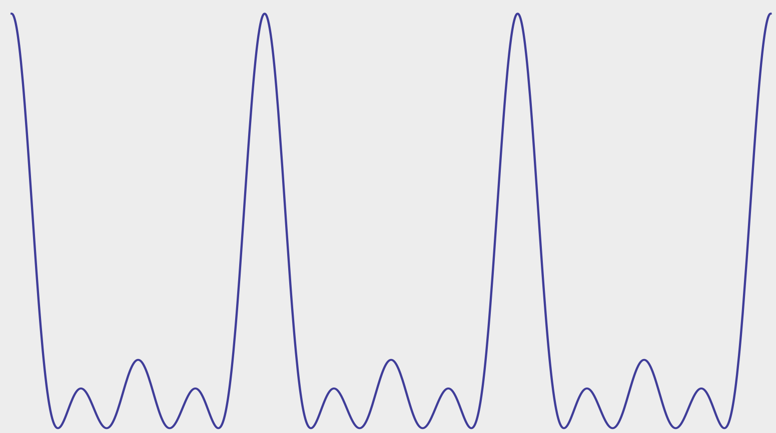
Local Quantum Cryptography

Mark Zhandry (Princeton & NTT Research)

Based on joint work with Ryan Amos, Marios Georgiou, and Aggelos Kiayias

Quantum Background

Quantum Period Finding
[Simon'94,Shor'95]

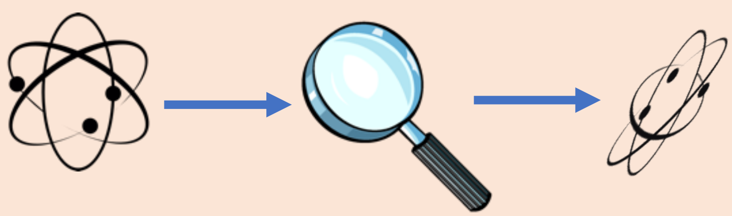


Factoring: $N=pq \rightarrow p, q$

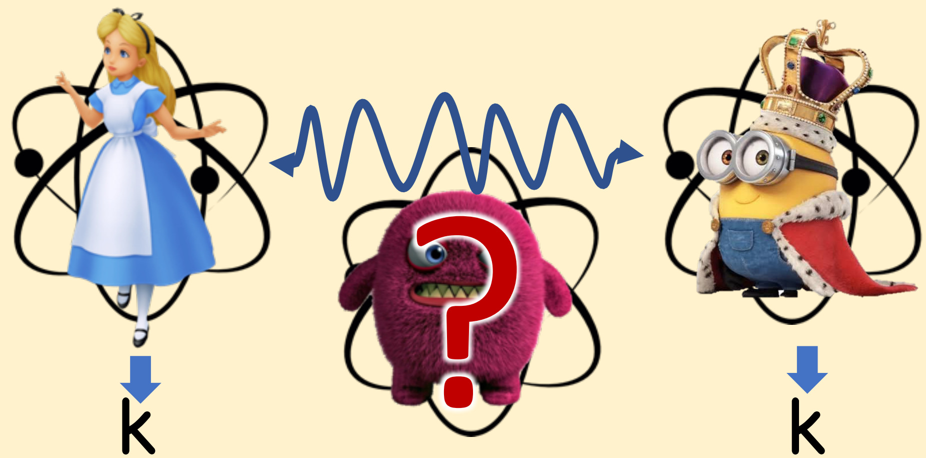
DLog: $g, g^a \rightarrow a$

Quantum Background

Observer Effect

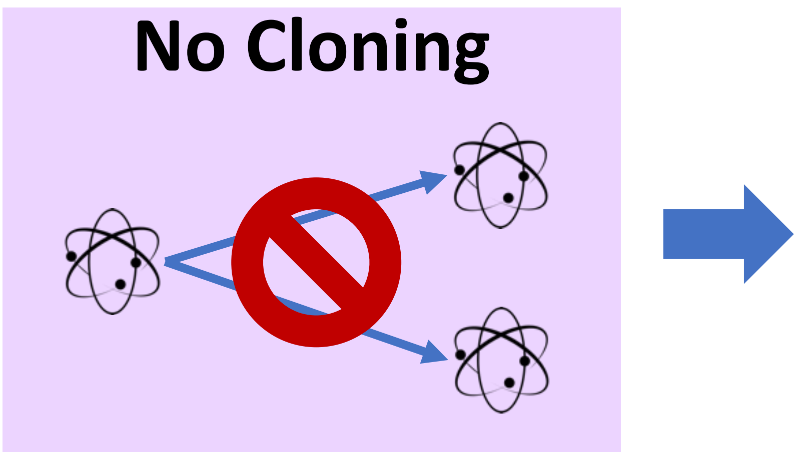


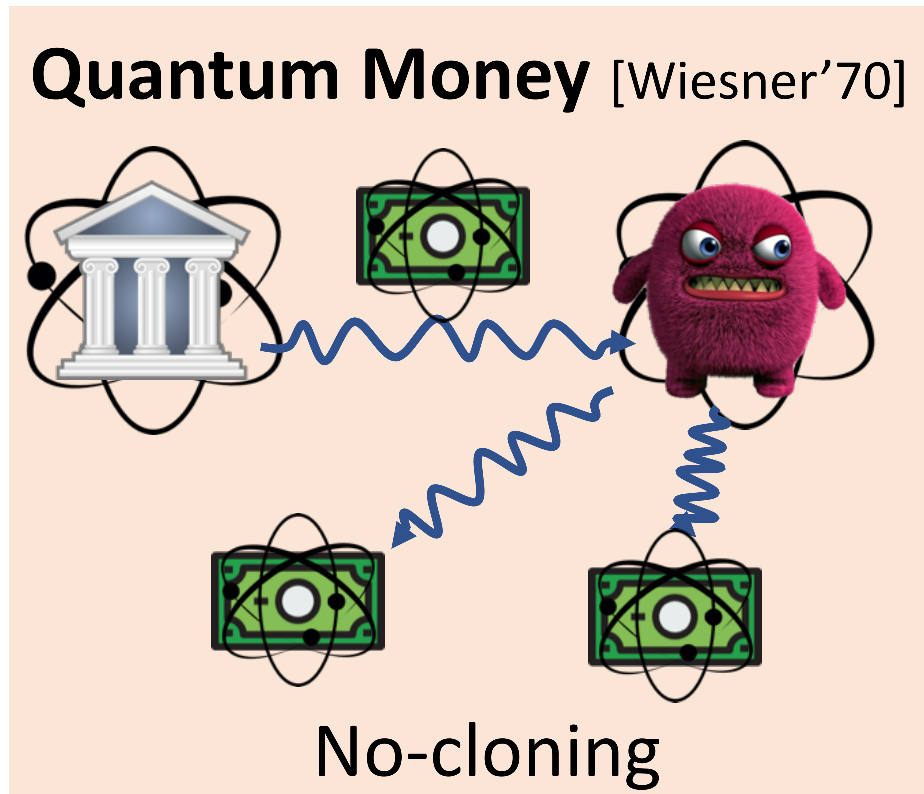
QKD [Bennett-Brassard'84]



Observer effect → Eavesdropping detection

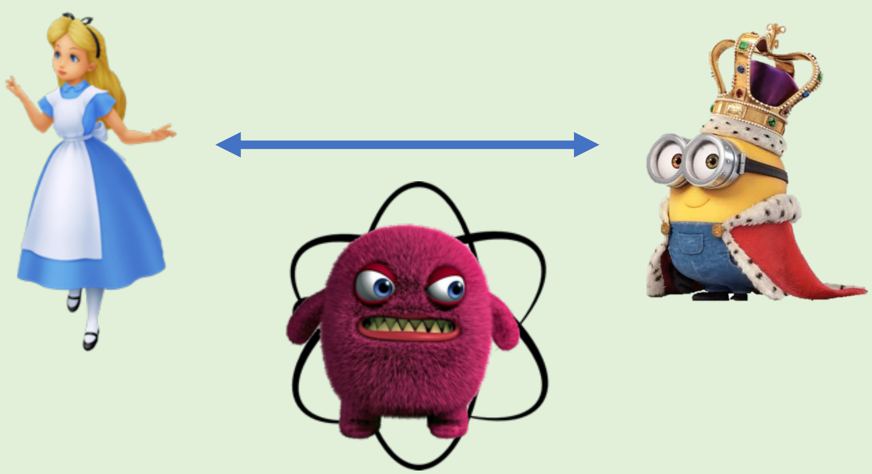
Quantum Background





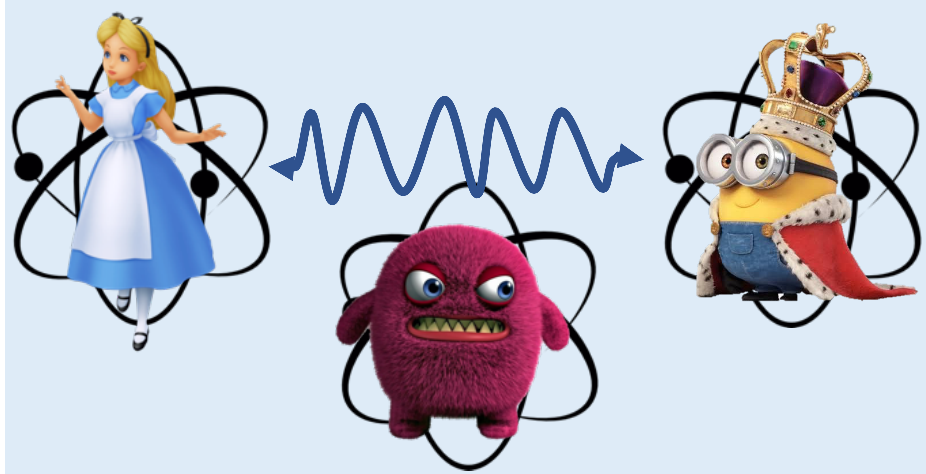
Post-Quantum vs Quantum Crypto

Post-Quantum Crypto:



Protect classical crypto from quantum attacks

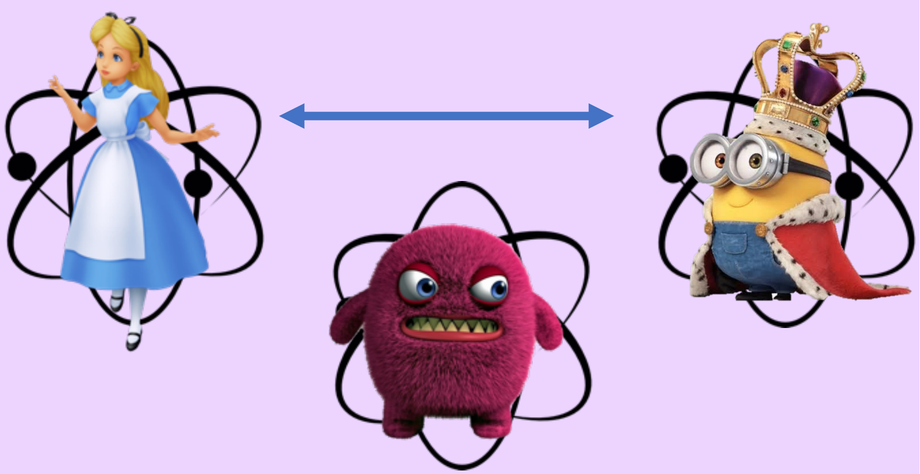
Quantum Crypto:



Use quantum effects to do new things

Emerging Area: Local Quantum Crypto

Local Quantum Crypto:



Everyone's quantum, communication classical

Main Question:
Is anything interesting possible?

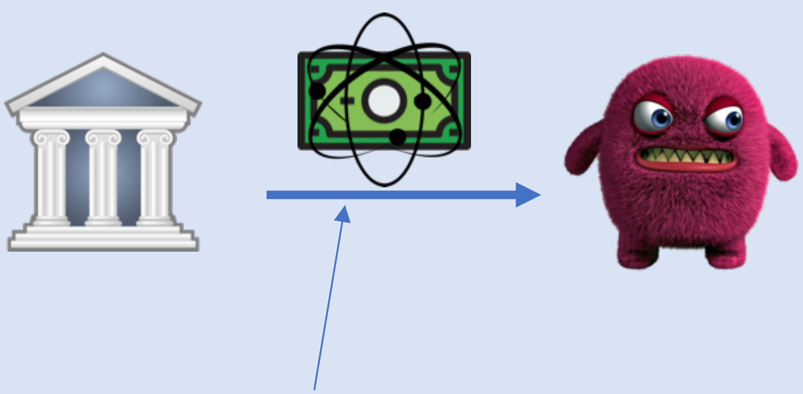
Prior Work: (Verifiable) Delegation

Mahadev'18 x2, Brakerski-Christiano-Mahadev-Vazirani-Vidick'18]

(I Don't really count multi-device setting: requires entanglement)

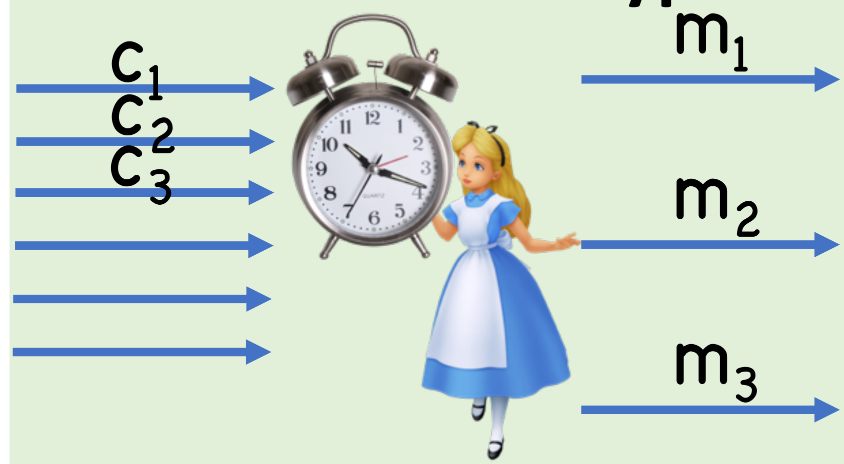
Two Motivating Examples

“Classical” Quantum Money



Send quantum money over classical channels?

Rate-Limited Decryption



Can only decrypt single ciphertext every T minutes

This Work: Two Questions

Q1: Can quantum keys yield any interesting crypto?

Q2: Can quantum states be sent over classical channels?

Disclaimer

Strong computational assumptions:

- Obfuscation (VBB)
- *Extractable* witness encryption
- *Recursively composable* zk-SNARKs
- Post-quantum proofs of (sequential) work

Part 1: One-Shot Signatures and Applications

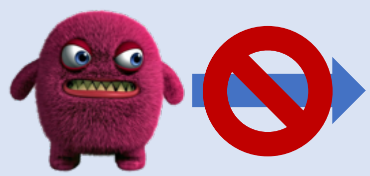
Tool: One-Shot Signatures

Syntax:

(pk,sk) ← Gen()

σ ← Sign(sk,m)

0/1 ← Ver(pk,m,σ)

Security:  $(pk, m_0, m_1, \sigma_0, \sigma_1)$ s.t.

$m_0 \neq m_1$,

$Ver(pk, m_0, \sigma_0) = 1$, and

$Ver(pk, m_1, \sigma_1) = 1$

Impossibility of One-Shot Signatures?

Attack?

- $(pk, sk) \leftarrow Gen()$
- $\sigma_0 \leftarrow Sign(sk, m_0)$
- $\sigma_1 \leftarrow Sign(sk, m_1)$

Idea!

What if **sk** is "used up" to produce σ_0?

- Makes no sense classically (rewinding)
- But quantumly, maybe?

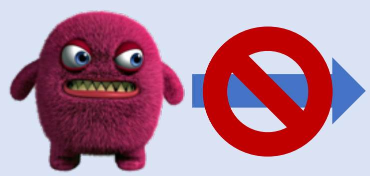
One-Shot Signatures (Quantum)

Syntax:

(pk, sk) ← Gen()

σ ← Sign(sk ,m)

0/1 ← Ver(pk,m,σ)

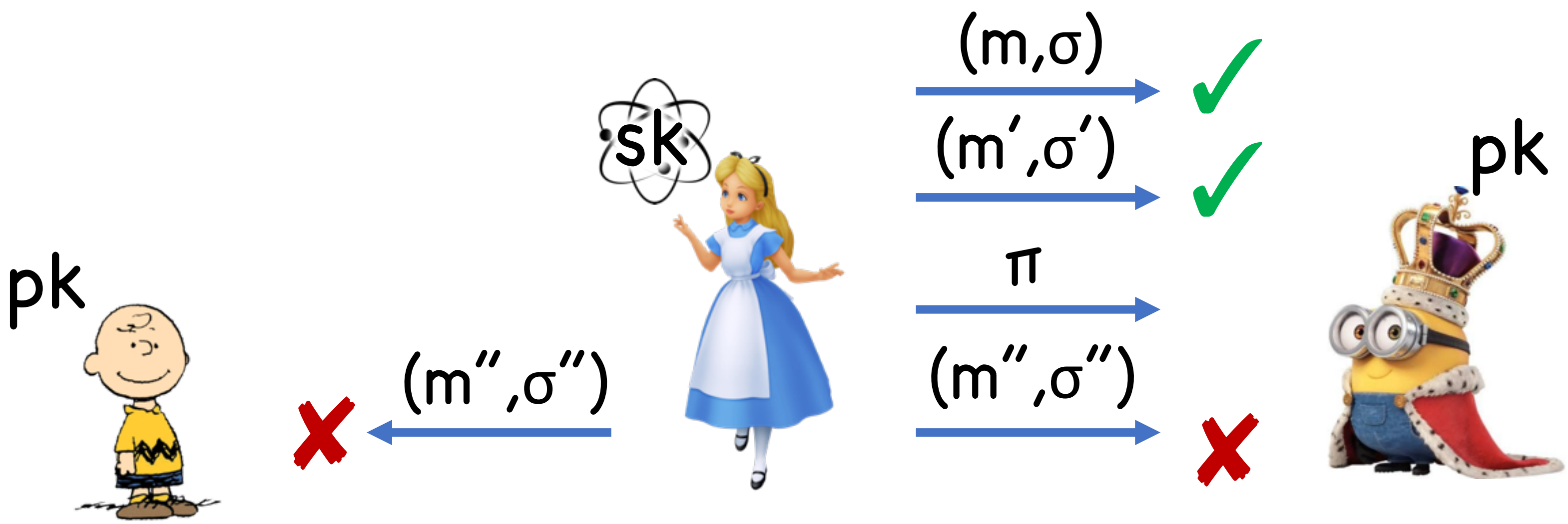
Security: 

$(pk, m_0, m_1, \sigma_0, \sigma_1)$ s.t.
$m_0 \neq m_1$,
$Ver(pk, m_0, \sigma_0)=1$, and
$Ver(pk, m_1, \sigma_1)=1$

For now, assume ∃ OSS. Will construct later

OSS Apps: Burnable Signatures

Goal: Prove that you destroyed your signing key



Solution: Signature Chaining

sk_0

pk_0

Solution: Signature Chaining

(assume message is part of sig)

$\sigma_{pk_0}(m_1 || pk_1)$,

pk_0

Solution: Signature Chaining

$\sigma_{pk_0}(m_1 || pk_1)$,

$\sigma_{pk_1}(m_2 || pk_2)$,

pk_0

Solution: Signature Chaining

$\sigma_{pk_0}(m_1||pk_1)$,
$\sigma_{pk_1}(m_2||pk_2)$,
$\sigma_{pk_2}(\perp)$

pk_0

Proof Idea:

Valid post-burn signature → Forked chain → OSS Forgery

Caveats

|signature| grows with #(messages)

Fix: SNARKs

|sk| grows with #(messages)

Fix: *Recursively Composable* SNARKs

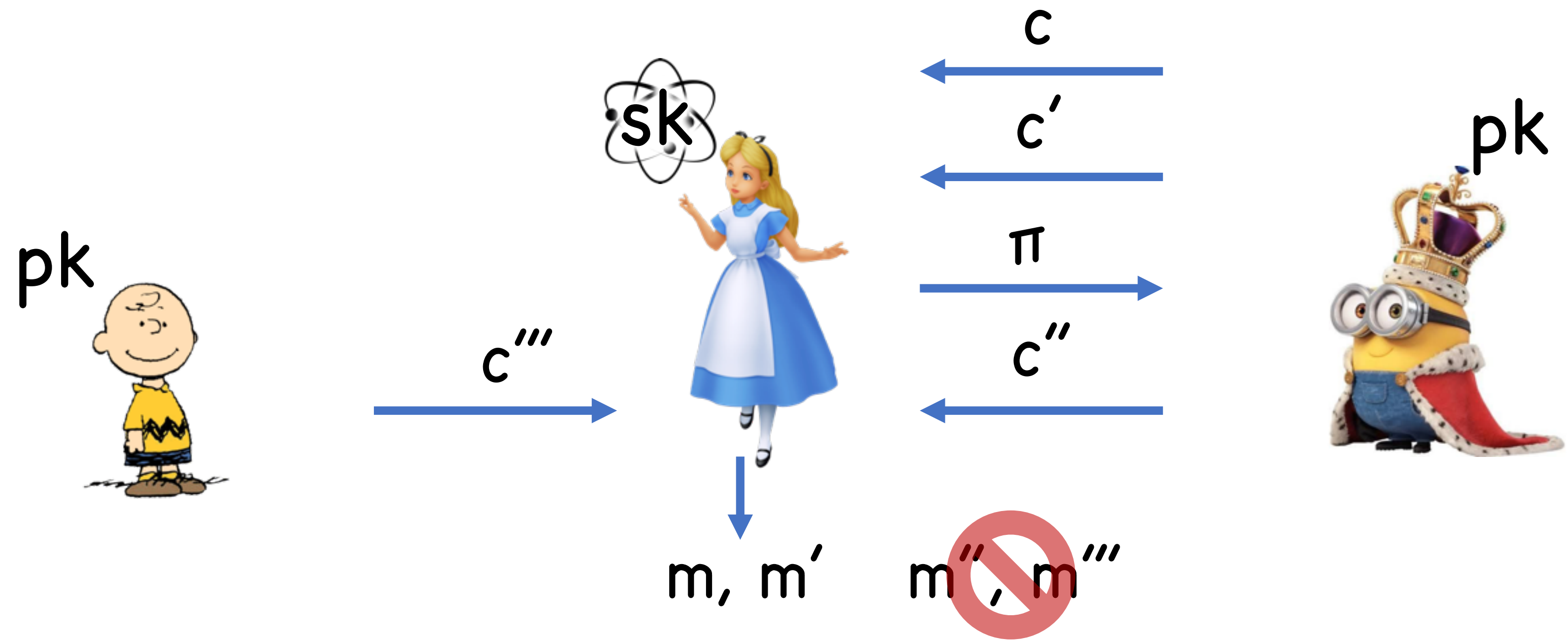
Stateful Signing

Natural for quantum keys

(reading key may disturb it)

OSS Apps: Burnable *Decryption*

Goal: Prove that you destroyed your decryption key



Burnable Sigs → Burnable Decryption

Tool: (Extractable) Witness Encryption

c ← WE.Enc(NP statement x , m)
m ← WE.Dec(x, witness w, c)

Security: c hides m, unless you "know" a witness

Burnable Sigs → Burnable Decryption

r ← Message space

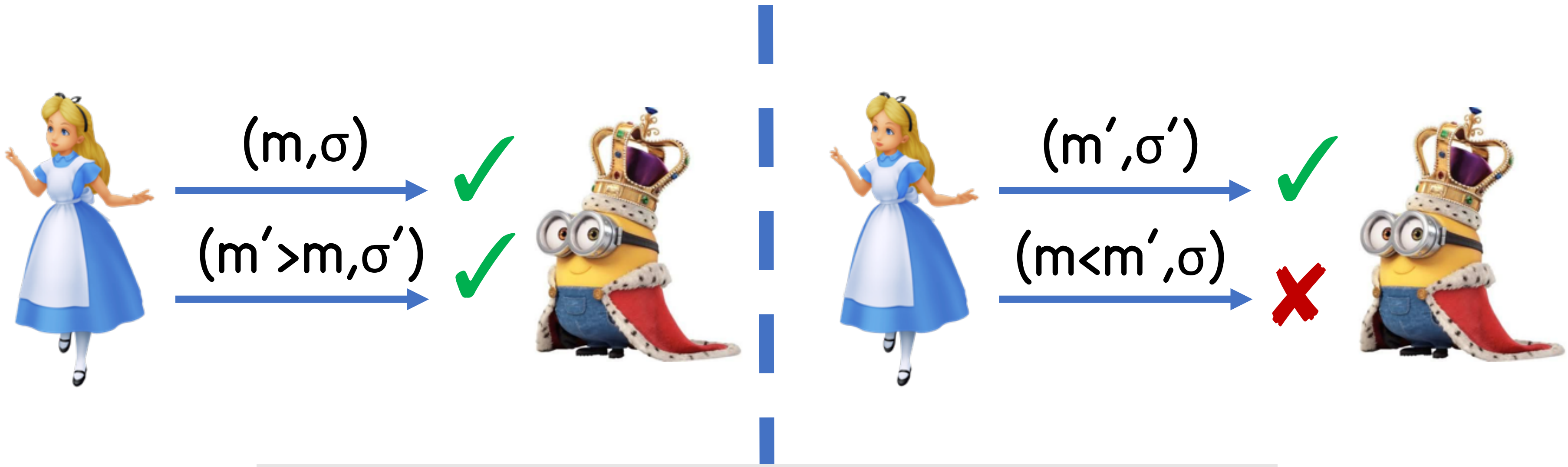
c ← WE.Enc(“r has a sig” , m)

pk

Actually, OSS works directly

OSS Apps: Ordered Signatures

Goal: Only sign messages in increasing order



Same construction as burnable sigs, *Ver* checks message order

OSS Apps: Ordered Signatures

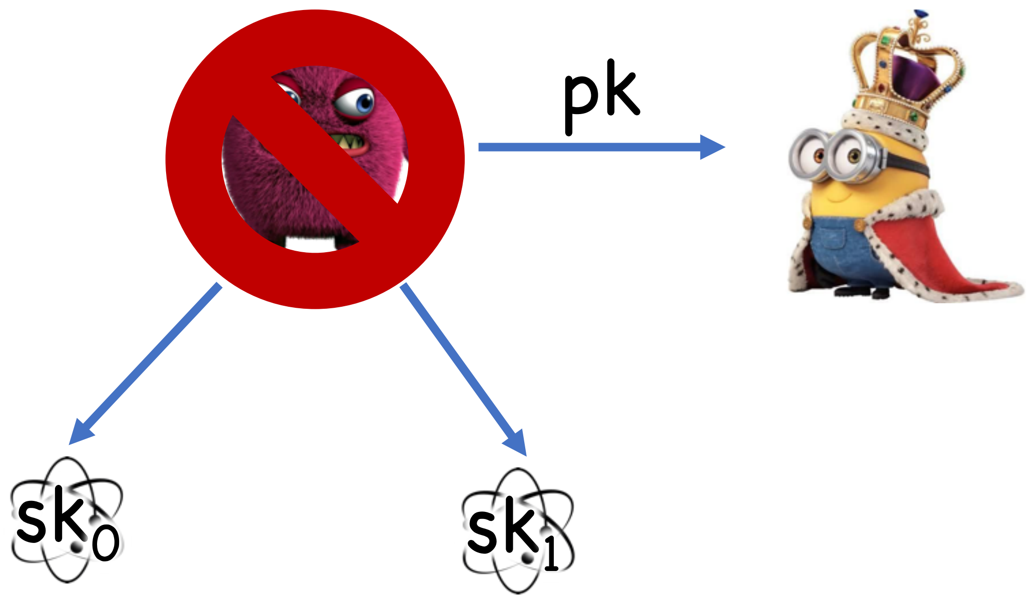
m = (timestamp, document)

((T, *) ,σ) → ✓

((T'<T, D) ,σ) →

If Bob accepts, Alice must have "known" D at time T

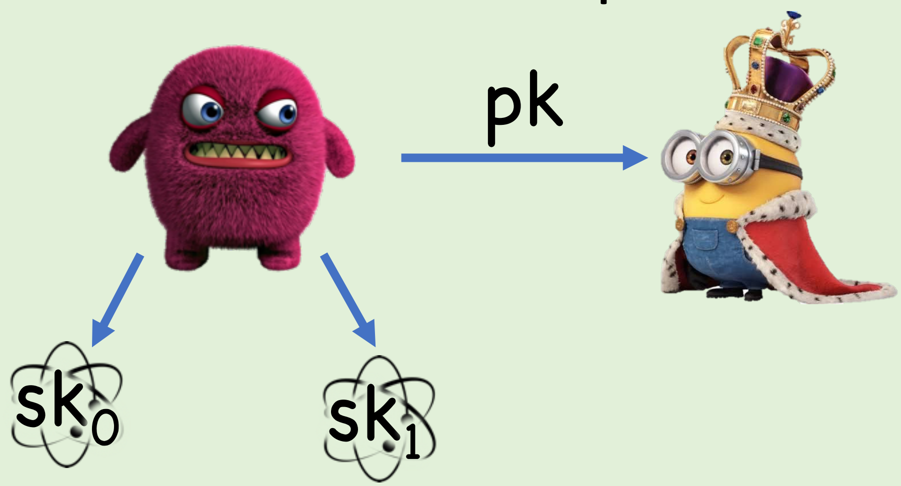
OSS Apps: Single-Signer Signatures



Honest sk can sign any number of messages

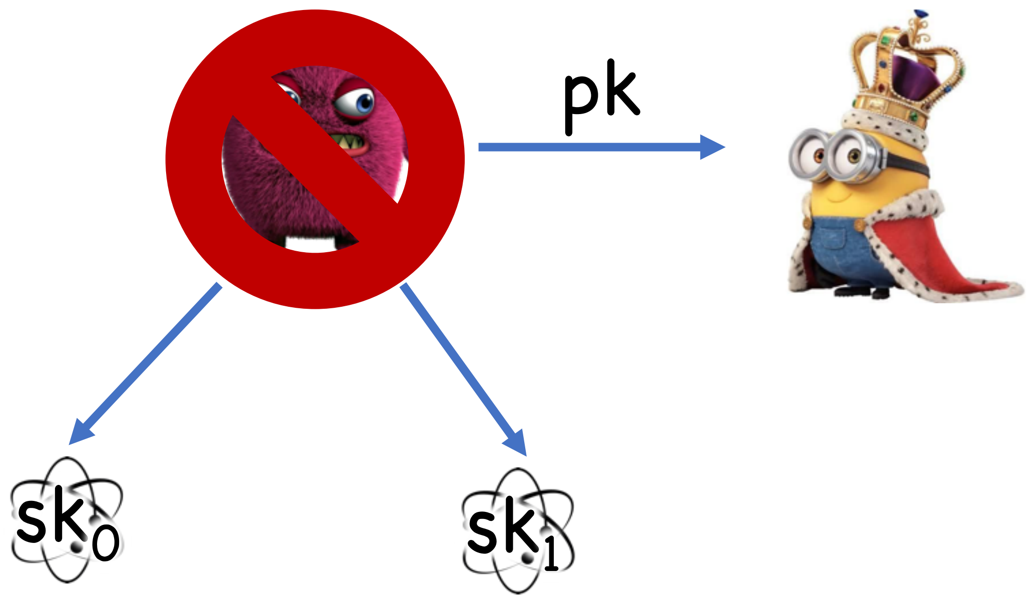
Ordered Sigs → Single-Signer Sigs

Proof: Used timestamped version



Secret keys must respect ordering, so can't sign independently

OSS Apps: Single-Decryptor Encryption



Same as single-signer sigs, except now secret keys are for decrypting

Single-Signer → Single-Decryptor

pk

r ← Message space

c ← WE.Enc(“r has a sig” , m)

Again, OSS works directly

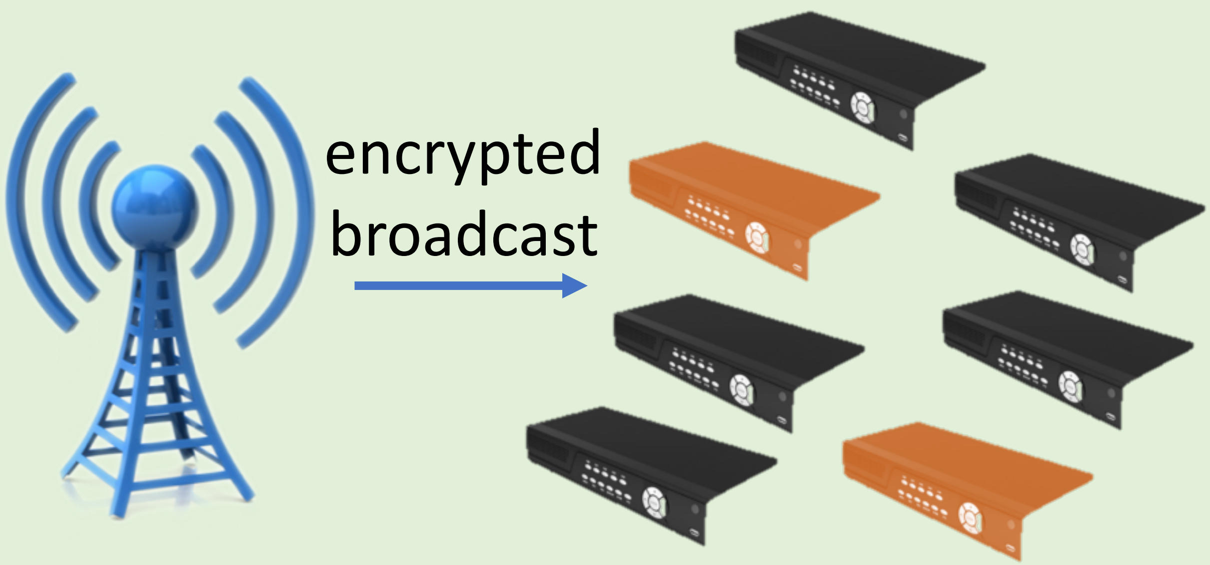
Single-Decryptor App: Traitor Prevention

Recall Traitor Tracing [Chor-Fiat-Naor'94]:

Goal: Given pirate decoder, can identify the traitor(s)

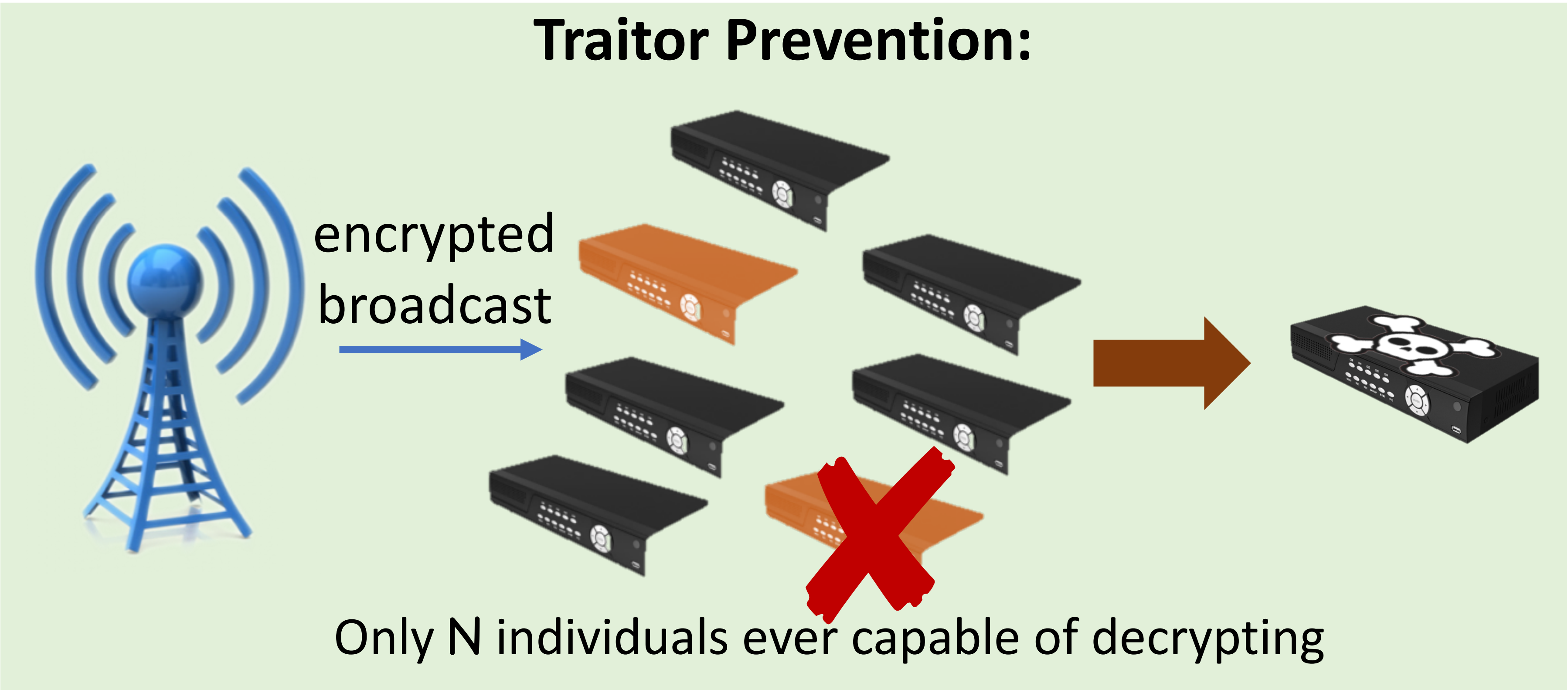
Single-Decryptor App: Traitor Prevention

Traitor Prevention:



Only N individuals ever capable of decrypting

Single-Decryptor App: Traitor Prevention



OSS Apps: Quantum Money*

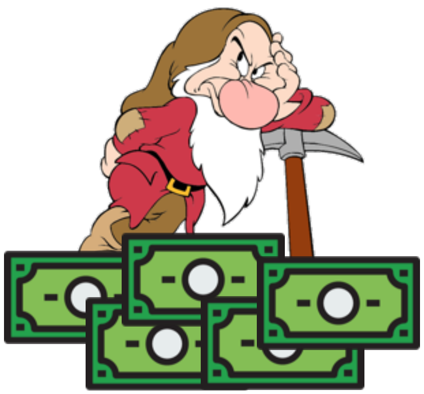
sk_{mint}

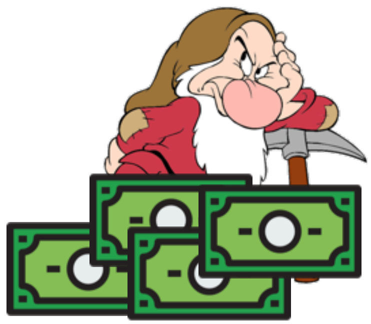
 $= pk,\ \sigma_{pk_{mint}}(pk),\ sk$

Verification: check $\sigma_{pk_{mint}}(pk)$, that sk can sign random message

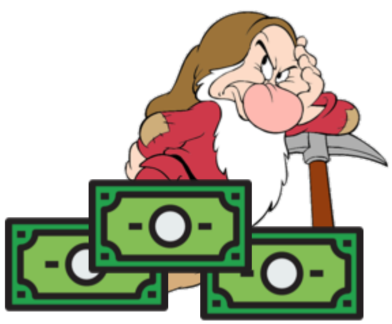
*Technically not "local" quantum crypto; will revisit later

OSS Apps: Cryptocurrency sans Blockchain





Mining entirely local!



OSS → Cryptocurrency w/o Blockchain

Tool: Proofs of Work (PoW)

$\pi \leftarrow PoW(ch, T)$, takes time T

$0/1 \leftarrow Ver(ch, T, \pi)$

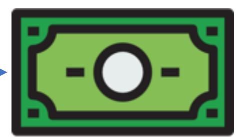
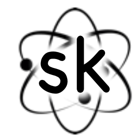
Time << T

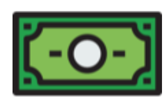
$ch \leftarrow \$$

π

$Ver(ch, T, \pi) = 1$

OSS → Cryptocurrency w/o Blockchain

 →  = pk, PoW(pk,T), 

Verification: check that  can sign random message, PoW valid

OSS Apps: Delay Signatures

Can only sign single message every T minutes

Application:

- Limit rate (quantum) money is created

OSS → Delay Signatures

Tool: Proofs of Sequential Work (PoSW)

$\pi \leftarrow PoSW(ch,T)$, takes sequential time T

$0/1 \leftarrow Ver(ch,T,\pi)$

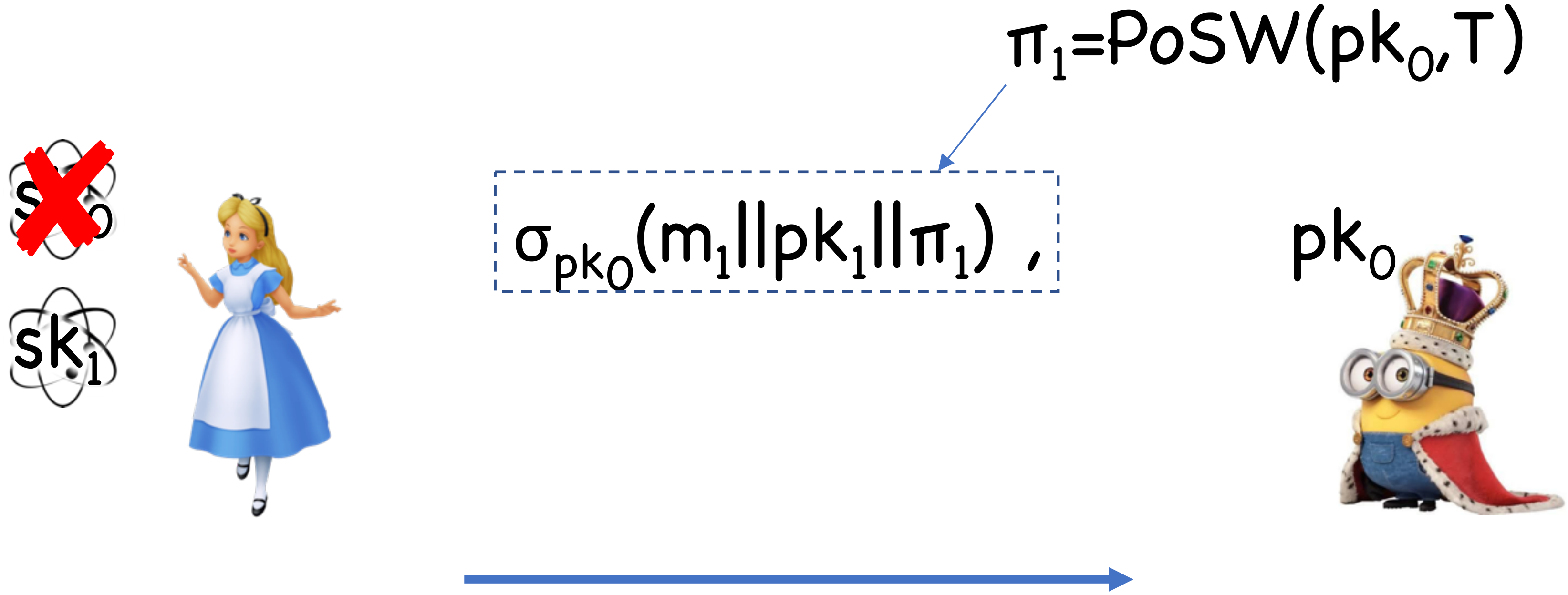
Sequential time << T

$ch \leftarrow \$$

π

$Ver(ch,T,\pi)=1$

OSS → Delay Signatures



OSS → Delay Signatures

$\pi_2 = \mathrm{PoSW}(\sigma_{pk_0}, T)$

$\sigma_{pk_0}(m_1 || pk_1 || \pi_1)$,

$\sigma_{pk_1}(m_2 || pk_2 || \pi_2)$,

pk_0

OSS Apps: Delay Decryption

Can only decrypt single ciphertext every T minutes

Application:

- Tiered broadcast subscriptions

Delay Sigs → Delay Decryption

r ← Message space

c ← WE.Enc(“r has a sig” , m)

pk

Part 2: Classically Sending Quantum States

Quantum States over Classical Channels?

Rejected Solution:
Send classical description of state

What if don't know classical description?

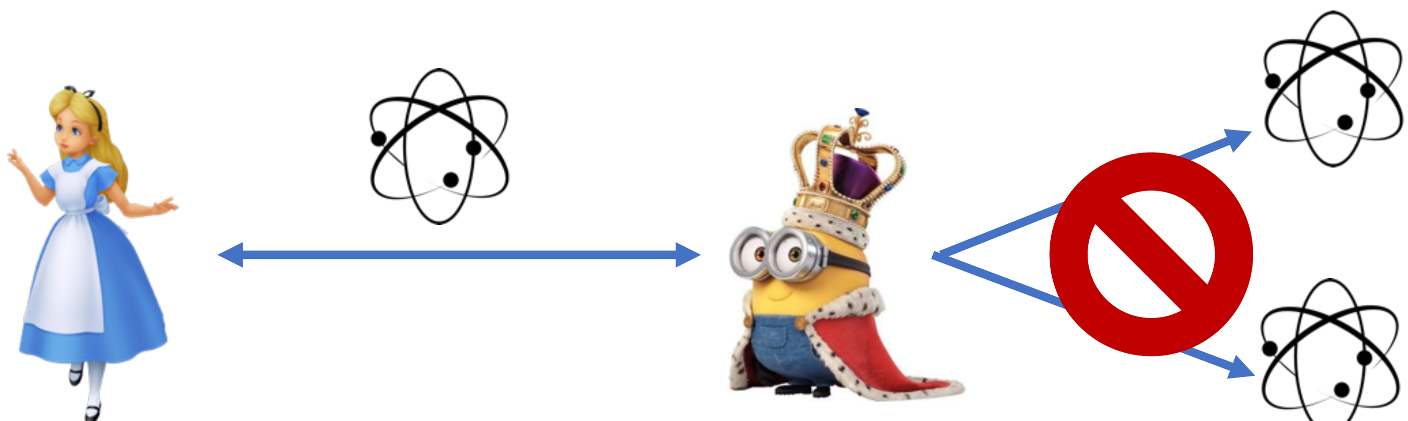
Rejected Solution:
Use quantum teleportation

Requires quantum entanglement

No In General: Could use to create entanglement via classical channel

Quantum States over Classical Channels?

Q2': Can any *unclonable* state be sent over a classical channel?



Q2 Rephrased

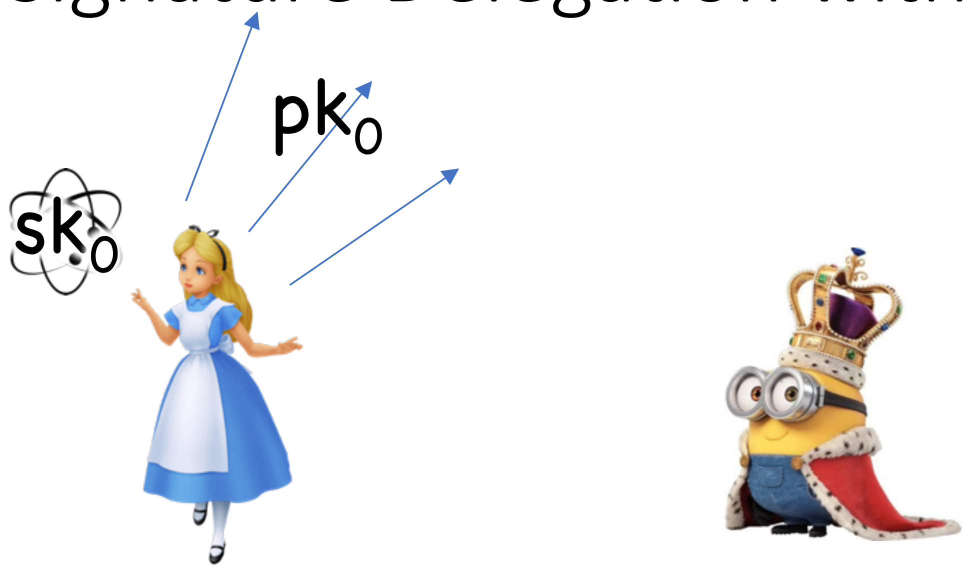
Q2’: Can any *unclonable* state be sent over a classical channel?

No, if single message from Alice to Bob

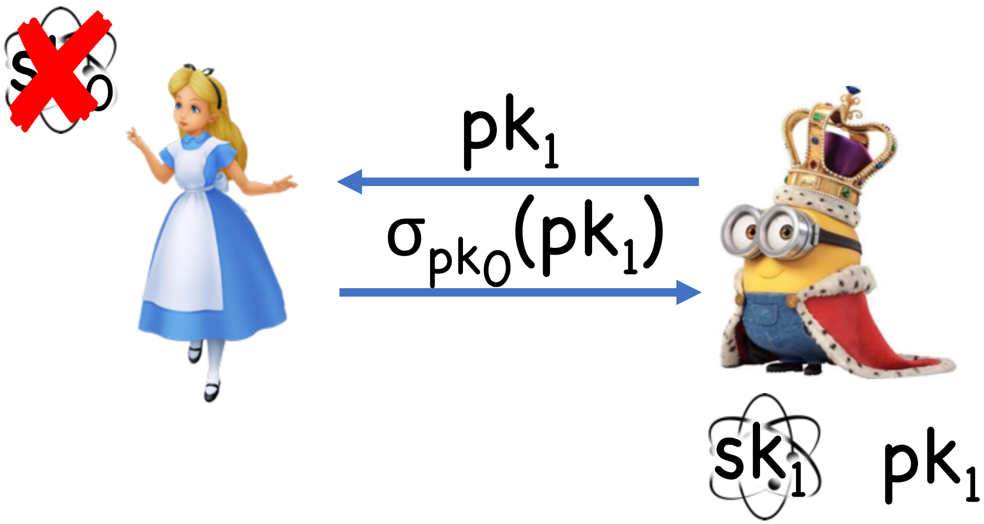
No, if computationally unbounded

What if interaction + computational assumptions?

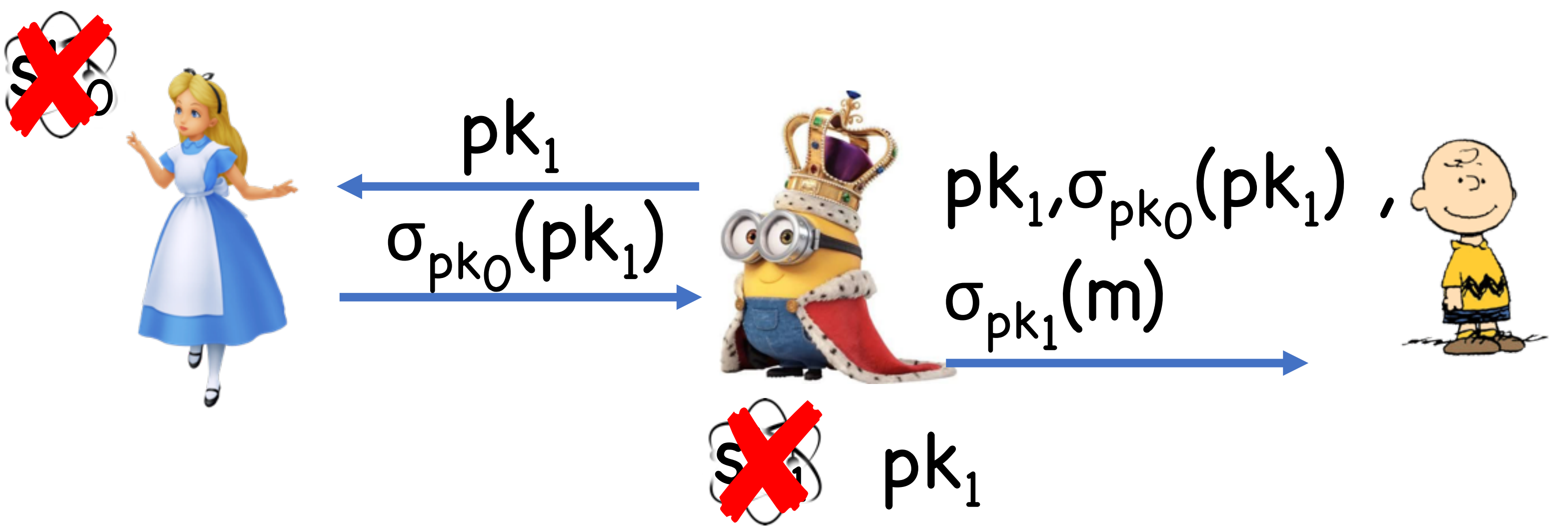
Signature Delegation with OSS



Signature Delegation with OSS



Signature Delegation with OSS



Alice effectively sent her unclonable state to Bob over classical channel

Signature Delegation

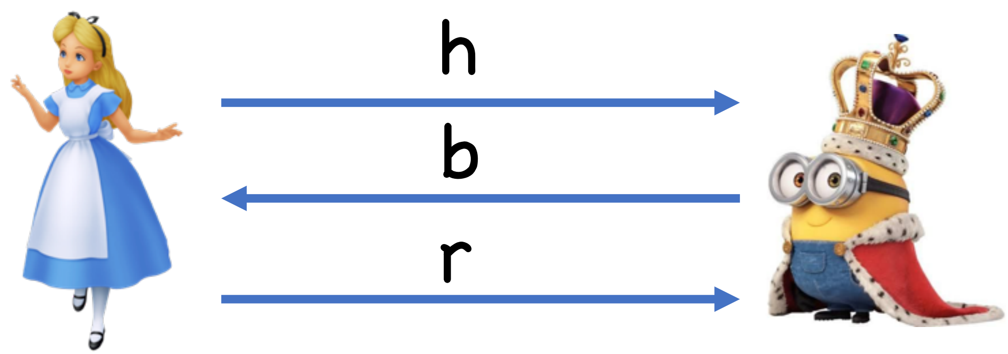
Using recursively composable **zk**-SNARKS, received state is computationally indistinguishable from original

Can apply to all of our schemes, to send quantum keys/money over classical channels

Part 3: Constructing OSS

Unequivocal Hash Functions

Closely related to concepts from [Ambainis-Rosmanis-Unruh'14,Unruh'16]



$H(b,r) = h?$

Classically:
col. resistance → unequiv. hash (rewinding)

Quantumly: maybe not

Equivocal Hash Functions

Equivocal Hash = Col. Resistance + ! Unequivocal

Easy Thm: Equivocal Hash → OSS

[Ambainis-Rosmanis-Unruh’14,Unruh’16]:
Construction relative to *quantum* oracle

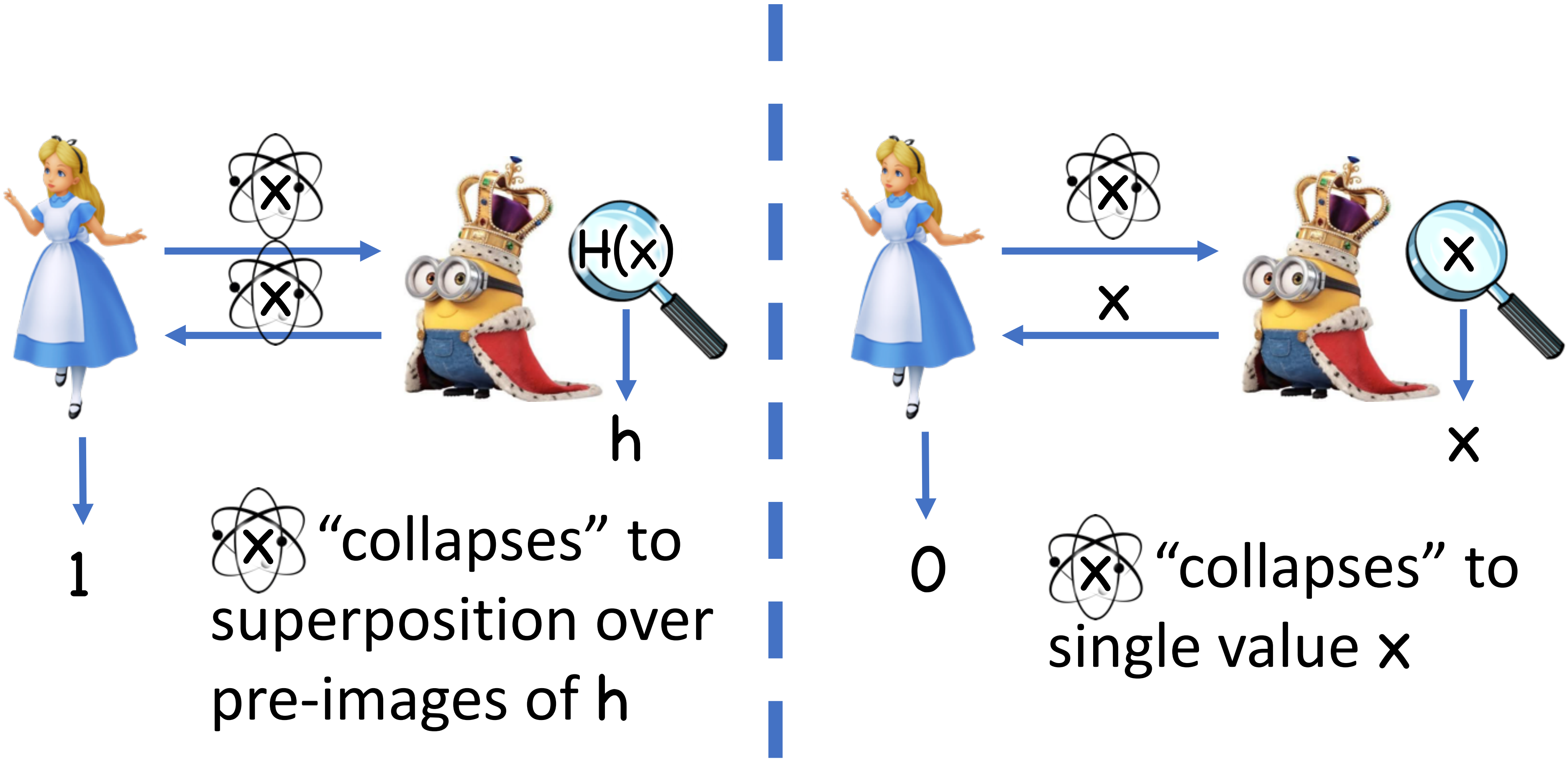
But, no clear idea how to instantiate

Our Result

Thm: Equivocal hash relative to *classical* oracle

Can heuristically instantiate w/ iO

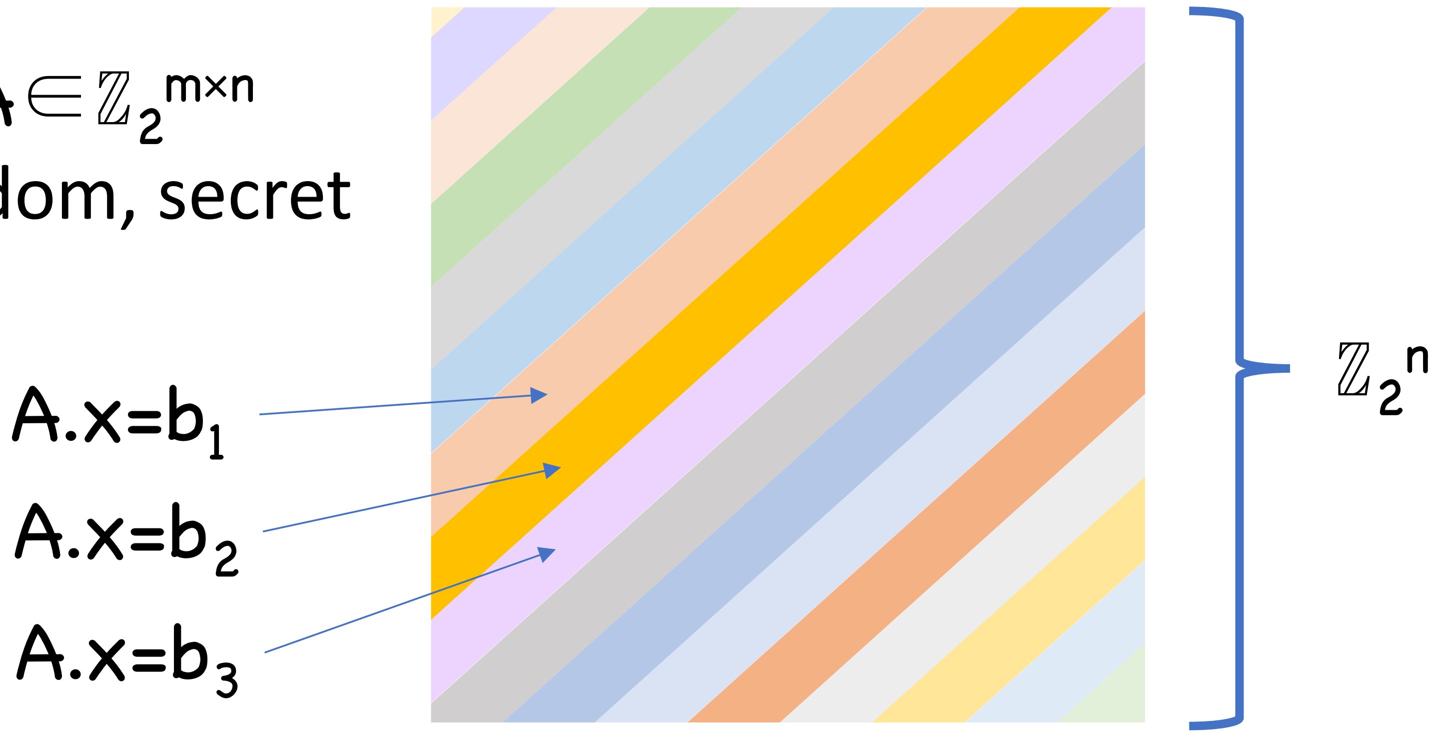
Simpler Goal: Non-Collapsing Hash



A First (Broken) Attempt

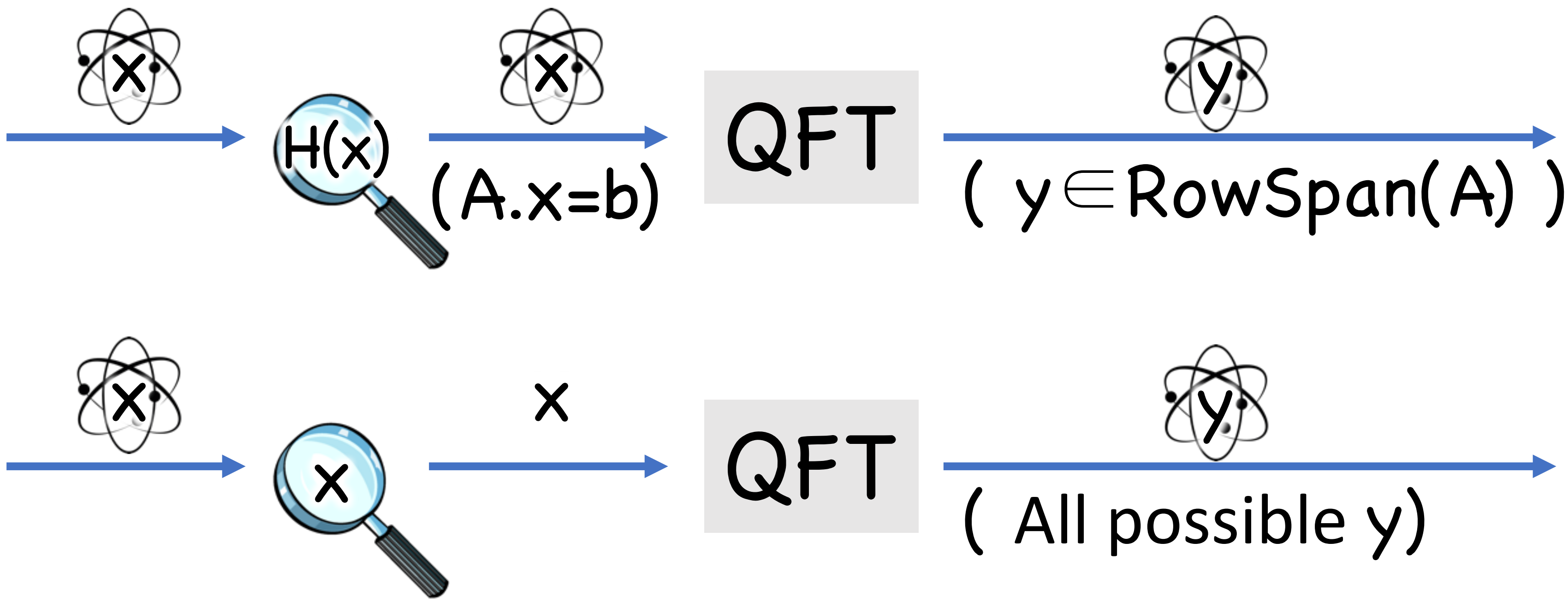
$A \in \mathbb{Z}_2^{m \times n}$

random, secret



H: assign each "slice" a random output

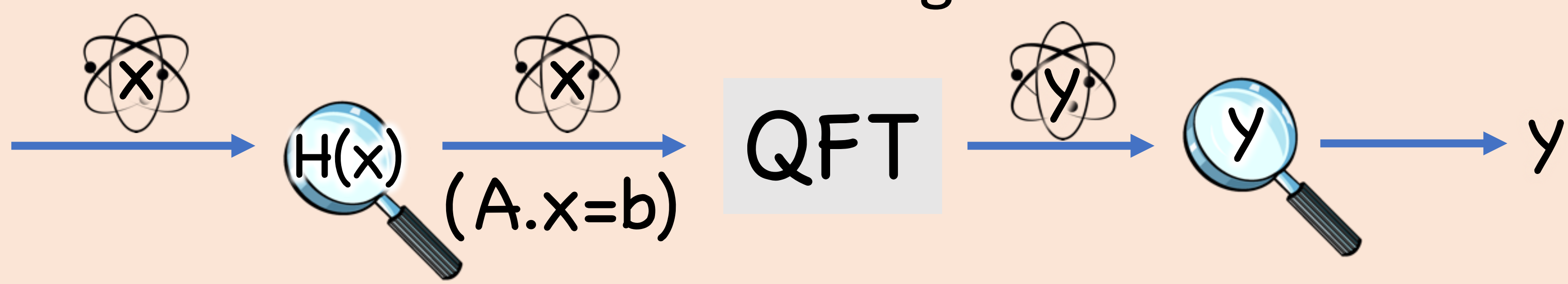
A First (Broken) Attempt



Inc. oracle O which checks for membership in RowSpan(A)

Problem: Periodic → Not Collision Resistant!

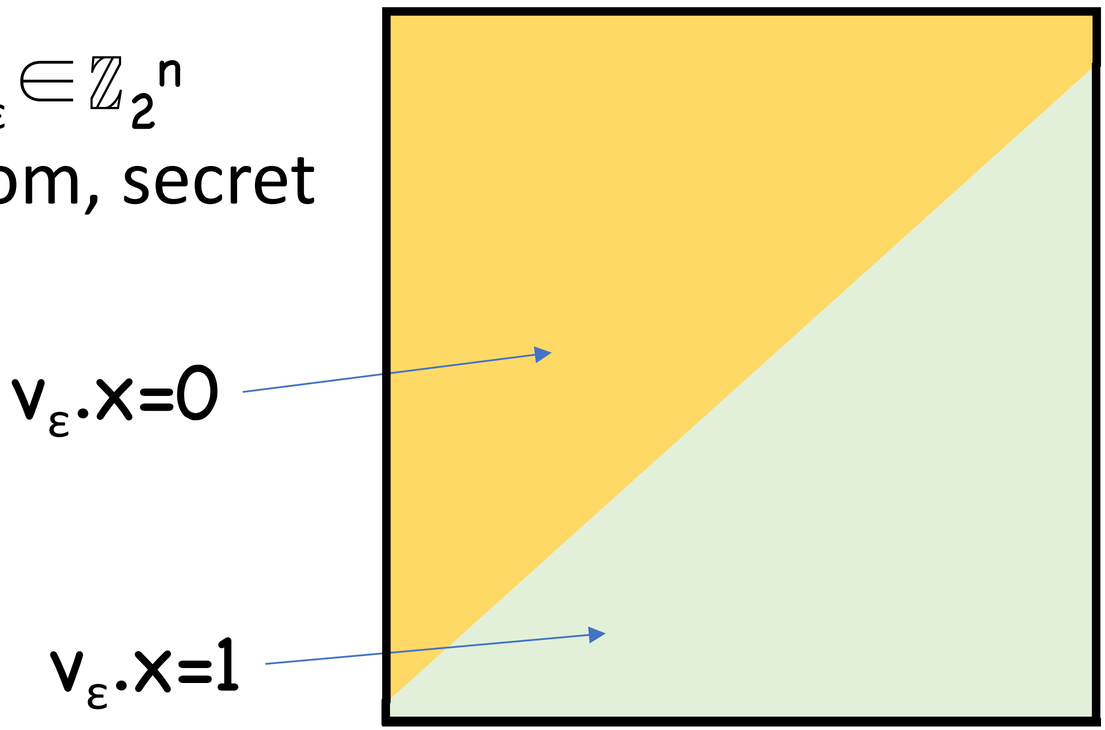
Simon's Alg:



Recall: $y \in \text{RowSpan}(A)$ ➡ Repeat several times: reconstruct A

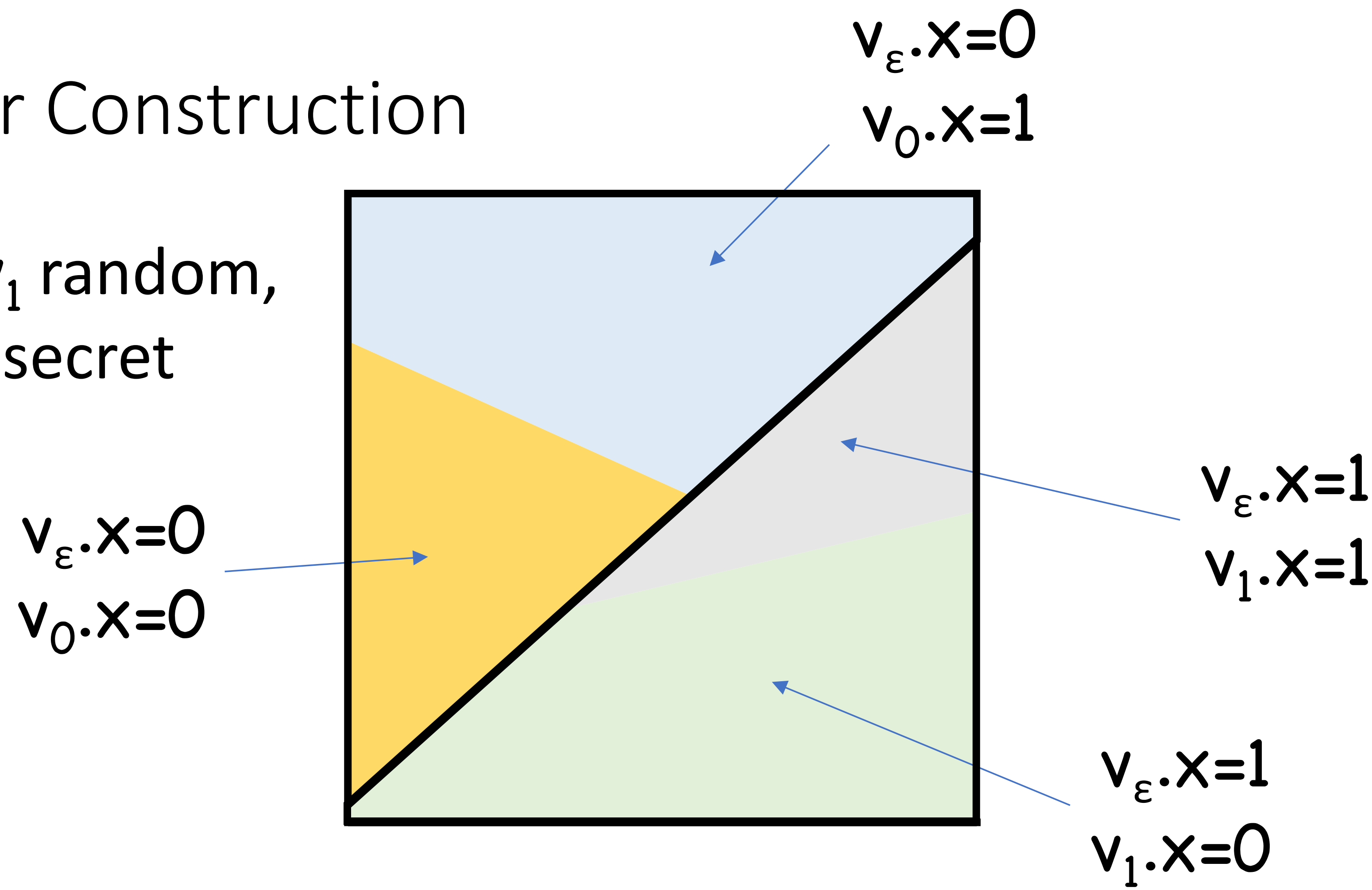
Our Construction

$v_\varepsilon \in \mathbb{Z}_2{}^n$
random, secret



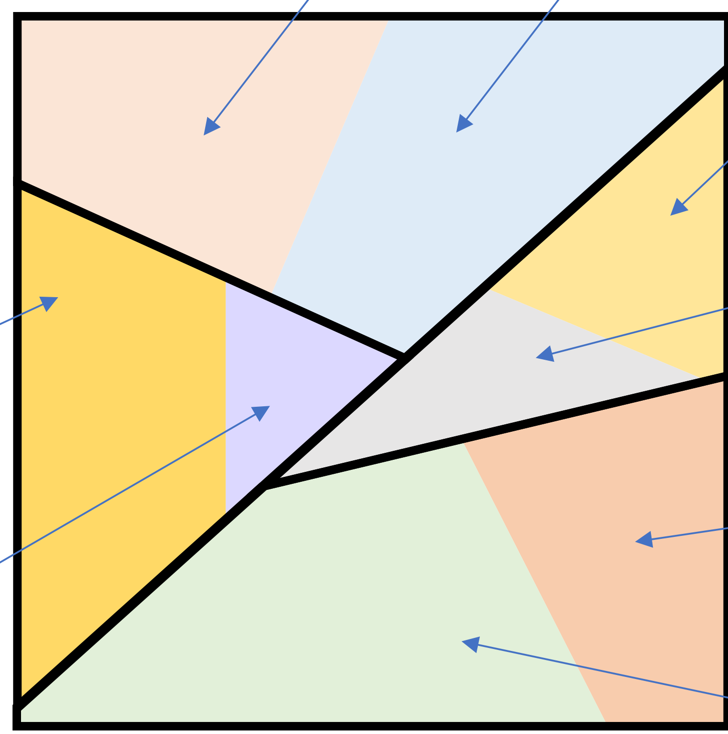
Our Construction

v_0, v_1 random, secret



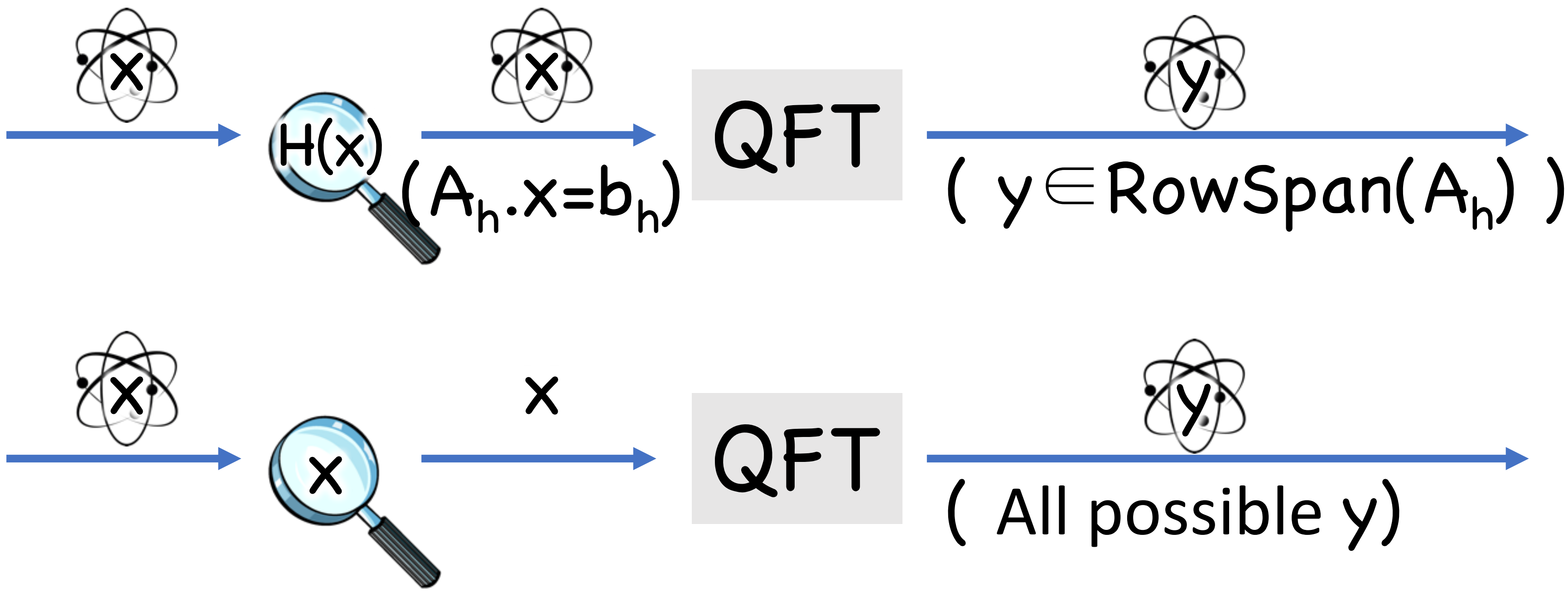
Our Construction

$v_{00}, v_{01}, v_{10}, v_{11}$
random, secret



•••

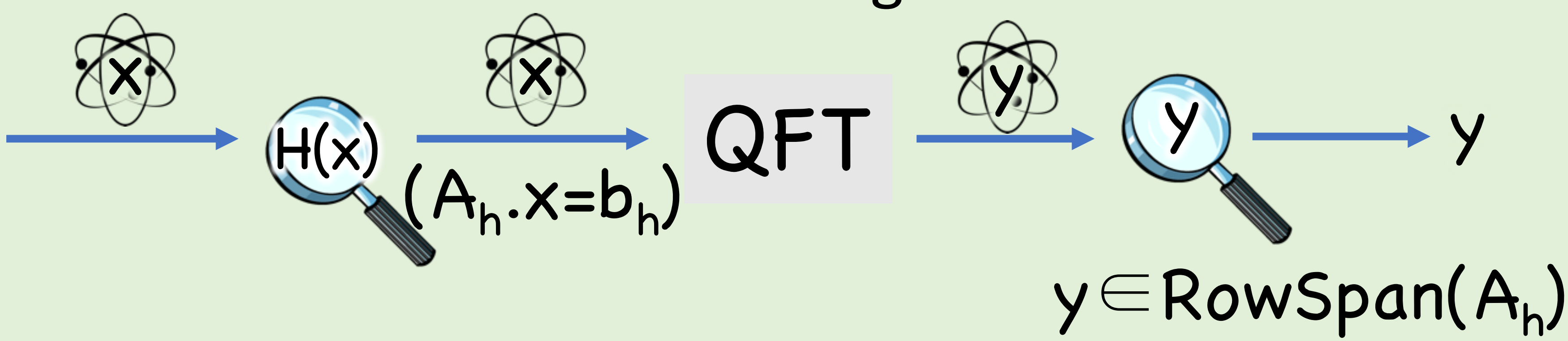
Our Construction



Inc. oracle O which checks for membership in $RowSpan(A_h)$

Simon's Algorithm?

Simon's Alg:



Repeat ➡ Different h ➡ Different A_h

➡ Never able to reconstruct A_h ✗

Our Construction

Thm: If H,*O* given as oracles, then collision resistant

With some extra work, can also equivocate

Future Directions?

Better assumptions?

- Even iO + LWE + LPN + Isogenies + …?

More apps?

- Fancier crypto (e.g. functional enc)?
- Classically send copy-protected programs?

Thanks!