Quantum Information for Developers, 11-14 September 2018

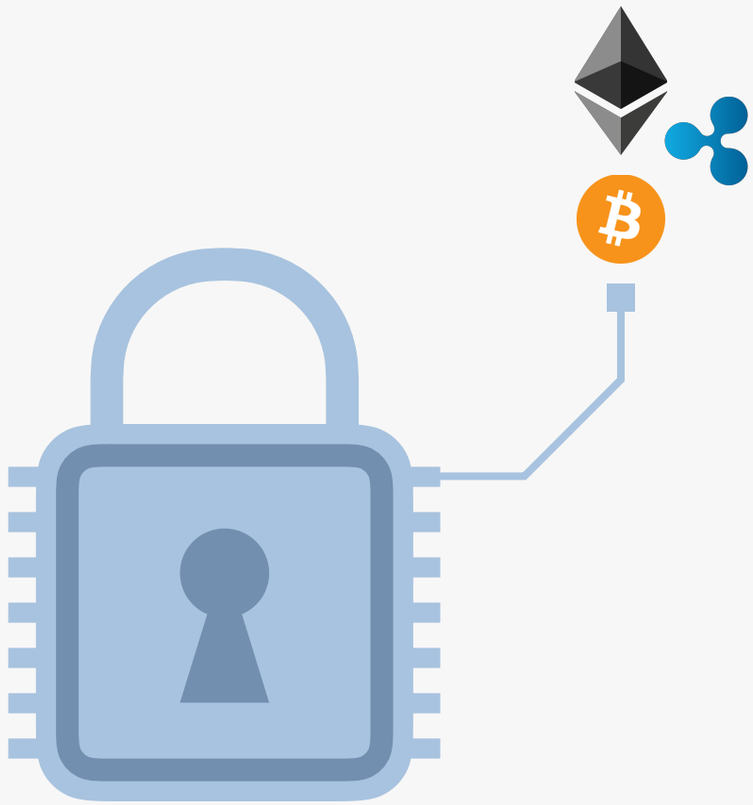
Quantum Cryptography

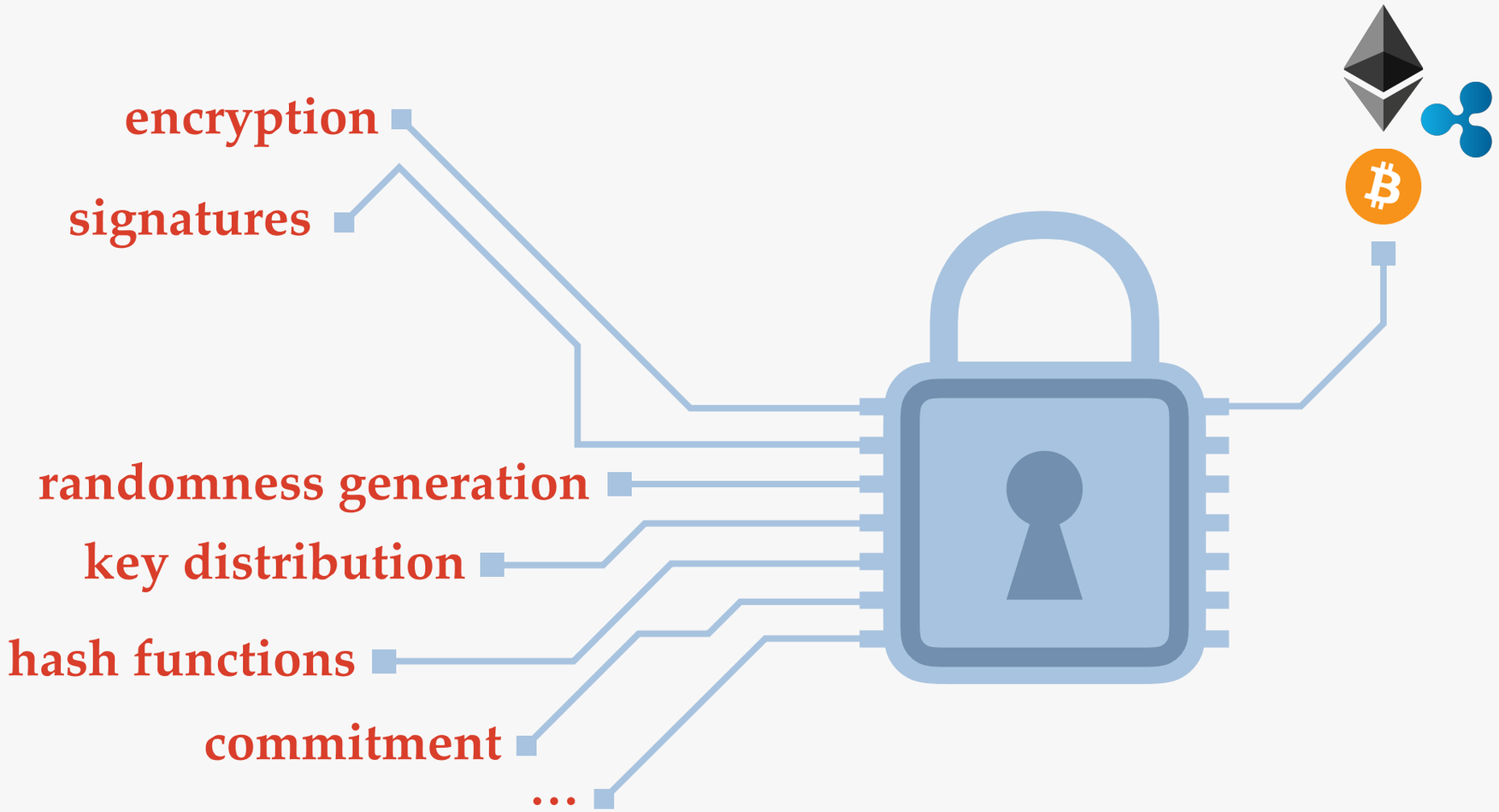
Advanced track

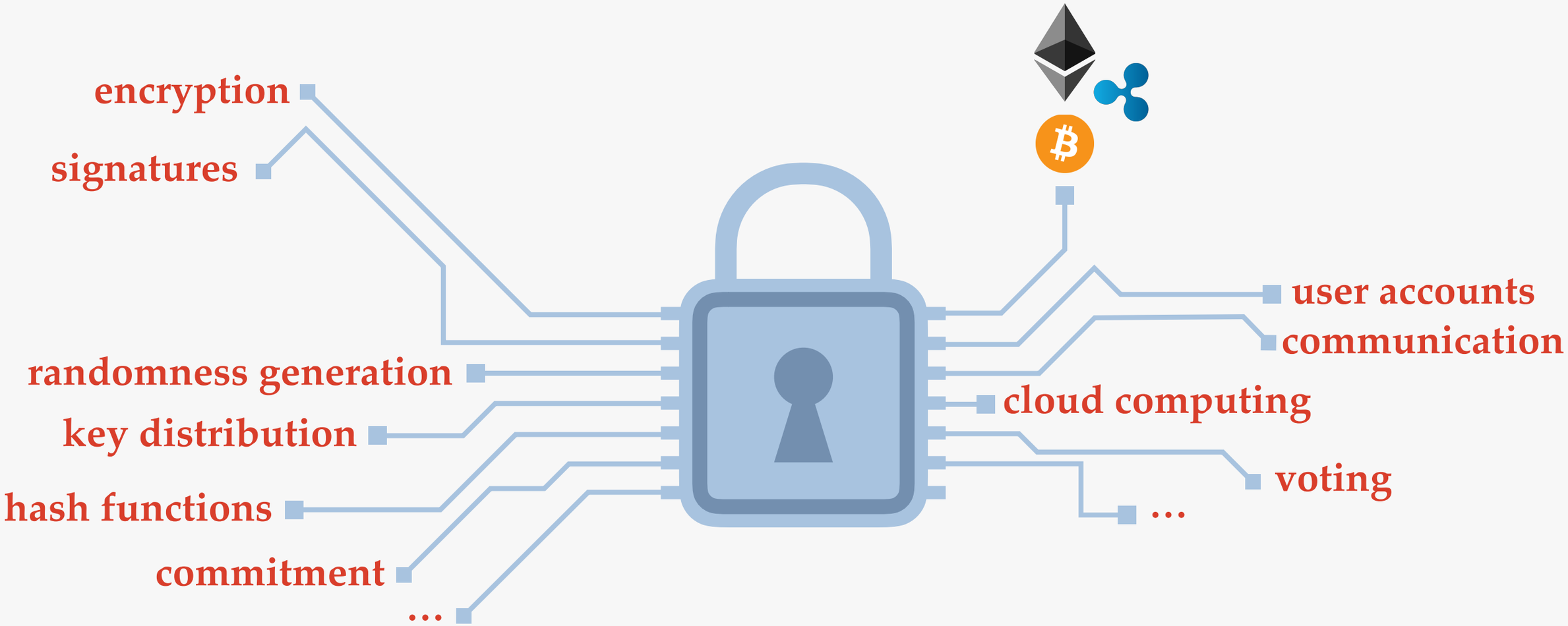
Yfke Dulek

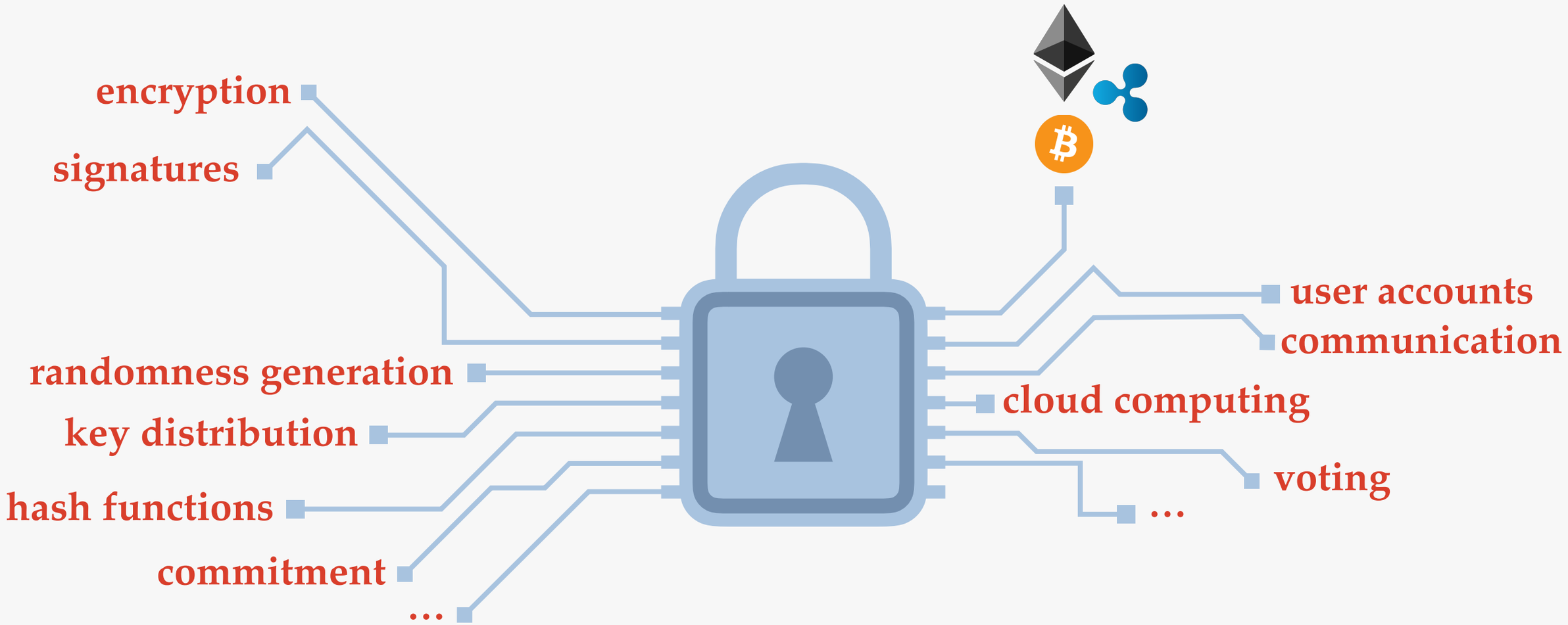
QuSoft Research Center for Quantum Software
CWI

www.qusoft.org
www.cwi.nl

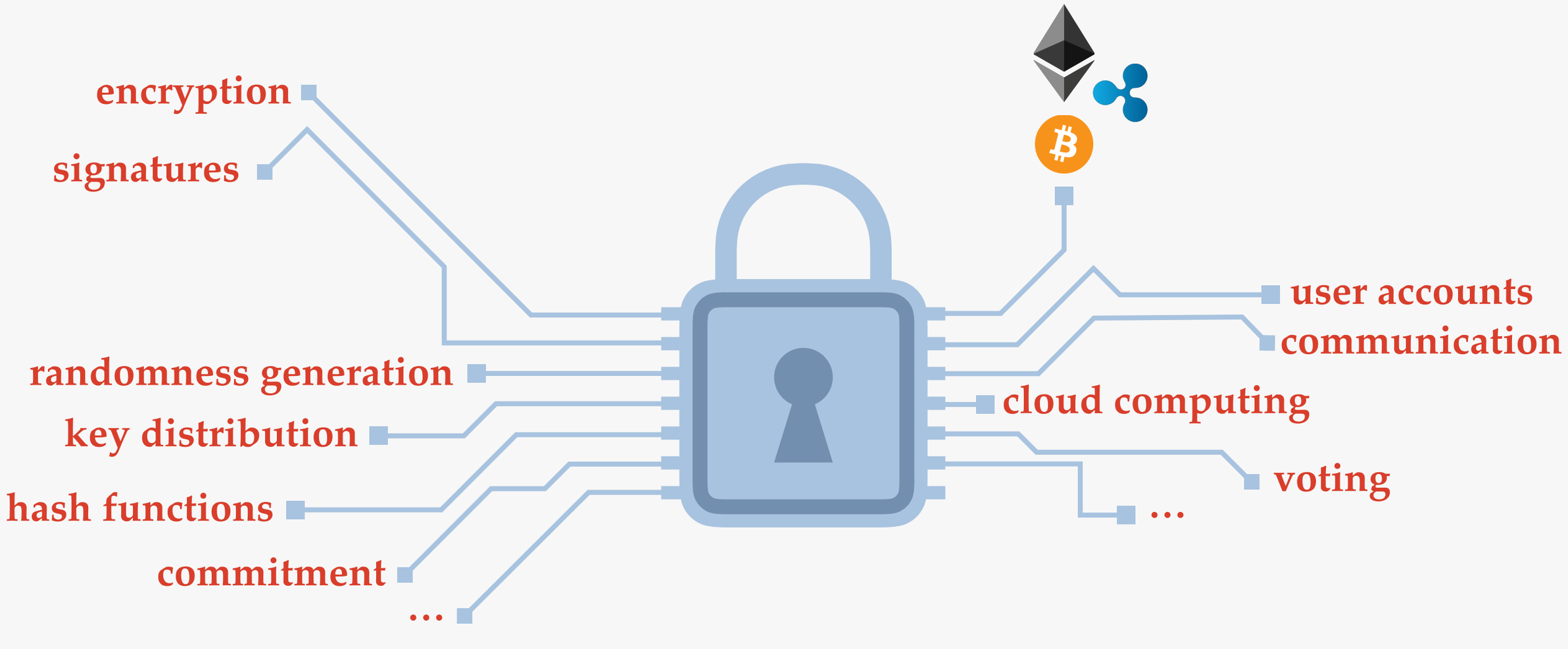



encryption
signatures
randomness generation
key distribution
hash functions
commitment
...

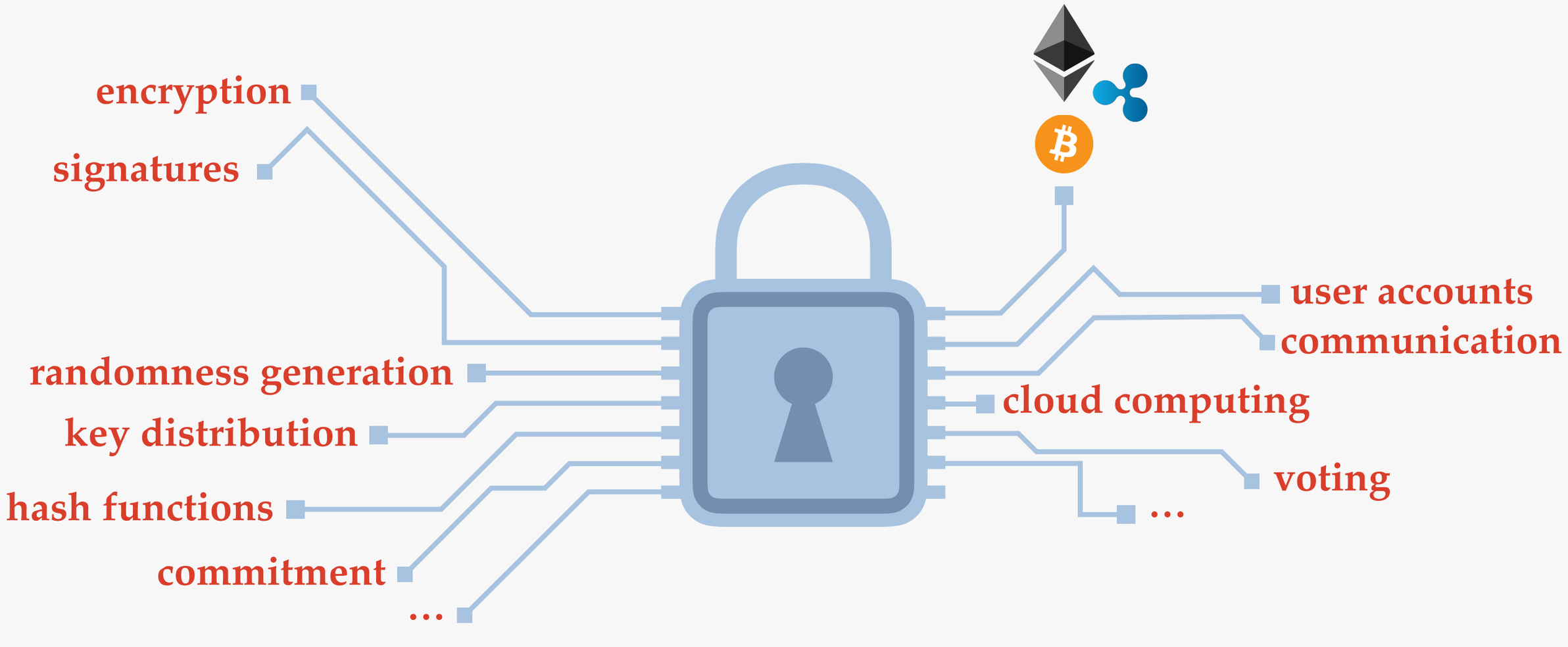

encryption
signatures
randomness generation
key distribution
hash functions
commitment
...
user accounts
communication
cloud computing
voting
...



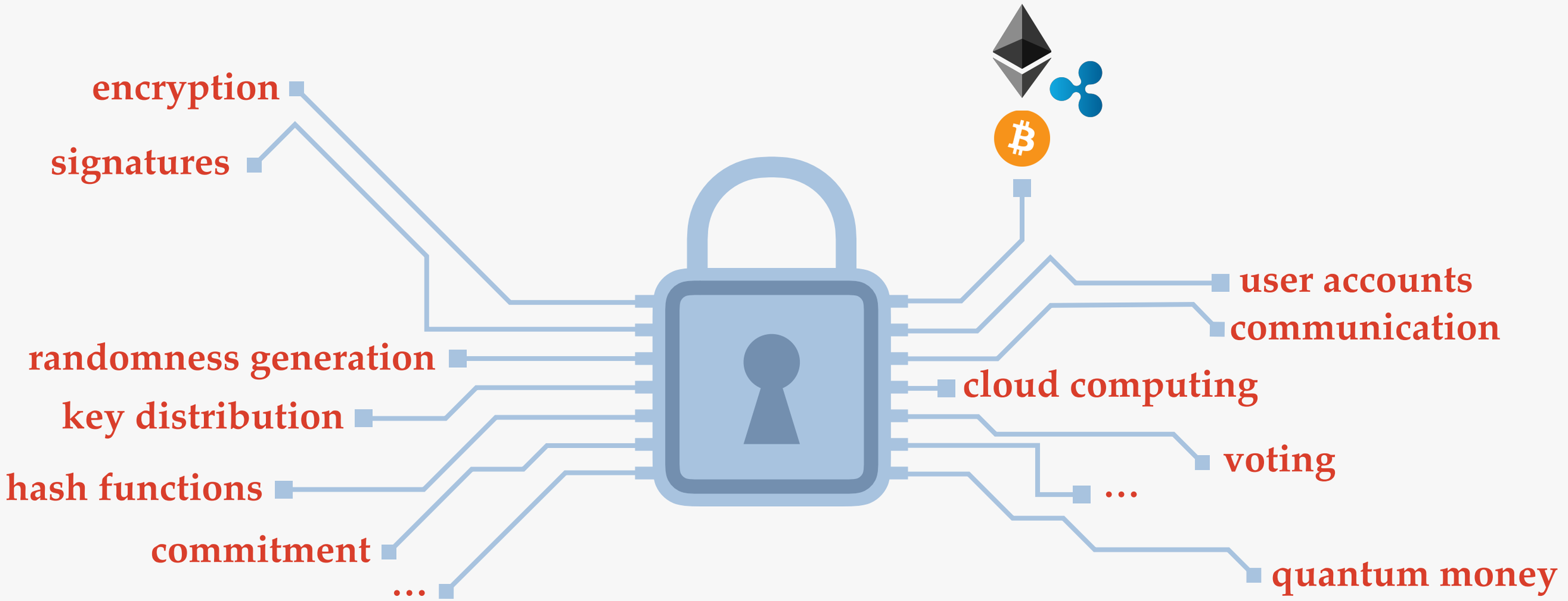
- Can quantum computing **break** the primitives?



- Can quantum computing **break** the primitives?
- Can quantum computing **improve** the primitives?



- Can quantum computing **break** the primitives?
- Can quantum computing **improve** the primitives?
- What **quantum applications** are desirable?

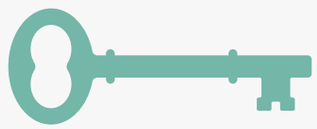


- Can quantum computing **break** the primitives?
- Can quantum computing **improve** the primitives?
- What **quantum applications** are desirable?

Outline

- Primitives:
  I. Encryption (privacy)
  II. Authentication
  III. Secret sharing
- Application:
  I. Cloud computing

Primitive I: encryption



Encryption: definition

Encryption: definition

Alice

Bob

Encryption: definition

Alice

Bob

$m \in \{0,1\}^n$

Encryption: definition

Alice

Bob

$m \in \{0,1\}^n$

k

$k \in \{0,1\}^\ell$

Encryption: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

$c = \mathrm{Enc}(m,k) \in \{0,1\}^d$

Bob

k

Encryption: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

$c = \mathrm{Enc}(m,k) \in \{0,1\}^d$

Bob

k

$m' = \mathrm{Dec}(c,k)$

Encryption: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

$c = \mathrm{Enc}(m,k) \in \{0,1\}^d$

Bob

k

$m' = \mathrm{Dec}(c,k)$

- correct: $m' = m$

Encryption: definition

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

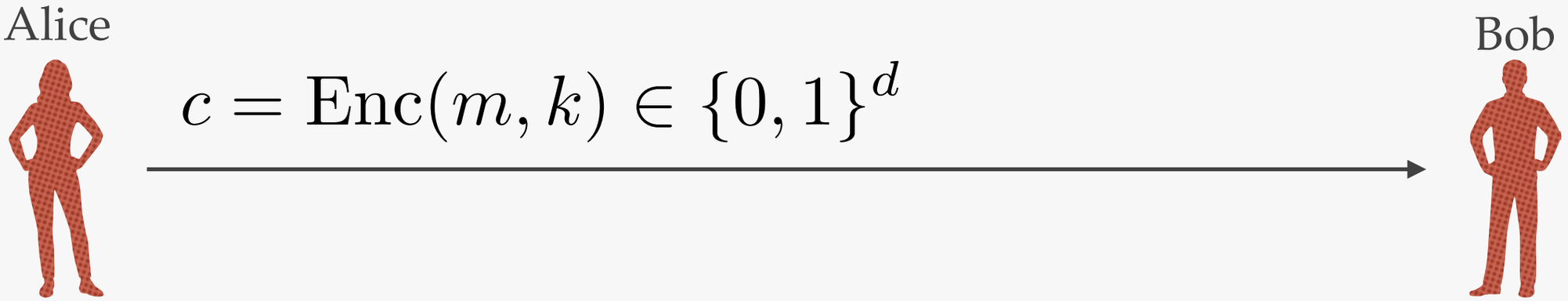
$c = \mathrm{Enc}(m,k) \in \{0,1\}^d$

k

$m' = \mathrm{Dec}(c,k)$

- correct: $m' = m$
- efficient: Enc and Dec are polynomial-time

Encryption: definition



- correct: $m' = m$
- efficient: Enc and Dec are polynomial-time
- private: c hides all the information about m

Example: one-time pad (also known as Vernam cipher)

Example: one-time pad (also known as Vernam cipher)

$$m = \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0$$

Example: one-time pad (also known as Vernam cipher)

$$m = \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0$$

$$k = \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0$$

Example: one-time pad (also known as Vernam cipher)

$$m = \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0$$

$$k = \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0$$

- Key k is selected uniformly random (independent of m)

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
\qquad \oplus \text{ (ADDITION MODULO 2)}
$$

- Key k is selected uniformly random (independent of m)

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}
$$

$\oplus$ (ADDITION MODULO 2)

- Key k is selected uniformly random (independent of m)

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}
$$

$\oplus$ (ADDITION MODULO 2)

- Key k is selected uniformly random (independent of m)
- Dec is the same as Enc

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0
\end{array}
$$

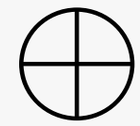
(ADDITION MODULO 2)

- Key k is selected uniformly random (independent of m)
- Dec is the same as Enc

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

(ADDITION MODULO 2)



- Key k is selected uniformly random (independent of m)
- Dec is the same as Enc

Example: one-time pad (also known as Vernam cipher)

$$
\begin{array}{lcccccccc}
m = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
c = & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
k = & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
\hline
m' = & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
$$

$\oplus$ (ADDITION MODULO 2)

- Key k is selected uniformly random (independent of m)
- Dec is the same as Enc

More formally: ‘privacy’

More formally: ‘privacy’

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

More formally: ‘privacy’

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

- Semantic definition of privacy:

More formally: ‘privacy’

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

- Semantic definition of privacy:

adversary

More formally: 'privacy'

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

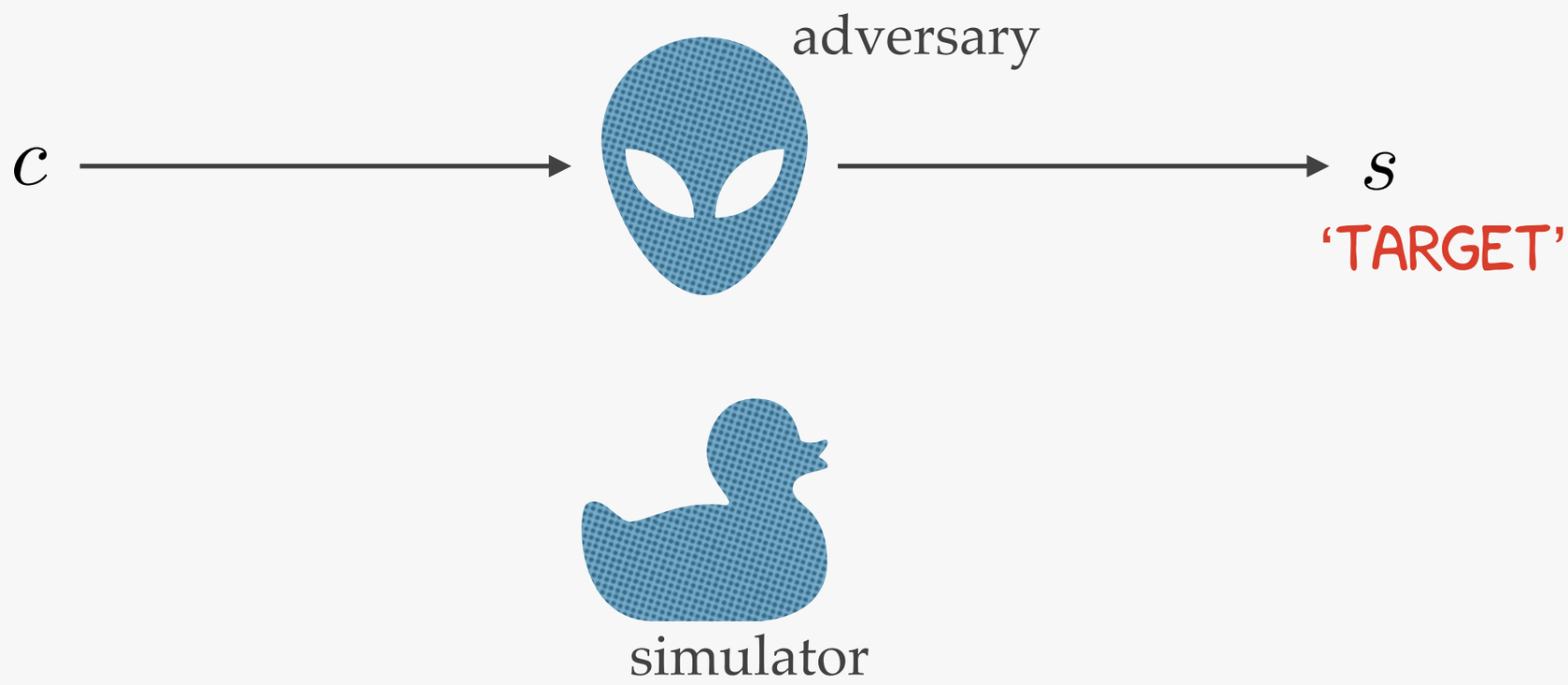
- Semantic definition of privacy:

More formally: 'privacy'

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

- Semantic definition of privacy:

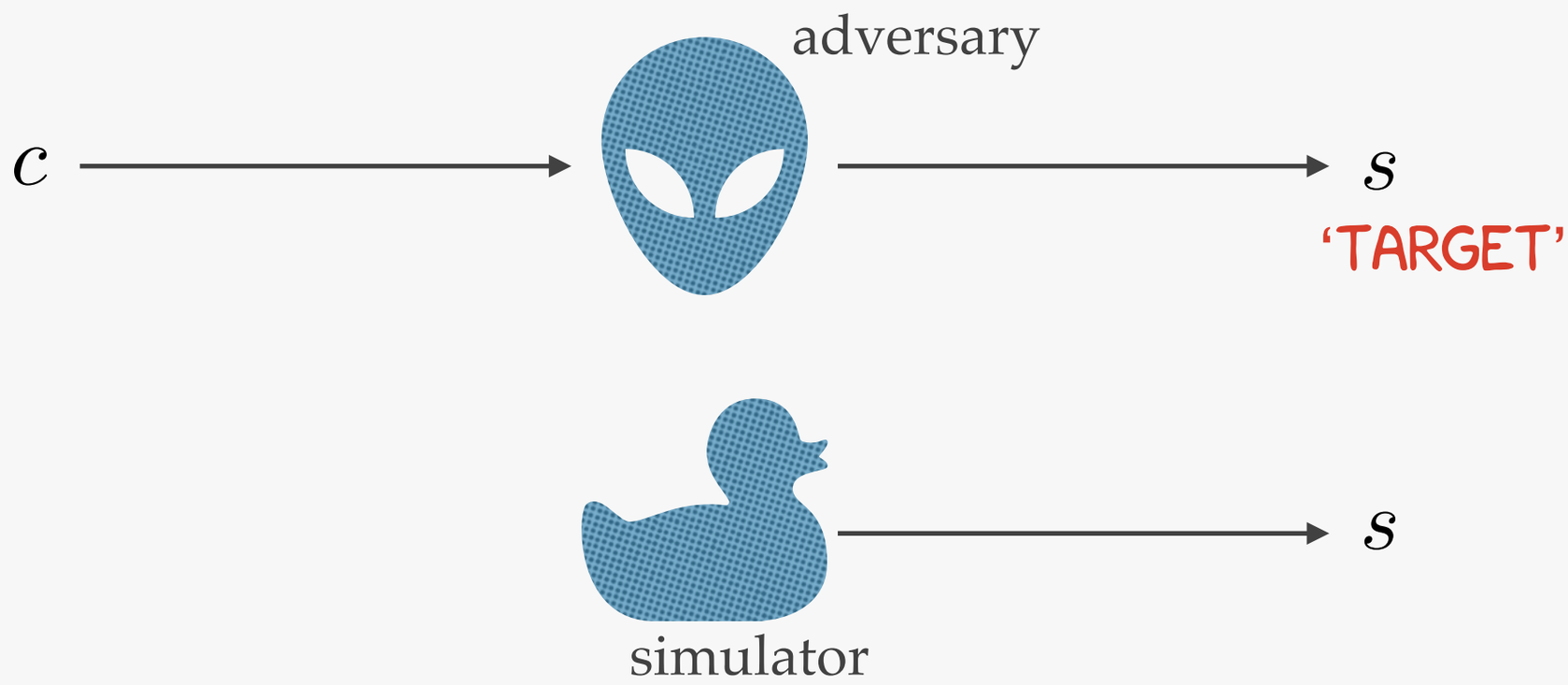


More formally: 'privacy'

- Probability-theoretic definition of privacy:

$$\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$$

- Semantic definition of privacy:



One-time pad is private (proof)

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

  - $\forall k \Pr[K = k] = 2^{-\ell}$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

  - $\forall k \Pr[K = k] = 2^{-\ell}$

    $\forall k \Pr[K = k | M = m] = 2^{-\ell}$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

  - $\forall k \Pr[K = k] = 2^{-\ell}$
  - $\forall k \Pr[K = k | M = m] = 2^{-\ell}$
  - $\exists! k : m \oplus k = c$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

  - $\forall k \Pr[K = k] = 2^{-\ell}$
  - $\forall k \Pr[K = k | M = m] = 2^{-\ell}$
  - $\exists! k : m \oplus k = c$

  $\Big\}$ $\Pr[C = c | M = m] = 2^{-\ell}$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

  - $\forall k \Pr[K = k] = 2^{-\ell}$

    $\left.\begin{array}{l} \forall k \Pr[K = k | M = m] = 2^{-\ell} \\ \exists! k : m \oplus k = c \end{array}\right\} \Pr[C = c | M = m] = 2^{-\ell}$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

- $\forall k \Pr[K = k] = 2^{-\ell}$

$$\left.\begin{array}{l} \forall k \Pr[K = k | M = m] = 2^{-\ell} \\ \exists! k : m \oplus k = c \end{array}\right\} \Pr[C = c | M = m] = 2^{-\ell}$$

- $\Pr[C = c] = \sum_{m'} \Pr[M = m'] \cdot \Pr[C = c | M = m'] =$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

- $\forall k \Pr[K = k] = 2^{-\ell}$

$$\left.\begin{array}{l} \forall k \Pr[K = k | M = m] = 2^{-\ell} \\ \exists! k : m \oplus k = c \end{array}\right\} \Pr[C = c | M = m] = 2^{-\ell}$$

- $\Pr[C = c] = \sum_{m'} \Pr[M = m'] \cdot \Pr[C = c | M = m'] =$

$$\sum_{m'} \Pr[M = m'] \cdot 2^{-\ell} =$$

One-time pad is private (proof)

- Claim: $\forall m \forall c : \Pr[M = m \mid C = c] = \Pr[M = m]$
- Proof: Fix m and c. Using Bayes' theorem:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]}$$

- $\forall k \Pr[K = k] = 2^{-\ell}$

$$\left.\begin{array}{l} \forall k \Pr[K = k | M = m] = 2^{-\ell} \\ \exists! k : m \oplus k = c \end{array}\right\} \Pr[C = c | M = m] = 2^{-\ell}$$

- $\Pr[C = c] = \sum_{m'} \Pr[M = m'] \cdot \Pr[C = c | M = m'] =$

$$\sum_{m'} \Pr[M = m'] \cdot 2^{-\ell} = 2^{-\ell}$$

One-time pad: drawbacks

One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)

One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once:

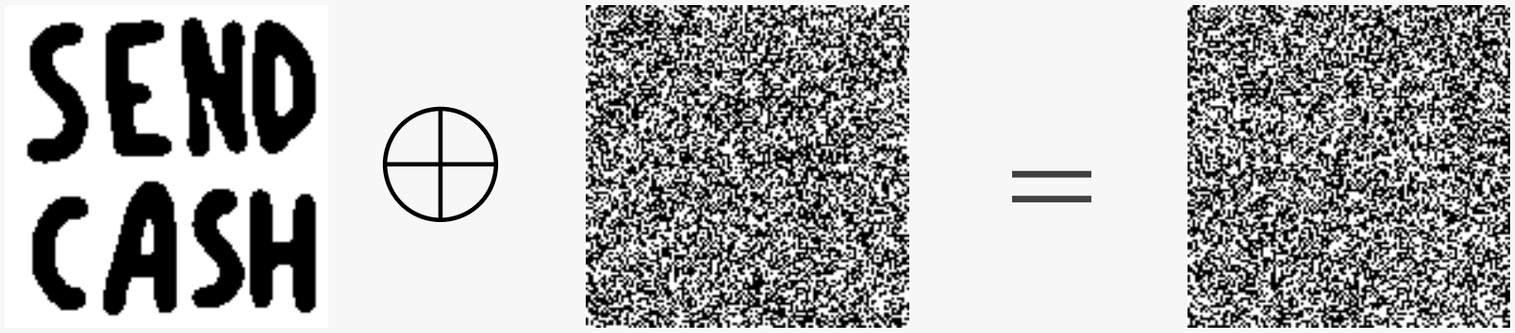
One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once:

(images from cryptosmith.com , 31/05/2008)

One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once:



(images from cryptosmith.com , 31/05/2008)

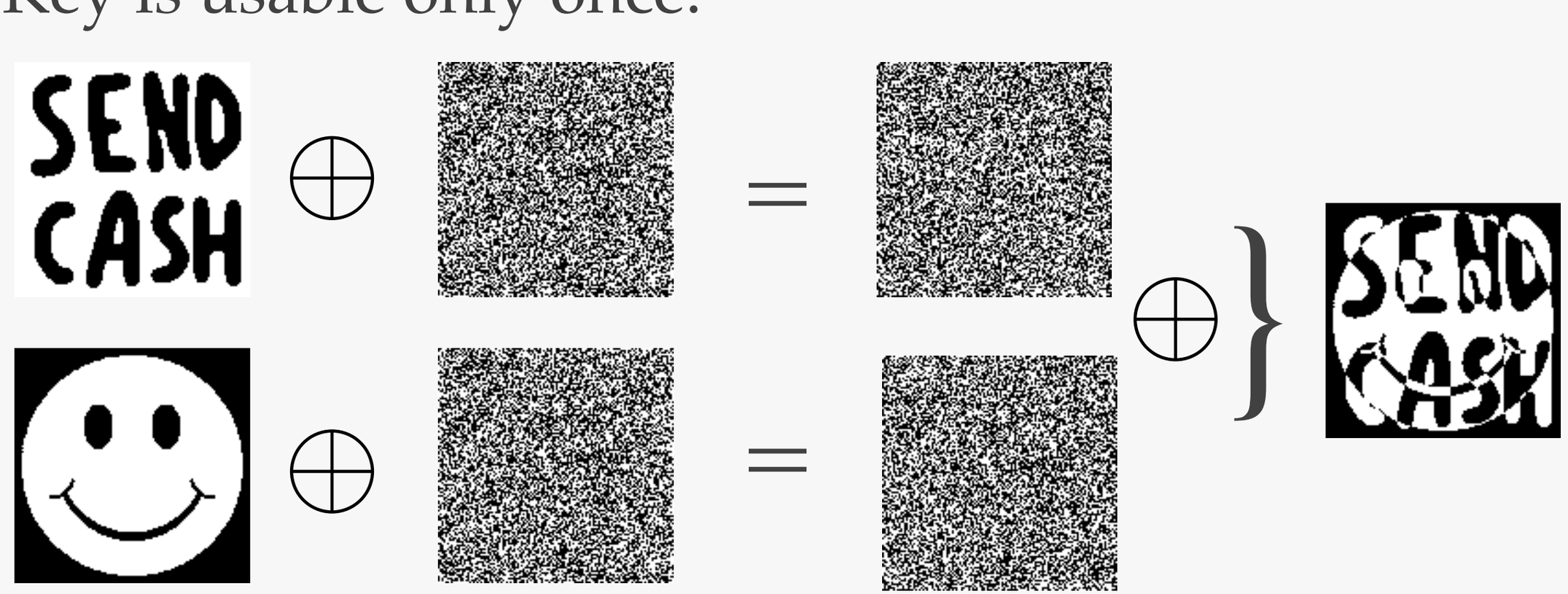
One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once:

(images from cryptosmith.com , 31/05/2008)

One-time pad: drawbacks

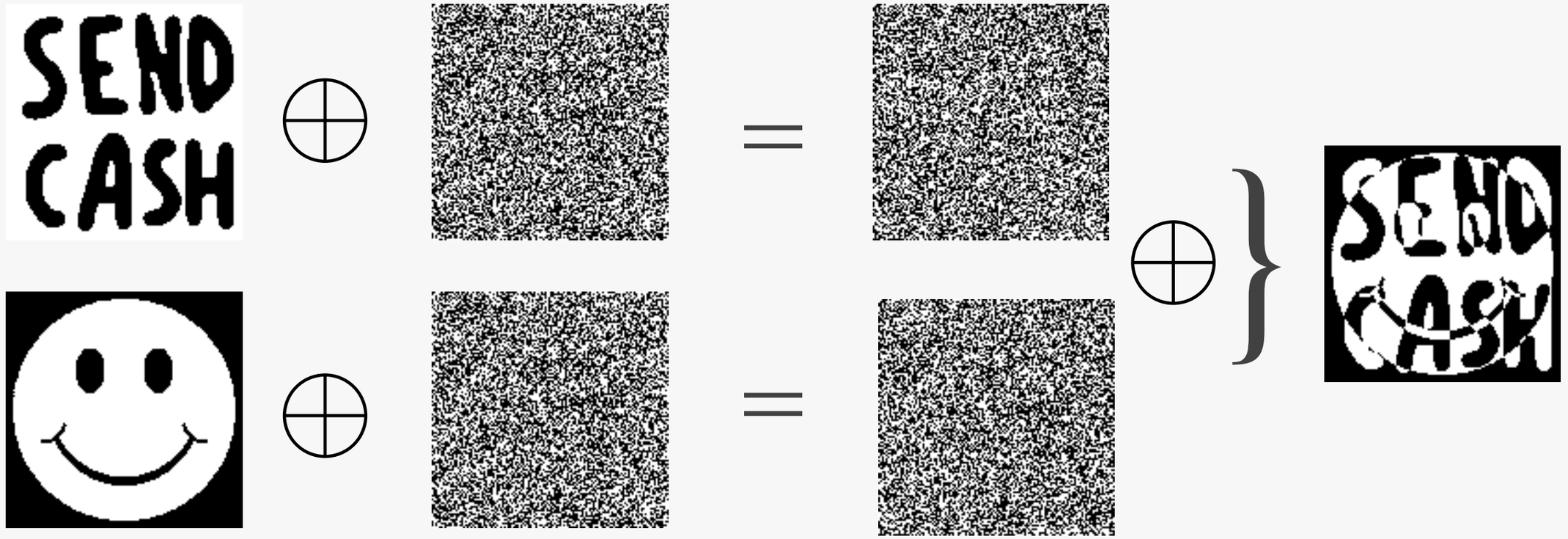
- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once:



(images from cryptosmith.com , 31/05/2008)

One-time pad: drawbacks

- Long keys: how do Alice and Bob share the key? (another primitive: key distribution)
- Key is usable only once: $(m_1 \oplus k) \oplus (m_2 \oplus k) = m_1 \oplus m_2$



(images from cryptosmith.com , 31/05/2008)

Alternative: public-key encryption

Alternative: public-key encryption

- Computational security: message is not hidden, but takes a lot of resources (time, electricity) to compute

Alternative: public-key encryption

- Computational security: message is not hidden, but takes a lot of resources (time, electricity) to compute
- Quantum computers can compute it faster!

Alternative: public-key encryption

- Computational security: message is not hidden, but takes a lot of resources (time, electricity) to compute
- Quantum computers can compute it faster!
- 2 ways to go:

Alternative: public-key encryption

- Computational security: message is not hidden, but takes a lot of resources (time, electricity) to compute
- Quantum computers can compute it faster!
- 2 ways to go:
  - Use harder problems (post-quantum cryptography)

Alternative: public-key encryption

- Computational security: message is not hidden, but takes a lot of resources (time, electricity) to compute
- Quantum computers can compute it faster!
- 2 ways to go:
  - Use harder problems (post-quantum cryptography)
  - Use one-time pad, but use quantum computers to distribute the keys (QKD)

Quantum encryption: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

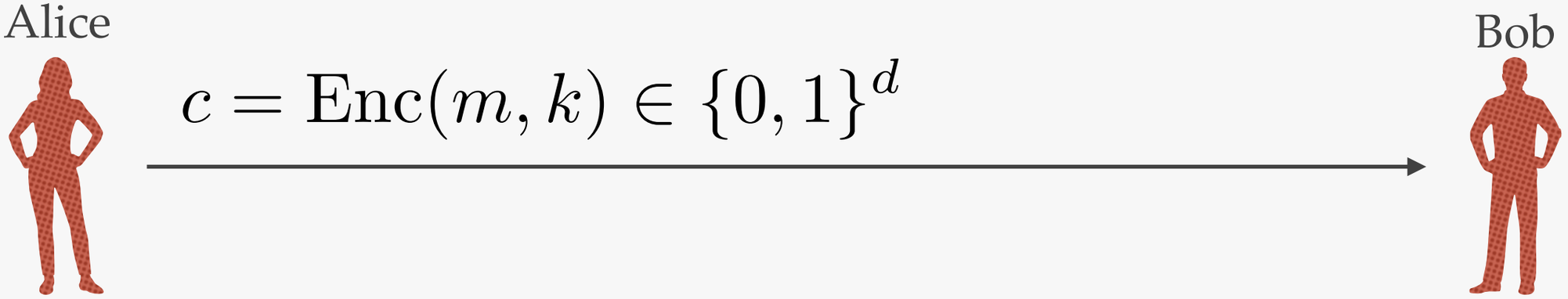
$c = \mathrm{Enc}(m,k) \in \{0,1\}^d$

Bob

k

$m' = \mathrm{Dec}(c,k)$

Quantum encryption: definition



Quantum encryption: definition

Alice

ρ (n qubits)

$k \in \{0,1\}^\ell$

$\sigma = \mathrm{Enc}(\rho, k)$ (d qubits)

Bob

k

$m' = \mathrm{Dec}(c, k)$

Quantum encryption: definition

Alice

ρ (n qubits)

$k \in \{0,1\}^\ell$

$\sigma = \mathrm{Enc}(\rho, k)$ (d qubits)

Bob

k

$\rho' = \mathrm{Dec}(\sigma, k)$

Quantum encryption: definition

Alice

ρ (n qubits)

$k \in \{0,1\}^\ell$

$\sigma = \mathrm{Enc}(\rho, k)$ (d qubits)

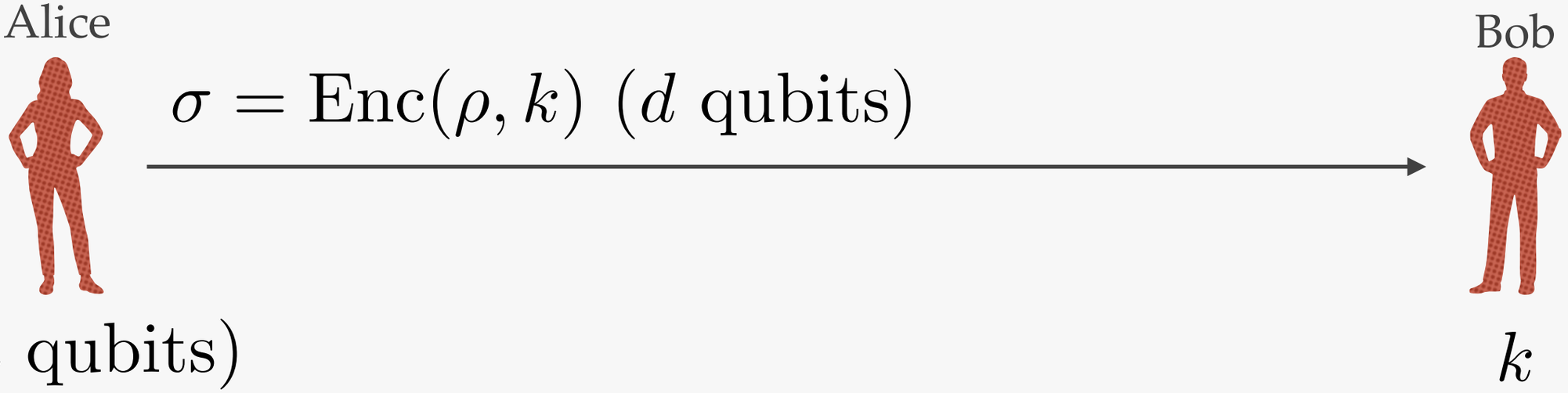
Bob

k

$\rho' = \mathrm{Dec}(\sigma, k)$

- correct: $\rho \approx \rho'$ ($\|\rho - \rho'\|_1 < \varepsilon$)

Quantum encryption: definition



- correct: $\rho \approx \rho'$ $\quad$ ($\|\rho - \rho'\|_1 < \varepsilon$)
- efficient: Enc and Dec are quantum polynomial-time

Quantum encryption: definition

Alice

$\sigma = \mathrm{Enc}(\rho, k)$ (d qubits)

Bob

ρ (n qubits)

$k \in \{0,1\}^\ell$

k

$\rho' = \mathrm{Dec}(\sigma, k)$

- correct: $\rho \approx \rho'$ ($\|\rho - \rho'\|_1 < \varepsilon$)
- efficient: Enc and Dec are quantum polynomial-time
- private: σ hides all the information about ρ

Quantum encryption: privacy

Quantum encryption: privacy

- First attempt:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}\mathrm{Enc}(\rho,k) \approx \frac{\mathbb{I}}{2^d}$$

Quantum encryption: privacy

- First attempt:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}\mathrm{Enc}(\rho,k) \approx \frac{\mathbb{I}}{2^d}$$

- Accounting for side information:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}(\mathrm{Enc}\otimes\mathbb{I})(\rho_{ME},k) \approx \frac{\mathbb{I}}{2^d}\otimes\rho_E$$

Quantum encryption: privacy

- Accounting for side information:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}(\mathrm{Enc}\otimes\mathbb{I})(\rho_{ME},k) \approx \frac{\mathbb{I}}{2^d}\otimes\rho_E$$

Quantum encryption: privacy

- Accounting for side information:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}(\mathrm{Enc}\otimes\mathbb{I})(\rho_{ME},k)\approx\frac{\mathbb{I}}{2^d}\otimes\rho_E$$

- Semantic definition:

Quantum encryption: privacy

- Accounting for side information:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}(\mathrm{Enc}\otimes\mathbb{I})(\rho_{ME},k) \approx \frac{\mathbb{I}}{2^d}\otimes\rho_E$$

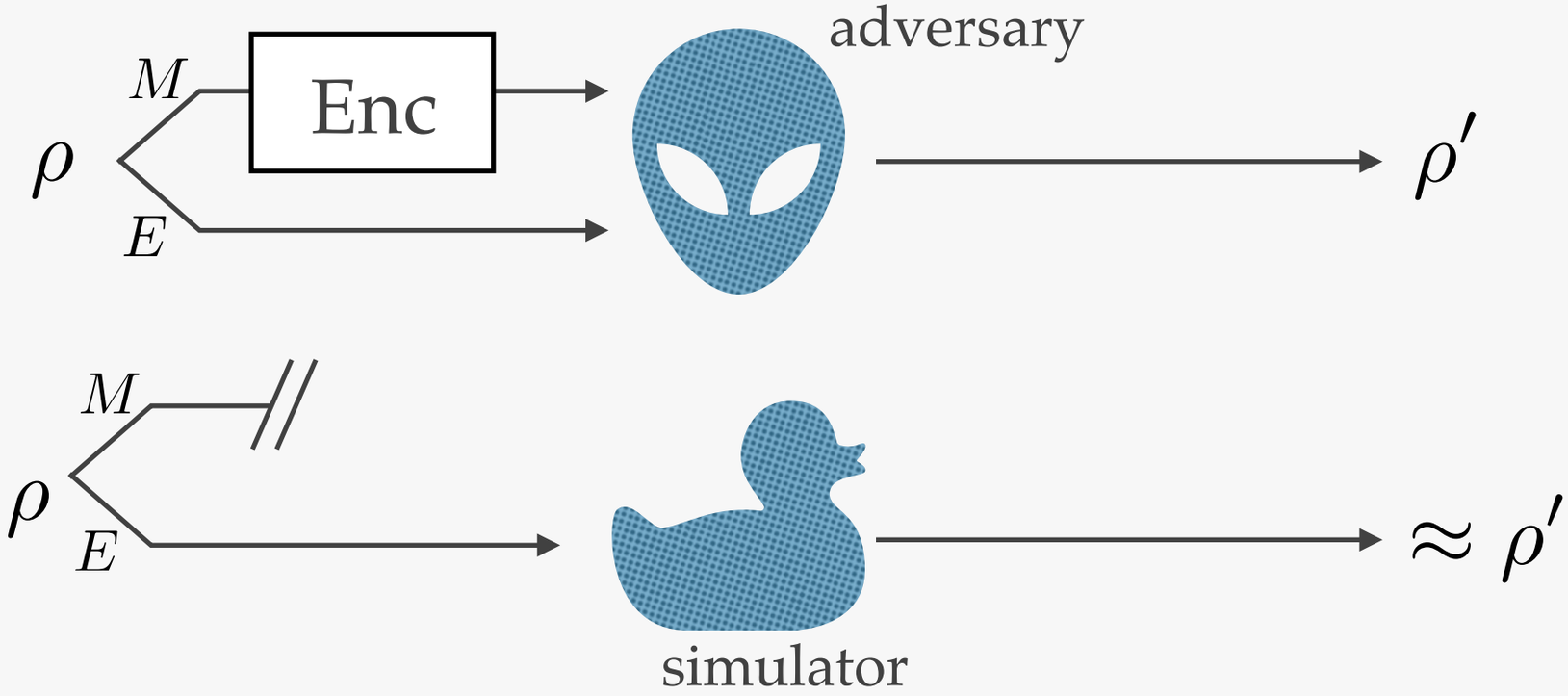
- Semantic definition:

Quantum encryption: privacy

- Accounting for side information:

$$\sum_{k\in\{0,1\}^\ell} \frac{1}{2^\ell}(\mathrm{Enc}\otimes\mathbb{I})(\rho_{ME},k)\approx\frac{\mathbb{I}}{2^d}\otimes\rho_E$$

- Semantic definition:



Quantum one-time pad (QOTP)

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

  but: $X^0|+\rangle = X^1|+\rangle = |+\rangle$ not encrypted!

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

  but: $X^0|+\rangle = X^1|+\rangle = |+\rangle$ not encrypted!

- Bit flips + phase flips:

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

  but: $X^0|+\rangle = X^1|+\rangle = |+\rangle$ not encrypted!

- Bit flips + phase flips:

$$\mathrm{Enc}(\rho, k) = X^{k_1} Z^{k_2} \rho Z^{k_2} X^{k_1} \qquad k \in \{0,1\}^2$$

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

  but: $X^0|+\rangle = X^1|+\rangle = |+\rangle$ not encrypted!

- Bit flips + phase flips:

$$\mathrm{Enc}(\rho, k) = X^{k_1} Z^{k_2} \rho Z^{k_2} X^{k_1} \qquad k \in \{0,1\}^2$$
$$\mathrm{Dec}(\rho, k) = X^{k_1} Z^{k_2} \rho Z^{k_2} X^{k_1}$$

Quantum one-time pad (QOTP)

- First idea: bit flip (X^{k_i} for every qubit i)

  but: $X^0|+\rangle = X^1|+\rangle = |+\rangle$ not encrypted!

- Bit flips + phase flips:

$$\mathrm{Enc}(\rho, k) = X^{k_1} Z^{k_2} \rho Z^{k_2} X^{k_1} \qquad k \in \{0,1\}^2$$
$$\mathrm{Dec}(\rho, k) = X^{k_1} Z^{k_2} \rho Z^{k_2} X^{k_1}$$

- $2n$ bits of key for n qubits

QOTP is secure (proof idea)

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho, k)$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) \;=\; \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4} P\rho P^\dagger$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_{P} \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_{P} \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4} P\rho P^\dagger$$

$$= \sum_P \frac{1}{8} P(\mathbb{I} + \alpha_1 X + \alpha_2 Y + \alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I} +$$

$$\sum_P \frac{1}{8} P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4} P\rho P^\dagger$$

$$= \sum_P \frac{1}{8} P(\mathbb{I} + \alpha_1 X + \alpha_2 Y + \alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I} +$$

$$\sum_P \frac{1}{8} P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I} +$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{\alpha_1}{8}PXP^\dagger = \frac{1}{8}(X+XXX+ZXZ+XZXZX) = 0$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I} + 0 +$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{\alpha_1}{8}PXP^\dagger = \frac{1}{8}(X+XXX+ZXZ+XZXZX) = 0$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I}+0+0+$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{\alpha_1}{8}PXP^\dagger = \frac{1}{8}(X+XXX+ZXZ+XZXZX) = 0$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I}+0+0+0=\frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{\alpha_1}{8}PXP^\dagger = \frac{1}{8}(X+XXX+ZXZ+XZXZX)=0$$

QOTP is secure (proof idea)

Singe-qubit state ρ (no side information)

$$\sum_{k\in\{0,1\}^2} \frac{1}{4}\mathrm{Enc}(\rho,k) = \sum_{P\in\{I,X,Z,XZ\}} \frac{1}{4}P\rho P^\dagger$$

$$= \sum_P \frac{1}{8}P(\mathbb{I}+\alpha_1 X+\alpha_2 Y+\alpha_3 Z)P^\dagger$$

$$= \frac{1}{2}\mathbb{I}+0+0+0=\frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{1}{8}P\mathbb{I}P^\dagger = \frac{1}{2}\mathbb{I}$$

$$\sum_P \frac{\alpha_1}{8}PXP^\dagger = \frac{1}{8}(X+XXX+ZXZ+XZXZX)=0$$

Summary (Primitive I)

Summary (Primitive I)

- One-time pad hides all information about the message, but requires a **long key**.

Summary (Primitive I)

- One-time pad hides all information about the message, but requires a **long key**.
- Long keys can be distributed using QKD, or we can use (quantum-safe) public-key cryptography

Summary (Primitive I)

- One-time pad hides all information about the message, but requires a **long key**.
- Long keys can be distributed using QKD, or we can use (quantum-safe) public-key cryptography
- Quantum one-time pad hides all information about a **quantum** message using **classical** (but still long) keys.

Summary (Primitive I)

- One-time pad hides all information about the message, but requires a **long key**.
- Long keys can be distributed using QKD, or we can use (quantum-safe) public-key cryptography
- Quantum one-time pad hides all information about a **quantum** message using **classical** (but still long) keys.
  - We used the **uncertainty principle** to make sure we only need finite keys.

Primitive II: Authentication

Authentication: definition

Authentication: definition

Alice

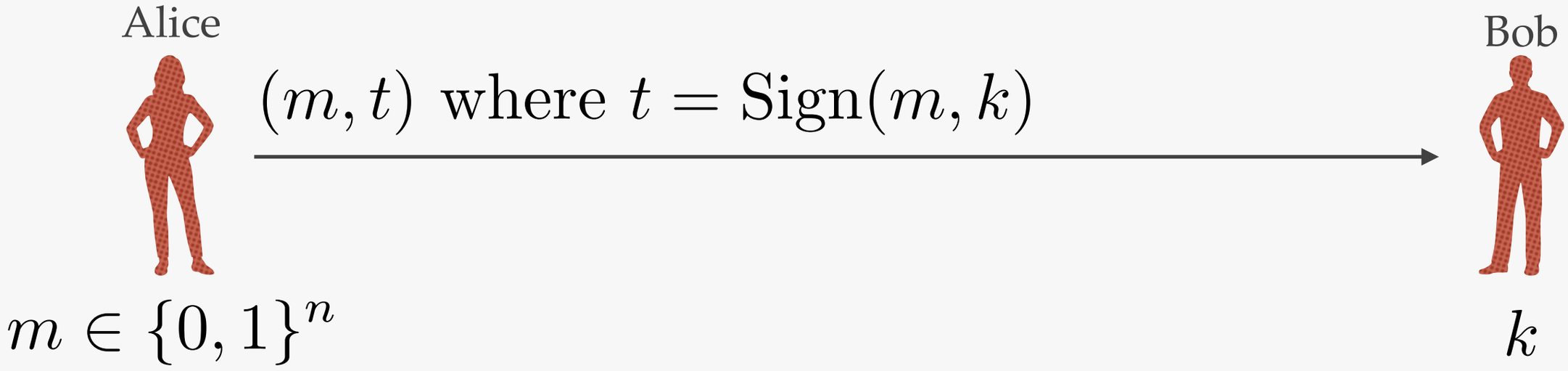
Bob

$m \in \{0,1\}^n$

k

$k \in \{0,1\}^\ell$

Authentication: definition



Authentication: definition

Alice

(m, t) where $t = \text{Sign}(m, k)$

Bob

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

k

$\text{Check}(m, t, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

Authentication: definition


Alice

(m, t) where $t = \text{Sign}(m, k)$


Bob

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

k

$\text{Check}(m, t, k)$
outputs one of:
- $accept : m$
- $reject : \perp$

- correct: $\text{Check}(m, \text{Sign}(m, k), k) =$ "accept: m"

Authentication: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

(m, t) where $t = \text{Sign}(m, k)$

Bob

k

$\text{Check}(m, t, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

- correct: $\text{Check}(m, \text{Sign}(m, k), k) =$ "accept: m"
- efficient: Sign and Check are polynomial-time

Authentication: definition

Alice

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

(m, t) where $t = \text{Sign}(m, k)$

Bob

k

$\text{Check}(m, t, k)$
outputs one of:
- $accept : m$
- $reject : \perp$

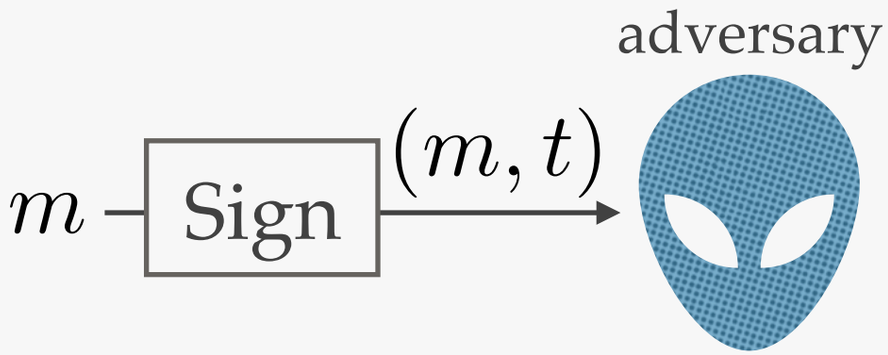
- correct: $\text{Check}(m, \text{Sign}(m, k), k) =$ "accept: m"
- efficient: Sign and Check are polynomial-time
- unforgeability: adversary cannot alter m without being detected

Authentication: unforgeability

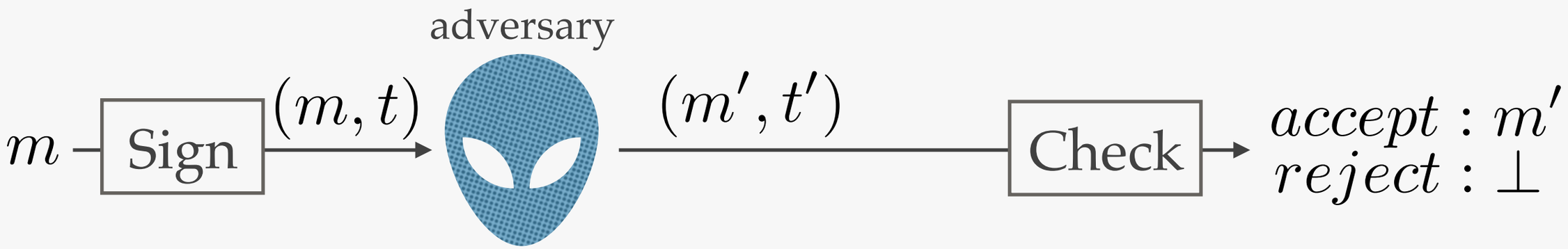
Authentication: unforgeability

adversary

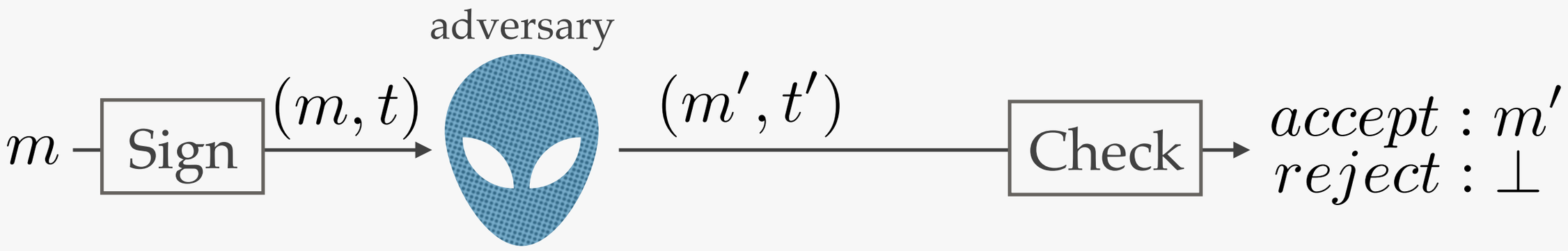
Authentication: unforgeability



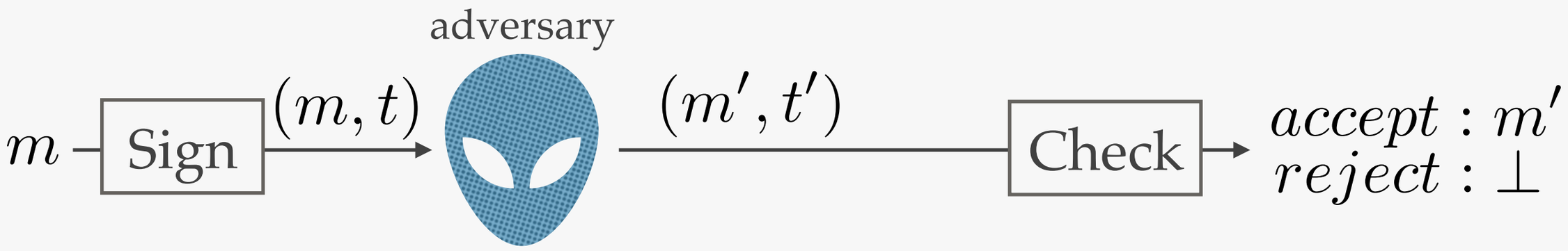
Authentication: unforgeability

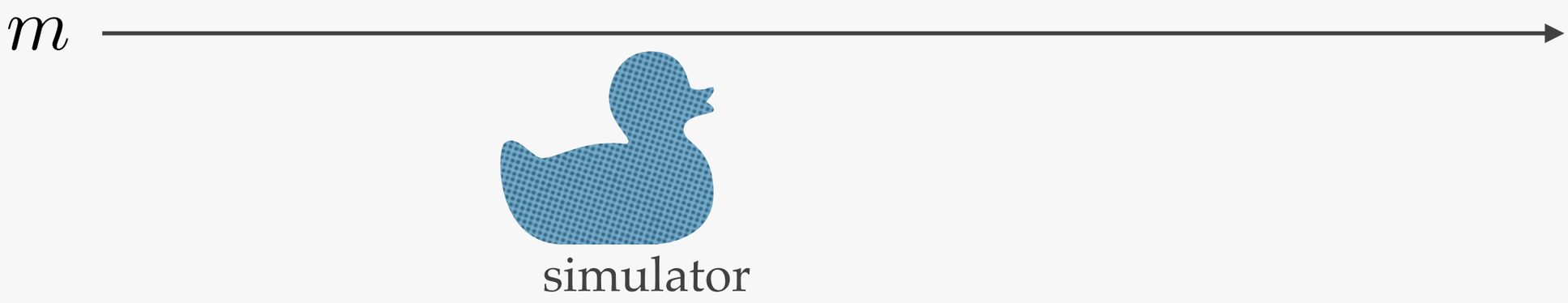


Authentication: unforgeability

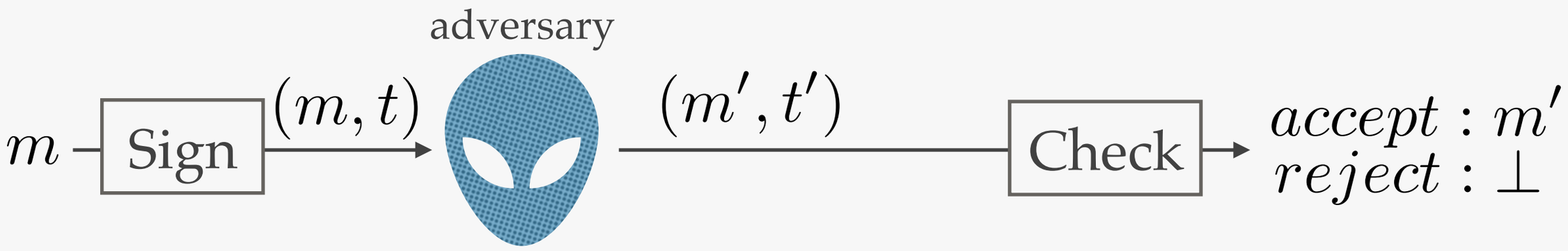


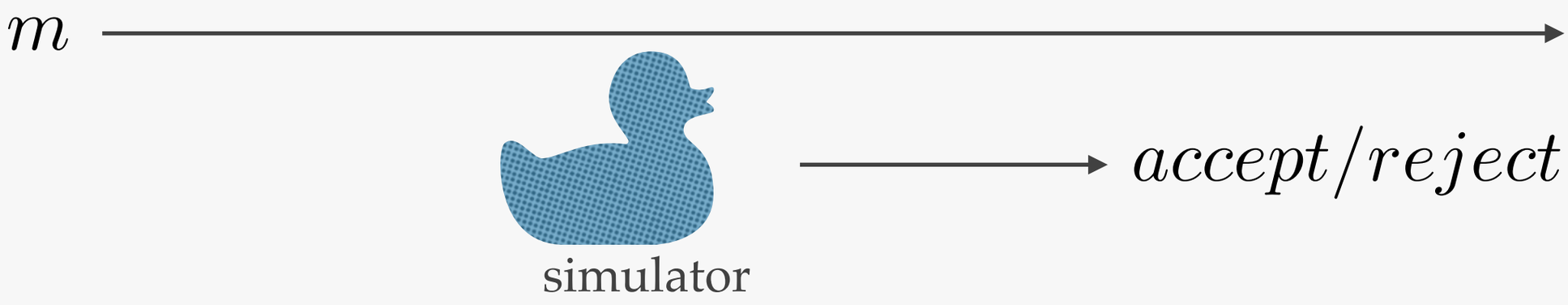
Authentication: unforgeability



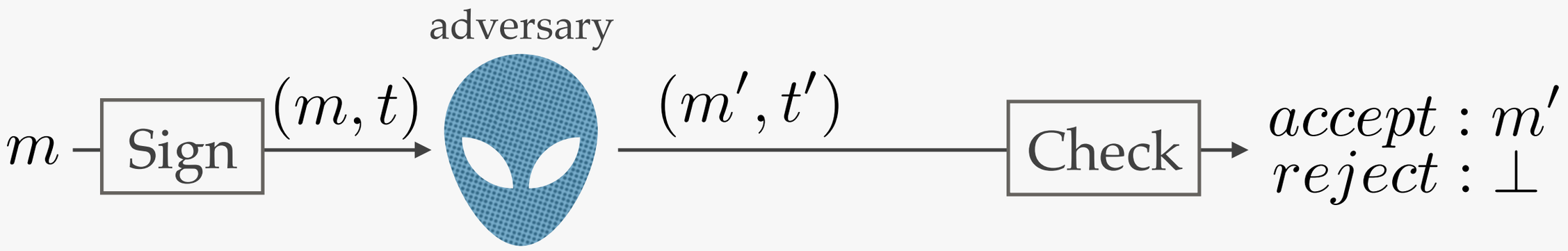


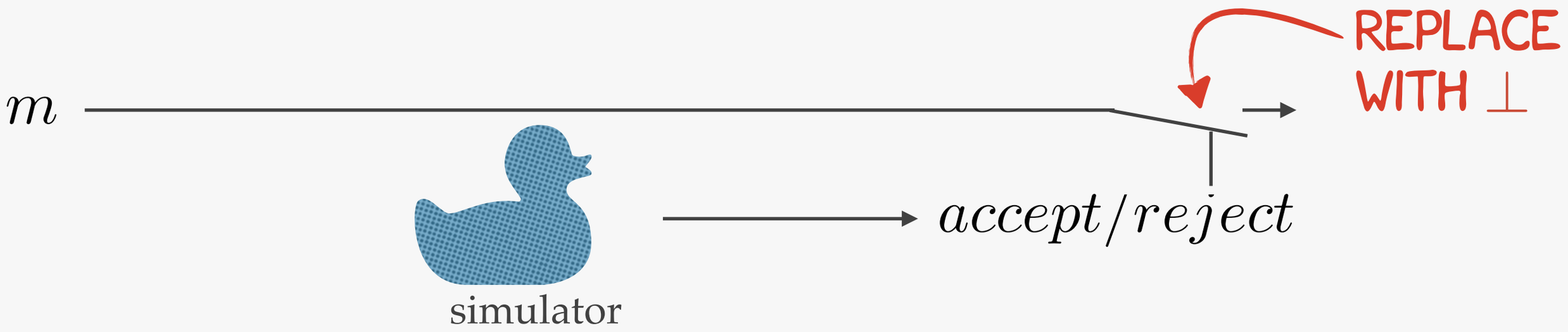
Authentication: unforgeability





Authentication: unforgeability





Authentication: example

Authentication: example

- Using pseudorandom function family $\{PRF_k\}_k$

Authentication: example

- Using pseudorandom function family $\{PRF_k\}_k$

$$\text{Sign}(m, k) := PRF_k(m)$$

Authentication: example

- Using pseudorandom function family $\{PRF_k\}_k$

$$\text{Sign}(m, k) := PRF_k(m)$$
$$\text{Check}(m, t, k) := (PRF_k(m) \stackrel{?}{=} t)$$

Authentication: example

- Using pseudorandom function family $\{PRF_k\}_k$

$$\text{Sign}(m, k) := PRF_k(m)$$
$$\text{Check}(m, t, k) := (PRF_k(m) \stackrel{?}{=} t)$$

- Security relies on the pseudorandomness: can a quantum computer tell the difference from “real” randomness?

Authentication: example

- Using pseudorandom function family $\{PRF_k\}_k$

$$\text{Sign}(m, k) := PRF_k(m)$$
$$\text{Check}(m, t, k) := (PRF_k(m) \stackrel{?}{=} t)$$

- Security relies on the pseudorandomness: can a quantum computer tell the difference from "real" randomness?
  - Quantum-secure PRFs [Zha12]

[Zha12] Mark Zhandry: How to construct quantum random functions

Authenticating a quantum state

Alice

(m, t) where $t = \mathrm{Sign}(m, k)$

Bob

$m \in \{0,1\}^n$

$k \in \{0,1\}^\ell$

k

$\mathrm{Check}(m, t, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

Authenticating a quantum state

Alice

(m, t) where $t = \mathrm{Sign}(m, k)$

ρ (n qubits)

$k \in \{0,1\}^\ell$

Bob

k

$\mathrm{Check}(m, t, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

Authenticating a quantum state

Alice

ρ (n qubits)

$k \in \{0,1\}^\ell$

$\sigma = \mathrm{Sign}(\rho, k)$

Bob

k

$\mathrm{Check}(m, t, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

Authenticating a quantum state

Alice

$\sigma = \mathrm{Sign}(\rho, k)$

ρ (n qubits)

$k \in \{0,1\}^\ell$

Bob

k

$\mathrm{Check}(\sigma, k)$

outputs one of:

- $accept : m$
- $reject : \perp$

Authenticating a quantum state

Alice

$\sigma = \mathrm{Sign}(\rho, k)$

Bob

ρ (n qubits)

$k \in \{0,1\}^\ell$

k

$\mathrm{Check}(\sigma, k)$

outputs one of:

- $accept : \rho$
- $reject : \perp$

Authenticating a quantum state

Alice

ρ (n qubits)

$k \in \{0,1\}^\ell$

$\sigma = \mathrm{Sign}(\rho, k)$

Bob

k

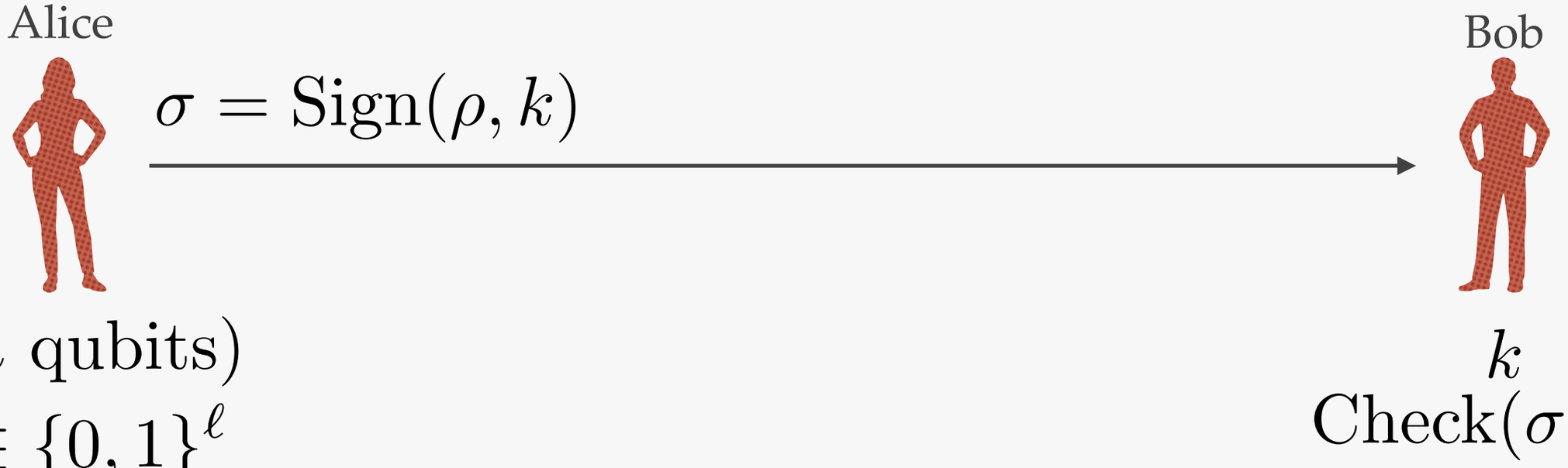
$\mathrm{Check}(\sigma, k)$

outputs one of:

- $accept : \rho$
- $reject : \perp$

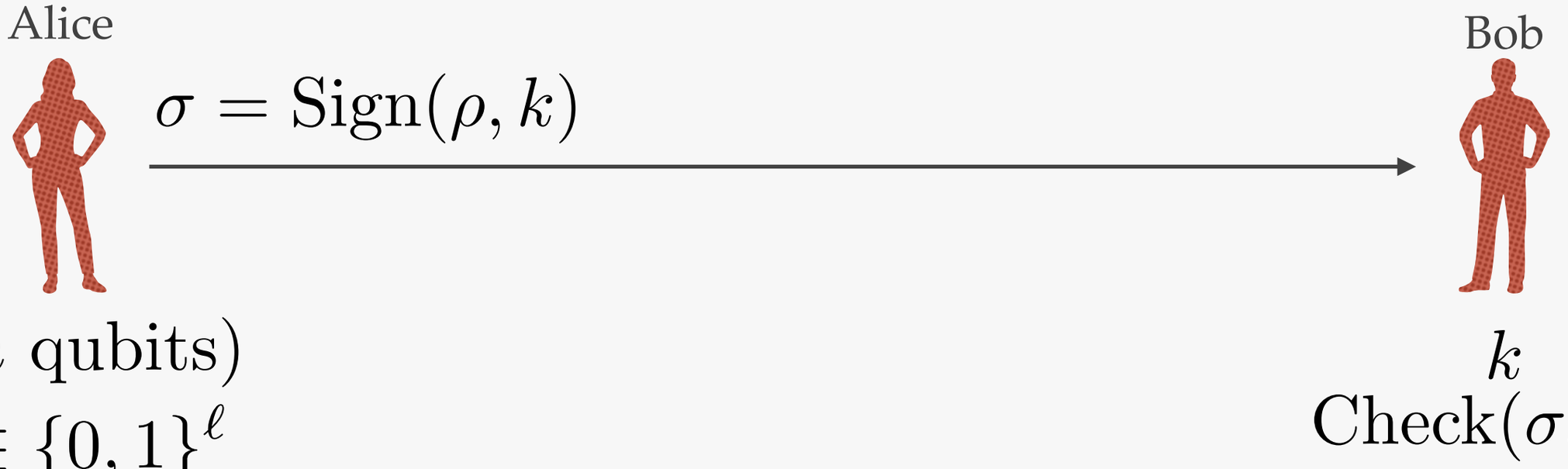
- Authentication implies encryption:

Authenticating a quantum state



- Authentication implies encryption:
  - Measure 0/1 basis → disturb +/- basis

Authenticating a quantum state



- Authentication implies encryption:
  - Measure 0/1 basis → disturb +/- basis
  - Conversely: authenticate +/- basis → encrypt 0/1 basis

Example: trap code

[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

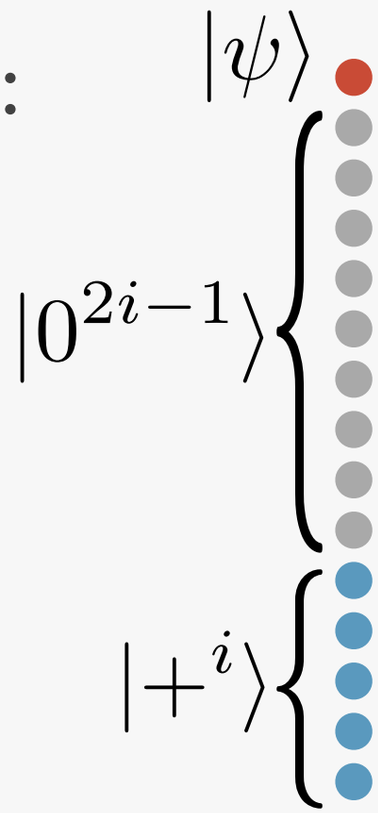
Example: trap code

Sign:

[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Example: trap code

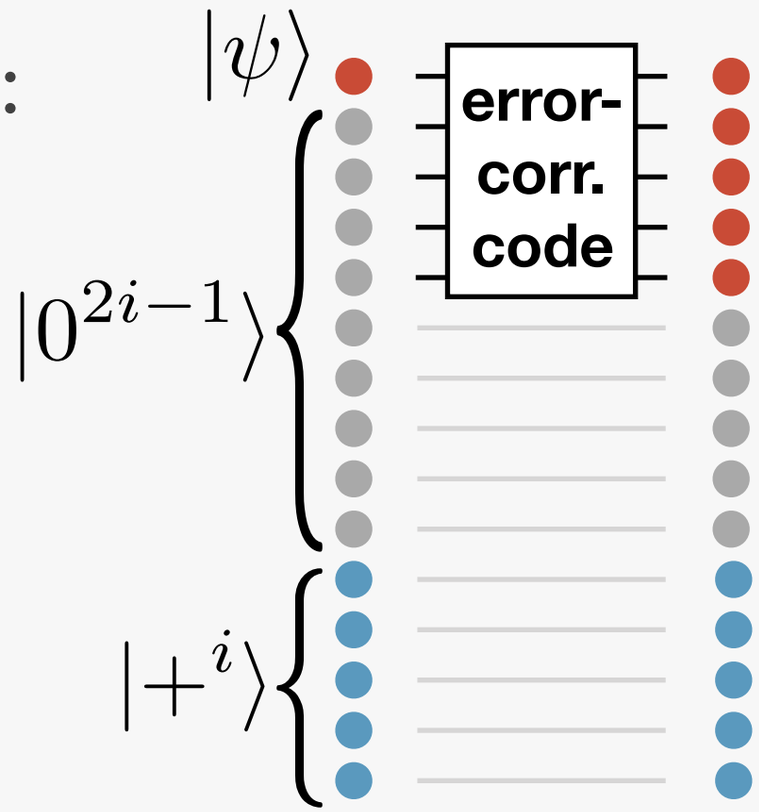
Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Example: trap code

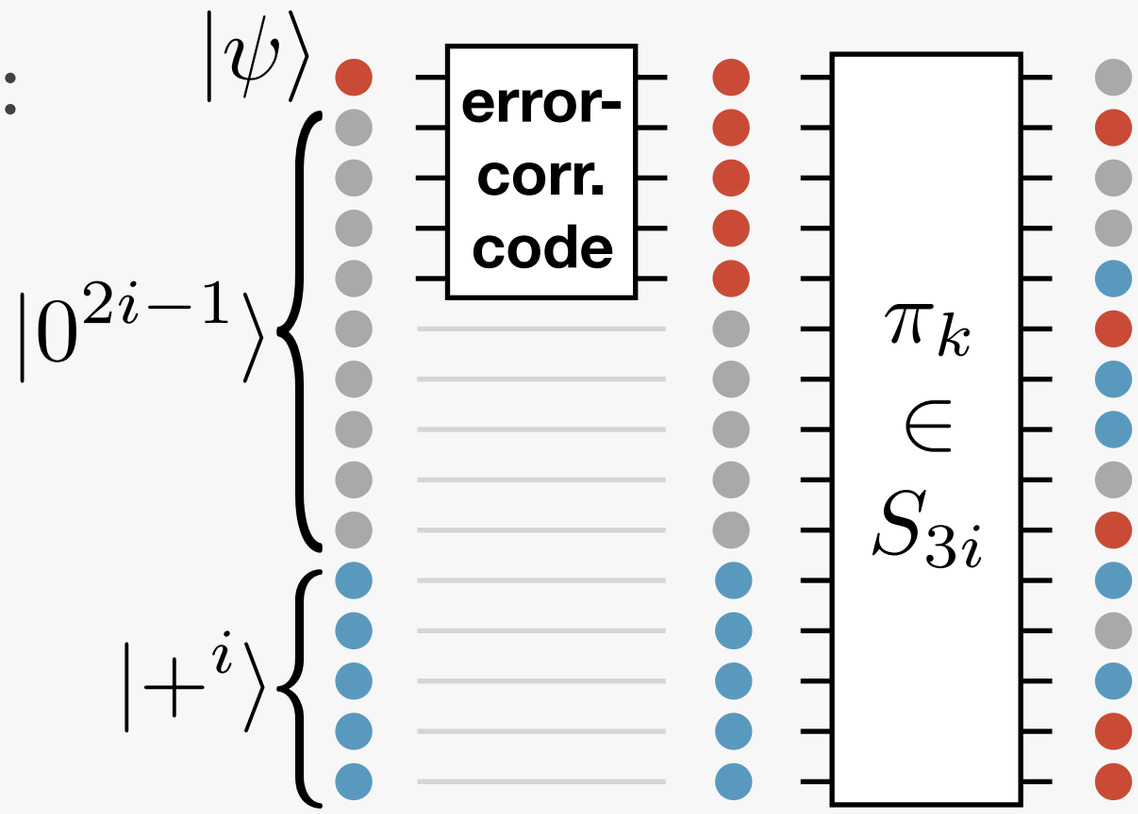
Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Example: trap code

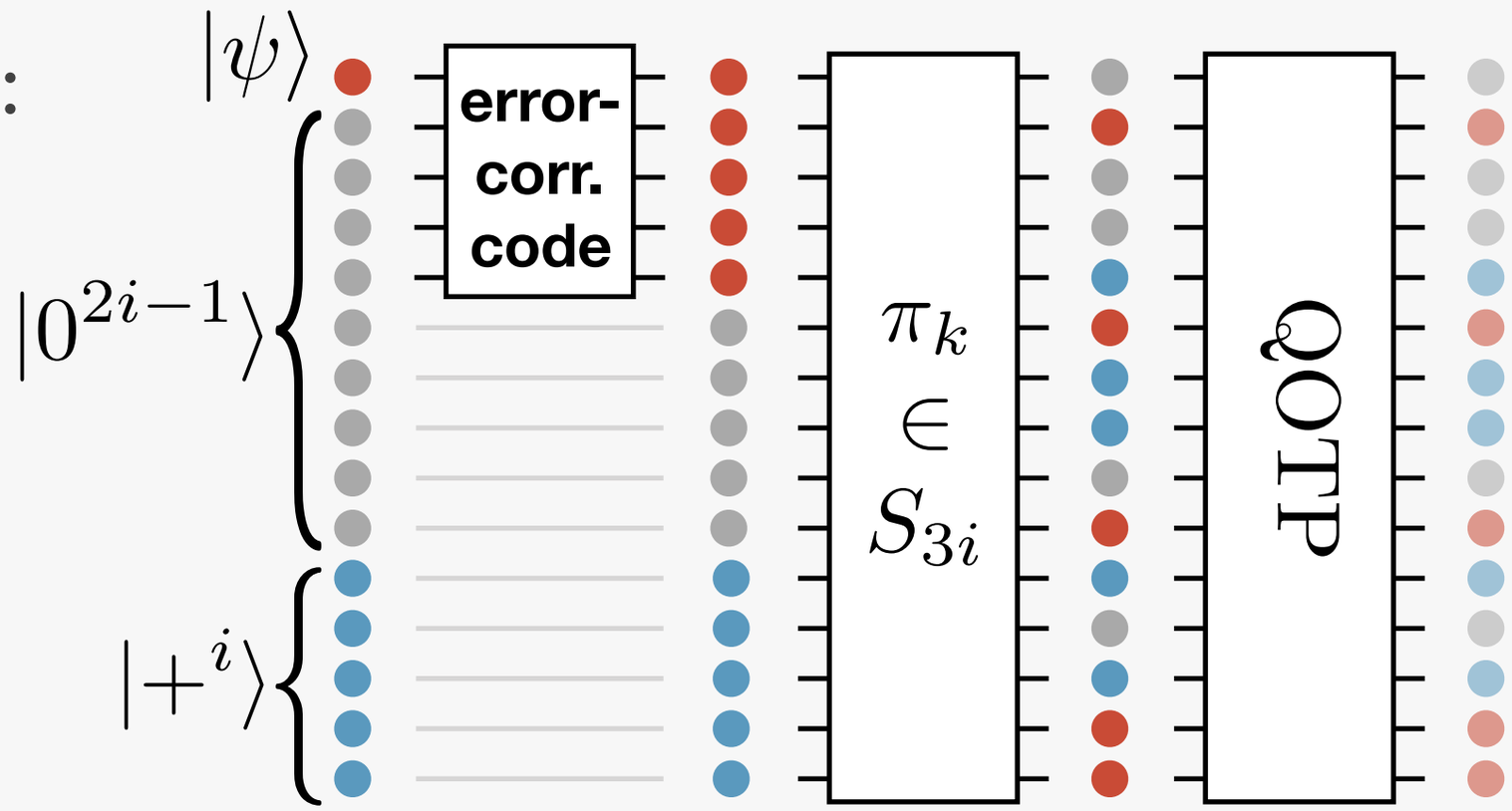
Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Example: trap code

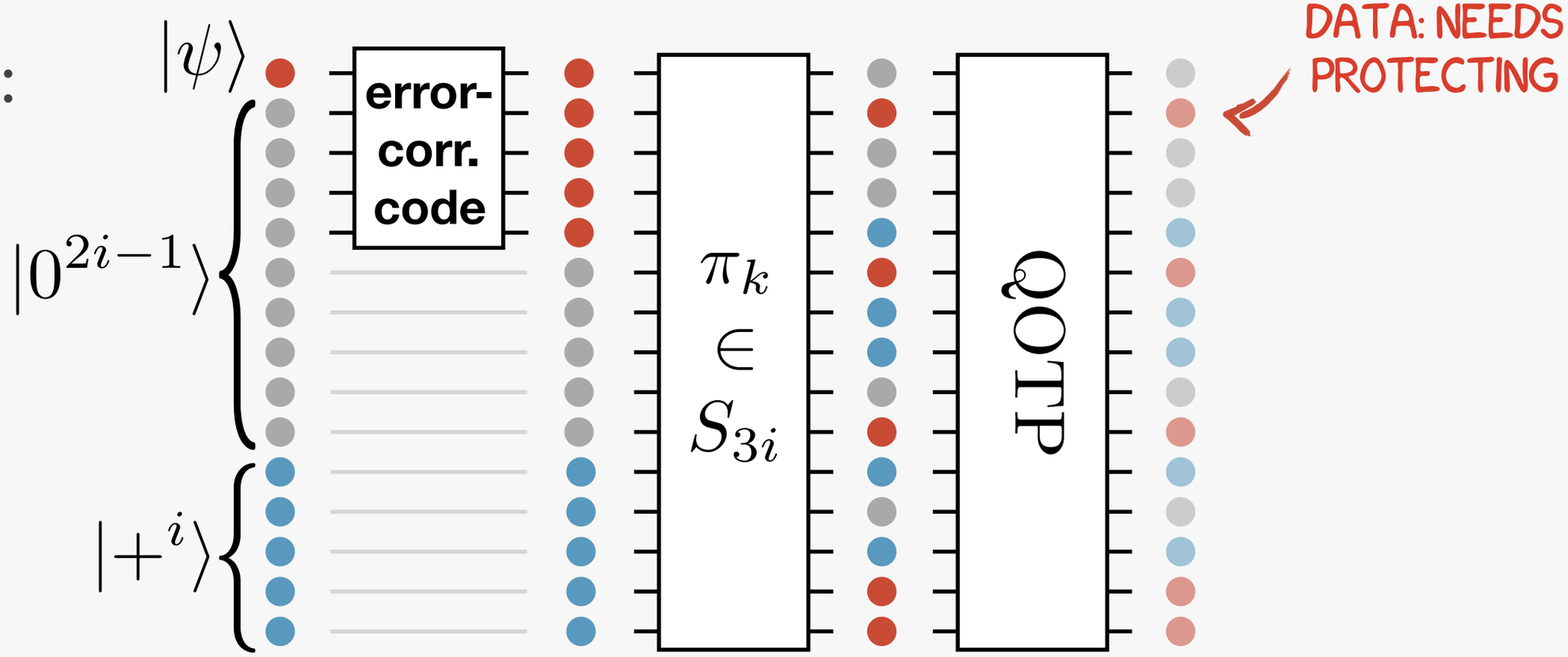
Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

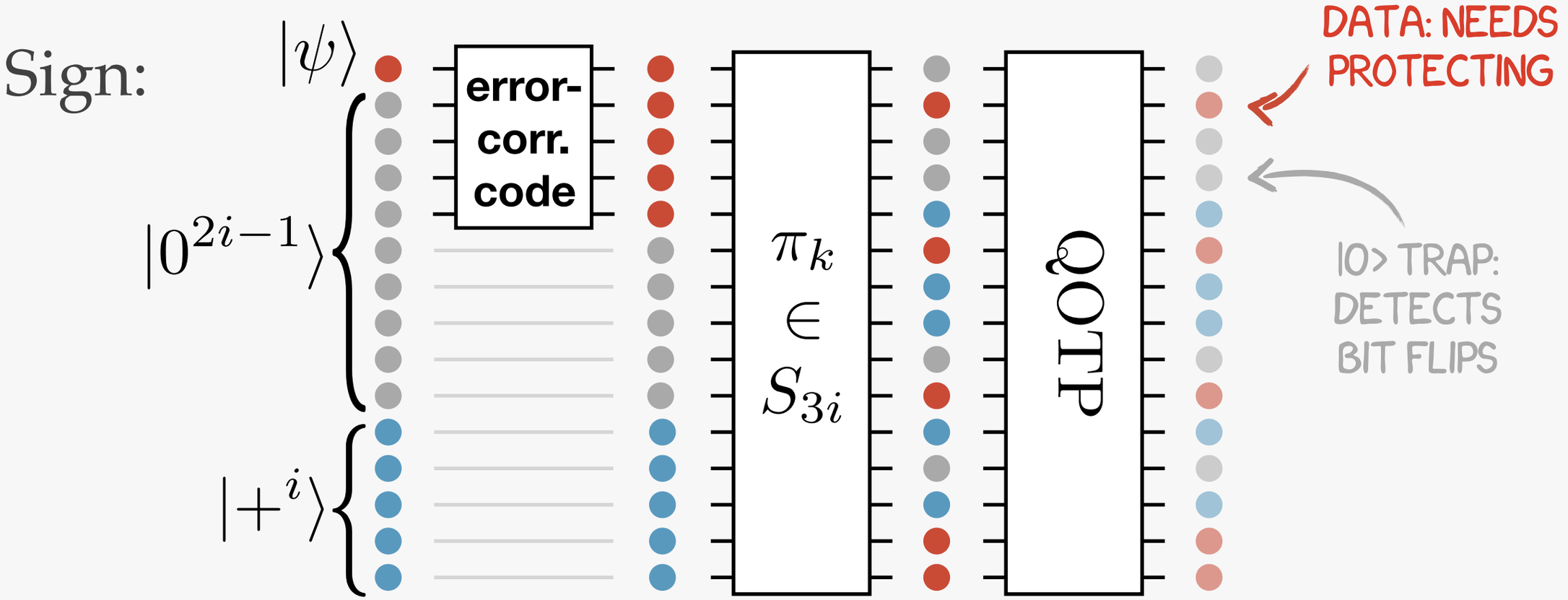
Example: trap code

Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

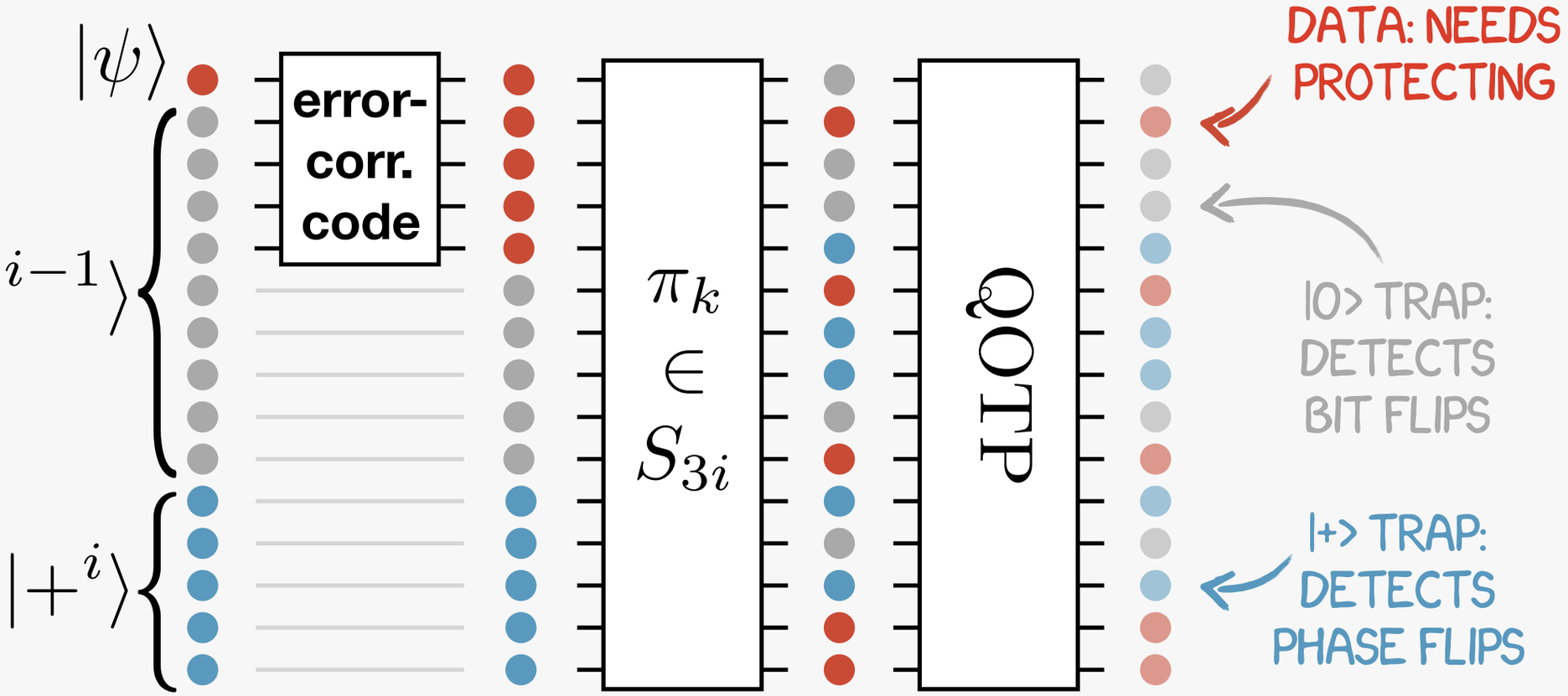
Example: trap code



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

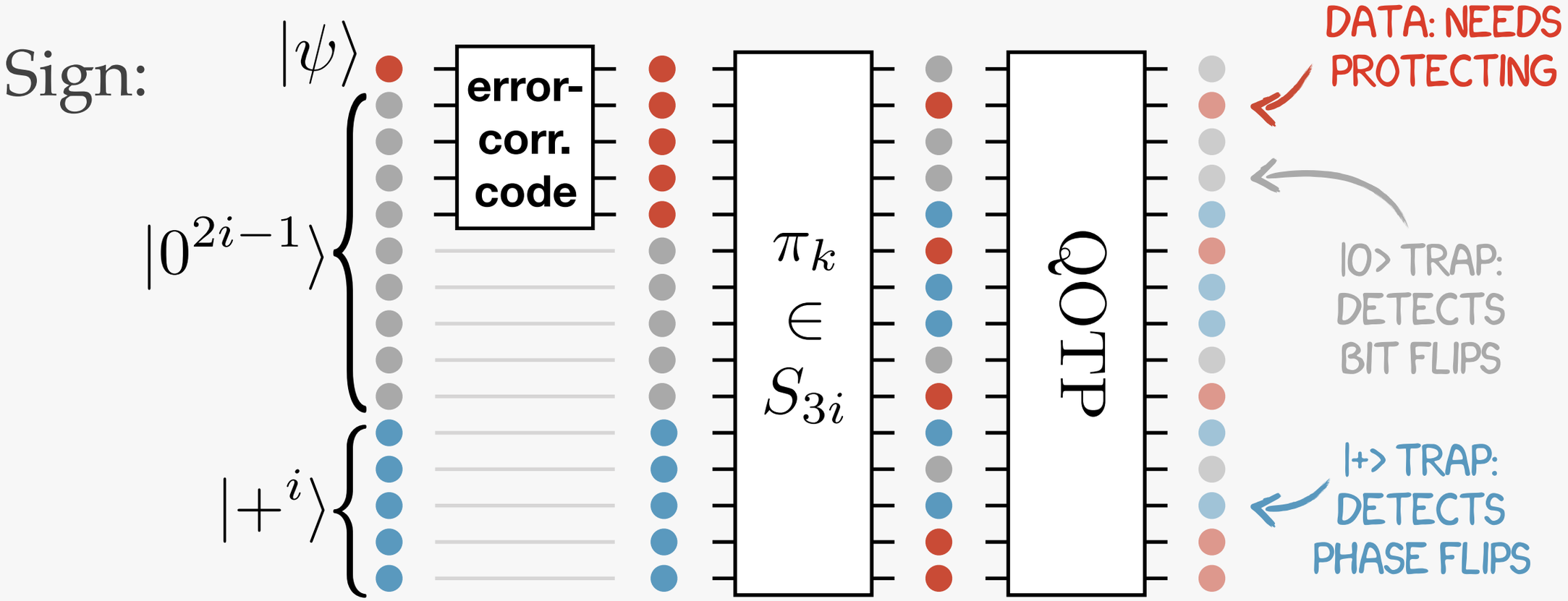
Example: trap code

Sign:



[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Example: trap code



Check: undo QOTP, undo permutation, check for errors, and measure all traps (dummy qubits)

[BGS13] Broadbent et al. (2013): Quantum One-Time Programs

Summary (Primitive II)

Summary (Primitive II)

- Authentication makes sure a message is not altered after it is signed.

Summary (Primitive II)

- Authentication makes sure a message is not altered after it is signed.
- Pseudorandom function families can sign classical messages, but they need to be **post-quantum secure**.

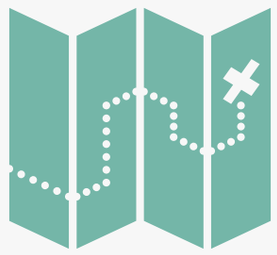
Summary (Primitive II)

- Authentication makes sure a message is not altered after it is signed.
- Pseudorandom function families can sign classical messages, but they need to be **post-quantum secure**.
- Quantum messages can be authenticated by protecting the computational and Hadamard basis separately.

Summary (Primitive II)

- Authentication makes sure a message is not altered after it is signed.
- Pseudorandom function families can sign classical messages, but they need to be **post-quantum secure**.
- Quantum messages can be authenticated by protecting the computational and Hadamard basis separately.
- Quantum authentication implies quantum encryption.

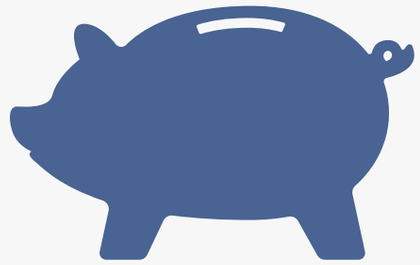
Primitive III: Secret Sharing



Secret sharing: definition

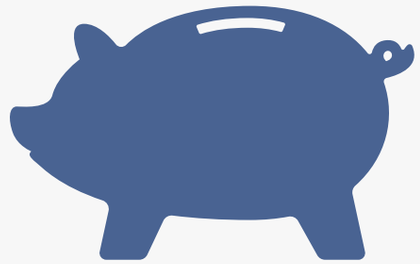
Secret sharing: definition

Secret sharing: definition



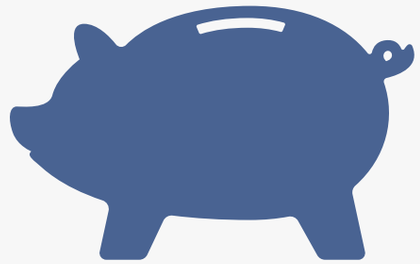
Secret sharing: definition

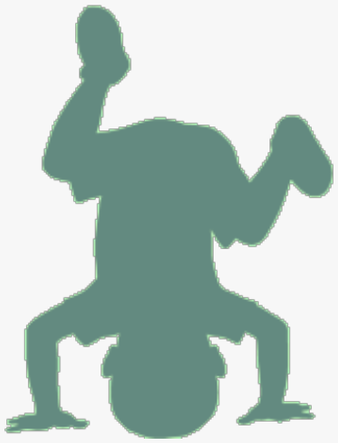
$k = 0010110001$



Secret sharing: definition

$k = 0010110001$

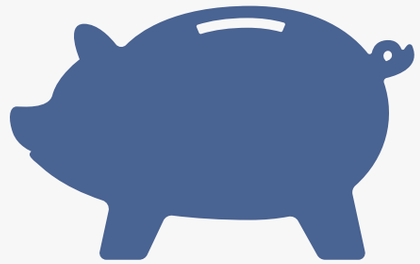


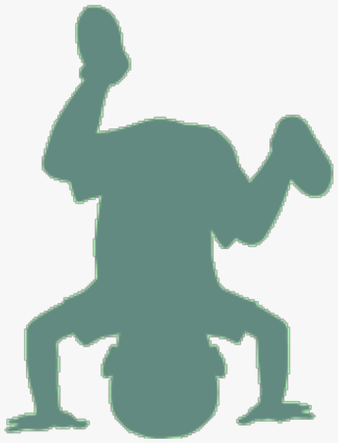


Secret sharing: definition

$$k = 0010110001$$

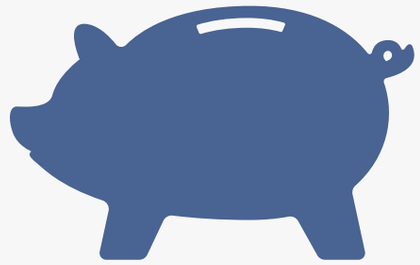
$$s_1 \quad s_2$$

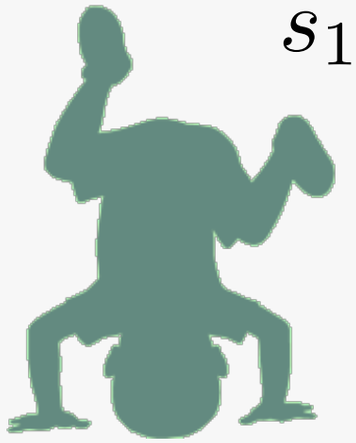




Secret sharing: definition

$k = 0010110001$



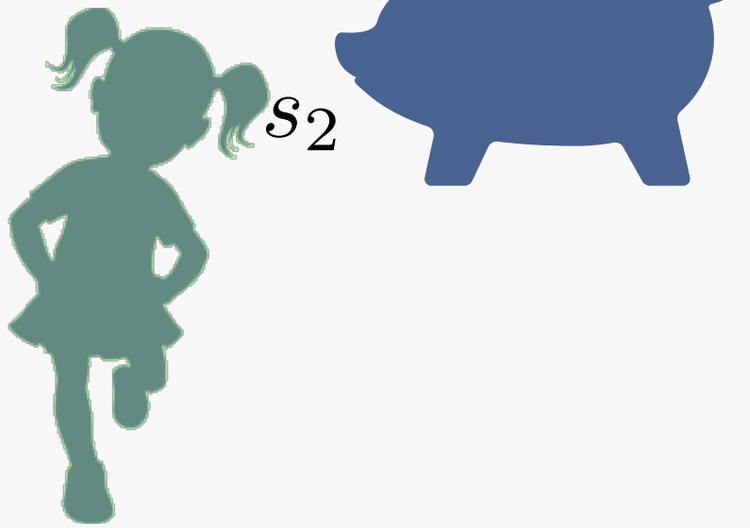


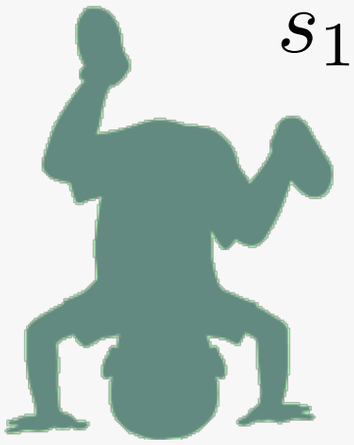
s_1

s_2

Secret sharing: definition

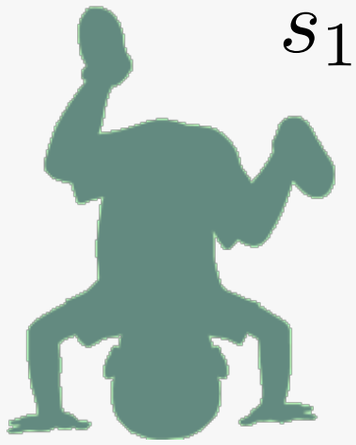
$k = 0010110001$



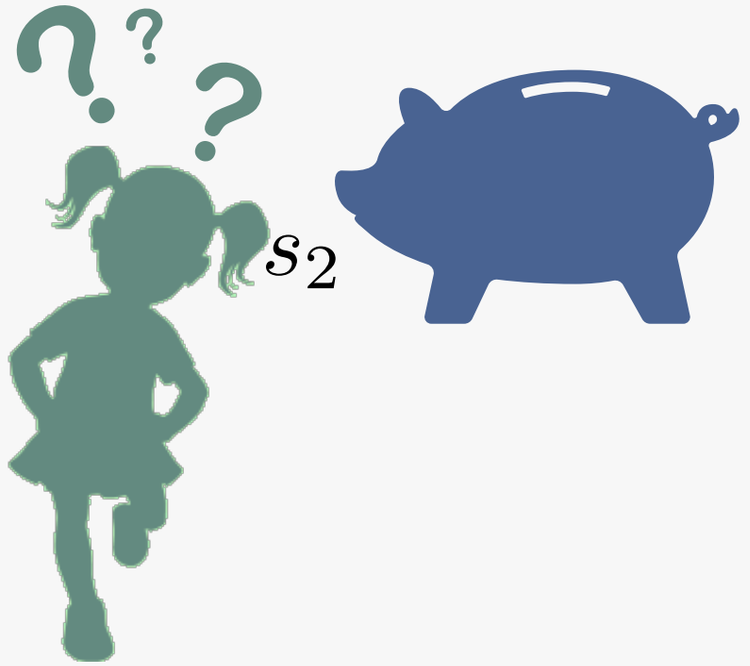


Secret sharing: definition

$k = 0010110001$



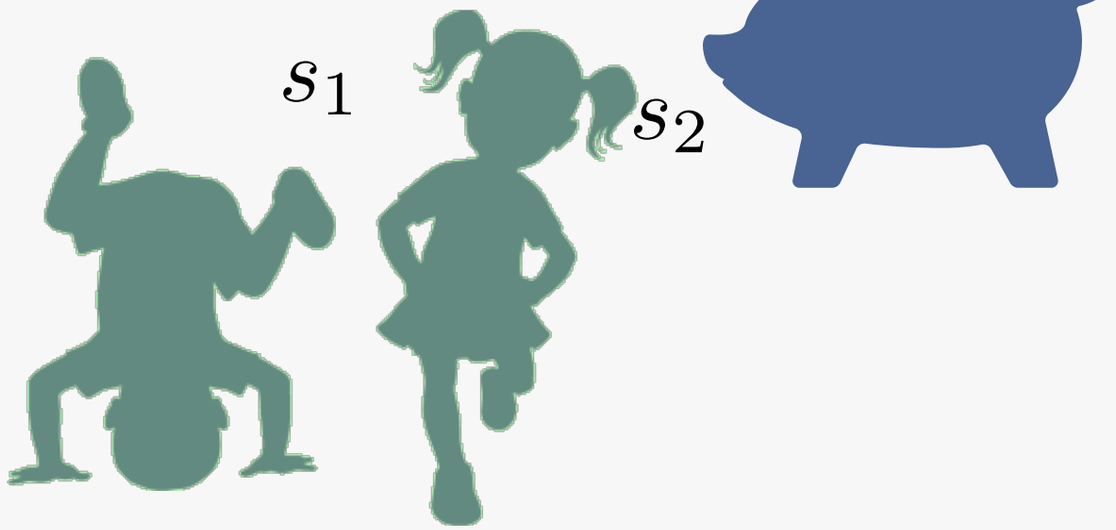
s_1



s_2

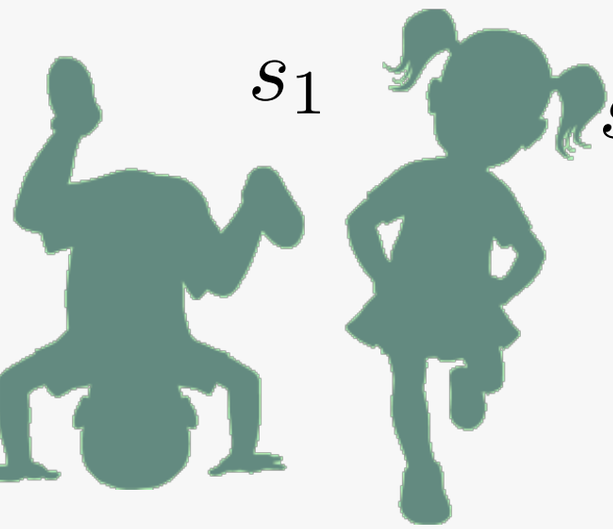
Secret sharing: definition

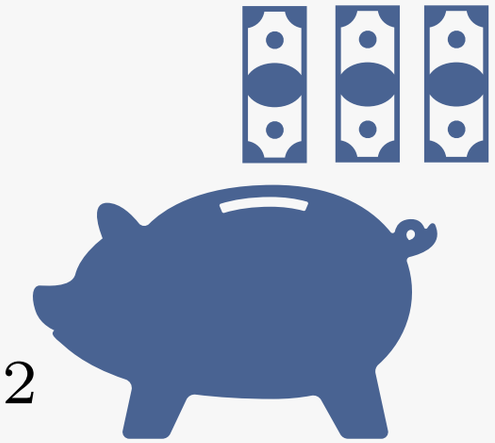
$k = 0010110001$



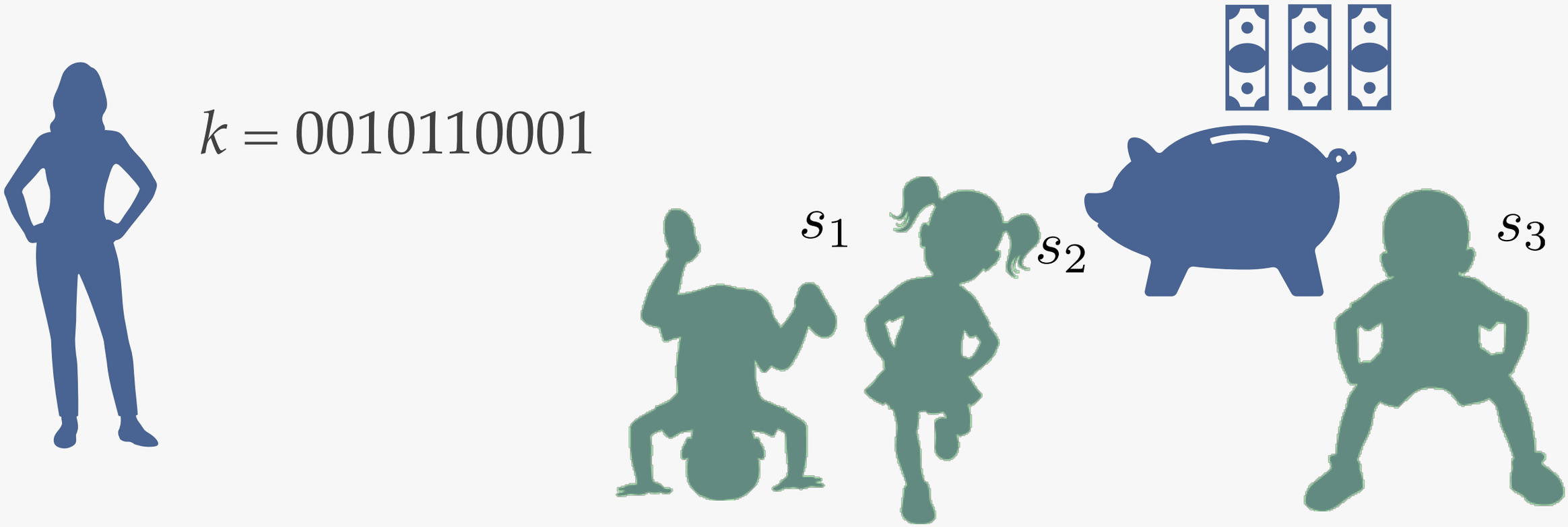
Secret sharing: definition

$k = 0010110001$

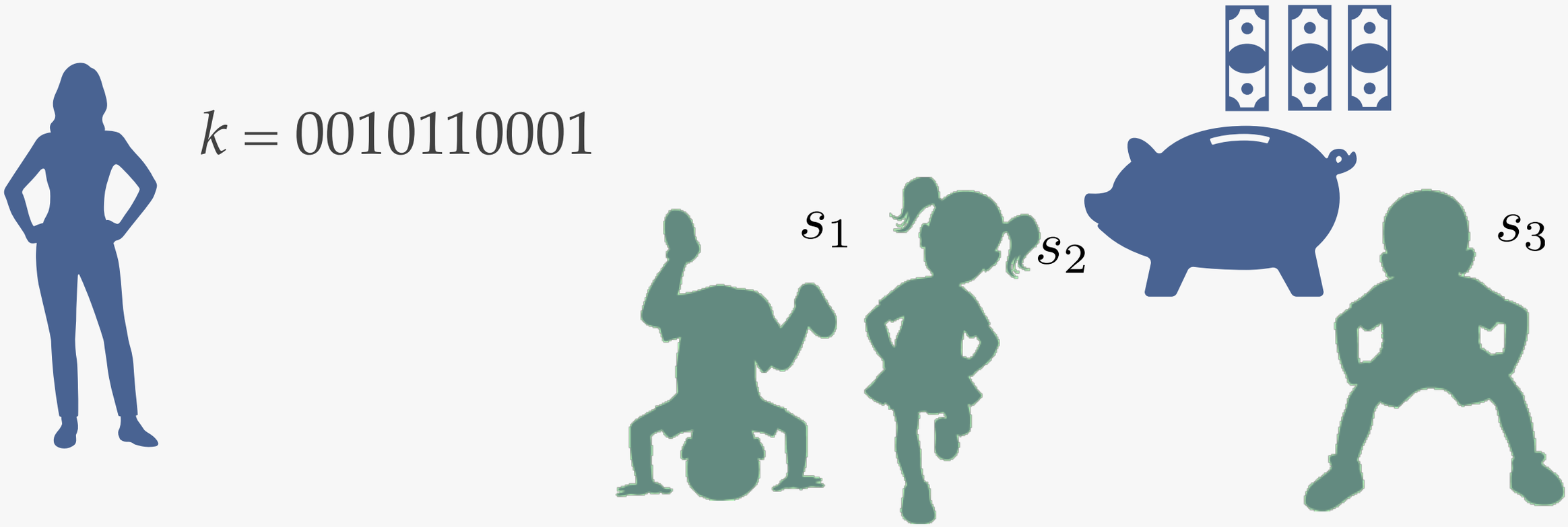




Secret sharing: definition

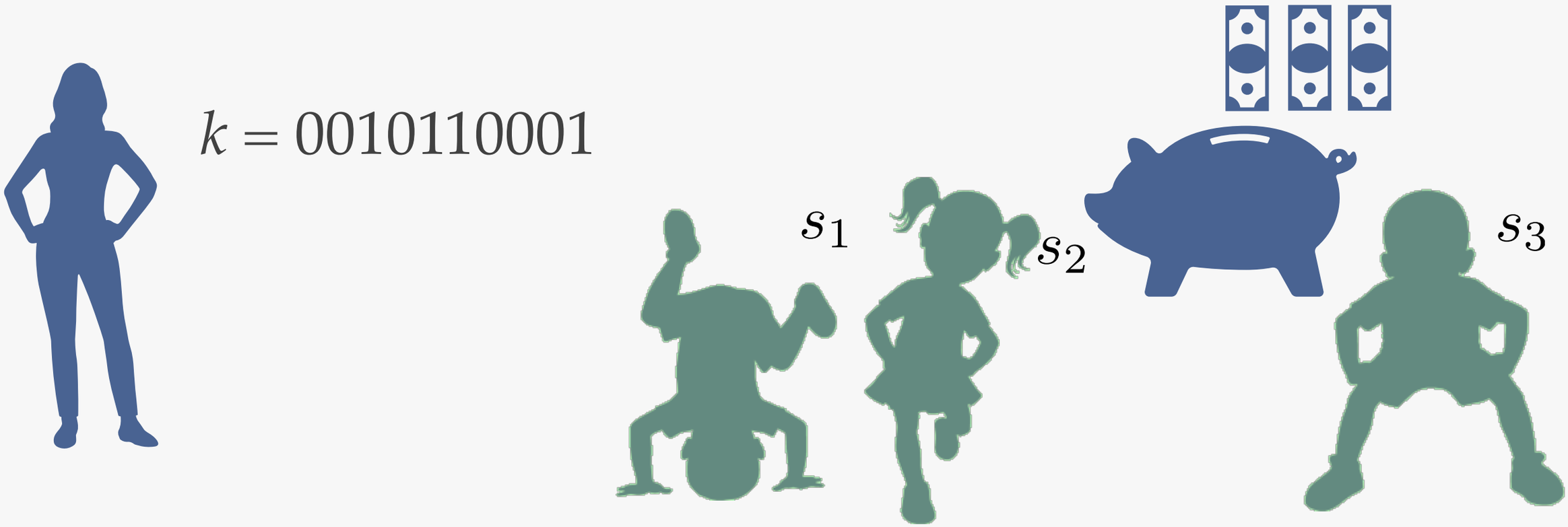


Secret sharing: definition



- Dealer: knows secret (0010110001), hands out shares

Secret sharing: definition



- Dealer: knows secret (0010110001), hands out shares
- (n,t) secret-sharing: any t out of n players can recover the secret

Secret sharing example (t=n)

Secret sharing example (t=n)

Secret sharing example (t=n)

$k = 0010110001$

Secret sharing example (t=n)

$k = 0010110001$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n - 1$

Secret sharing example (t=n)

$k = 0010110001$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n-1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

$s_1 =$

$s_2 =$

$s_3 =$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n - 1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

$s_1 = 1011000001$

$s_2 = 1101001110$

$s_3 =$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n-1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

$$s_1 = 1011000001$$
$$s_2 = 1101001110$$
$$s_3 = 0100111110$$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n-1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

$$s_1 = 1011000001$$
$$s_2 = 1101001110$$
$$s_3 = 0100111110 \oplus$$

(ADDITION MODULO 2)

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n-1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing example (t=n)

$k = 0010110001$

$$
\begin{aligned}
s_1 &= 1011000001 \\
s_2 &= 1101001110 \\
s_3 &= 0100111110 \quad \oplus \quad \text{(ADDITION MODULO 2)} \\
&\underline{\qquad\qquad\qquad\qquad} \\
&0010110001
\end{aligned}
$$

- Draw *n-1* random strings $r_1, r_2, ..., r_{n-1}$
- Set $s_i := r_i$ for $i \leq n-1$
- Set $s_n := k \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$

Secret sharing: considerations

Secret sharing: considerations

- For t = n, the randomness protocol is secure (<t players learn *nothing* about the key k)

Secret sharing: considerations

- For t = n, the randomness protocol is secure (<t players learn *nothing* about the key k)
- For t < n, more complicated protocol is needed. e.g. Shamir (polynomials) or Blakley (hyperplanes)

Secret sharing: considerations

- For t = n, the randomness protocol is secure (<t players learn *nothing* about the key k)
- For t < n, more complicated protocol is needed. e.g. Shamir (polynomials) or Blakley (hyperplanes)
- How to distribute the secrets? What about eavesdropping?

Quantumly sharing classical secrets

$k \in \{0, 1\}$

Quantumly sharing classical secrets

$k \in \{0,1\}$
$r_1 \oplus r_2$

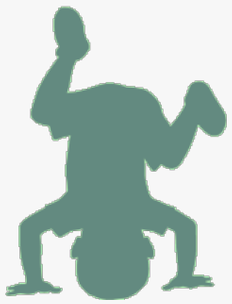
r_1

r_2

Quantumly sharing classical secrets

$k \in \{0,1\}$
$r_1 \oplus r_2$
$k \oplus r_1 \oplus r_2$

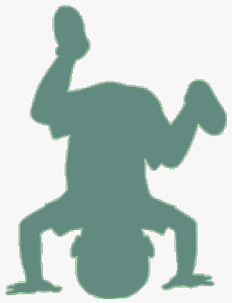
r_1



r_2

Quantumly sharing classical secrets

$k \in \{0,1\}$
$r_1 \oplus r_2$



r_1

$k \oplus r_1 \oplus r_2$

r_2

Quantumly sharing classical secrets

$k \in \{0,1\}$
$r_1 \oplus r_2$

r_1

r_2

Quantumly sharing classical secrets

$k \in \{0, 1\}$
$r_1 \oplus r_2$

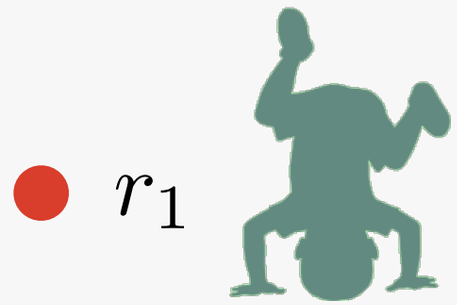
r_1

r_2

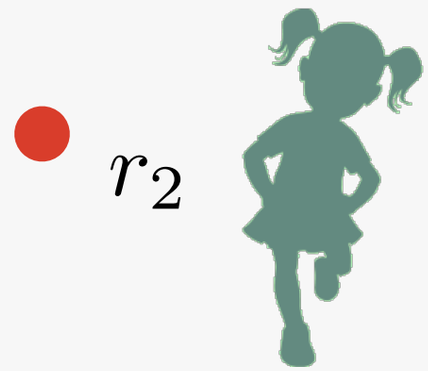
$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

Quantumly sharing classical secrets

$k \in \{0, 1\}$
$r_1 \oplus r_2$



r_1

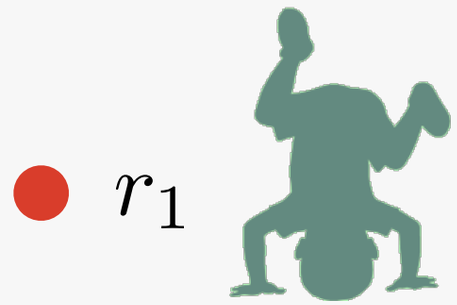


r_2

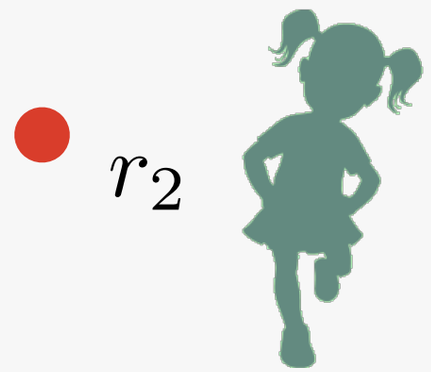
$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

Quantumly sharing classical secrets

$k \in \{0, 1\}$
$r_1 \oplus r_2$



r_1



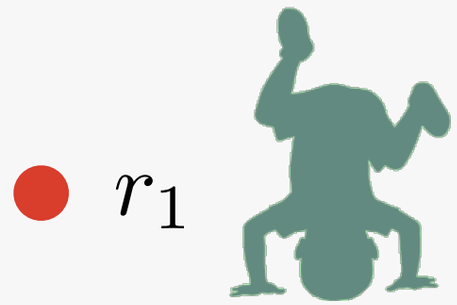
r_2

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

- measure in X $(|+\rangle, |-\rangle)$ or Y basis $(|0\rangle + i|1\rangle, |0\rangle - i|1\rangle)$

Quantumly sharing classical secrets

$k \in \{0, 1\}$
$r_1 \oplus r_2$



r_1

r_2

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

- measure in X $(|+\rangle, |-\rangle)$ or Y basis $(|0\rangle + i|1\rangle, |0\rangle - i|1\rangle)$
- publicly share the measurement basis (but not the result)

Example: X X X measurement

Example: X X X measurement

- Suppose everyone measures in X basis (prob 1/8):

Example: X X X measurement

- Suppose everyone measures in X basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

Example: X X X measurement

- Suppose everyone measures in X basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle + |-\rangle)^{\otimes 3} + (|+\rangle - |-\rangle)^{\otimes 3})$$

|0> = (|+> + |->) / √2
|1> = (|+> - |->) / √2

Example: X X X measurement

- Suppose everyone measures in X basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle + |-\rangle)^{\otimes 3} + (|+\rangle - |-\rangle)^{\otimes 3})$$

$$= \frac{1}{2}(|+++\rangle + |+--\rangle + |-+-\rangle + |--+\rangle)$$

|0> = (|+> + |->) / √2
|1> = (|+> - |->) / √2

INTER-FERENCE

Example: X X X measurement

- Suppose everyone measures in X basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle + |-\rangle)^{\otimes 3} + (|+\rangle - |-\rangle)^{\otimes 3})$$

$$= \frac{1}{2}(|+++\rangle + |+--\rangle + |-+-\rangle + |--+\rangle)$$

|0> = (|+> + |->) / √2
|1> = (|+> - |->) / √2

INTER-FERENCE

$$|r_1 \oplus r_2 \quad r_1 \quad r_2\rangle$$

Example: X Y Y measurement

Example: X Y Y measurement

- Suppose Alice measures in X basis, Bob and Charlie in Y basis (prob 1/8):

Example: X Y Y measurement

- Suppose Alice measures in X basis, Bob and Charlie in Y basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

Example: X Y Y measurement

- Suppose Alice measures in X basis, Bob and Charlie in Y basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle - |-\rangle) \otimes (|0_Y\rangle + |1_Y\rangle)^{\otimes 2}$$

$$+(|+\rangle - |-\rangle) \otimes (-i|0_Y\rangle + i|1_Y\rangle)^{\otimes 2}$$

$|0\rangle = |+\rangle + |-\rangle$

$|1\rangle = |+\rangle - |-\rangle$

$|0\rangle = |0_Y\rangle + |1_Y\rangle$

$|1\rangle = -i\,|0_Y\rangle + i\,|1_Y\rangle$

Example: X Y Y measurement

- Suppose Alice measures in X basis, Bob and Charlie in Y basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle - |-\rangle) \otimes (|0_Y\rangle + |1_Y\rangle)^{\otimes 2}$$

$$+(|+\rangle - |-\rangle) \otimes (-i|0_Y\rangle + i|1_Y\rangle)^{\otimes 2}$$

$$= \frac{1}{2}(|+0_Y 1_Y\rangle + |+1_Y 0_Y\rangle + |-0_Y 0_Y\rangle + |-1_Y 1_Y\rangle)$$

|0> = |+> + |->
|1> = |+> - |->
|0> = |0Y> + |1Y>
|1>= -i |0Y> + i |1Y>

INTER-FERENCE

Example: X Y Y measurement

- Suppose Alice measures in X basis, Bob and Charlie in Y basis (prob 1/8):

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes 3} + |1\rangle^{\otimes 3})$$

$$= \frac{1}{4}((|+\rangle - |-\rangle) \otimes (|0_Y\rangle + |1_Y\rangle)^{\otimes 2}$$

$$+(|+\rangle - |-\rangle) \otimes (-i|0_Y\rangle + i|1_Y\rangle)^{\otimes 2}$$

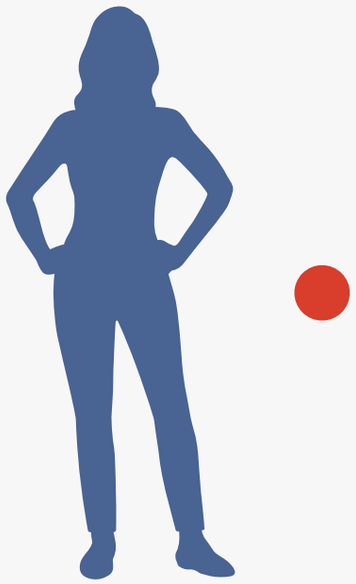
$$= \frac{1}{2}(|+0_Y 1_Y\rangle + |+1_Y 0_Y\rangle + |-0_Y 0_Y\rangle + |-1_Y 1_Y\rangle)$$

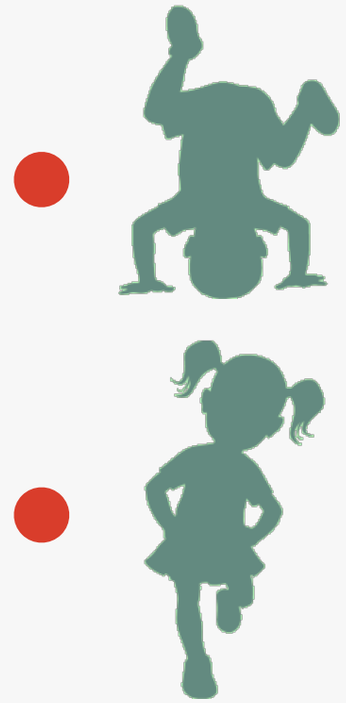
$$|r_1 \oplus r_2 \oplus 1 \quad r_1 \quad r_2\rangle$$

|0> = |+> + |->
|1> = |+> - |->
|0> = |0Y> + |1Y>
|1>= -i |0Y> + i |1Y>

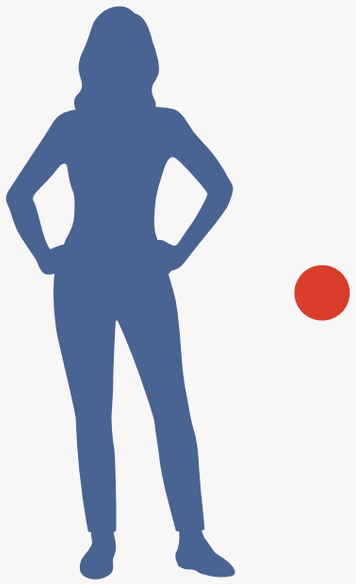
INTER-FERENCE

Quantumly sharing classical secrets



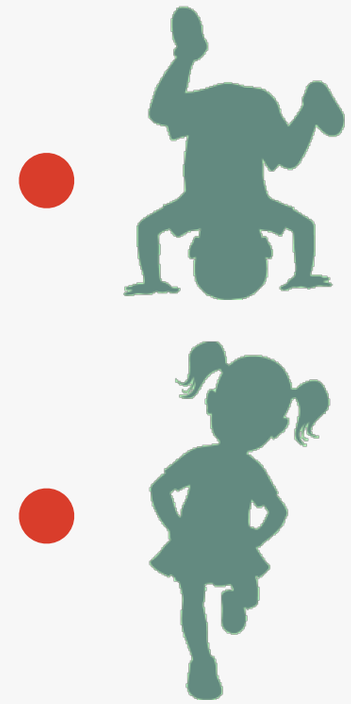


Quantumly sharing classical secrets

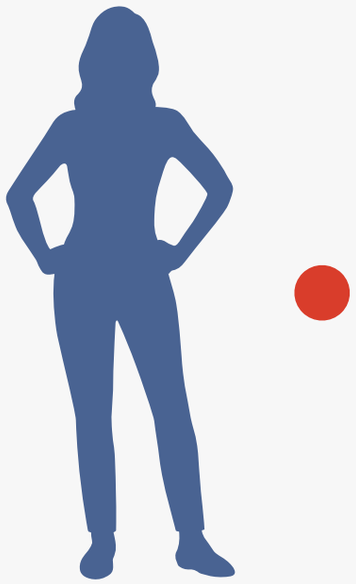


✓ measurements with correlation:

✗ measurements without correlation:



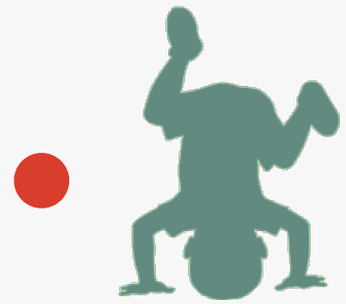
Quantumly sharing classical secrets

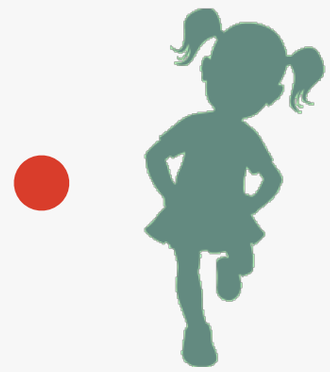


✓ measurements with correlation:

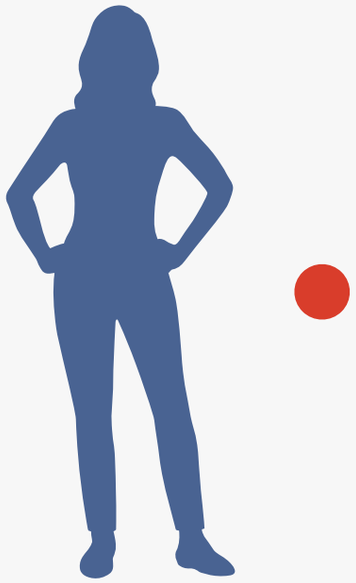
XXX
XYY
YXY
YYX

✗ measurements without correlation:





Quantumly sharing classical secrets

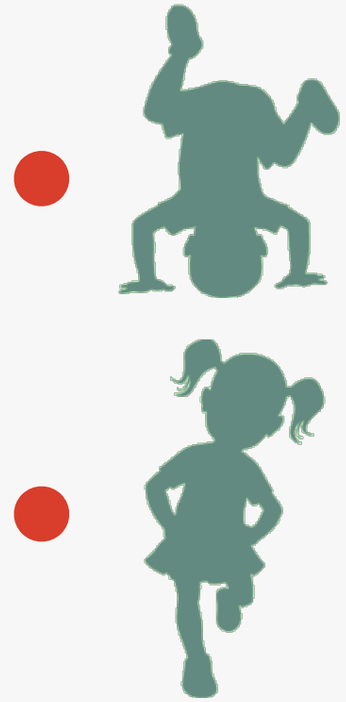


✓ measurements with correlation:

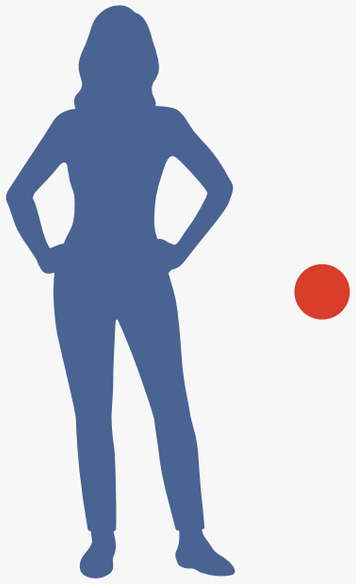
XXX
XYY
YXY
YYX

✗ measurements without correlation:

XXY
XYX
YXX
YYY



Quantumly sharing classical secrets



✓ measurements with correlation:

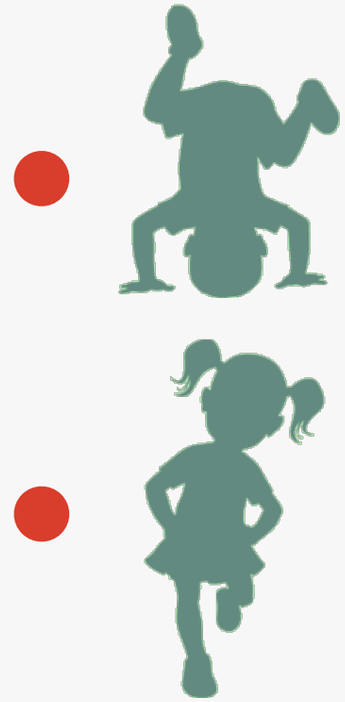
XXX
XYY
YXY
YYX

prob. 1/2

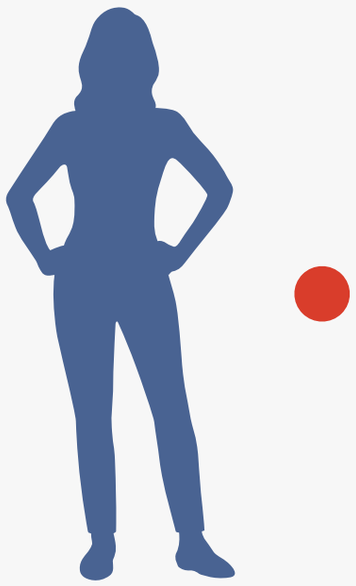
✗ measurements without correlation:

XXY
XYX
YXX
YYY

prob. 1/2



Quantumly sharing classical secrets



✓ measurements with correlation:

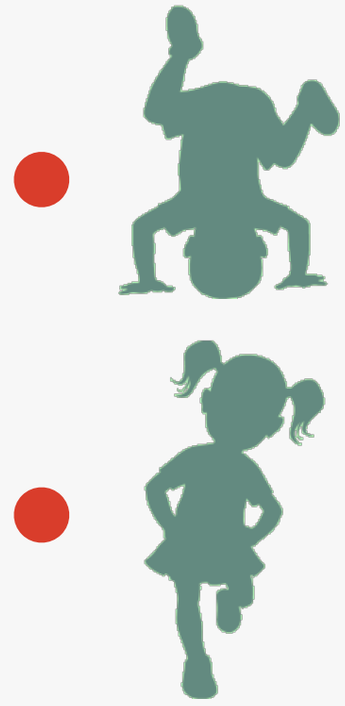
XXX
XYY
YXY
YYX

prob. 1/2

✗ measurements without correlation:

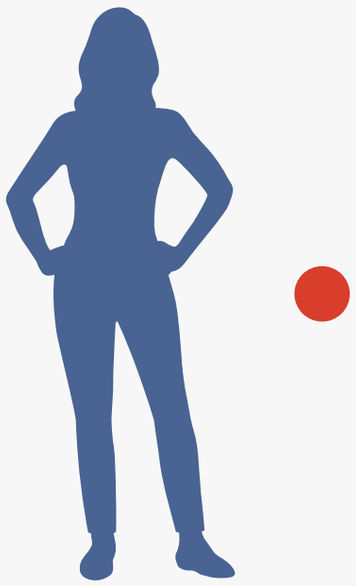
XXY
XYX
YXX
YYY

prob. 1/2



- If the bases don’t match: try again (**amplification**)

Quantumly sharing classical secrets



✓ measurements with correlation:

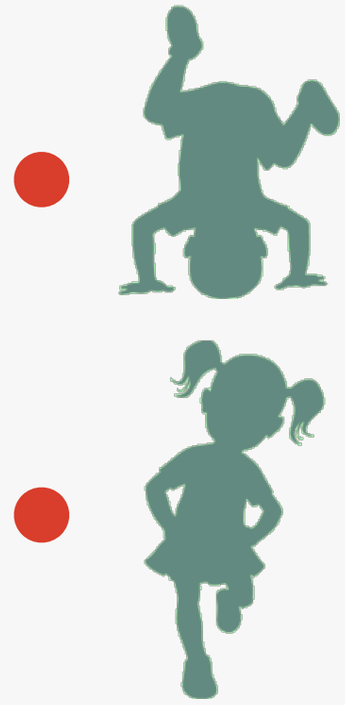
XXX
XYY
YXY
YYX

prob. 1/2

✗ measurements without correlation:

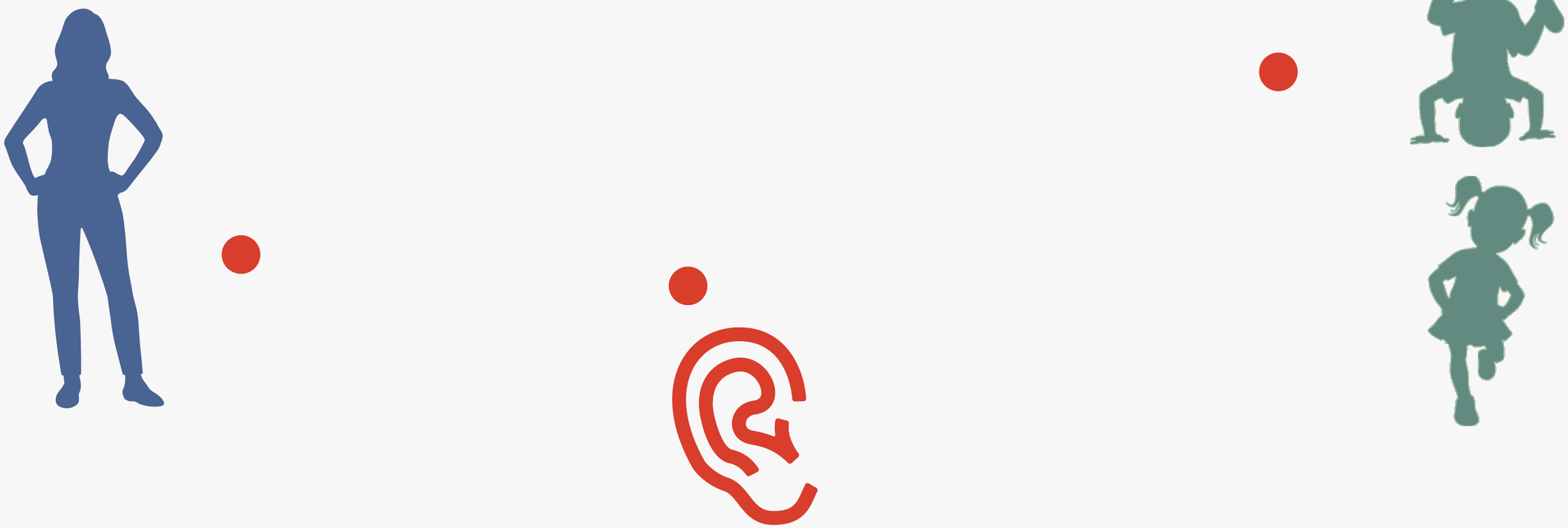
XXY
XYX
YXX
YYY

prob. 1/2



- If the bases don’t match: try again (**amplification**)
- What about eavesdropping?

Quantumly sharing classical secrets



- If the bases don’t match: try again (**amplification**)
- What about eavesdropping?

Quantumly sharing classical secrets

- If the bases don’t match: try again (**amplification**)
- What about eavesdropping?

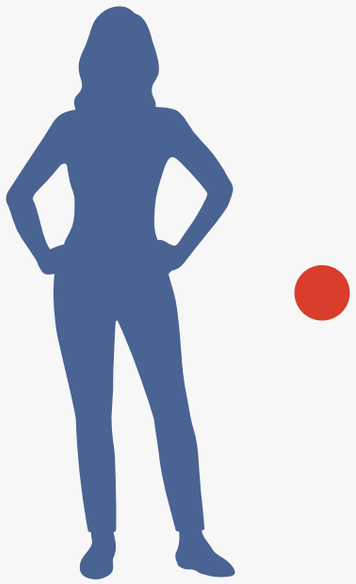
Quantumly sharing classical secrets

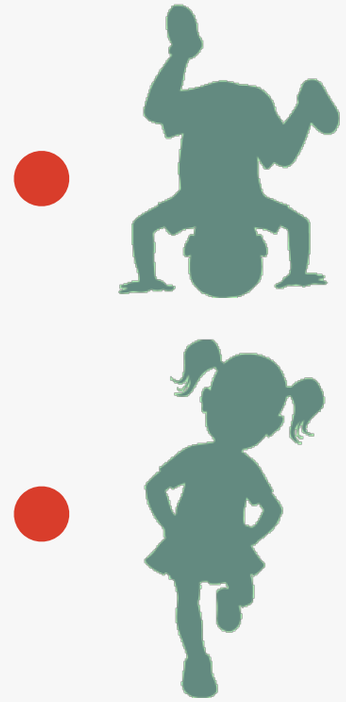
- If the bases don’t match: try again (**amplification**)
- What about eavesdropping?

Quantumly sharing classical secrets

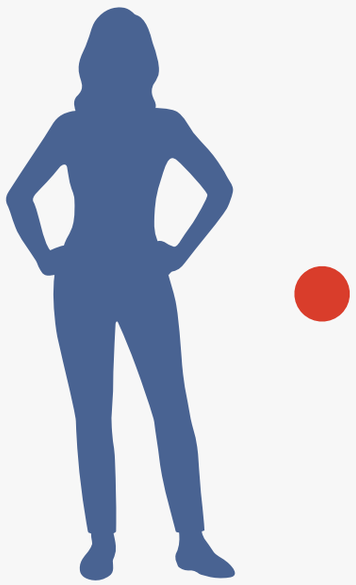
- If the bases don’t match: try again (**amplification**)
- What about eavesdropping?

Eavesdropper protection: test rounds

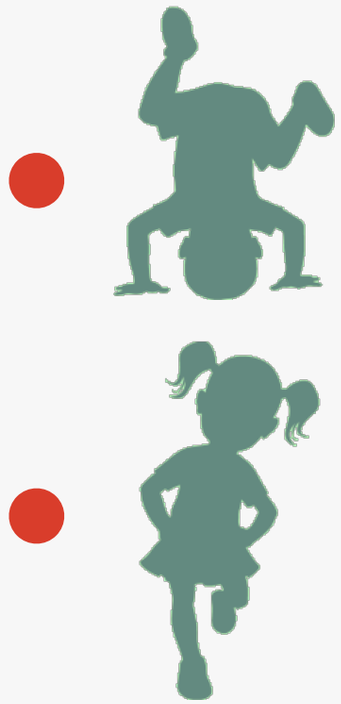




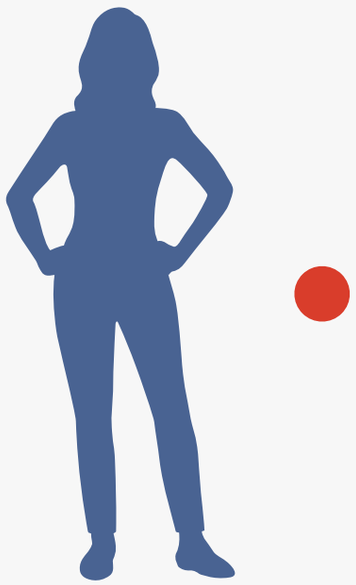
Eavesdropper protection: test rounds



Test round: same as regular round



Eavesdropper protection: test rounds

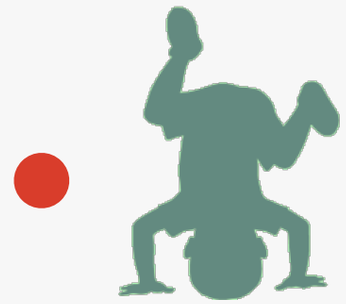


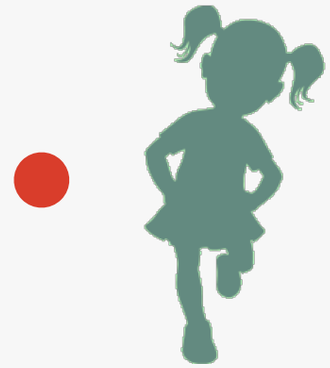
Test round: same as regular round

1/2

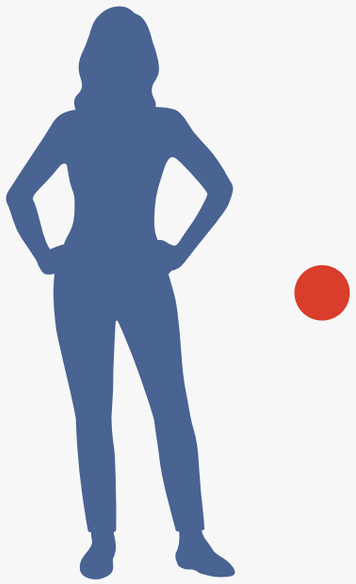
uncorrelated bases:

pass





Eavesdropper protection: test rounds



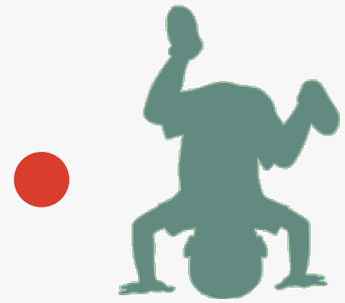
Test round: same as regular round

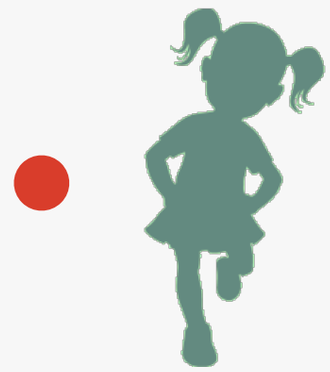
1/2

correlated bases:
share measurement results

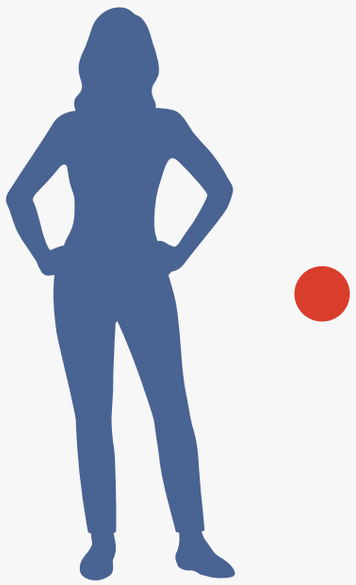
1/2

uncorrelated bases:
pass





Eavesdropper protection: test rounds



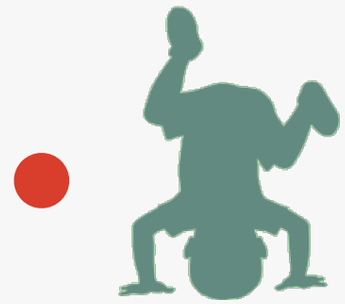
Test round: same as regular round

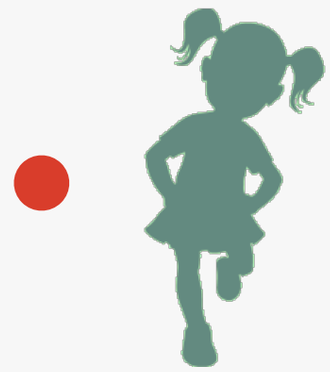
1/2

correlated bases:
share measurement results

1/2

uncorrelated bases:
pass



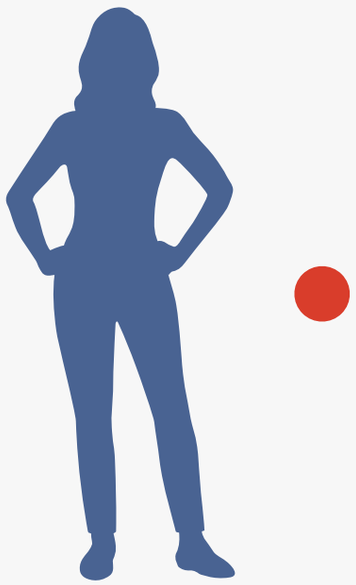


1/2

$$\text{dealer's result} = r_1 \oplus r_2$$

pass

Eavesdropper protection: test rounds



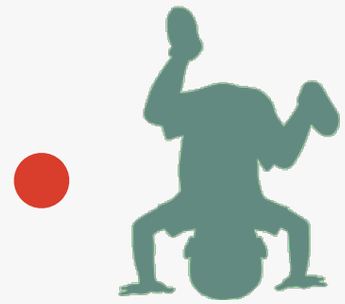
Test round: same as regular round

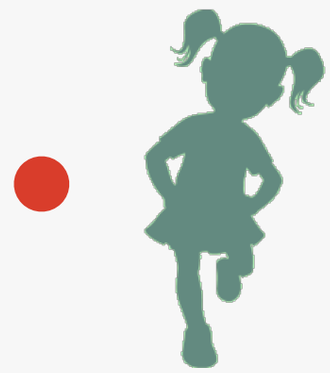
1/2

correlated bases:
share measurement results

1/2

uncorrelated bases:
pass





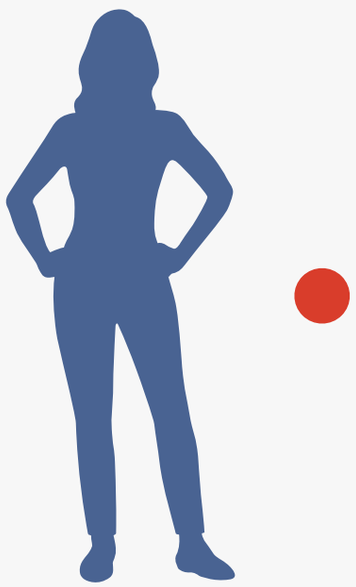
1/2

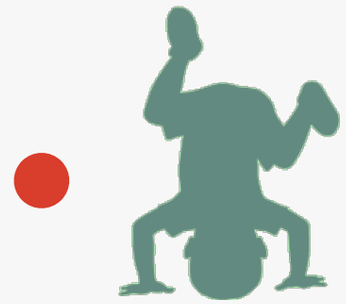
$\text{dealer's result} = r_1 \oplus r_2$
pass

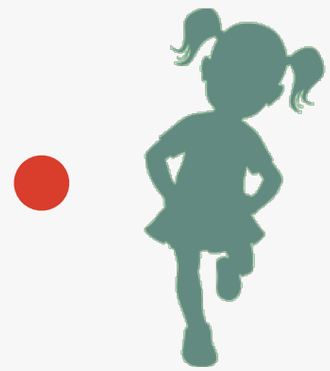
1/2

$\text{dealer's result} \neq r_1 \oplus r_2$
fail

Eavesdropper protection: test rounds







Test round: same as regular round

1/2

correlated bases:
share measurement results

1/2

uncorrelated bases:
pass

1/2

$\text{dealer's result} = r_1 \oplus r_2$
pass

1/2

$\text{dealer's result} \neq r_1 \oplus r_2$
fail

If the eavesdropper measures in a different basis than C, he is detected with probability 1/4. **Repeat** to **amplify**!

Summary (Primitive III)

Summary (Primitive III)

- Sharing classical secrets: only if t out of n parties come together, they can recover the secret

Summary (Primitive III)

- Sharing classical secrets: only if t out of n parties come together, they can recover the secret
- Distributing shares can be done using the GHZ state.

Summary (Primitive III)

- Sharing classical secrets: only if t out of n parties come together, they can recover the secret
- Distributing shares can be done using the GHZ state.
- We used **entanglement** to ensure correlation between the shares.

Summary (Primitive III)

- Sharing classical secrets: only if t out of n parties come together, they can recover the secret
- Distributing shares can be done using the GHZ state.
- We used **entanglement** to ensure correlation between the shares.
- We used the fact that **measurements disturb quantum states** to protect against eavesdropping.

Summary (Primitive III)

- Sharing classical secrets: only if t out of n parties come together, they can recover the secret
- Distributing shares can be done using the GHZ state.
  - We used **entanglement** to ensure correlation between the shares.
  - We used the fact that **measurements disturb quantum states** to protect against eavesdropping.
- Side note: it is also possible to share **quantum secrets**.

Application: Cloud Computing

Quantum Cloud Computing [Chi05]

[Chi05] Childs: Secure Assisted Quantum Computation

Quantum Cloud Computing [Chi05]

[Chi05] Childs: Secure Assisted Quantum Computation

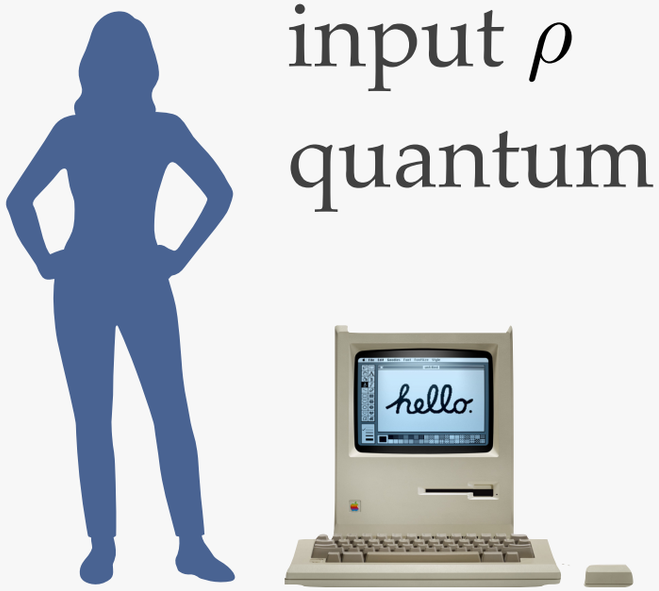
Quantum Cloud Computing [Chi05]

input ρ

quantum circuit C

[Chi05] Childs: Secure Assisted Quantum Computation

Quantum Cloud Computing [Chi05]

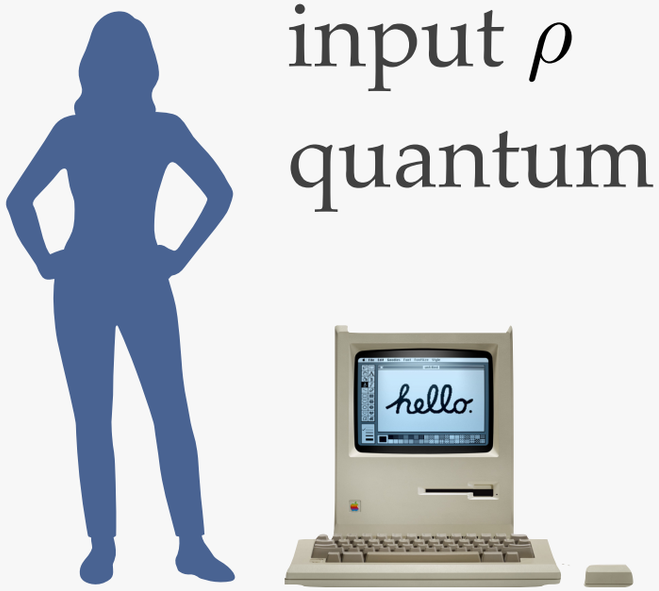


input ρ

quantum circuit C

[Chi05] Childs: Secure Assisted Quantum Computation

Quantum Cloud Computing [Chi05]

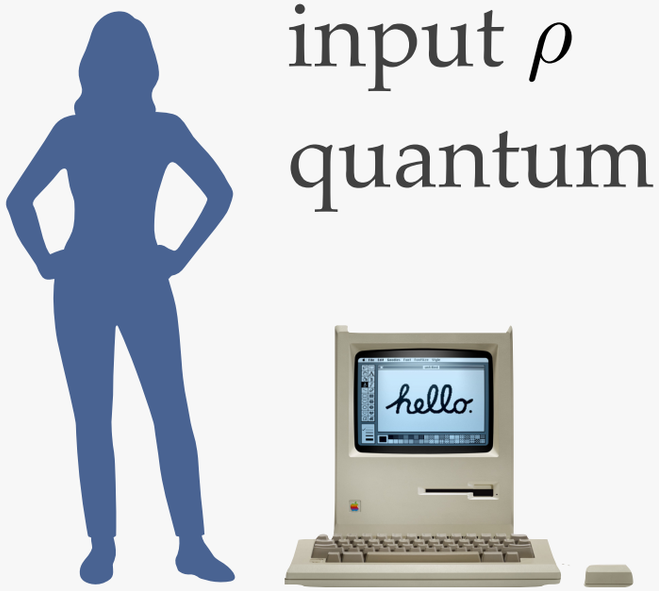


input ρ

quantum circuit C

[Chi05] Childs: Secure Assisted Quantum Computation

Quantum Cloud Computing [Chi05]

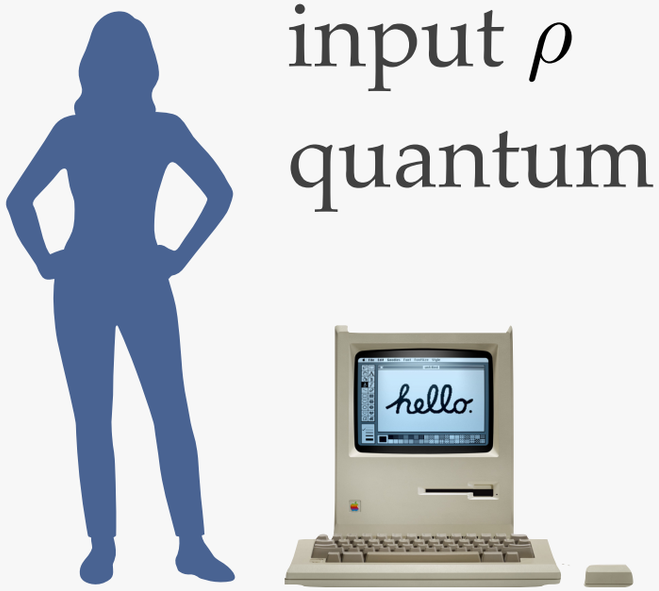


input ρ

quantum circuit C

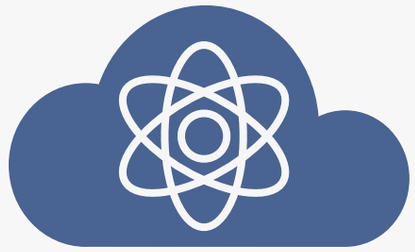
- Can Alice let the cloud perform C on ρ, without giving away any information about ρ (privacy)?

Quantum Cloud Computing [Chi05]



input ρ

quantum circuit C



- Can Alice let the cloud perform C on ρ, without giving away any information about ρ (privacy)?
- Can she do so efficiently?

[Chi05] Childs: Secure Assisted Quantum Computation

Q. cloud computing: variations

Q. cloud computing: variations

- No communication during the computation: (quantum) homomorphic encryption [DSS16]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits

Q. cloud computing: variations

- No communication during the computation: (quantum) homomorphic encryption [**D**SS16]
- Completely classical client [CGJV17]

[**D**SS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources

Q. cloud computing: variations

- No communication during the computation: (quantum) homomorphic encryption [**D**SS16]
- Completely classical client [CGJV17]
- The cloud server cannot learn the circuit/program: blind (quantum) computing [AS06]

[**D**SS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
- Completely classical client [CGJV17]
- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]
- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Preliminary: Clifford group

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g.
$$HIH^\dagger = I$$
$$HXH^\dagger = Z$$
$$HZH^\dagger = X$$
$$HYH^\dagger = -Y$$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g. $$\left.\begin{array}{c} HIH^\dagger = I \\ HXH^\dagger = Z \\ HZH^\dagger = X \\ HYH^\dagger = -Y \end{array}\right\} \; H \in \mathcal{C}_1$$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g. $$\left.\begin{array}{c} HIH^\dagger = I \\ HXH^\dagger = Z \\ HZH^\dagger = X \\ HYH^\dagger = -Y \end{array}\right\} \; H \in \mathcal{C}_1$$

$$\begin{aligned} HI &= IH \\ HX &= ZH \\ HZ &= XH \\ HY &= YH \end{aligned}$$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g. $$\left.\begin{array}{c} HIH^\dagger = I \\ HXH^\dagger = Z \\ HZH^\dagger = X \\ HYH^\dagger = -Y \end{array}\right\} \; H \in \mathcal{C}_1$$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g. $$\left.\begin{array}{c} HIH^\dagger = I \\ HXH^\dagger = Z \\ HZH^\dagger = X \\ HYH^\dagger = -Y \end{array}\right\} \; H \in \mathcal{C}_1$$

- $\mathcal{C}_n$ is generated by $H,\ CNOT,\ P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$

Preliminary: Clifford group

- Recall the Pauli group: $\mathcal{P}_1 := \{\pm(i)I, \pm(i)X, \pm(i)Y, \pm(i)Z\}$
- Clifford group: gates that commute with Pauli group:

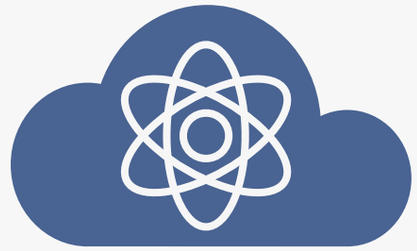
$$\mathcal{C}_n := \{U \in U(2^n) \mid \forall P \in \mathcal{P}_n : UPU^\dagger \in \mathcal{P}_n\}$$

e.g. $\left.\begin{array}{c} HIH^\dagger = I \\ HXH^\dagger = Z \\ HZH^\dagger = X \\ HYH^\dagger = -Y \end{array}\right\} \; H \in \mathcal{C}_1$

- $\mathcal{C}_n$ is generated by $H,\ CNOT,\ P = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
- For universal quantum computation, add $T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\pi i/4} \end{bmatrix}$

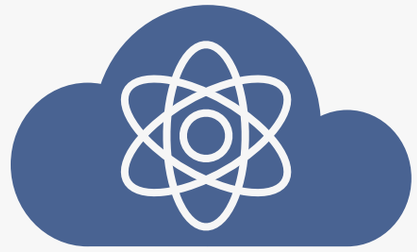
Cloud computing (for Cliffords)

input $|\psi\rangle$ (1 qubit)
quantum circuit C



Cloud computing (for Cliffords)

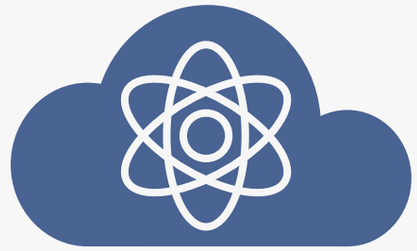
input $|\psi\rangle$ (1 qubit)
quantum circuit C



1. Alice encrypts $|\psi\rangle$ with one-time pad: $X^a Z^b |\psi\rangle$
(key: (a, b))

Cloud computing (for Cliffords)

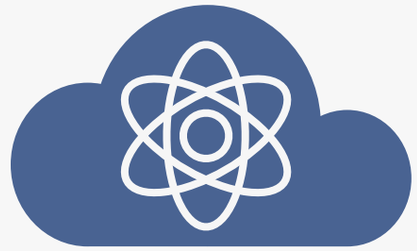
input $|\psi\rangle$ (1 qubit)
quantum circuit C



1. Alice encrypts $|\psi\rangle$ with one-time pad: $X^a Z^b |\psi\rangle$
   (key: (a, b))
2. For every gate G in circuit C:

Cloud computing (for Cliffords)

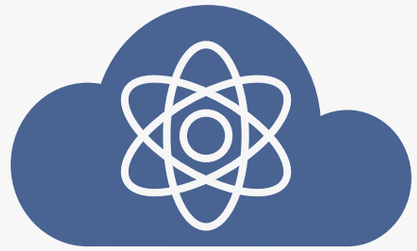
input $|\psi\rangle$ (1 qubit)
quantum circuit C



1. Alice encrypts $|\psi\rangle$ with one-time pad: $X^a Z^b |\psi\rangle$
   (key: (a, b))
2. For every gate G in circuit C:
   1. Server applies G to the ciphertext: $GX^a Z^b |\psi\rangle$

Cloud computing (for Cliffords)

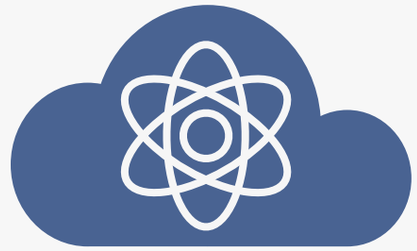
input $|\psi\rangle$ (1 qubit)
quantum circuit C



1. Alice encrypts $|\psi\rangle$ with one-time pad: $X^a Z^b |\psi\rangle$ (key: (a, b))
2. For every gate G in circuit C:
    1. Server applies G to the ciphertext: $GX^a Z^b |\psi\rangle$
    2. Alice updates her key $GX^a Z^b |\psi\rangle = X^{a'} Z^{b'} G |\psi\rangle$

Cloud computing (for Cliffords)

input $|\psi\rangle$ (1 qubit)
quantum circuit C



1. Alice encrypts $|\psi\rangle$ with one-time pad: $X^a Z^b |\psi\rangle$
   (key: (a, b))
2. For every gate G in circuit C:
   1. Server applies G to the ciphertext: $GX^a Z^b |\psi\rangle$
   2. Alice updates her key $GX^a Z^b |\psi\rangle = X^{a'} Z^{b'} G |\psi\rangle$
3. Alice decrypts one-time pad with updated key

Cloud computing (for non-Clifford)

Cloud computing (for non-Clifford)

- Clifford gates were “easy” because they commute with the quantum one-time pad

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a
- Childs 2005: server sends qubit to Alice, who swaps it out whenever $a = 0$. Server always applies P-correction.

Cloud computing (for non-Clifford)

- Clifford gates were "easy" because they commute with the quantum one-time pad
- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a
- Childs 2005: server sends qubit to Alice, who swaps it out whenever $a = 0$. Server always applies P-correction.
- Alternative: magic-state computation

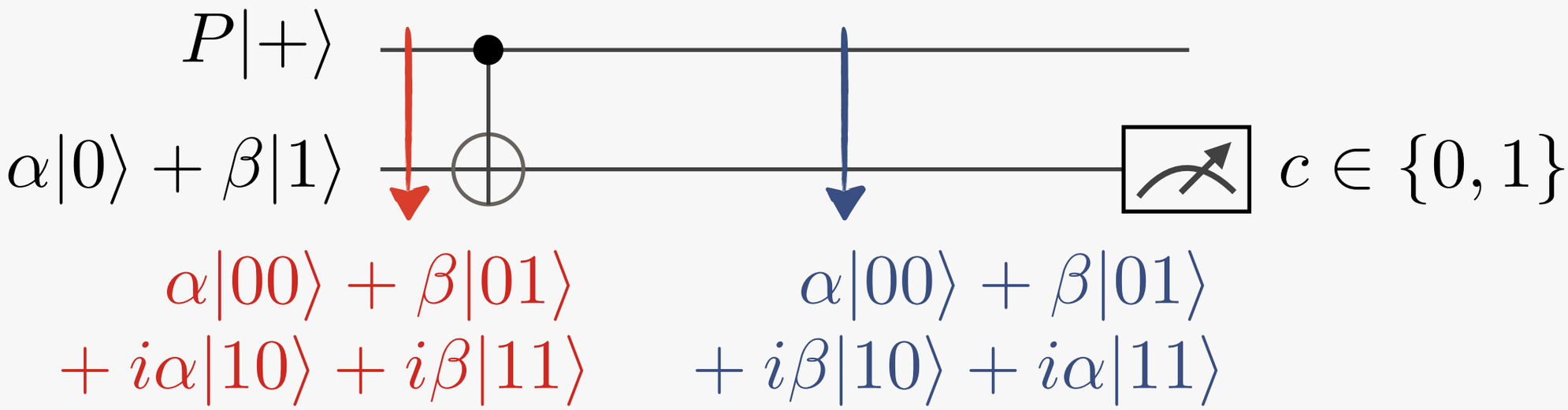
Magic-state computation

Magic-state computation

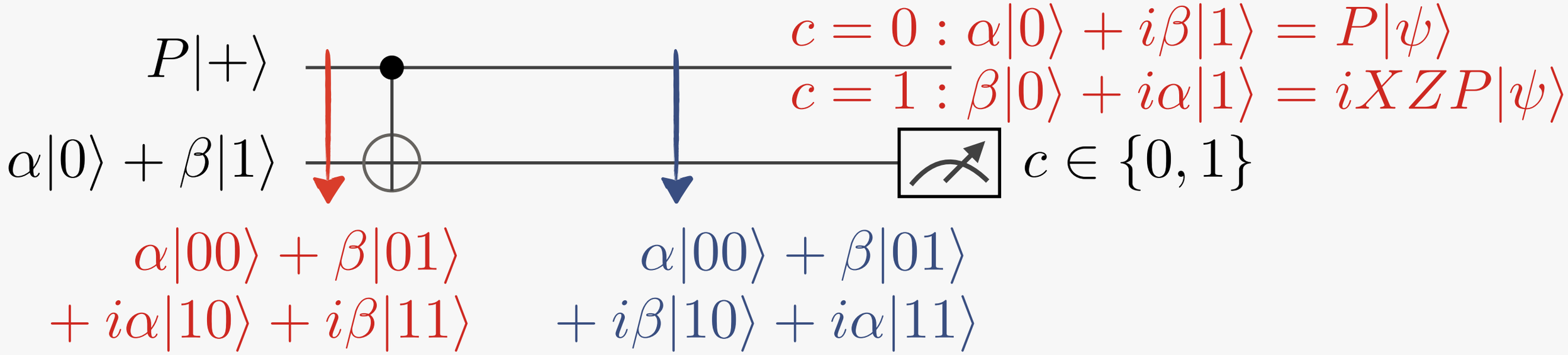
Magic-state computation

$$\alpha|00\rangle + \beta|01\rangle$$
$$+ i\alpha|10\rangle + i\beta|11\rangle$$

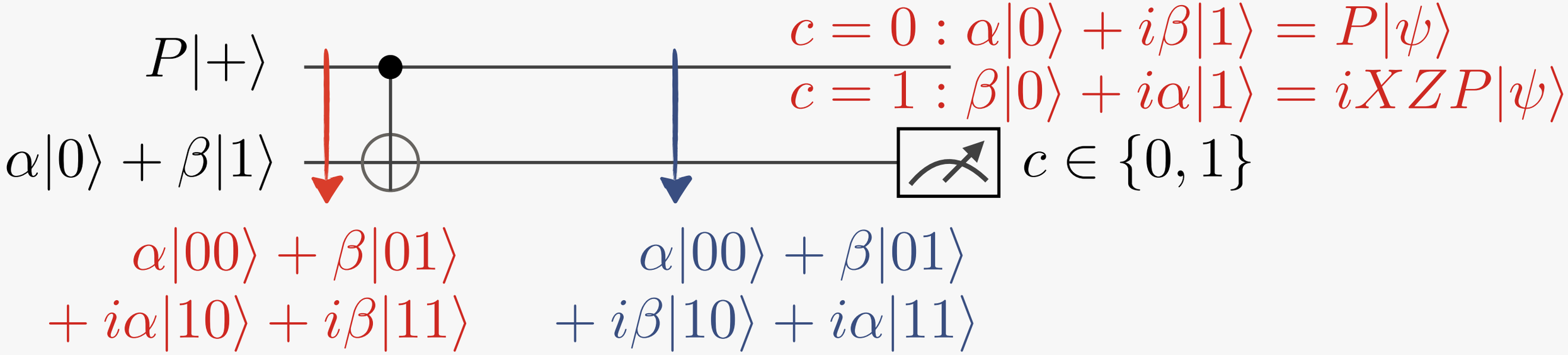
Magic-state computation



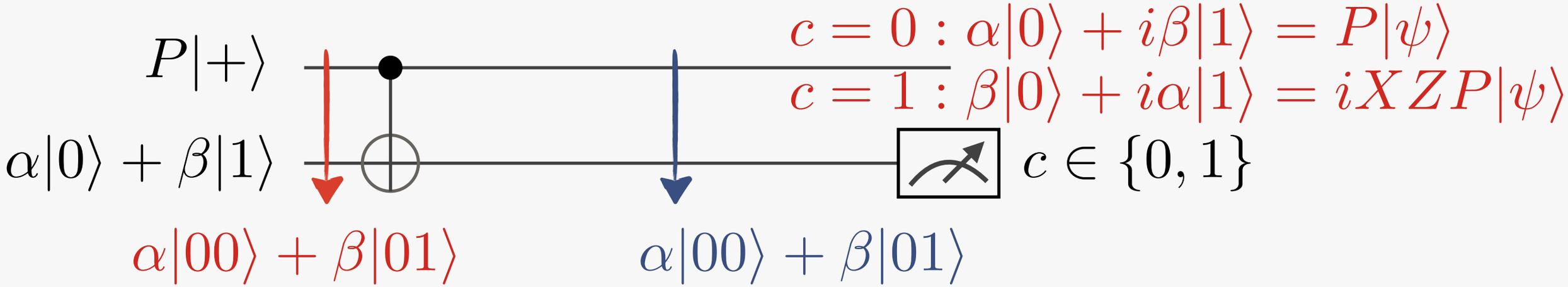
Magic-state computation

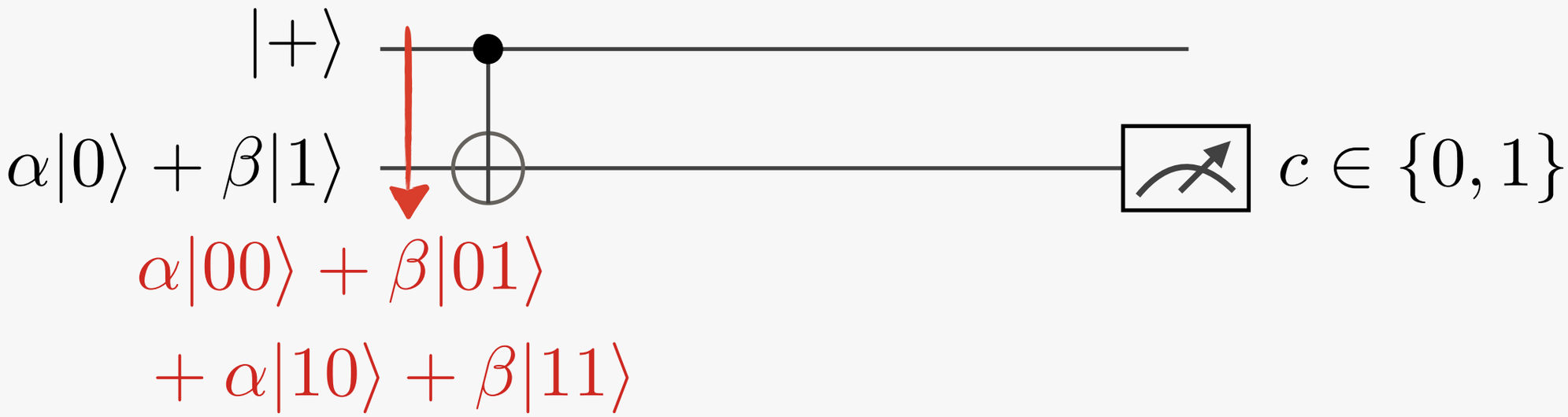


Magic-state computation

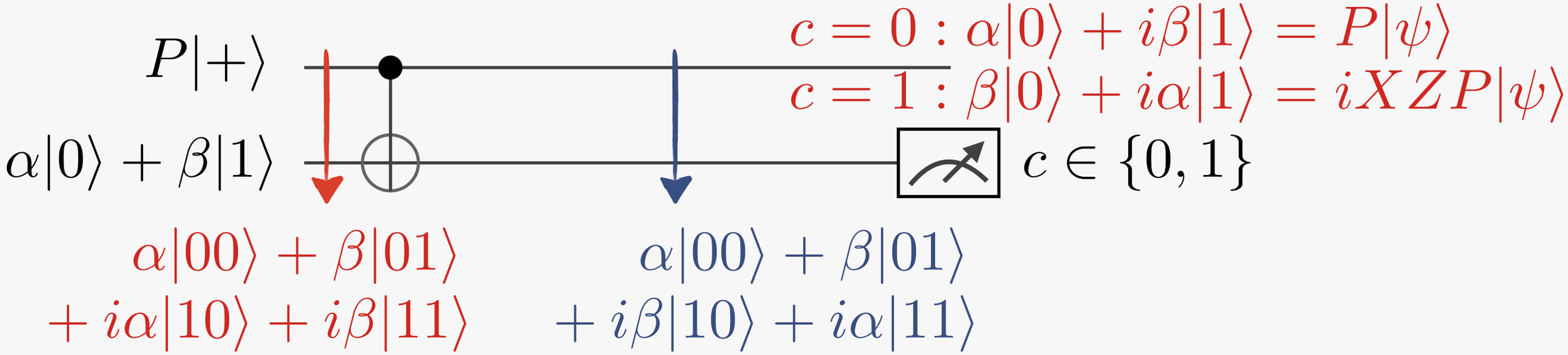


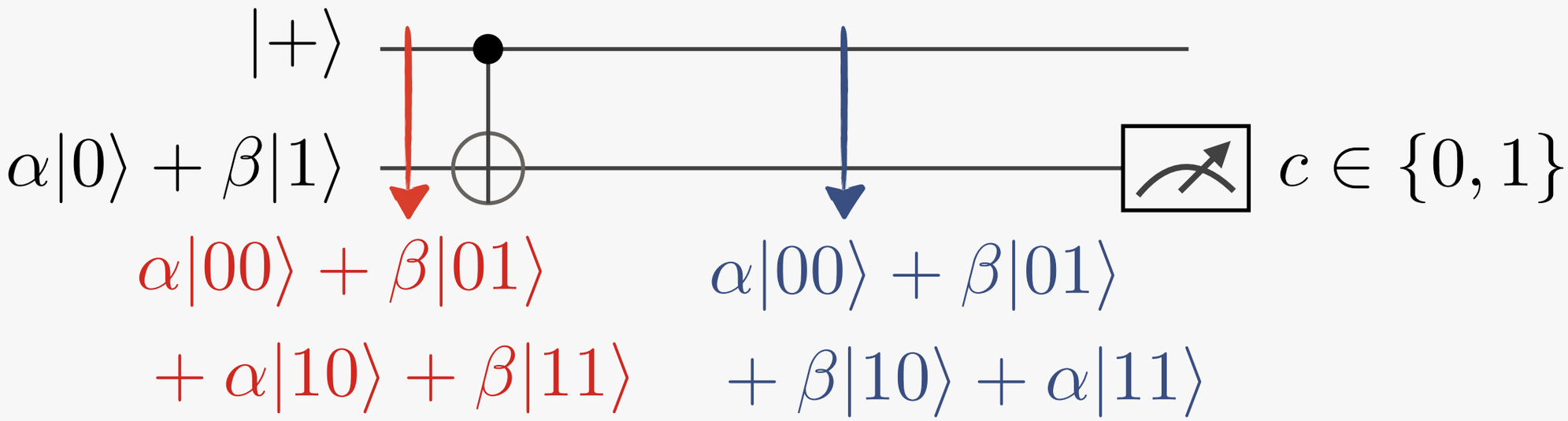
Magic-state computation



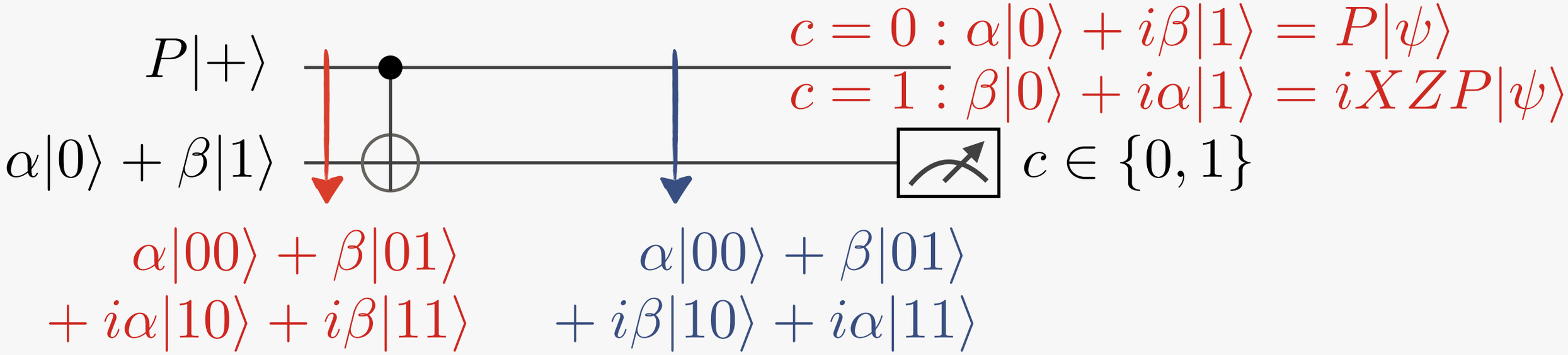


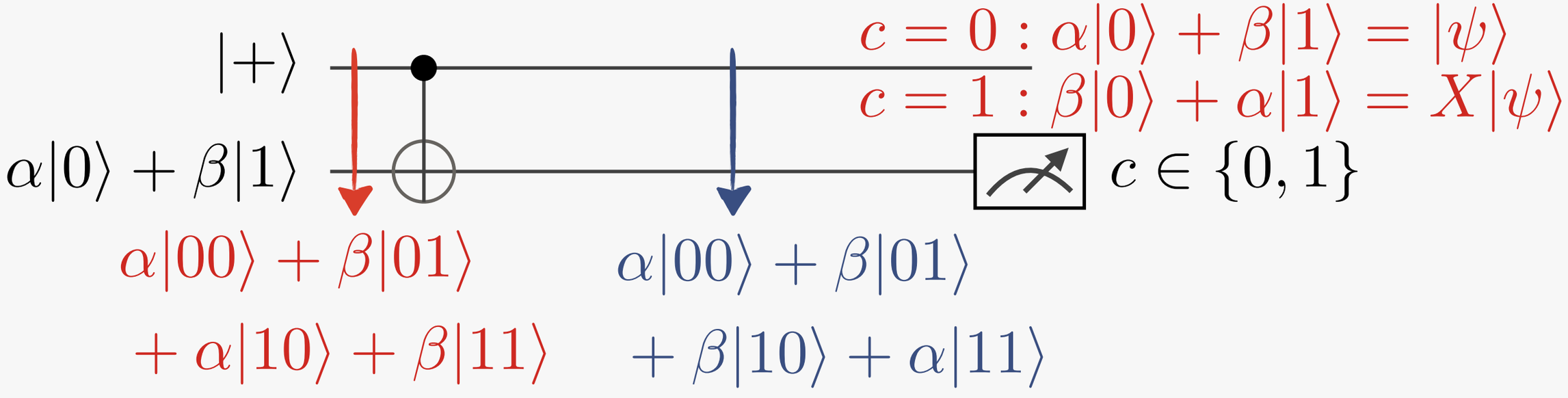
Magic-state computation



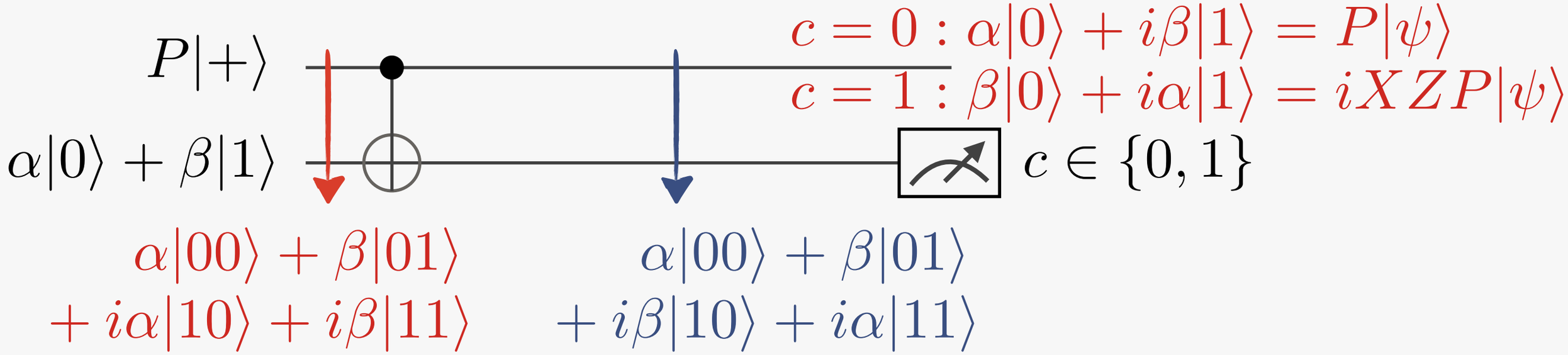


Magic-state computation

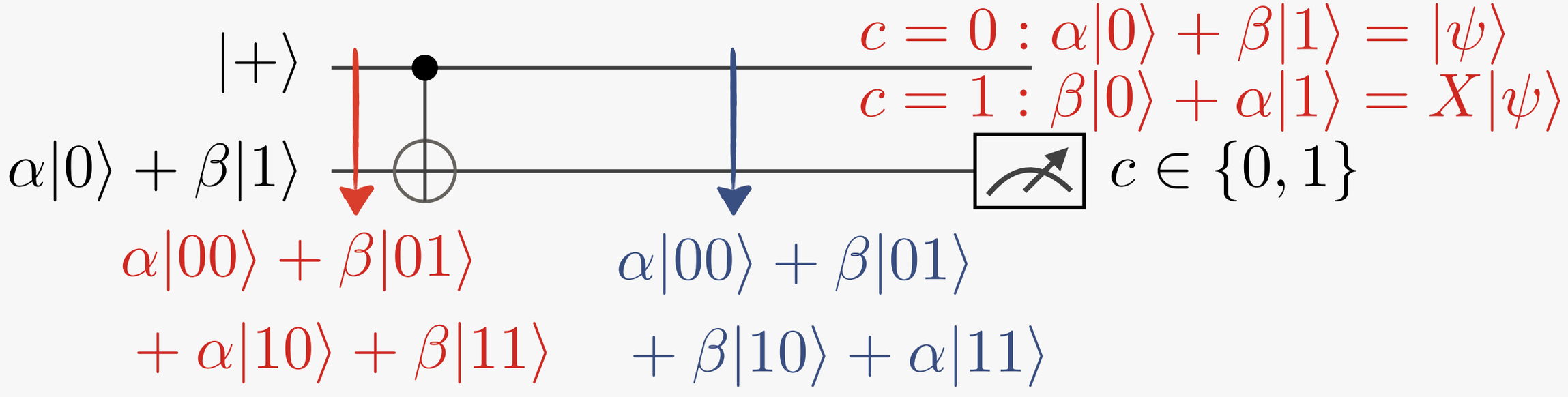




Magic-state computation



$P|+\rangle$ applies P, $|+\rangle$ applies identity (up to Paulis)



Cloud computing (for non-Clifford)

- T-gate: $TX^aZ^b|\psi\rangle = P^aX^aZ^bT|\psi\rangle$

  ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a

Cloud computing (for non-Clifford)

- T-gate: $TX^aZ^b|\psi\rangle = P^a X^a Z^b T|\psi\rangle$

  ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a

- Alice sends (encrypted version of) $P^a|+\rangle$

Cloud computing (for non-Clifford)

- T-gate: $TX^aZ^b|\psi\rangle = P^a X^a Z^b T|\psi\rangle$

  ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a

- Alice sends (encrypted version of) $P^a|+\rangle$

- Server performs CNOT and measurement, and sends the result ($c \in \{0,1\}$) to Alice

Cloud computing (for non-Clifford)

- T-gate: $TX^aZ^b|\psi\rangle = P^a X^a Z^b T|\psi\rangle$

  ERROR ENCRYPTED OUTPUT

- Server does not (and should not) know a

- Alice sends (encrypted version of) $P^a|+\rangle$

- Server performs CNOT and measurement, and sends the result ($c \in \{0,1\}$) to Alice

- Alice updates keys for CNOT and measurement outcome

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
- Completely classical client [CGJV17]
- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]
- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]

- Completely classical client [CGJV17]

- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]

- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
  - Prepare magic states in advance for a = 0 and a = 1
- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]
- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
    - Prepare magic states in advance for a = 0 and a = 1
    - Entangle them in a clever way such that the server, using Enc(a) only, can select the correct one

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
- Completely classical client [CGJV17]
- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]
- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]

- Completely classical client [CGJV17]

- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]

- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]

- Completely classical client [CGJV17]
  - Talk to two (noncommunicating) cloud servers

- The client sends the (classical) program
  blind (quantum) computing [AS06]

- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- Completely classical client [CGJV17]
  - Talk to two (noncommunicating) cloud servers
  - Use the second server to prepare the correct magic states

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]
- Completely classical client [CGJV17]
- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]
- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[DSS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Q. cloud computing: variations

- No communication during the computation:
  (quantum) homomorphic encryption [DSS16]

- Completely classical client [CGJV17]

- The cloud server cannot learn the circuit/program:
  blind (quantum) computing [AS06]

- Verification of the result:
  Quantum computing on authenticated data [Bro15]

[**D**SS16] Dulek et al.: Quantum Homomorphic Encryption for Polynomial-Size Circuits
[CGJV17] Coladangelo et al.: Verifier-on-a-Leash: new schemes for verifiable delegated quantum computation, with quasilinear resources
[AS06] Arrighi, Salvail: Blind Quantum Computation
[Bro15] Broadbent: How to verify a quantum computation

Computing on authenticated data

Computing on authenticated data

- Verify that C was applied (and not some other circuit)

Computing on authenticated data

- Verify that C was applied (and not some other circuit)
- Replace primitive: authentication instead of encryption

Computing on authenticated data

- Verify that C was applied (and not some other circuit)
- Replace primitive: authentication instead of encryption
- Update authentication key during computation [Bro15]

[Bro15] Broadbent: How to verify a quantum computation

Computing on authenticated data

- Verify that C was applied (and not some other circuit)
- Replace primitive: authentication instead of encryption
- Update authentication key during computation [Bro15]
- Can even be combined with homomorphic encryption! [A**D**SS17]

[Bro15] Broadbent: How to verify a quantum computation
[A**D**SS17] Alagic et al.: Quantum Fully Homomorphic Encryption with Verification

Summary (Application I)

Summary (Application I)

- Alice can **outsource** her quantum computation to an (untrusted) cloud service without giving up privacy.

Summary (Application I)

- Alice can **outsource** her quantum computation to an (untrusted) cloud service without giving up privacy.
- We used magic-state computation (which relies on **entanglement**) to correct for errors during computation.

Summary (Application I)

- Alice can **outsource** her quantum computation to an (untrusted) cloud service without giving up privacy.
- We used magic-state computation (which relies on **entanglement**) to correct for errors during computation.
- We can use **more complicated entanglement** to eliminate communication entirely (homomorphic encryption)

Summary (Application I)

- Alice can **outsource** her quantum computation to an (untrusted) cloud service without giving up privacy.
- We used magic-state computation (which relies on **entanglement**) to correct for errors during computation.
- We can use **more complicated entanglement** to eliminate communication entirely (homomorphic encryption)
- We can replace the **encryption** primitive with the **authentication** primitive in order to gain verification.

Summary (morning session)

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
  - Most symmetric-key cryptography (e.g. one-time pad) remains secure

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
  - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
  - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?
  - Yes: key distribution, secret sharing (eavesdropper protection)

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
    - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?
  - Yes: key distribution, secret sharing (eavesdropper protection)
- What **quantum applications** are desirable?

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
    - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?
  - Yes: key distribution, secret sharing (eavesdropper protection)
- What **quantum applications** are desirable?
  - Cloud computing: use **quantum primitives** such as encryption, authentication

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
    - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?
  - Yes: key distribution, secret sharing (eavesdropper protection)
- What **quantum applications** are desirable?
  - Cloud computing: use **quantum primitives** such as encryption, authentication
  - Use entanglement to correct errors

Summary (morning session)

- Can quantum computing **break** cryptographic primitives?
  - Yes: pseudorandom functions used in authentication
  - Yes: public-key cryptography
    - Most symmetric-key cryptography (e.g. one-time pad) remains secure
- Can quantum computing **improve** cryptographic primitives?
  - Yes: key distribution, secret sharing (eavesdropper protection)
- What **quantum applications** are desirable?
  - Cloud computing: use **quantum primitives** such as encryption, authentication
  - Use entanglement to correct errors

Thank you for your attention!