Secure Computation

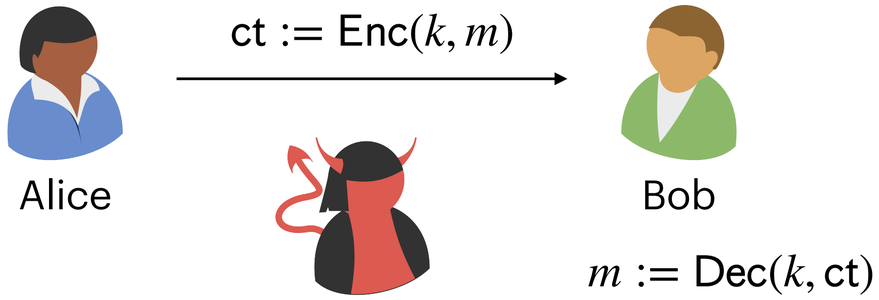
601.442/642 Modern Cryptography

16th April 2026

Recap

Recap

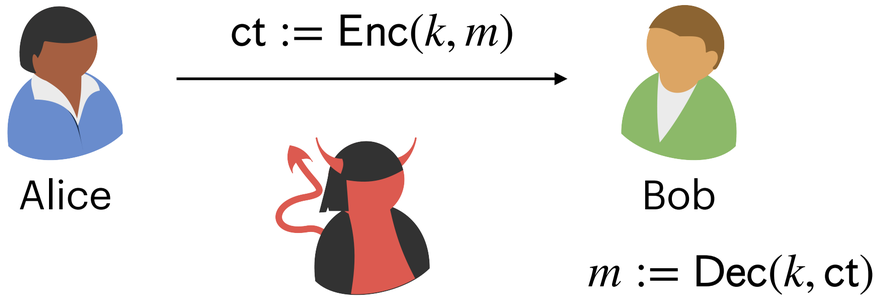
Encryption



Private communication in the presence of an external passive eavesdropper.

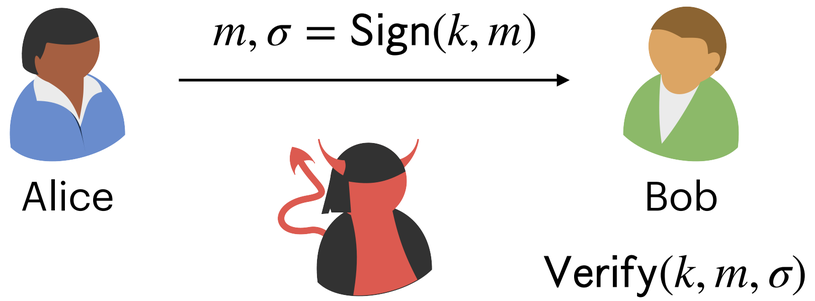
Recap

Encryption



Private communication in the presence of an external passive eavesdropper.

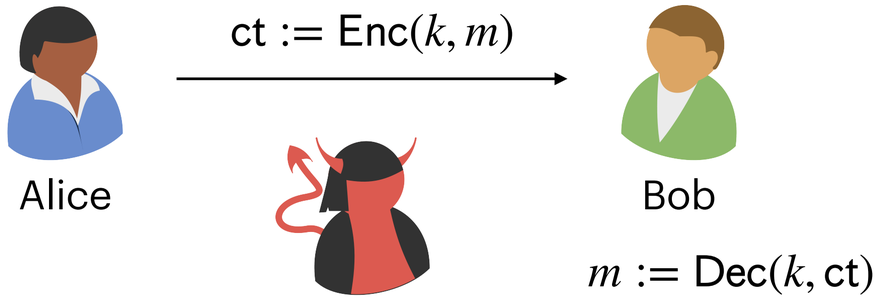
Authentication



Authentic communication in the presence of an external impersonator.

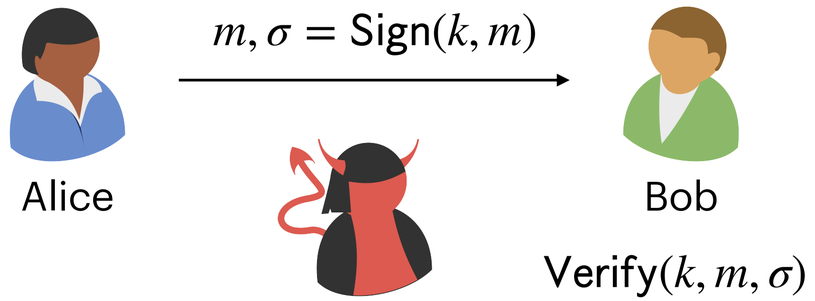
Recap

Encryption



Private communication in the presence of an external active eavesdropper.

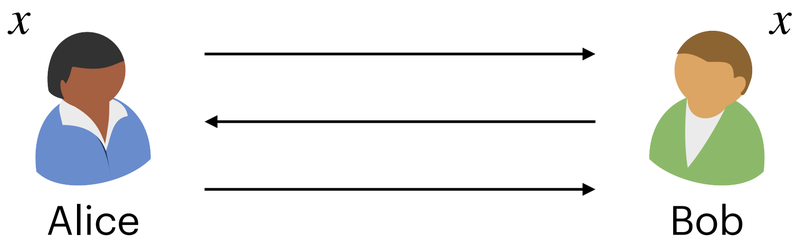
Authentication



Authentic communication in the presence of an external impersonator.

Recap

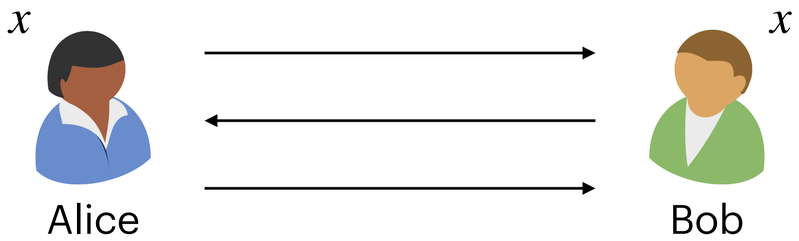
Zero Knowledge Proofs



Mutually distrusting parties. Alice wants to convince Bob of her claim.

Recap

Zero Knowledge Proofs



Mutually distrusting parties. Alice wants to convince Bob of her claim.

What else do mutually distrusting parties want to achieve?

Examples

Examples

- **Contact Discovery:** Alice registers on Signal and wants to find who among her contacts is also on Signal.
  - Alice does not wish to share her list of contacts with the server.
  - The server doesn’t want to reveal the contacts of all registered users.

Examples

- **Contact Discovery:** Alice registers on Signal and wants to find who among her contacts is also on Signal.
  - Alice does not wish to share her list of contacts with the server.
  - The server doesn’t want to reveal the contacts of all registered users.
- **Auctions:** In a sealed-bid auction, the auctioneer opens the envelopes — and can leak bids, insert phantom bids, or exploit bid data in future negotiations. Bidders must trust the auctioneer to behave honestly.

Examples

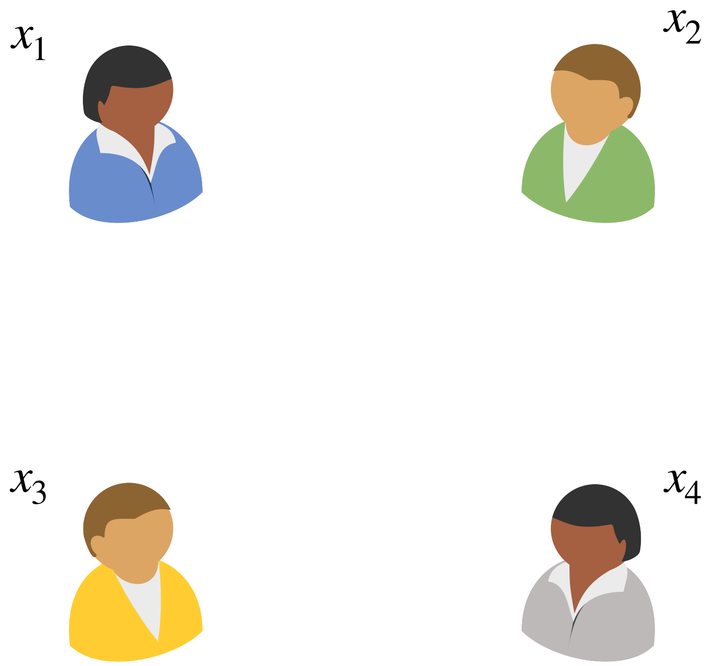
- **Contact Discovery:** Alice registers on Signal and wants to find who among her contacts is also on Signal.
  - Alice does not wish to share her list of contacts with the server.
  - The server doesn’t want to reveal the contacts of all registered users.
- **Auctions:** In a sealed-bid auction, the auctioneer opens the envelopes — and can leak bids, insert phantom bids, or exploit bid data in future negotiations. Bidders must trust the auctioneer to behave honestly.
- **Genome-Wide Association Studies:** A researcher wants to compute statistics across genomic data held by multiple hospitals, but regulations like HIPAA and GDPR prohibit the hospitals from sharing data with the researcher or with each other.

Examples

- **Contact Discovery:** Alice registers on Signal and wants to find who among her contacts is also on Signal.
  - Alice does not wish to share her list of contacts with the server.
  - The server doesn’t want to reveal the contacts of all registered users.
- **Auctions:** In a sealed-bid auction, the auctioneer opens the envelopes — and can leak bids, insert phantom bids, or exploit bid data in future negotiations. Bidders must trust the auctioneer to behave honestly.
- **Genome-Wide Association Studies:** A researcher wants to compute statistics across genomic data held by multiple hospitals, but regulations like HIPAA and GDPR prohibit the hospitals from sharing data with the researcher or with each other.
- **Threshold Signing:** A signing key stored on one server is a single point of failure. Splitting the key across multiple servers is safer. But then how do you sign?

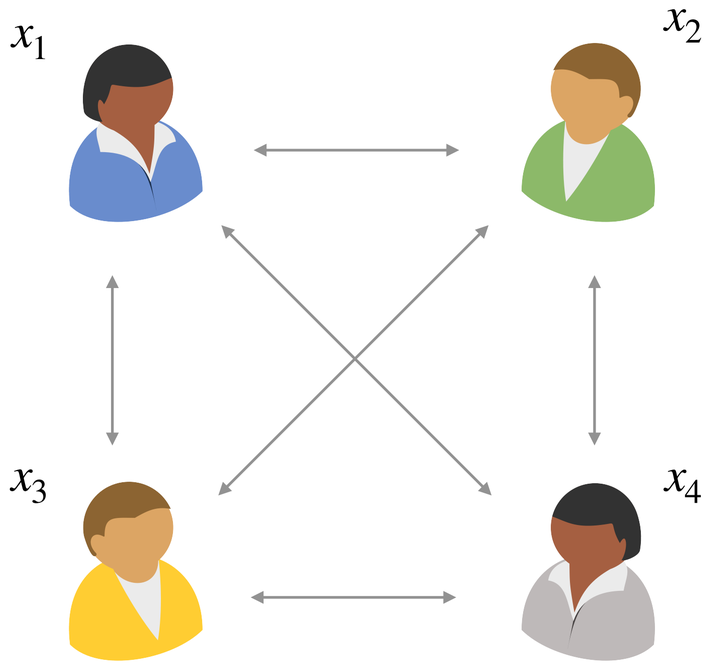
Secure Computation

Secure Computation



- Each party has its own private input x_i.

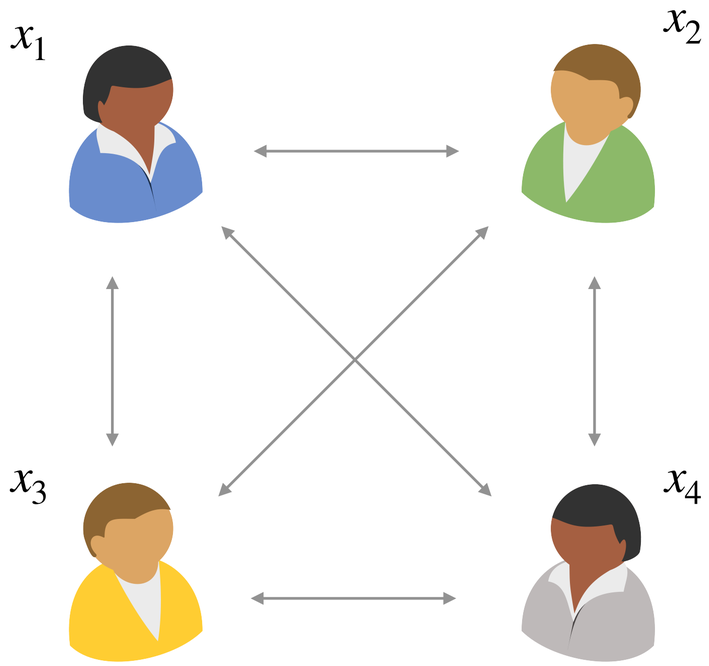
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- Each party has its own private input x_i.
- They communicate with each other and compute the output of a program P evaluated on their joint inputs.

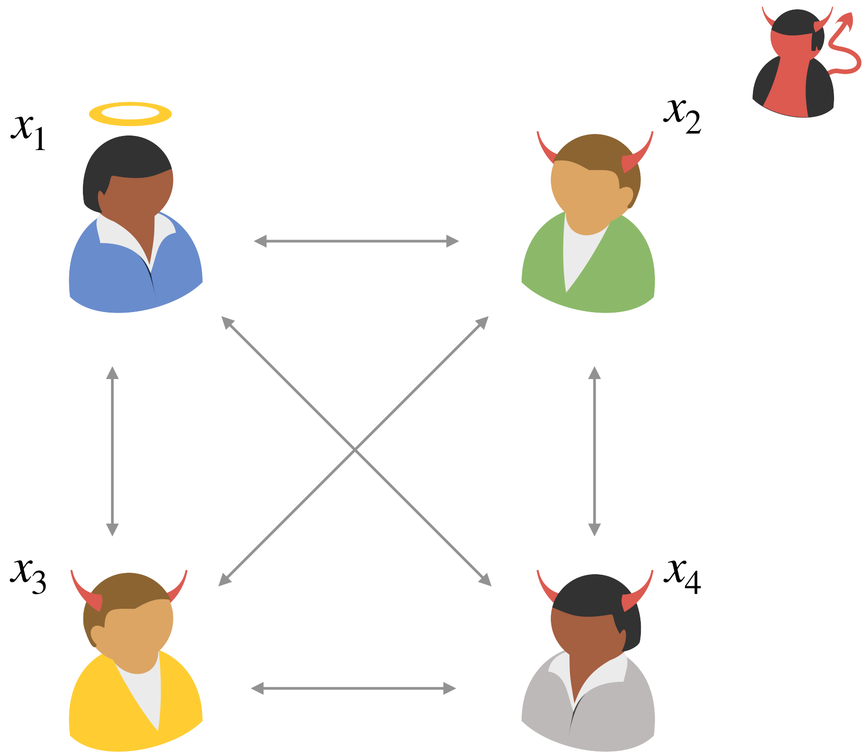
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- Each party has its own private input x_i.
- They communicate with each other and compute the output of a program P evaluated on their joint inputs.
- Each party should learn nothing beyond its input x_i and output z.

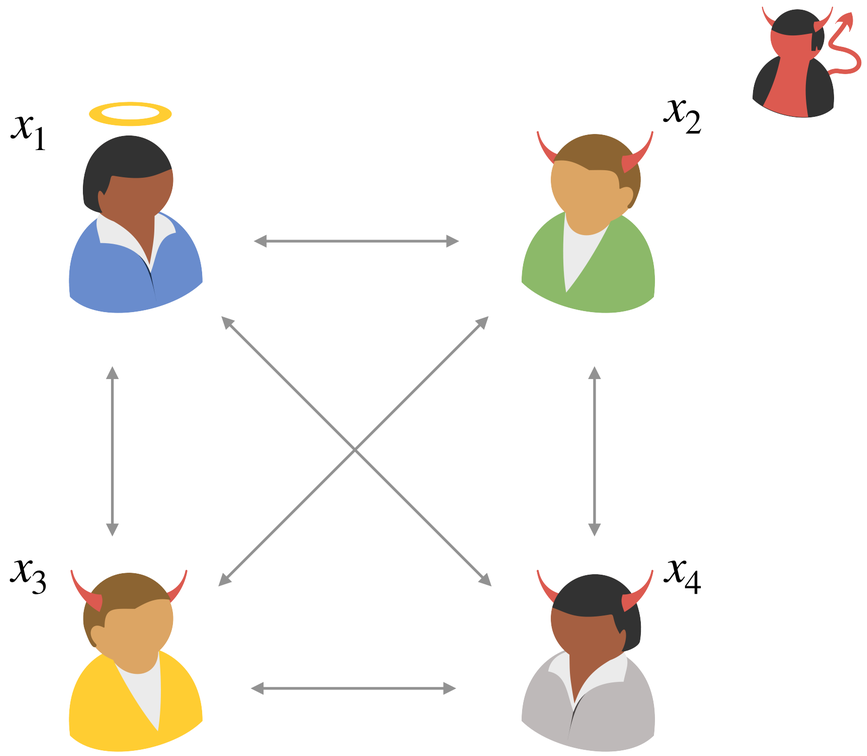
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.

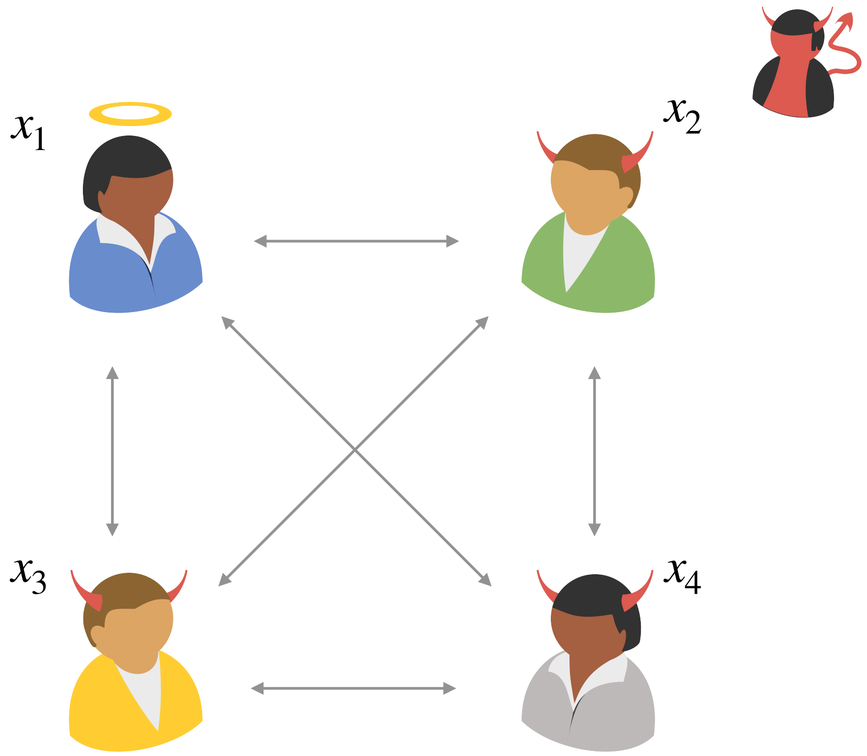
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.
- Two types of adversaries

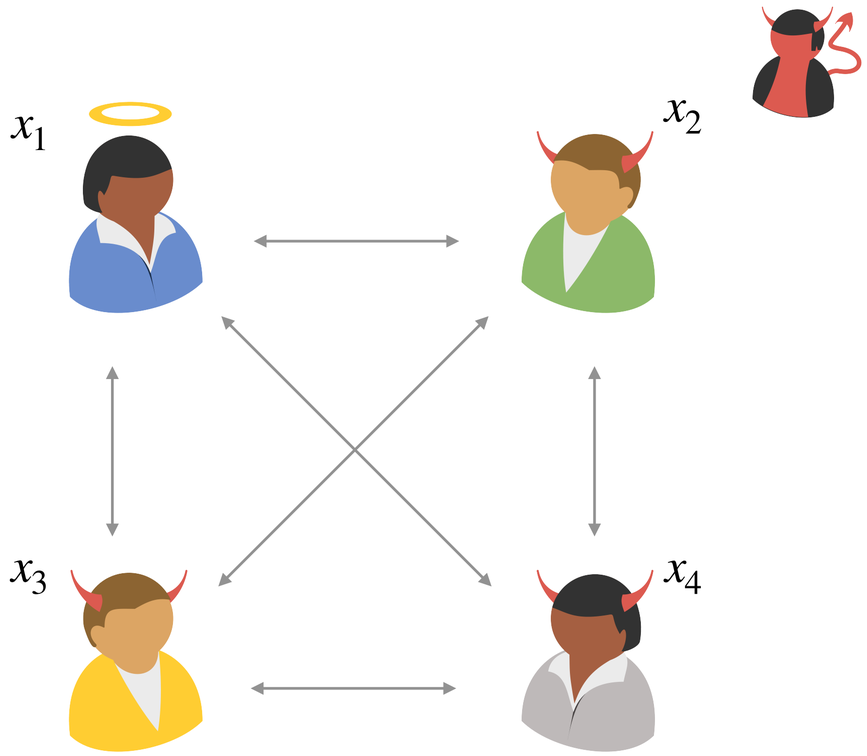
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.
- Two types of adversaries
  - Semi-honest
    - Corrupt parties follow the protocol description.
    - Adversary tries to learn as much as possible from the protocol transcript of corrupt parties.

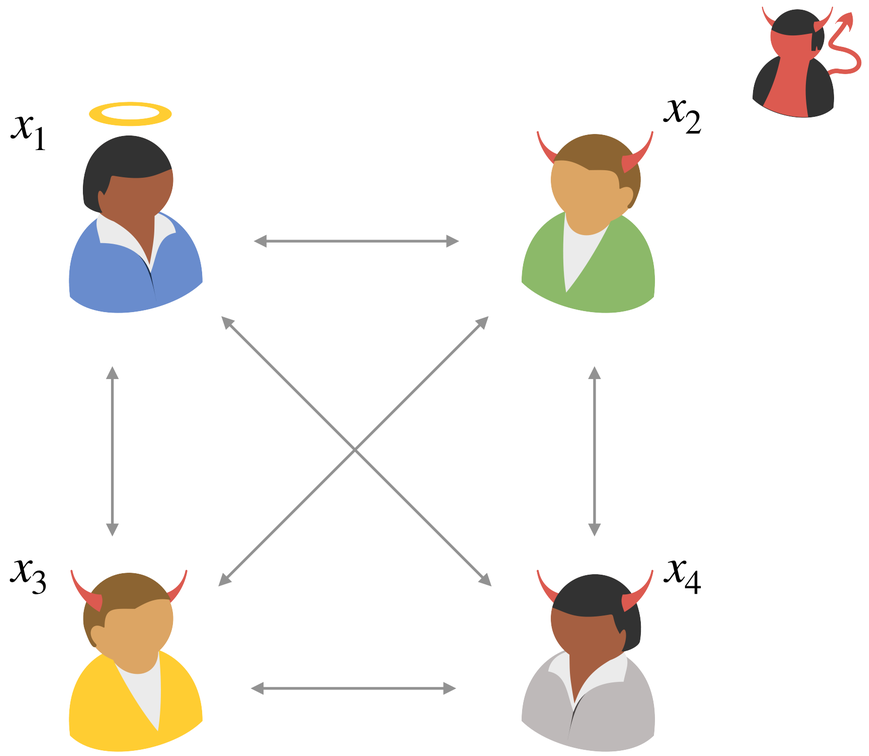
Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.
- Two types of adversaries
  - Semi-honest
    - Corrupt parties follow the protocol description.
    - Adversary tries to learn as much as possible from the protocol transcript of corrupt parties.
  - Malicious
    - Adversary can make corrupt parties arbitrarily deviate from the protocol description.

Secure Computation

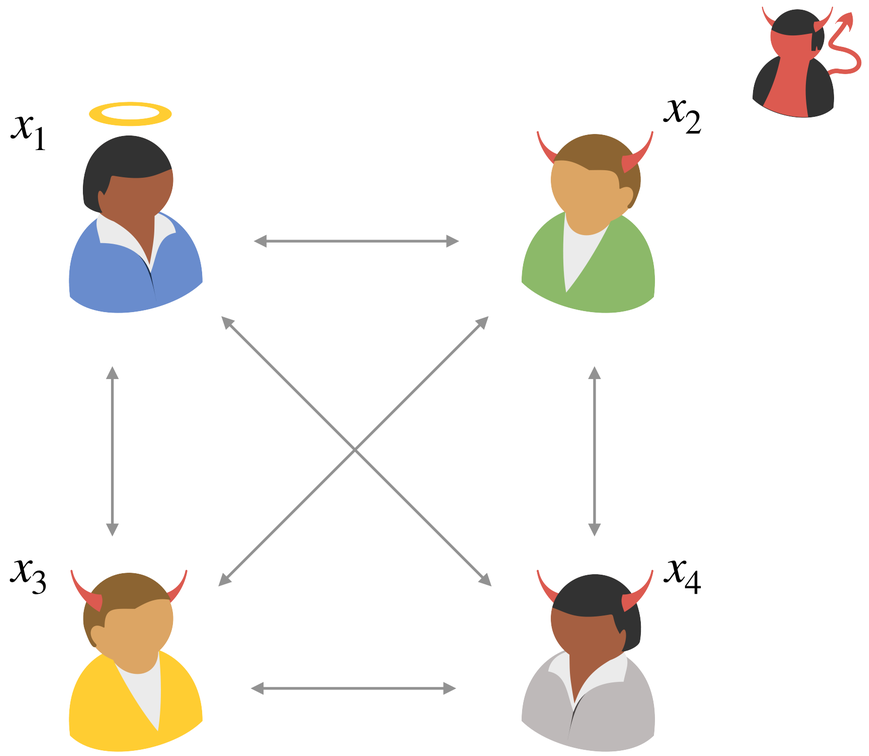


$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.
- Two types of adversaries
  - Semi-honest
    - Corrupt parties follow the protocol description.
    - Adversary tries to learn as much as possible from the protocol transcript of corrupt parties.
  - Malicious
    - Adversary can make corrupt parties arbitrarily deviate from the protocol description.

We will only consider semi-honest adversaries.

Secure Computation



$$z = P(x_1, x_2, x_3, x_4)$$

- **Adversarial Model:** A subset of parties are corrupt by a single adversary.
- Two types of adversaries
  - Semi-honest
    - Corrupt parties follow the protocol description.
    - Adversary tries to learn as much as possible from the protocol transcript of corrupt parties.
  - Malicious
    - Adversary can make corrupt parties arbitrarily deviate from the protocol description.

We will primarily focus on 2-party secure computation.

We will only consider semi-honest adversaries.

Defining Secure Computation

- How to define secure computation?

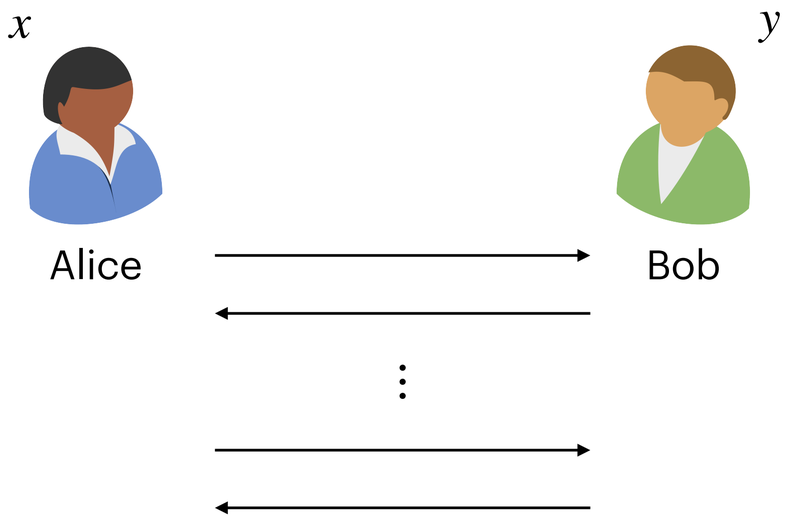
x

Alice

y

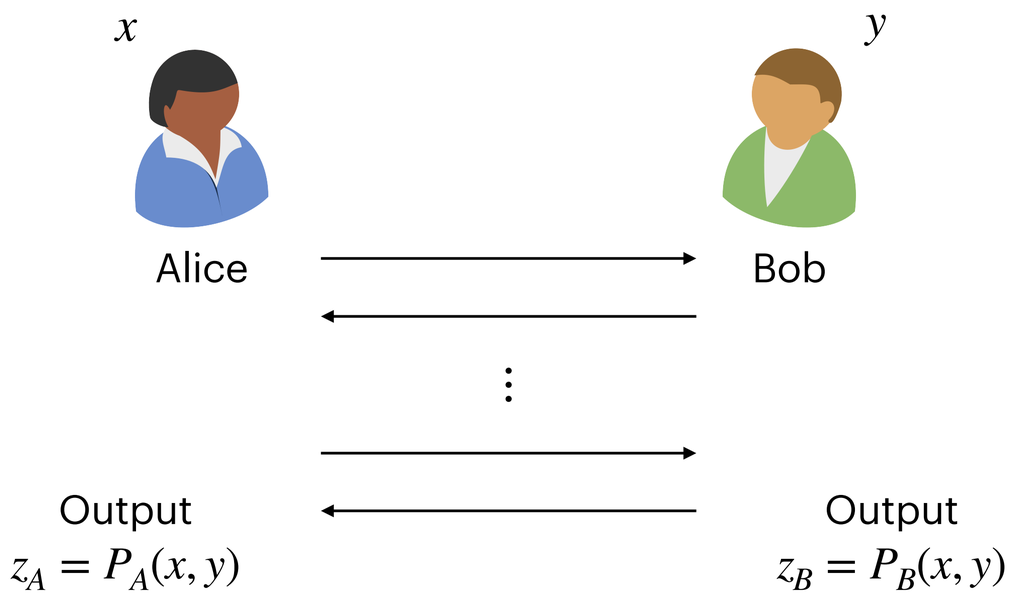
Bob

Defining Secure Computation



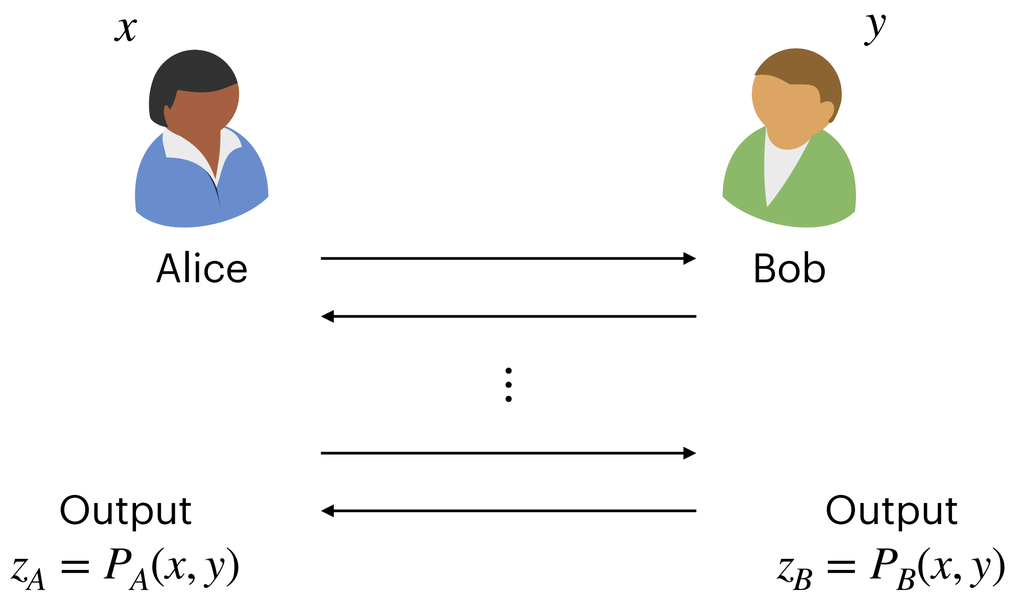
- How to define secure computation?

Defining Secure Computation



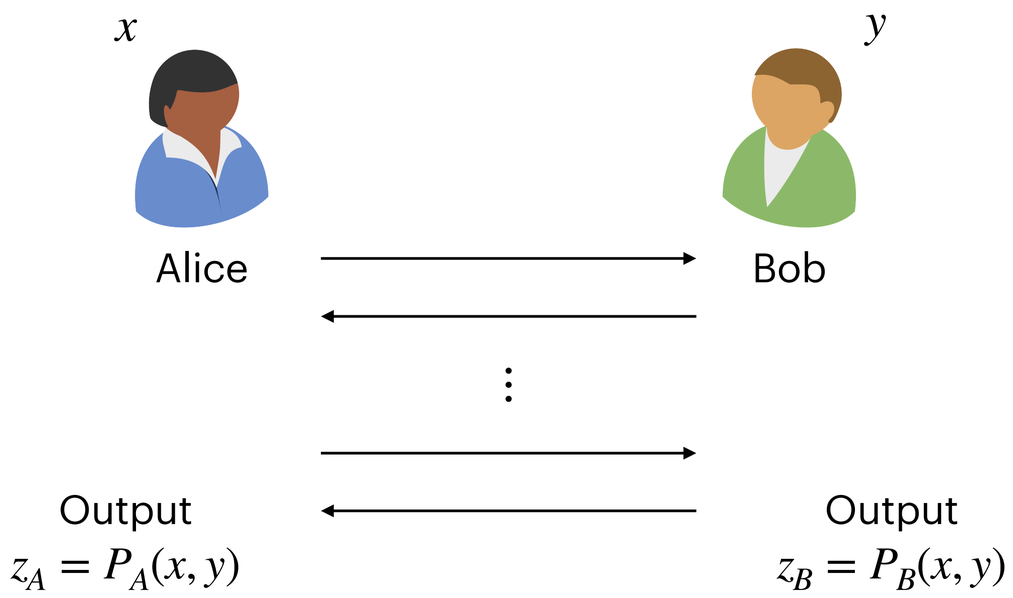
- How to define secure computation?

Defining Secure Computation



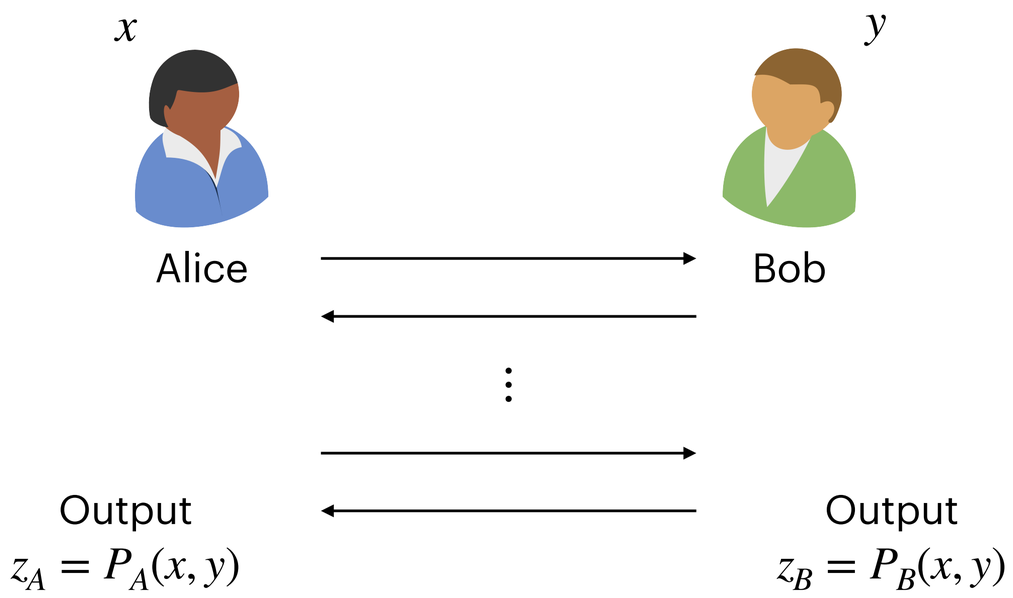
- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.

Defining Secure Computation



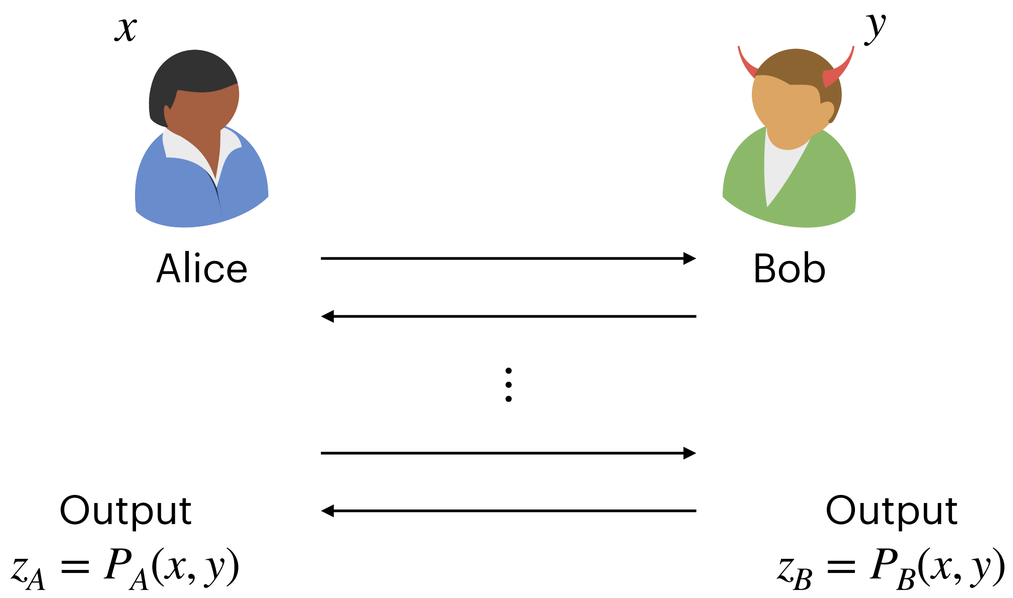
- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:**

Defining Secure Computation



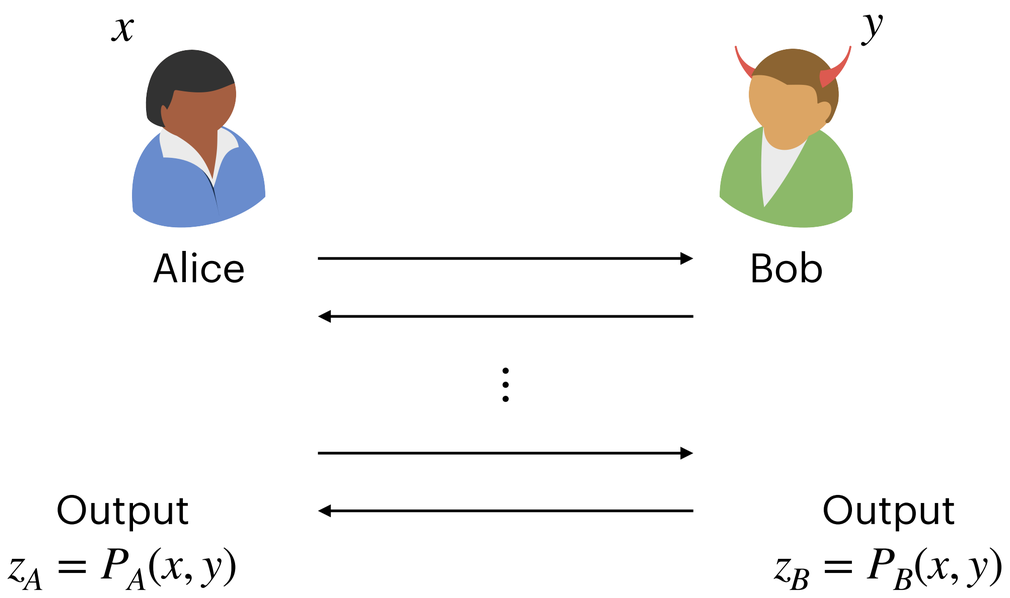
- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:** Simulation paradigm!

Defining Secure Computation



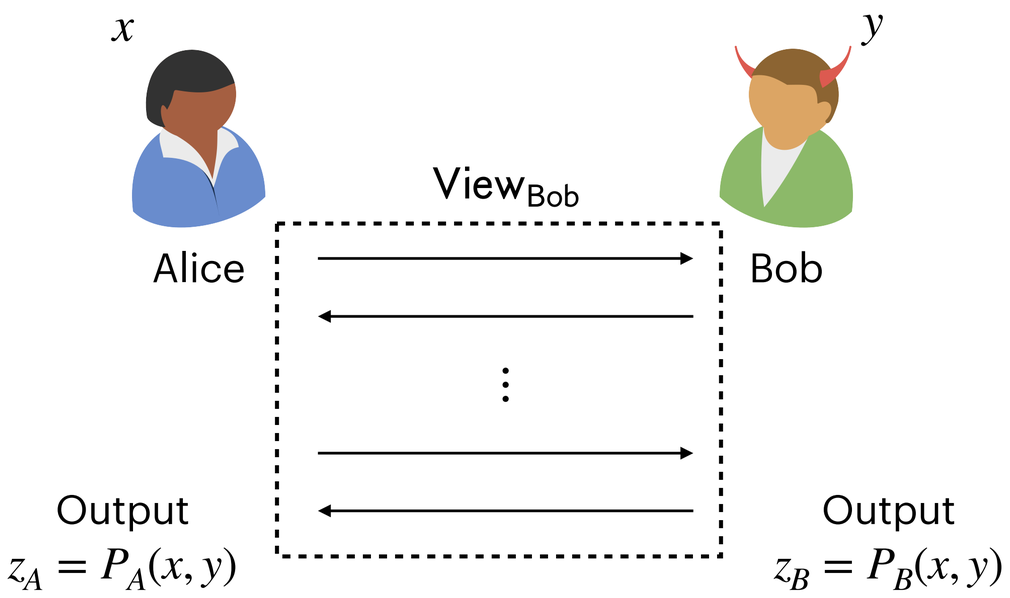
- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:** Simulation paradigm!
- **Simulation paradigm:** Anything a party could compute from its own inputs is not knowledge it gains by interacting with an external entity.

Defining Secure Computation



- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:** Simulation paradigm!
- **Simulation paradigm:** Anything a party could compute from its own inputs is not knowledge it gains by interacting with an external entity.
  - The adversary is allowed to learn the corrupt party's input y and final output z_B.

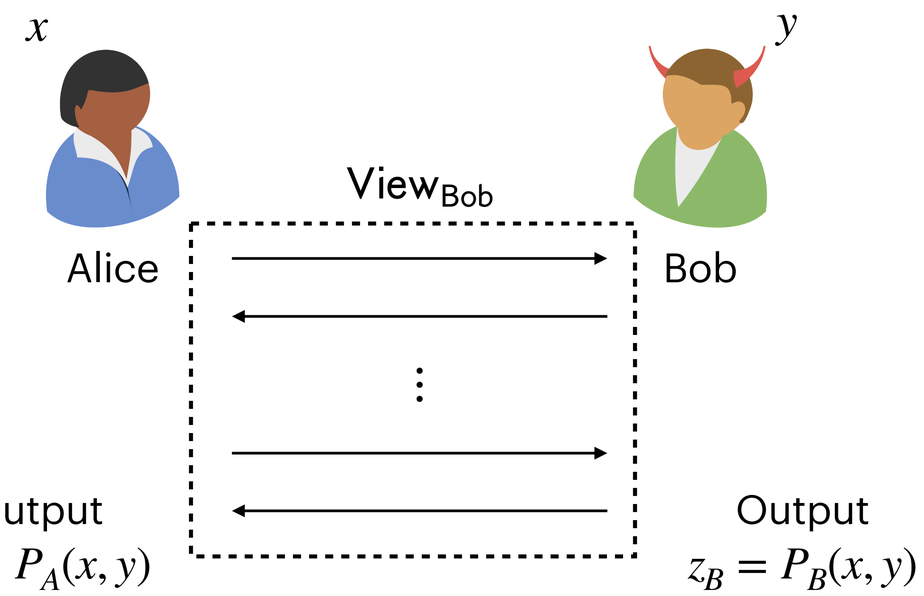
Defining Secure Computation



$\text{View}_{\text{Bob}} \equiv$ Bob's random tape, Alice's messages.

- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:** Simulation paradigm!
- **Simulation paradigm:** Anything a party could compute from its own inputs is not knowledge it gains by interacting with an external entity.
  - The adversary is allowed to learn the corrupt party's input y and final output z_B.

Defining Secure Computation



$\text{View}_{\text{Bob}} \equiv$ Bob's random tape, Alice's messages.

$$\text{View}_{\text{Bob}} \overset{c}{\approx} S(y, P_B(x, y))$$

- How to define secure computation?
  - **Goal:** Adversary must not learn anything beyond corrupt parties' input and function output.
  - **Solution:** Simulation paradigm!
- **Simulation paradigm:** Anything a party could compute from its own inputs is not knowledge it gains by interacting with an external entity.
  - The adversary is allowed to learn the corrupt party's input y and final output z_B.
  - There is an efficient simulator that can generate the corrupt party's view only using y and z_B, without the honest party's input x.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A) \} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)] \}, \text{ and}$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

Output of both parties at the end of the protocol.

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- **Correctness:** Alice and Bob compute $P_A(x, y)$ and $P_B(x, y)$ respectively upon running the protocol.
  - Captured by the definition since we require $z \overset{c}{\approx} \text{Output}[A(x) \leftrightarrow B(y)]$.

Semi-Honest Secure Two-Party Computation

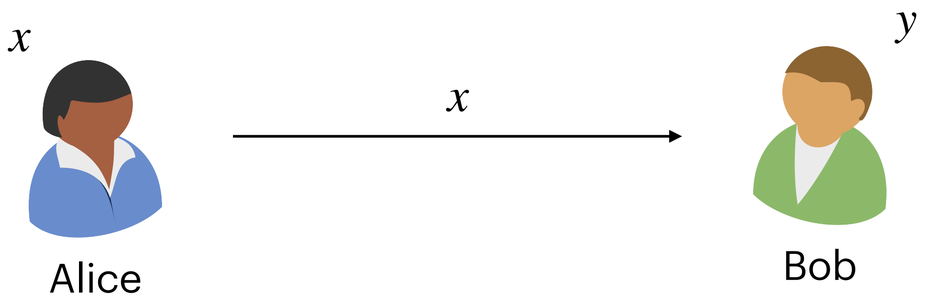
x

Alice

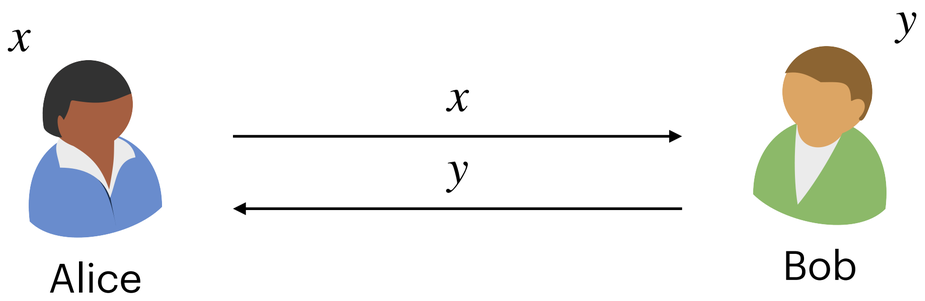
y

Bob

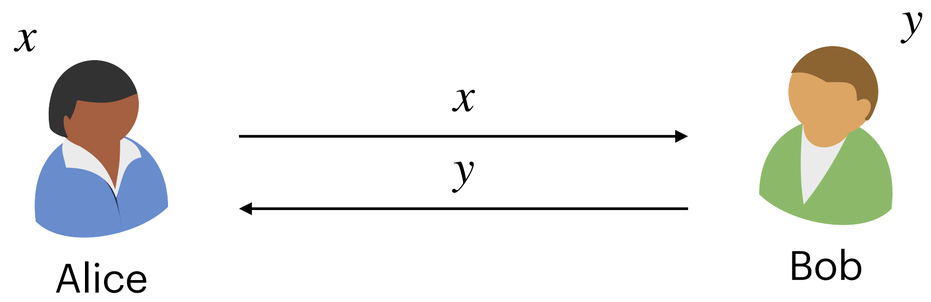
Semi-Honest Secure Two-Party Computation



Semi-Honest Secure Two-Party Computation



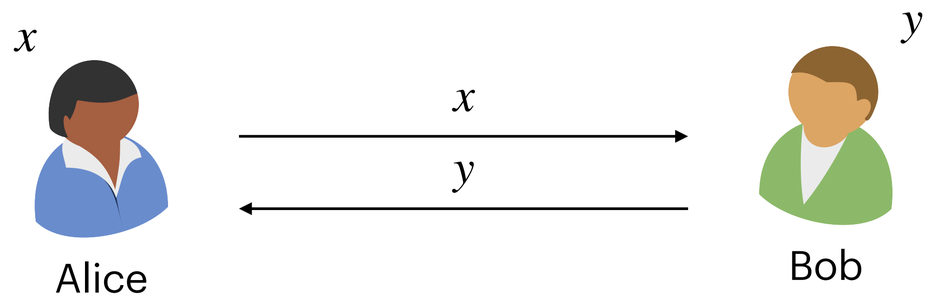
Semi-Honest Secure Two-Party Computation



Output $x + y$

Output $x + y$

Semi-Honest Secure Two-Party Computation

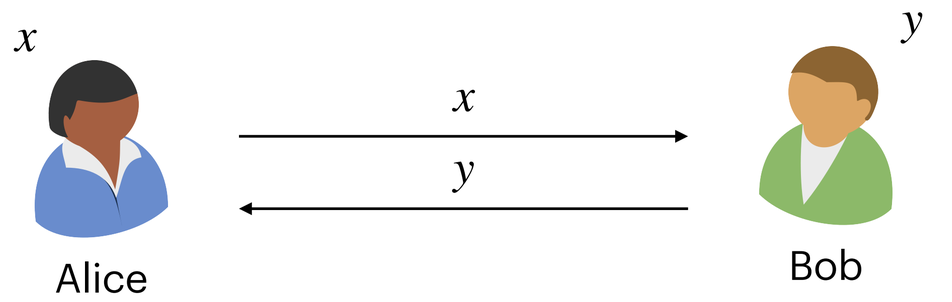


Output $x + y$

Output $x + y$

Is this protocol secure?

Semi-Honest Secure Two-Party Computation



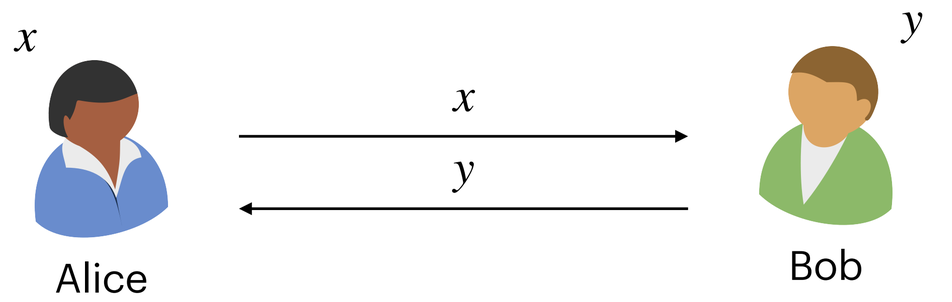
Output $x + y$

Output $x + y$

Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

Semi-Honest Secure Two-Party Computation



Output $x + y$

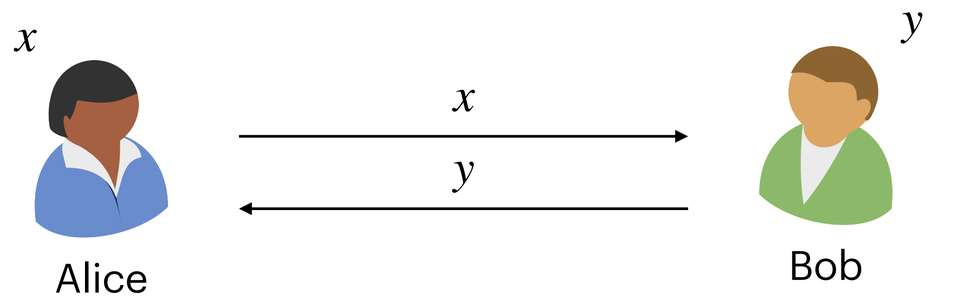
Output $x + y$

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.

Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

Semi-Honest Secure Two-Party Computation



Output $x + y$

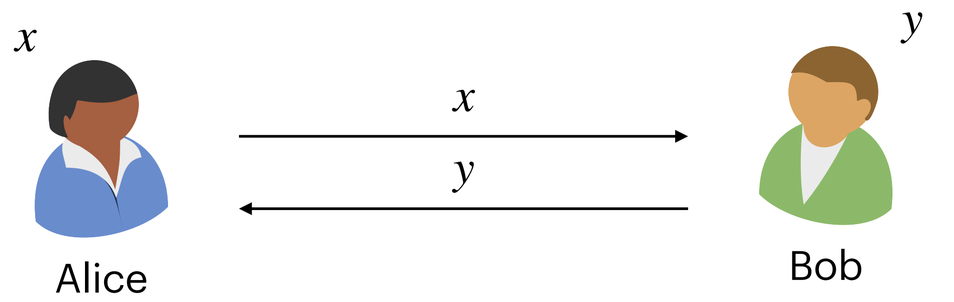
Output $x + y$

Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.

Semi-Honest Secure Two-Party Computation

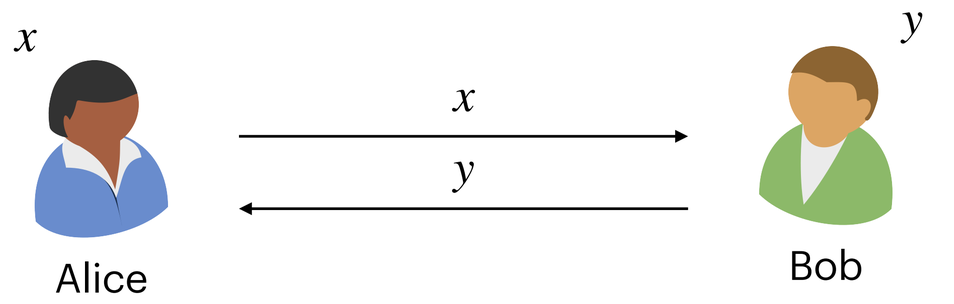


Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.
- **Another Example:** Say parties use a secure two-party protocol to compute the AND function $z = x \wedge y$.

Semi-Honest Secure Two-Party Computation



Output $x + y$

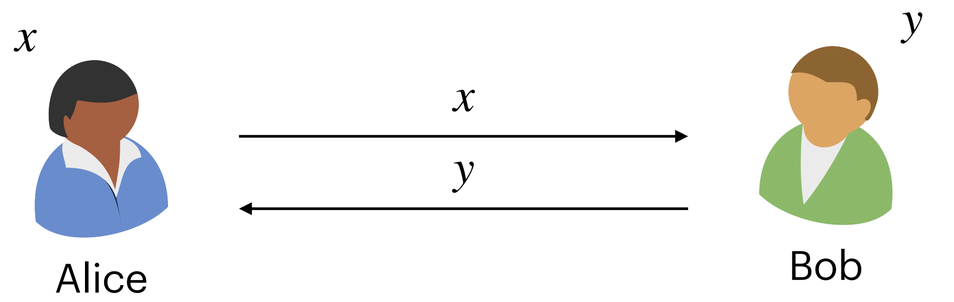
Output $x + y$

Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.
- **Another Example:** Say parties use a secure two-party protocol to compute the AND function $z = x \wedge y$.
  - If the corrupt party's input is $x = 1$,

Semi-Honest Secure Two-Party Computation

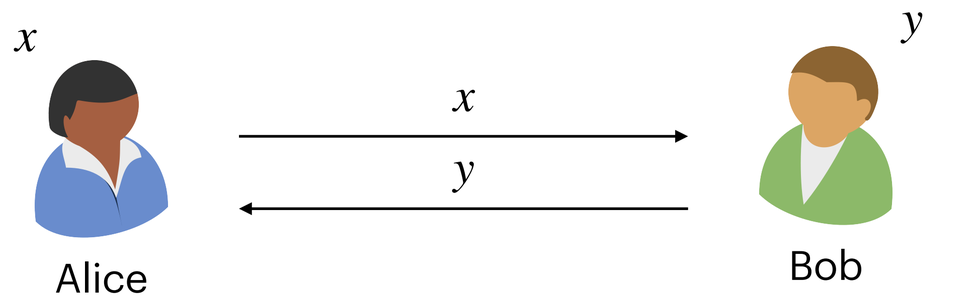


Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.
- **Another Example:** Say parties use a secure two-party protocol to compute the AND function $z = x \wedge y$.
  - If the corrupt party's input is $x = 1$, then the output is y. The adversary learns y even when using a secure protocol.

Semi-Honest Secure Two-Party Computation

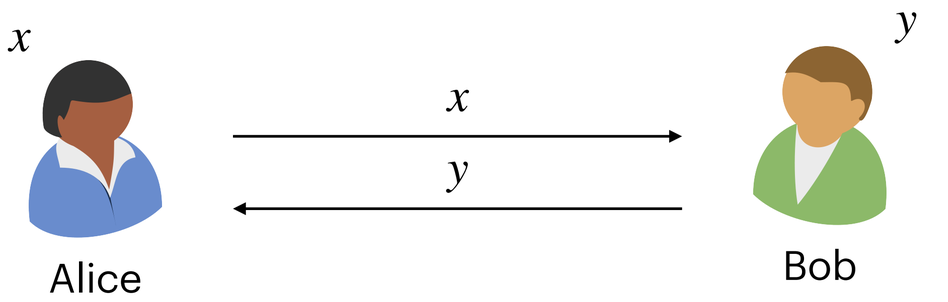


Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.
- **Another Example:** Say parties use a secure two-party protocol to compute the AND function $z = x \wedge y$.
  - If the corrupt party's input is $x = 1$, then the output is y. The adversary learns y even when using a secure protocol.
  - If the corrupt party's input is $x = 0$,

Semi-Honest Secure Two-Party Computation



Is this protocol secure?

- Yes! Adversary can learn honest party's input only from the corrupt party's input and the sum.
- Formally, $S_A(x, x + y)$ can simply compute y and generate Alice's view.

- The definition only says the adversary does not learn anything beyond what it could compute using the corrupt party's input and function output.
  - Privacy of the honest party's input depends on the function being computed.
- **Another Example:** Say parties use a secure two-party protocol to compute the AND function $z = x \wedge y$.
  - If the corrupt party's input is $x = 1$, then the output is y. The adversary learns y even when using a secure protocol.
  - If the corrupt party's input is $x = 0$, then the adversary does not know if $y = 1$ or $y = 0$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Joint distribution of simulated view and correctly computed output z must be indistinguishable from corrupt party's view and joint outputs in an execution of the protocol.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Joint distribution of simulated view and correctly computed output z must be indistinguishable from corrupt party's view and joint outputs in an execution of the protocol.
  - Necessary for randomized programs.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Joint distribution of simulated view and correctly computed output z must be indistinguishable from corrupt party's view and joint outputs in an execution of the protocol.
  - Necessary for randomized programs.
  - Example

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Joint distribution of simulated view and correctly computed output z must be indistinguishable from corrupt party's view and joint outputs in an execution of the protocol.
  - Necessary for randomized programs.
  - Example
    - A program that samples a uniformly random bitstring for Alice and outputs nothing to Bob.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Joint distribution of simulated view and correctly computed output z must be indistinguishable from corrupt party's view and joint outputs in an execution of the protocol.
  - Necessary for randomized programs.
  - Example
    - A program that samples a uniformly random bitstring for Alice and outputs nothing to Bob.
    - A protocol where Alice samples a uniformly random bitstring and sends it to Bob.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Captures semi-honest security because it only requires simulating the view in an honest execution of the protocol.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Captures semi-honest security because it only requires simulating the view in an honest execution of the protocol.
  - Extending the definition to capture security against malicious adversaries is non-trivial.

Semi-Honest Secure Two-Party Computation

Semi-Honest Secure Two-Party Computation

A protocol $\Pi = (A, B)$ securely computes a program $P = (P_A, P_B)$ in the presence of semi-honest adversaries if there exists PPT algorithms S_A and S_B such that for all $x, y \in \{0,1\}^*$ we have

$$\{S_A(1^\lambda, x, z_A), z\} \overset{c}{\approx} \{\text{View}_A[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\}, \text{ and}$$

$$\{S_B(1^\lambda, y, z_B), z\} \overset{c}{\approx} \{\text{View}_B[A(x) \leftrightarrow B(y)], \text{Output}[A(x) \leftrightarrow B(y)]\},$$

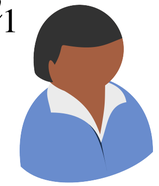
where $z_A \leftarrow P_A(x, y)$, $z_B \leftarrow P_B(x, y)$, and $z = (z_A, z_B)$.

- Captures semi-honest security because it only requires simulating the view in an honest execution of the protocol.
  - Extending the definition to capture security against malicious adversaries is non-trivial.
- This definition naturally extends to more than two parties.

Oblivious Transfer

Oblivious Transfer (OT)

m_0, m_1



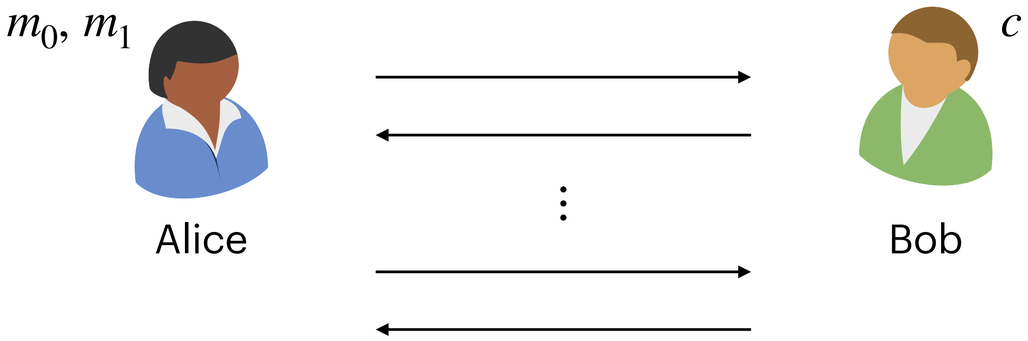
Alice

c

Bob

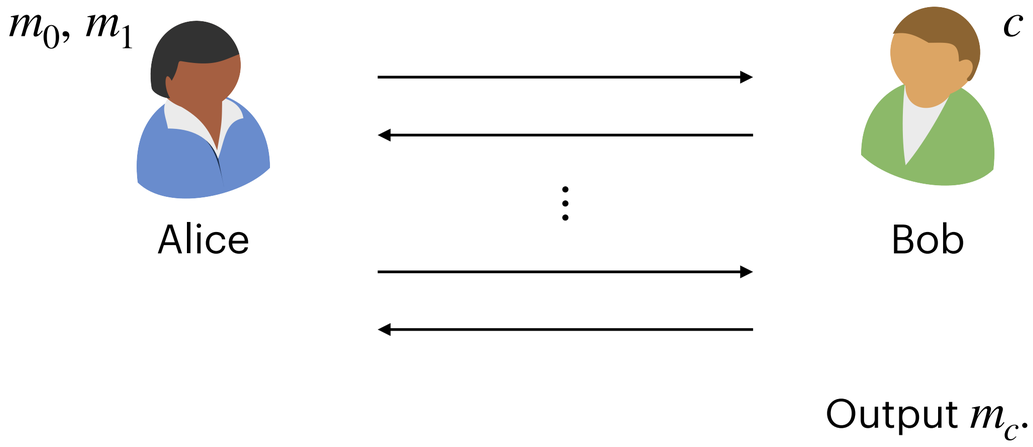
- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.

Oblivious Transfer (OT)



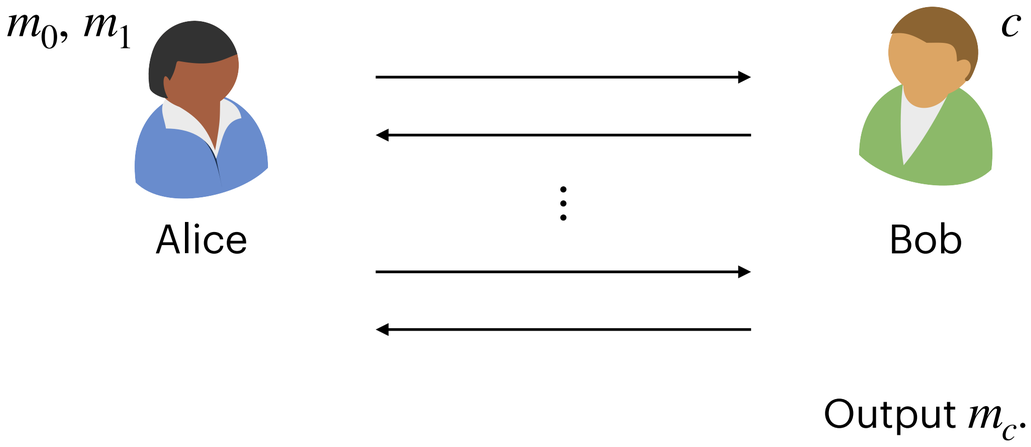
- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.

Oblivious Transfer (OT)



- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

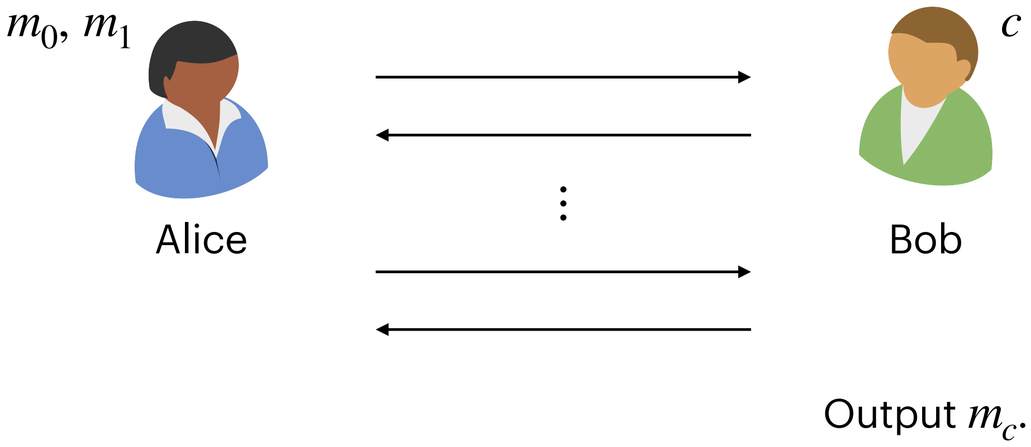
Oblivious Transfer (OT)



- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

Theorem [Yao'86]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure two-party computation of any PPT program.

Oblivious Transfer (OT)

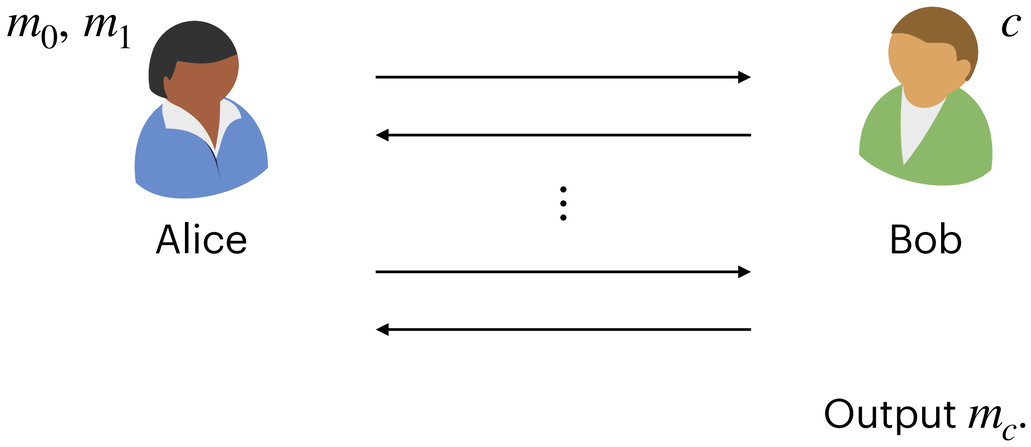


- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

Theorem [Yao'86]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure two-party computation of any PPT program.

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

Oblivious Transfer (OT)



- **Inputs**
  - Alice has two messages m_0 and m_1.
  - Bob has a choice bit $c \in \{0,1\}$.
- **Output:** Bob outputs m_c, Alice does not have any output.

Theorem [Yao'86]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure two-party computation of any PPT program.

Theorem [Goldreich-Micali-Wigderson'87]: Semi-honest secure OT $\Longrightarrow$ semi-honest secure n-party computation of any PPT program.

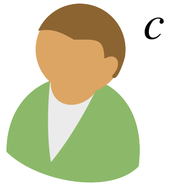
OT is necessary and complete for secure multi-party computation!

Constructing Oblivious Transfer Protocol

Constructing Oblivious Transfer Protocol

m_0, m_1

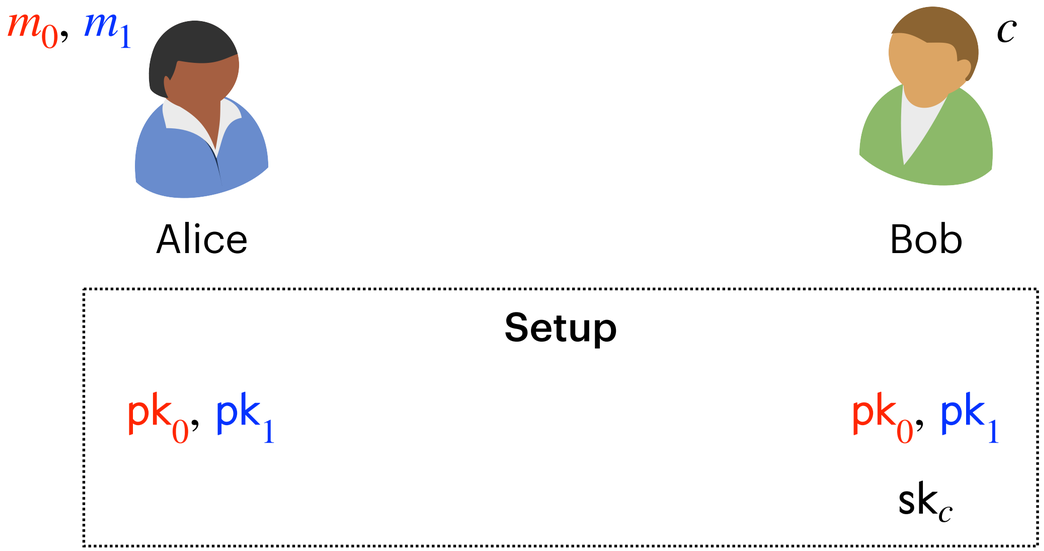
Alice



c

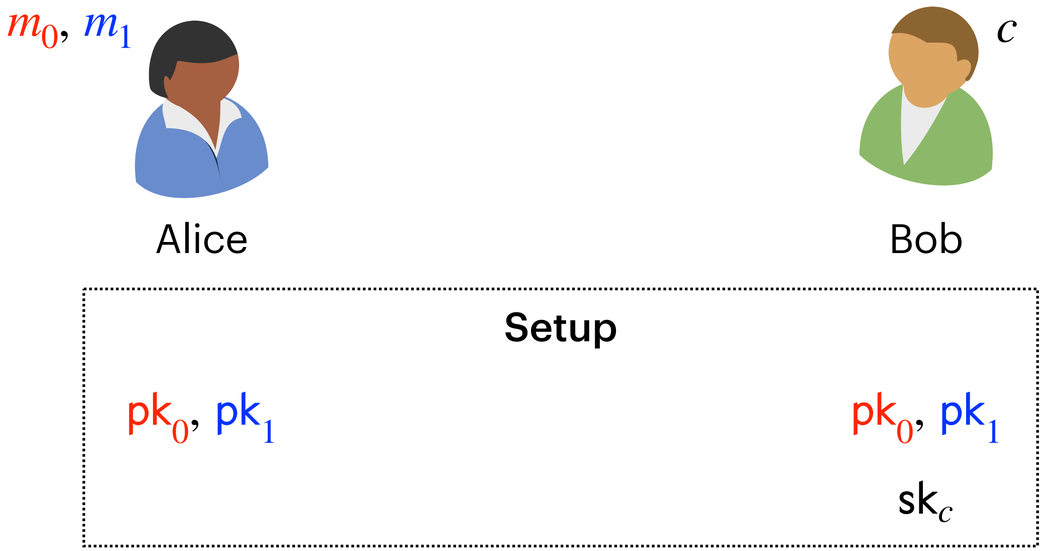
Bob

Constructing Oblivious Transfer Protocol



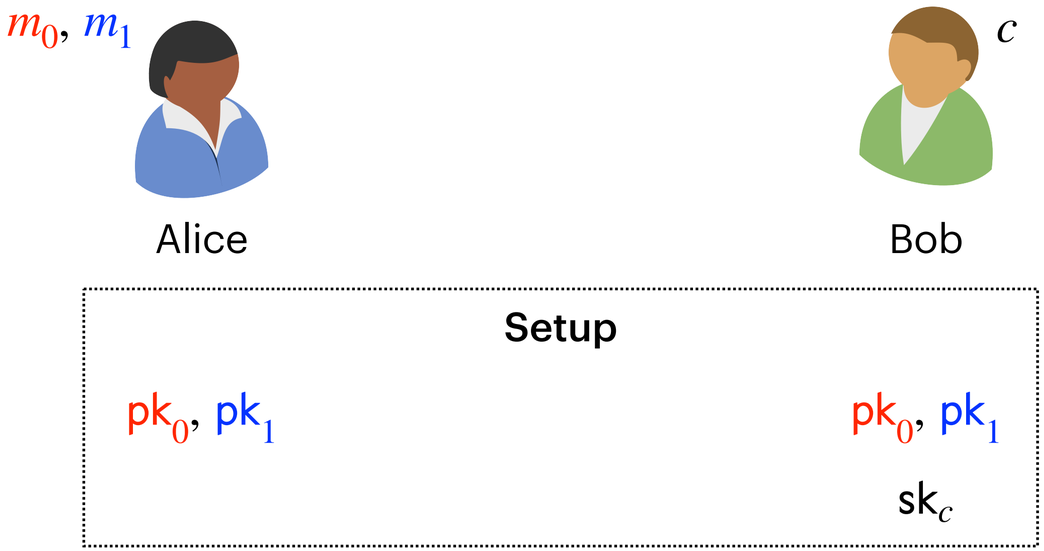
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.

Constructing Oblivious Transfer Protocol



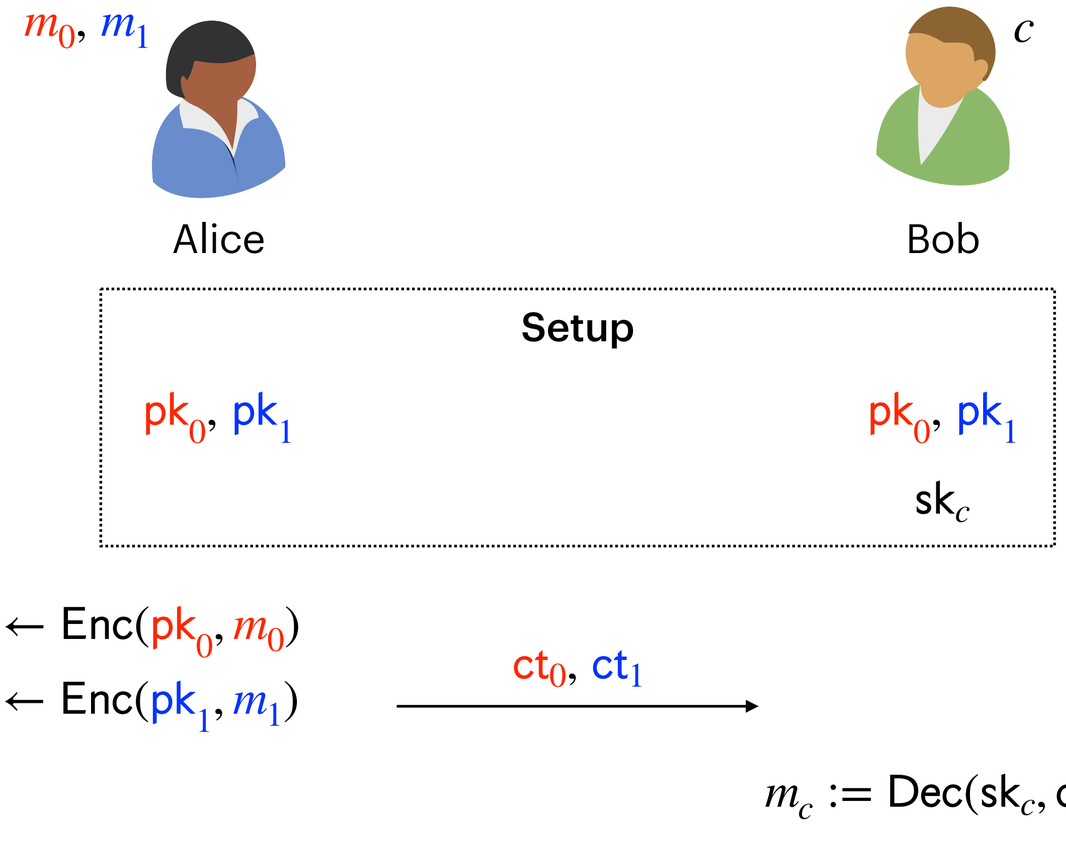
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?

Constructing Oblivious Transfer Protocol



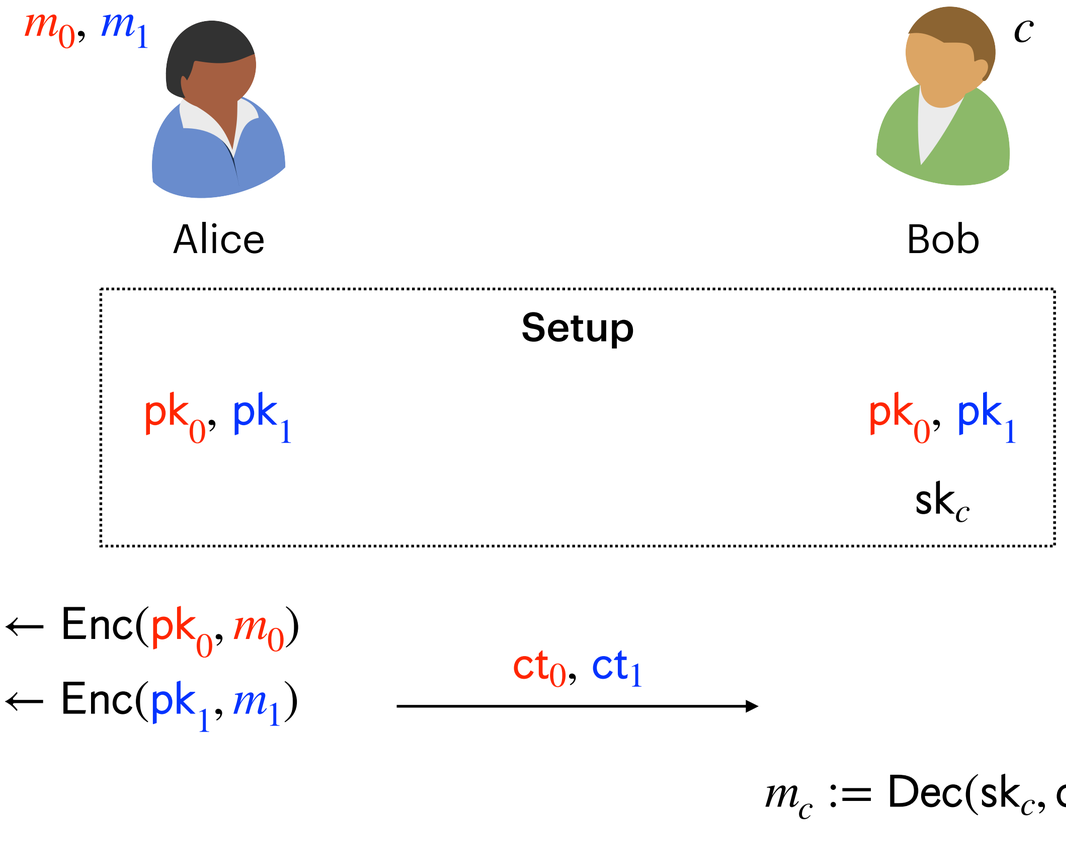
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?

Constructing Oblivious Transfer Protocol



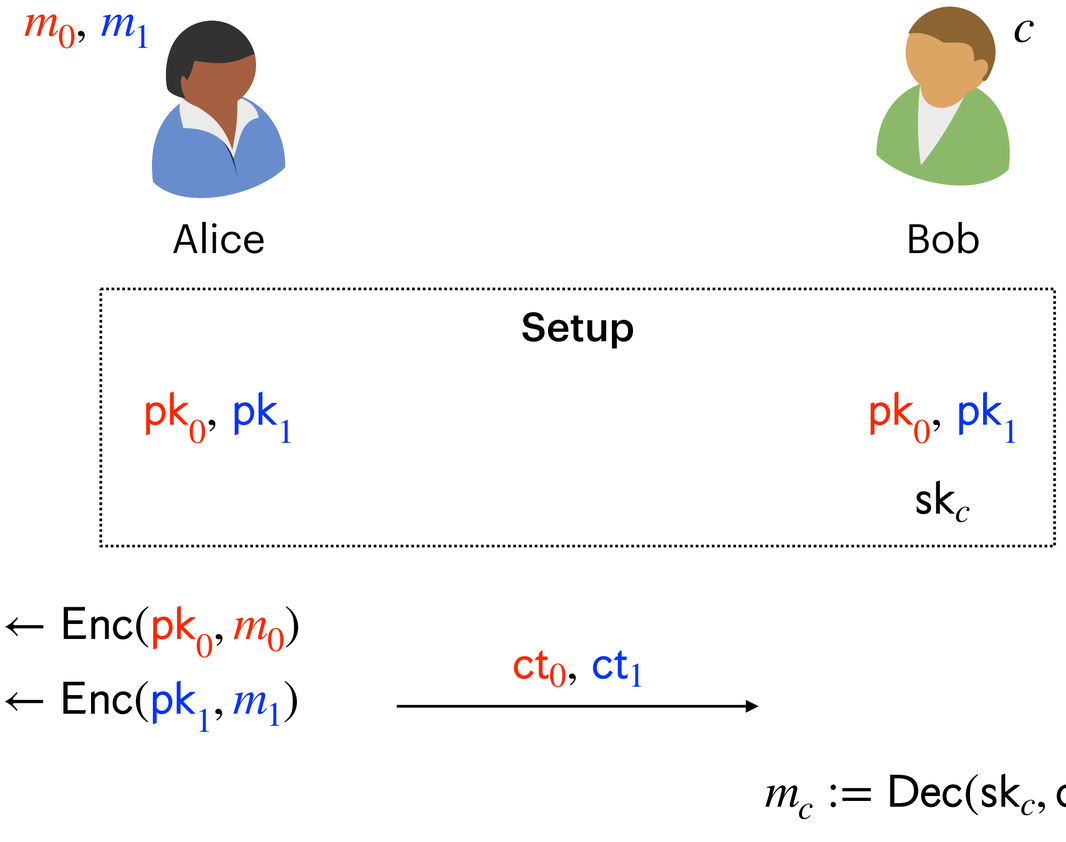
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?

Constructing Oblivious Transfer Protocol



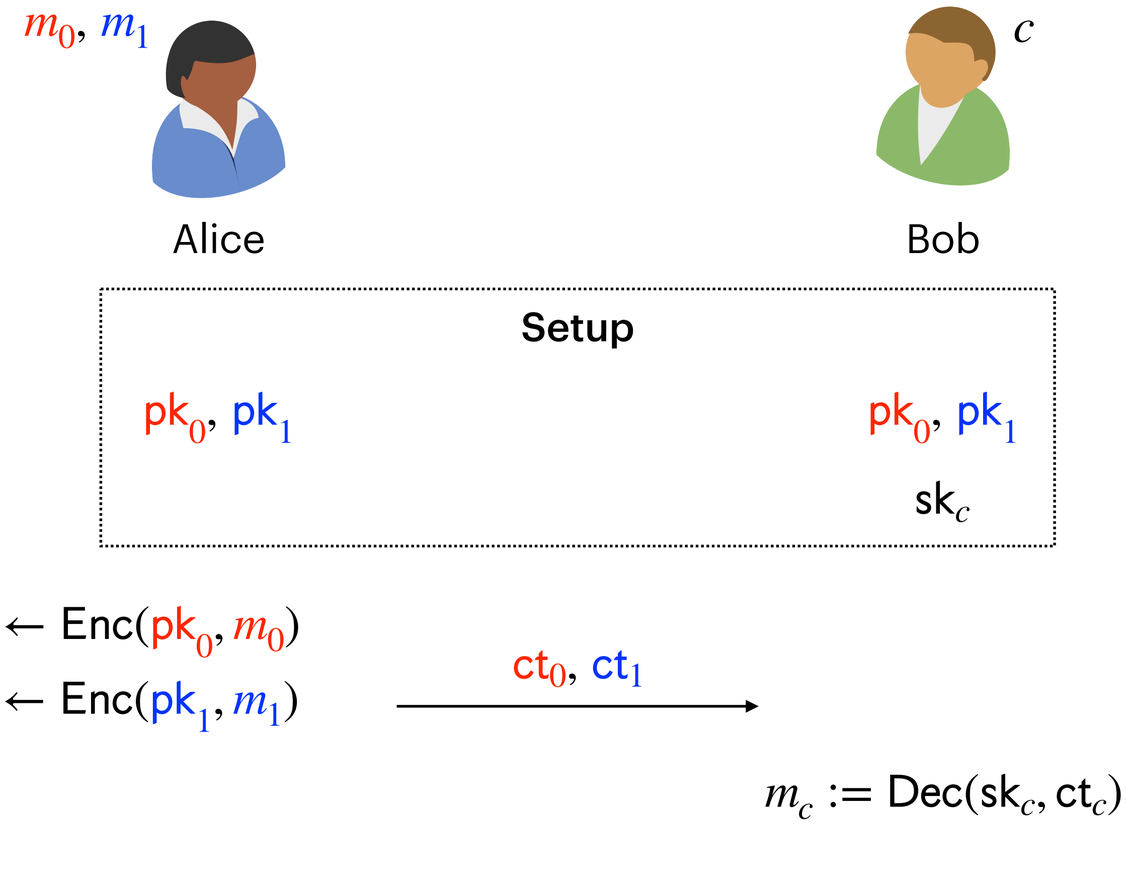
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?
- **Correctness:**

Constructing Oblivious Transfer Protocol



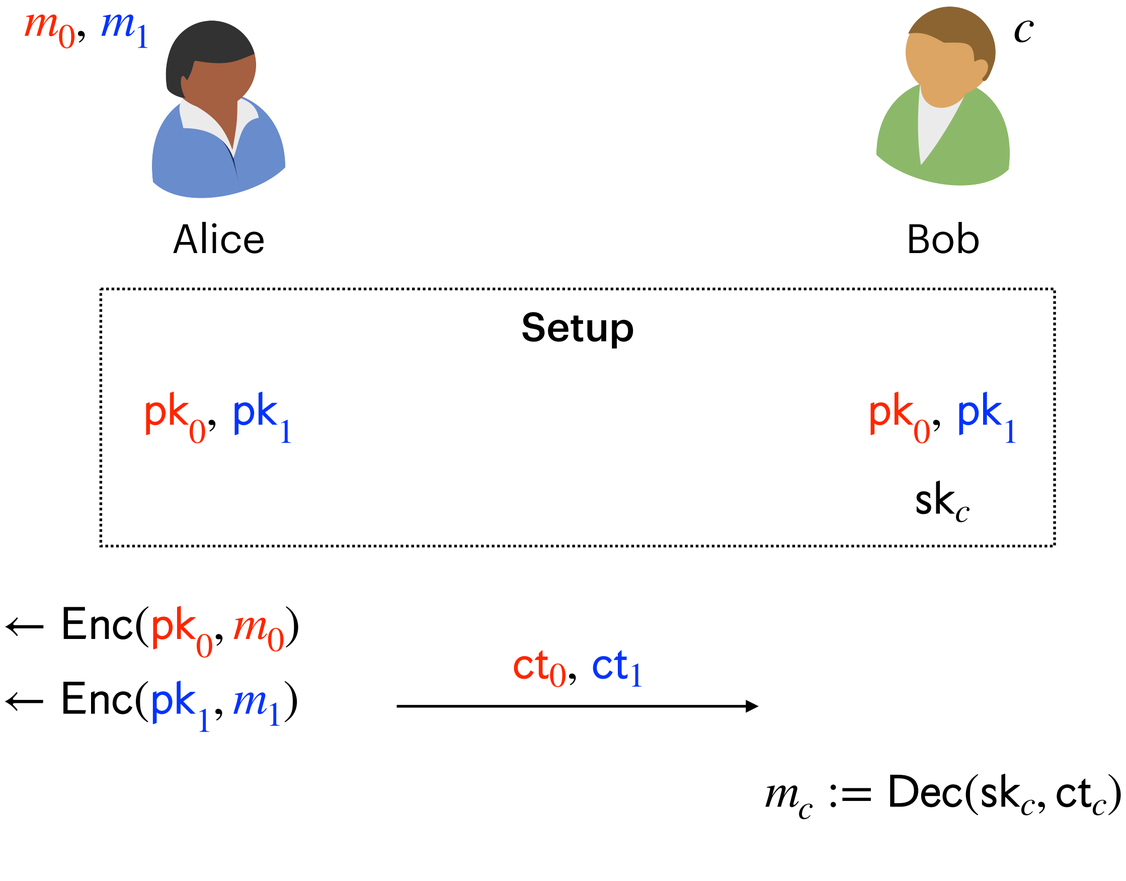
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?
- **Correctness:** Correctness of encryption scheme.

Constructing Oblivious Transfer Protocol



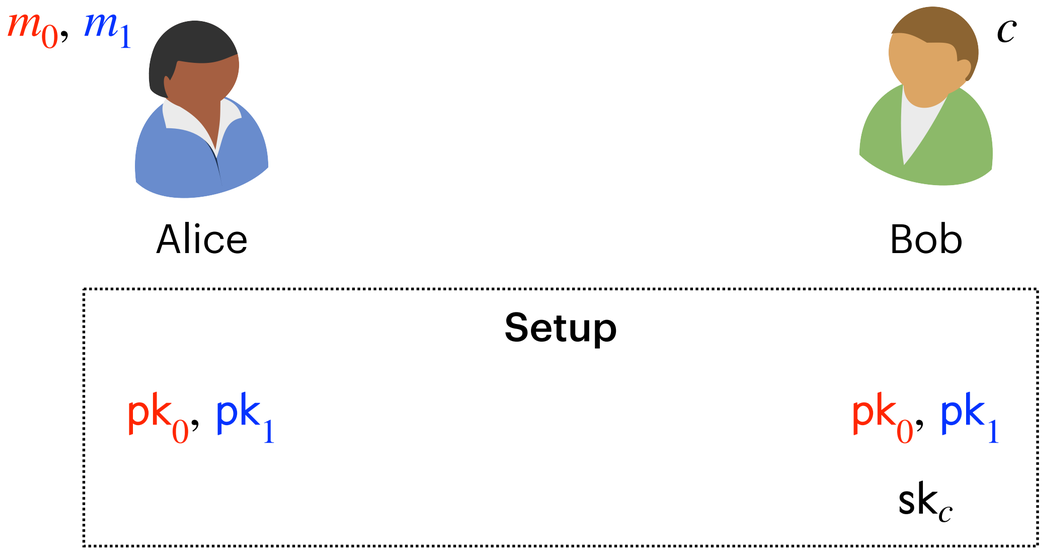
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?
- **Correctness:** Correctness of encryption scheme.
- **Security:**

Constructing Oblivious Transfer Protocol



- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?
- **Correctness:** Correctness of encryption scheme.
- **Security:** Bob does not have sk_{1-c}.

Constructing Oblivious Transfer Protocol



How can parties compute the setup?

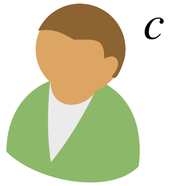
- **Approach:** Assume parties are given the following setup before the start of the protocol.
  - Alice and Bob are both given public keys pk_0 and pk_1.
  - Bob is given the secret key sk_c for pk_c.
- Can we use the setup to construct an OT protocol?
- **Correctness:** Correctness of encryption scheme.
- **Security:** Bob does not have sk_{1-c}.

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice

c

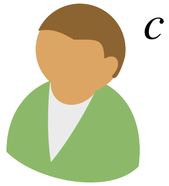


Bob

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

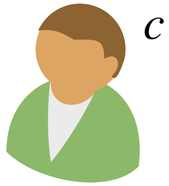
Bob

$(\mathbb{G}, g, p)$

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

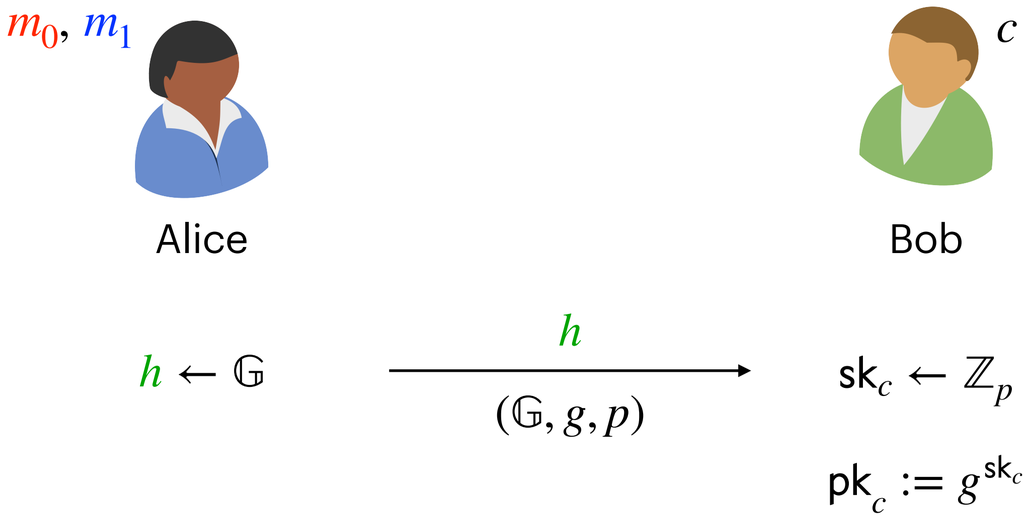
Bob

$\xrightarrow{(\mathbb{G}, g, p)}$

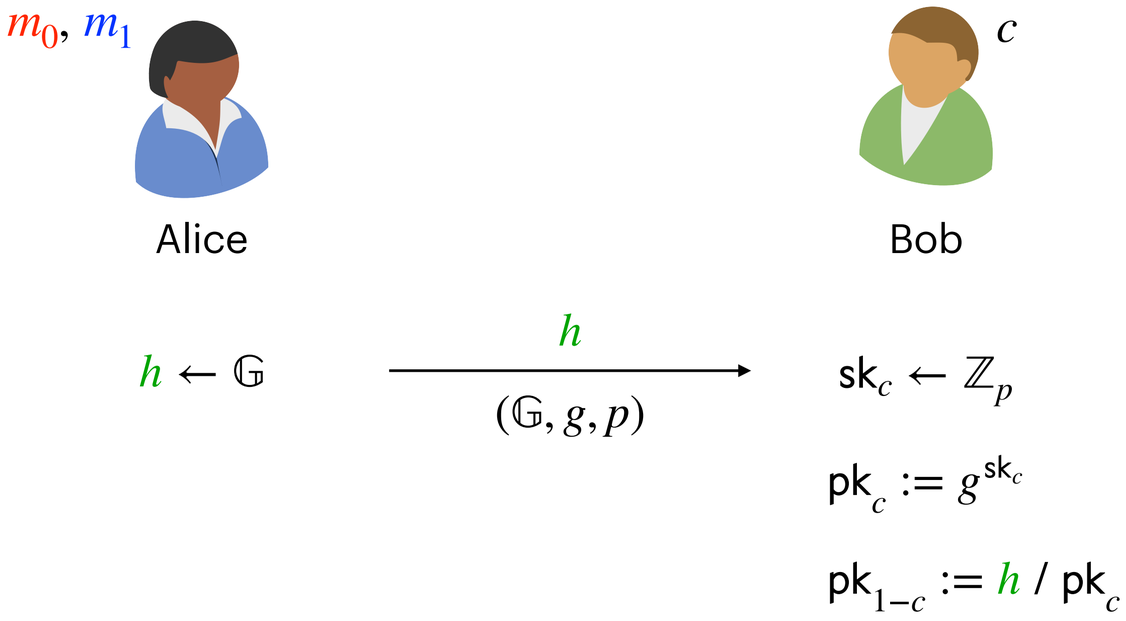
$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

Constructing Oblivious Transfer Protocol



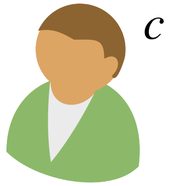
Constructing Oblivious Transfer Protocol



Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

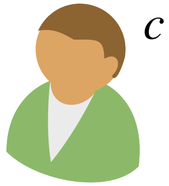
$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

Does Bob know the secret key sk_{1-c} for pk_{1-c}?

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

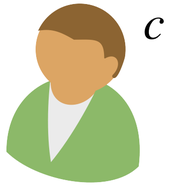
$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

pk_0

$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

Does Bob know the secret key sk_{1-c} for pk_{1-c}?

Constructing Oblivious Transfer Protocol



m_0, m_1

c

Alice

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

$\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$

pk_0

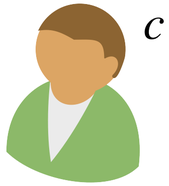
$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

Does Bob know the secret key sk_{1-c} for pk_{1-c}?

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

$\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$

pk_0

$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

$\mathsf{pk}_0, \mathsf{pk}_1$

$\mathsf{pk}_0, \mathsf{pk}_1$ sk_c

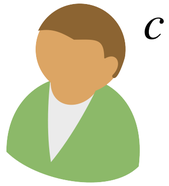
Does Bob know the secret key sk_{1-c} for pk_{1-c}?

Alice and Bob have computed the setup!

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice



c

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

$\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$

pk_0

$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

Does Bob know the secret key sk_{1-c} for pk_{1-c}?

$\mathsf{pk}_0, \mathsf{pk}_1$

$\mathsf{pk}_0, \mathsf{pk}_1$ sk_c

Alice and Bob have computed the setup!

$\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}_0, m_0)$

$\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_1, m_1)$

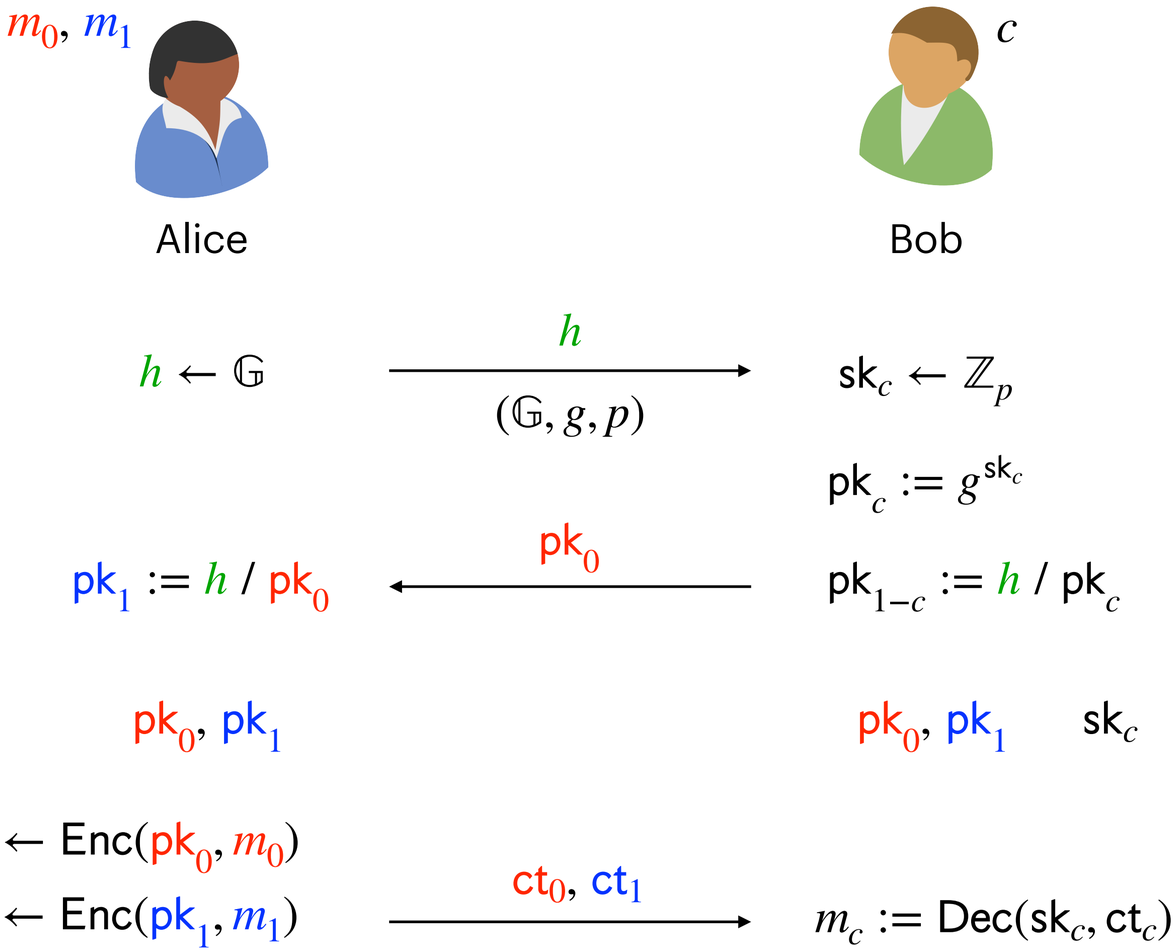
$\mathsf{ct}_0, \mathsf{ct}_1$

$m_c := \mathsf{Dec}(\mathsf{sk}_c, \mathsf{ct}_c)$

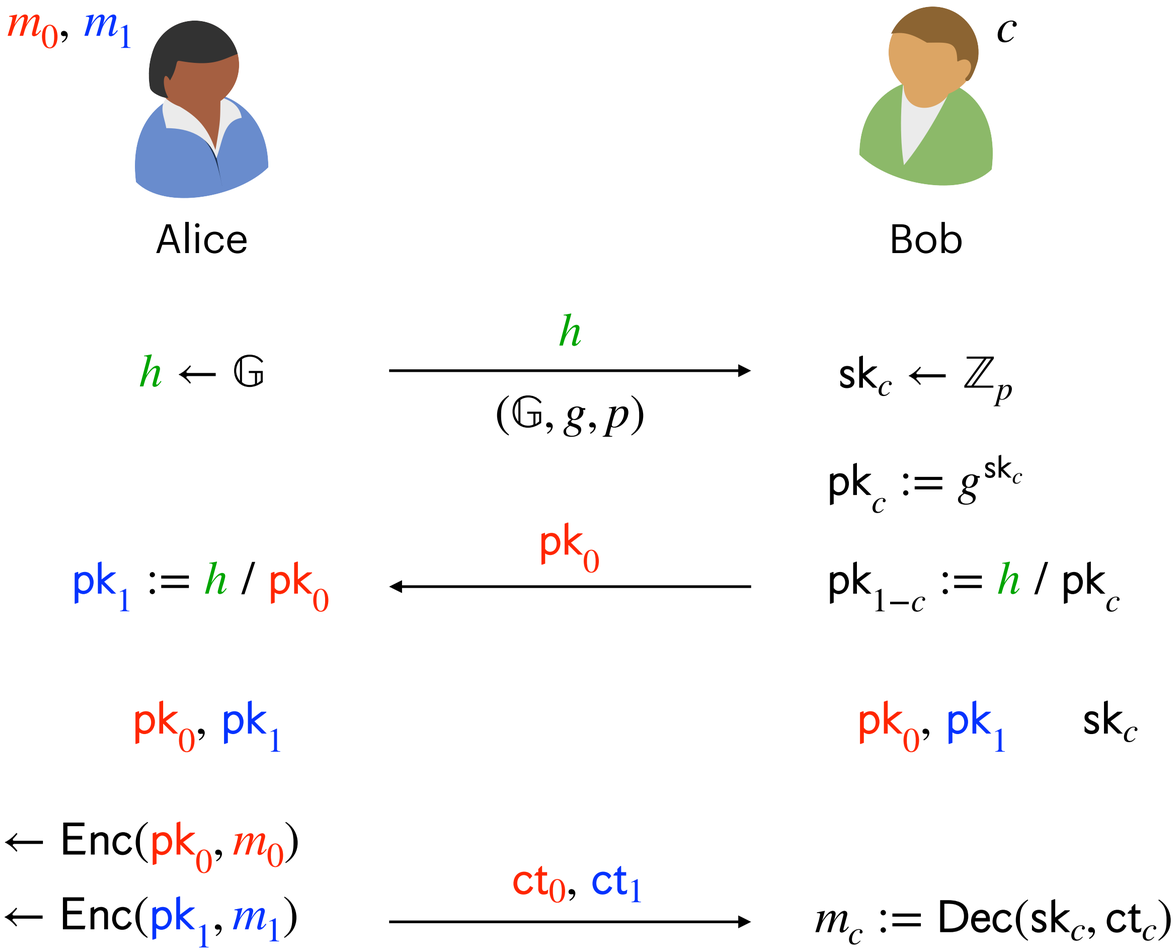
Alice uses ElGamal encryption to encrypt the messages.

Constructing Oblivious Transfer Protocol

- **Correctness:**

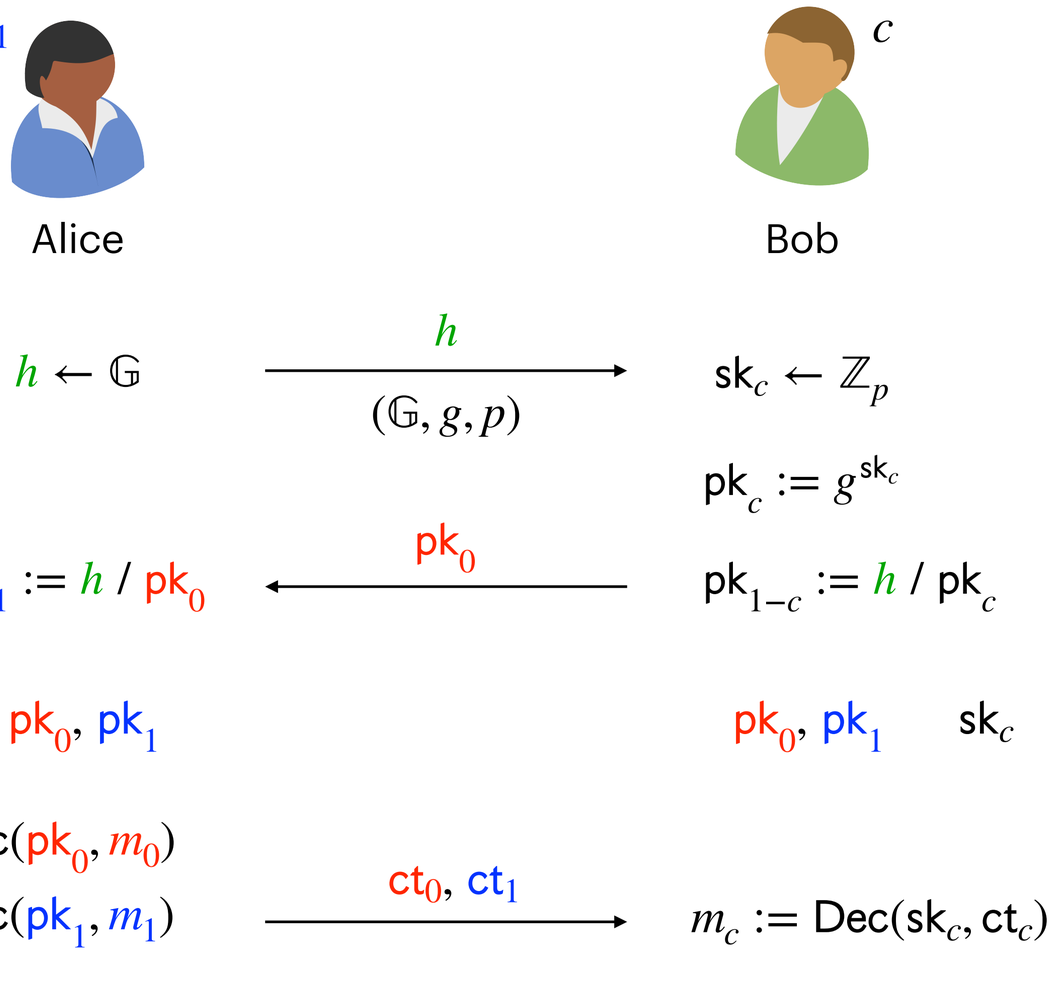


Constructing Oblivious Transfer Protocol



- **Correctness:**
  - Bob ensures that

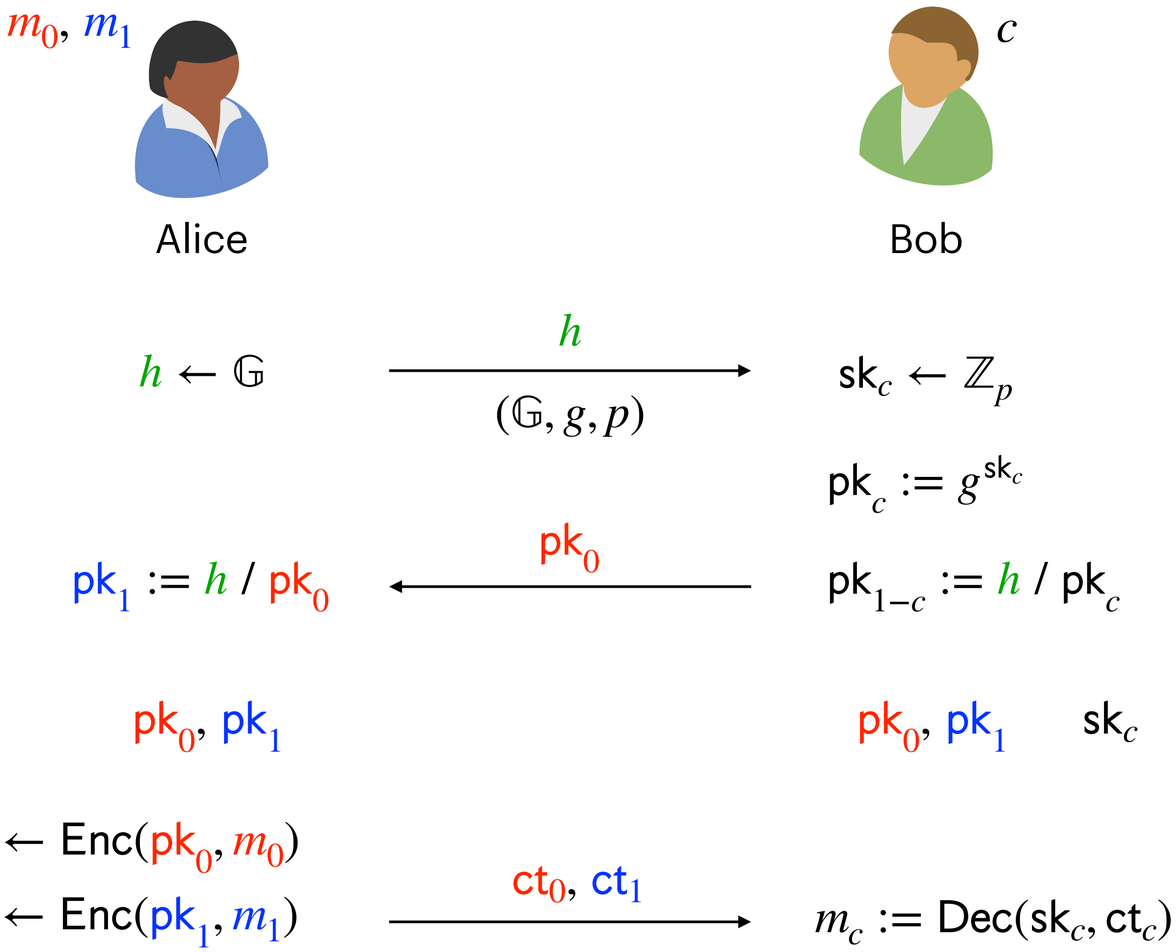
Constructing Oblivious Transfer Protocol



- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

Constructing Oblivious Transfer Protocol

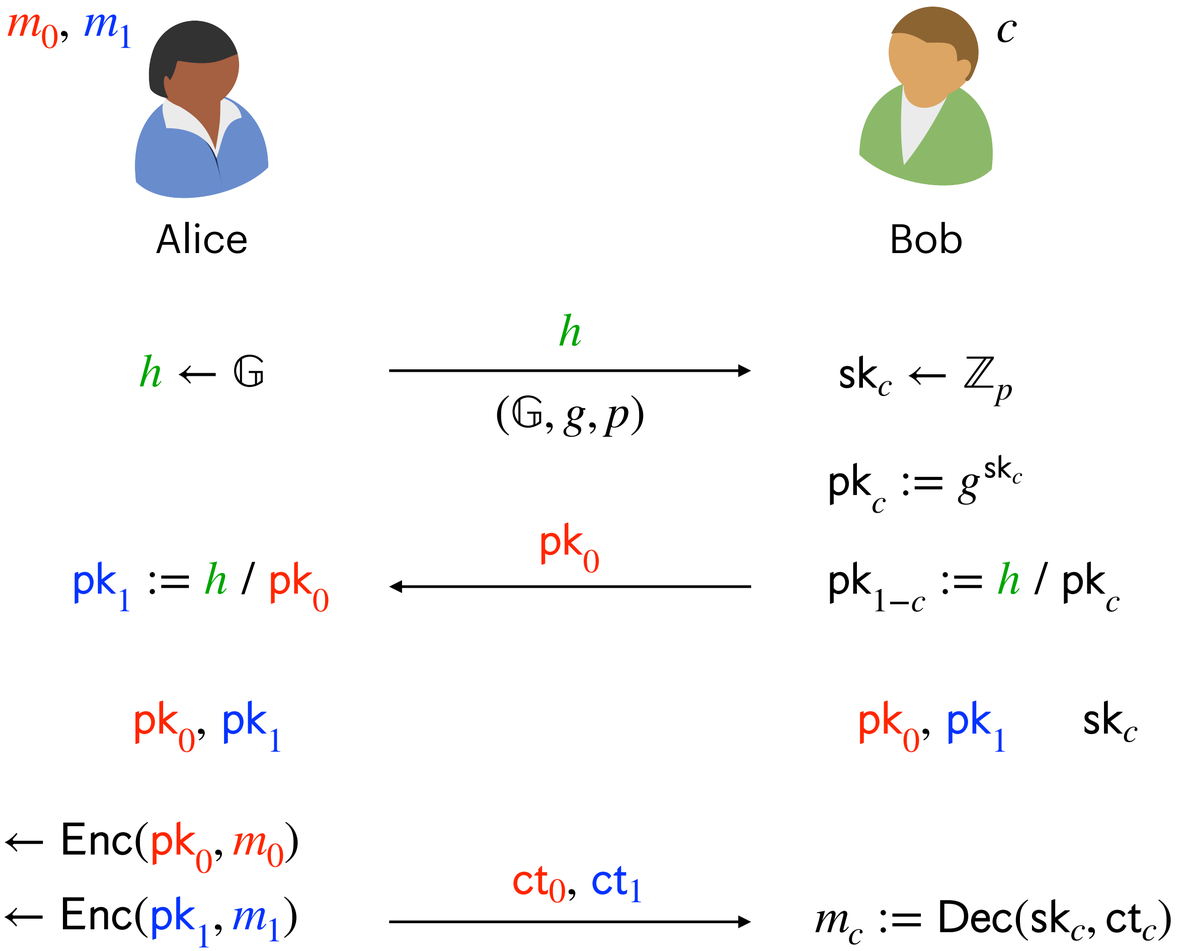


- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

  - Therefore, Alice computes the same public keys as Bob.

Constructing Oblivious Transfer Protocol

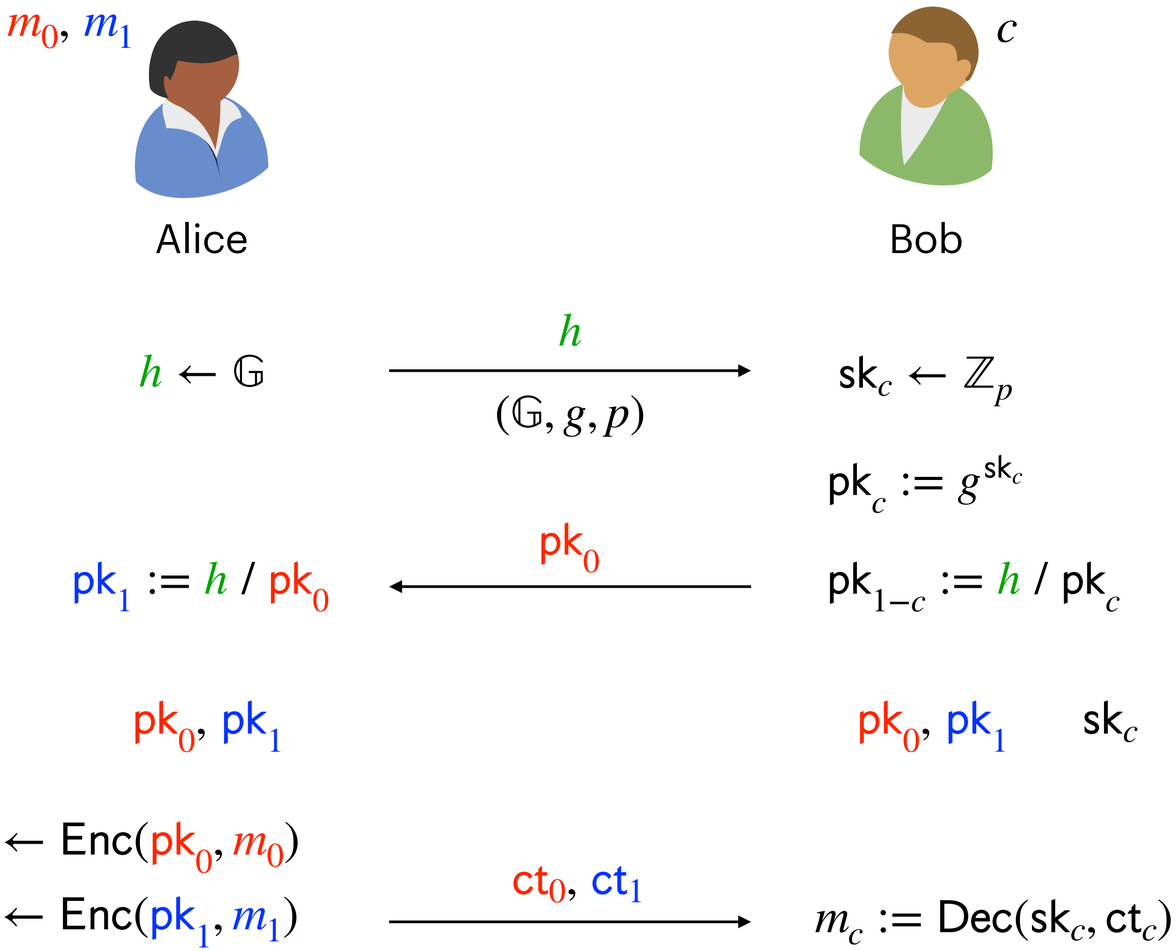


- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

  - Therefore, Alice computes the same public keys as Bob.
  - Bob outputs correct m_c due to correctness of decryption.

Constructing Oblivious Transfer Protocol

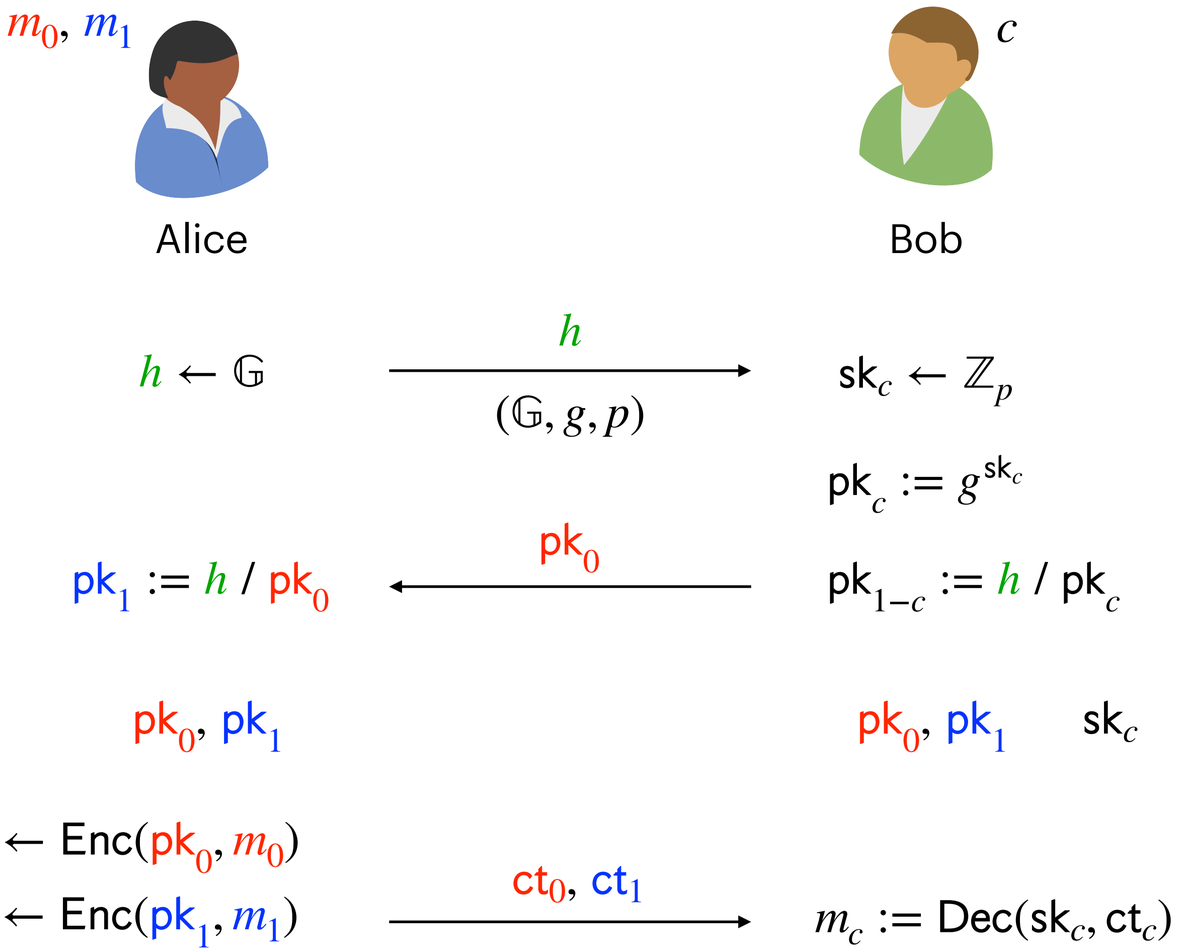


- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

  - Therefore, Alice computes the same public keys as Bob.
  - Bob outputs correct m_c due to correctness of decryption.
- **Security:**

Constructing Oblivious Transfer Protocol

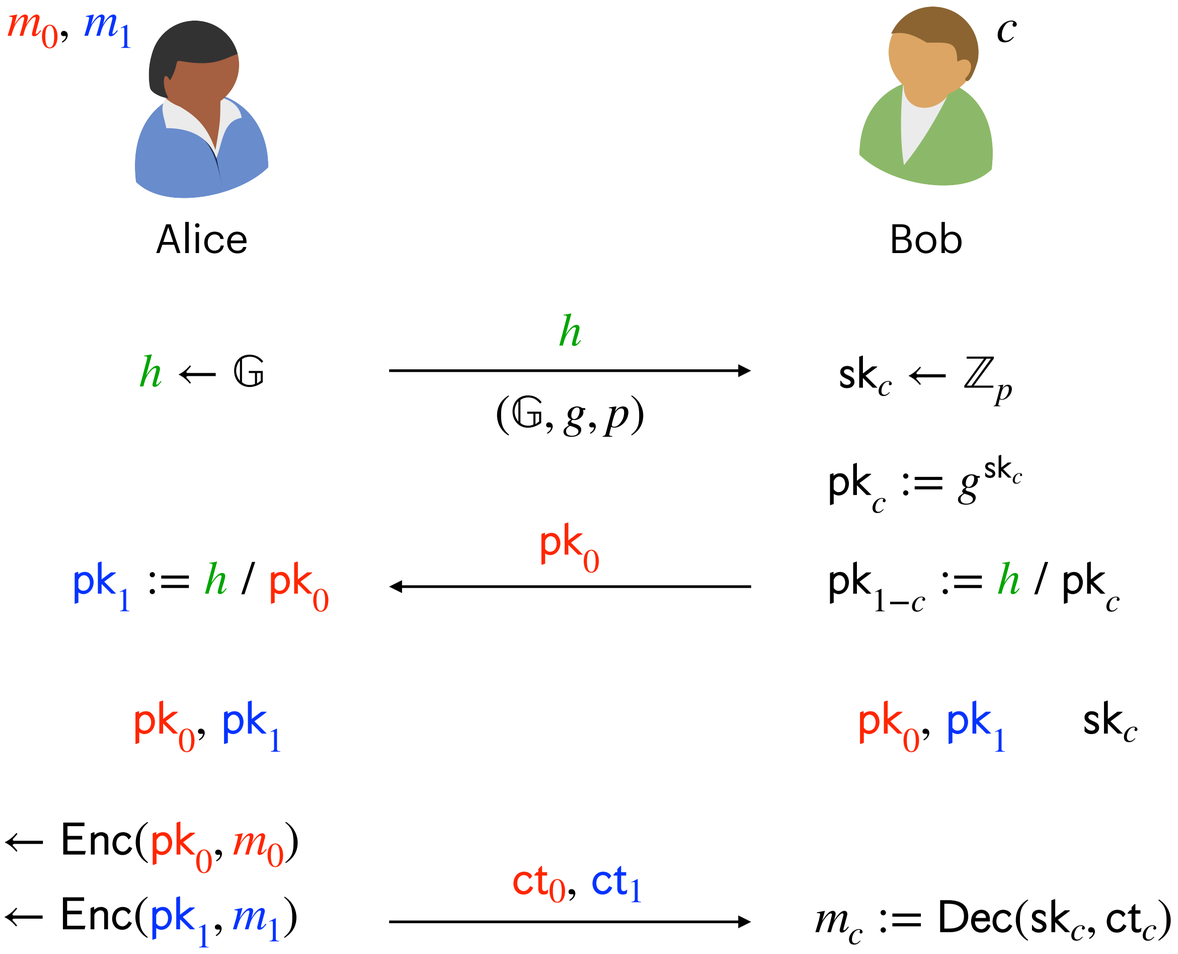


- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

  - Therefore, Alice computes the same public keys as Bob.
  - Bob outputs correct m_c due to correctness of decryption.
- **Security:**
  - h is a random group element chosen by Alice. Bob cannot efficiently compute its discrete log.

Constructing Oblivious Transfer Protocol

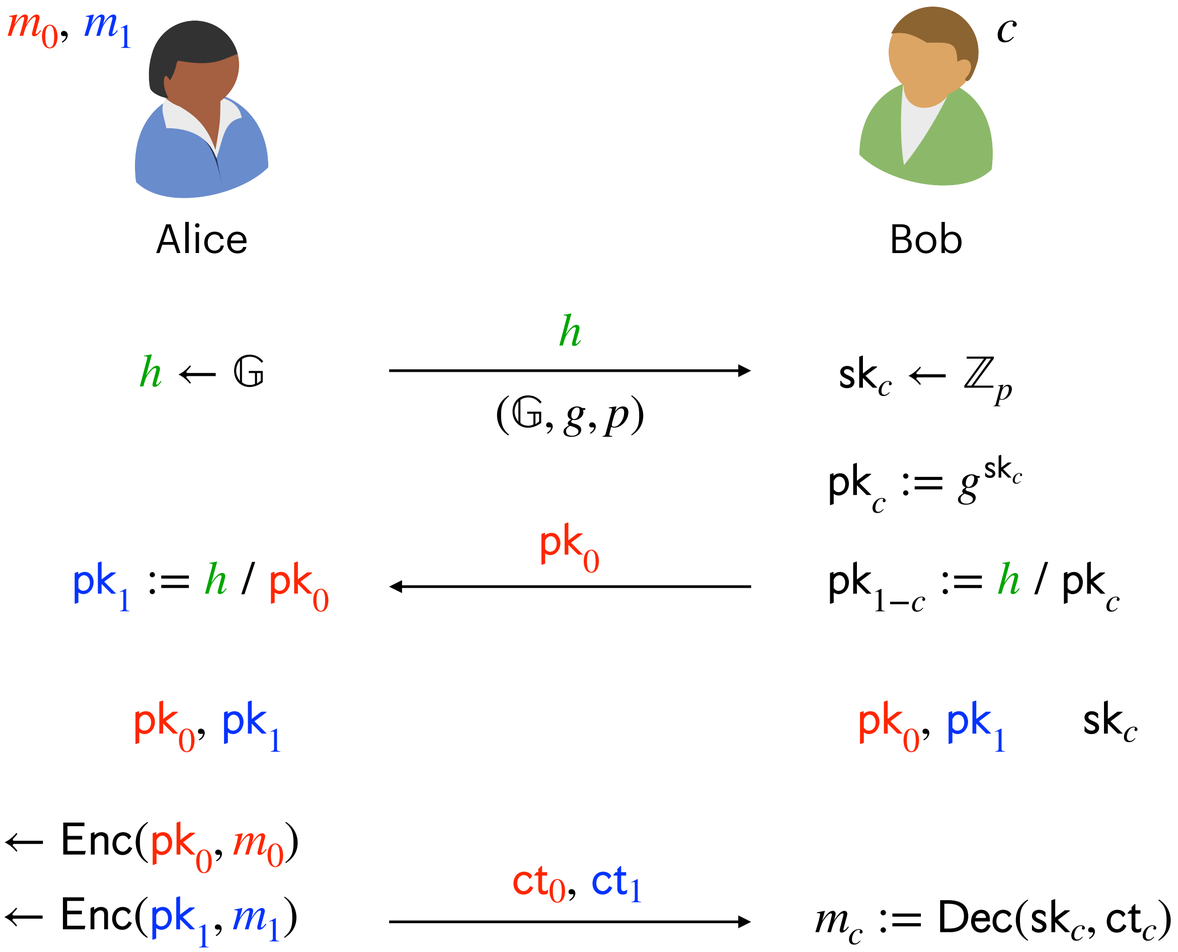


- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

  - Therefore, Alice computes the same public keys as Bob.
  - Bob outputs correct m_c due to correctness of decryption.
- **Security:**
  - h is a random group element chosen by Alice. Bob cannot efficiently compute its discrete log.
  - Bob cannot efficiently compute the discrete log of

Constructing Oblivious Transfer Protocol



- **Correctness:**
  - Bob ensures that

$$\mathsf{pk}_c \cdot \mathsf{pk}_{1-c} = \mathsf{pk}_0 \cdot \mathsf{pk}_1 = h.$$

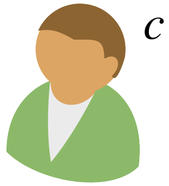
  - Therefore, Alice computes the same public keys as Bob.
  - Bob outputs correct m_c due to correctness of decryption.
- **Security:**
  - h is a random group element chosen by Alice. Bob cannot efficiently compute its discrete log.
  - Bob cannot efficiently compute the discrete log of

$$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c.$$

Constructing Oblivious Transfer Protocol

- **Simulator** $S_A(m_0, m_1)$

Alice



Bob

m_0, m_1 c

$h \leftarrow \mathbb{G}$

h →

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

$\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$

← pk_0

$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

$\mathsf{pk}_0, \mathsf{pk}_1$ $\mathsf{pk}_0, \mathsf{pk}_1$ sk_c

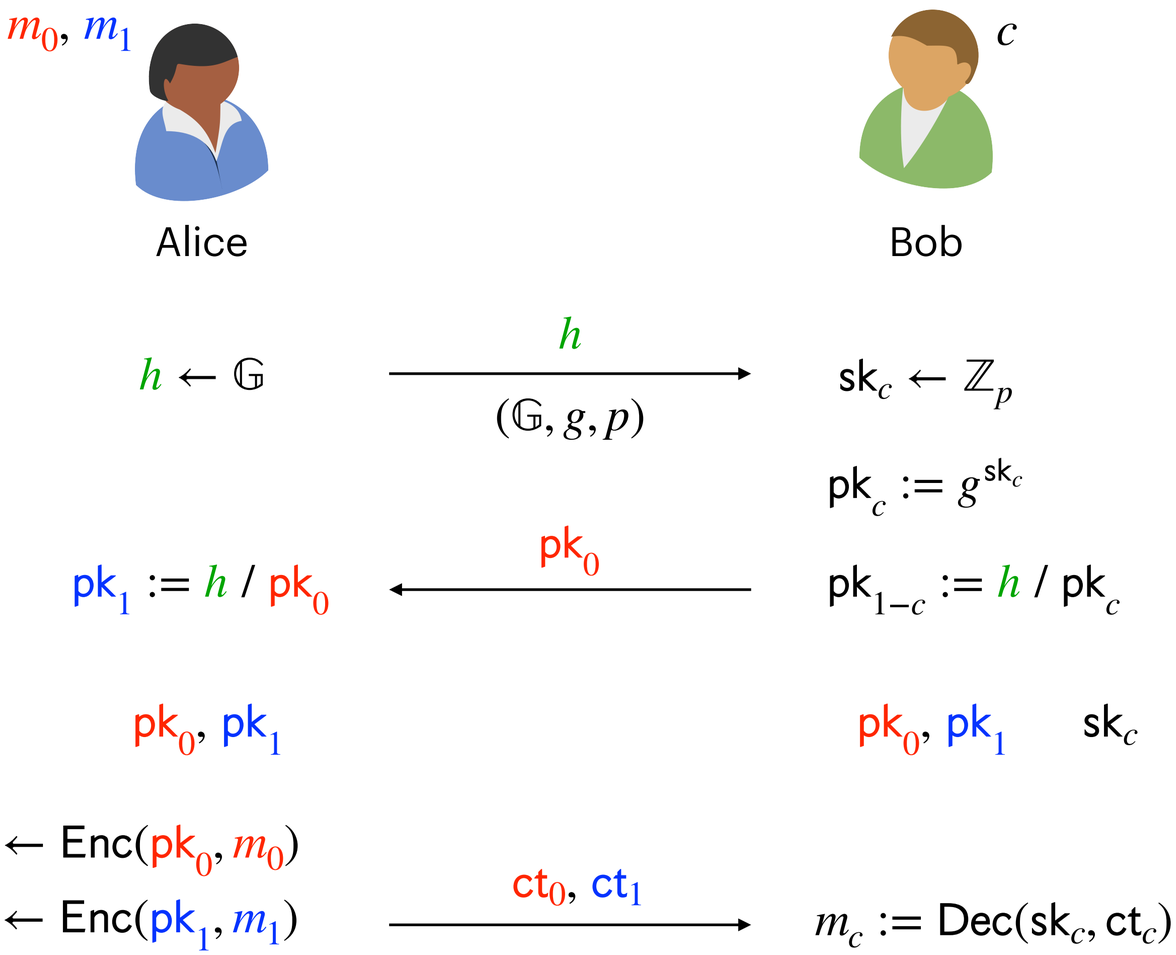
$\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}_0, m_0)$

$\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_1, m_1)$

$\mathsf{ct}_0, \mathsf{ct}_1$ →

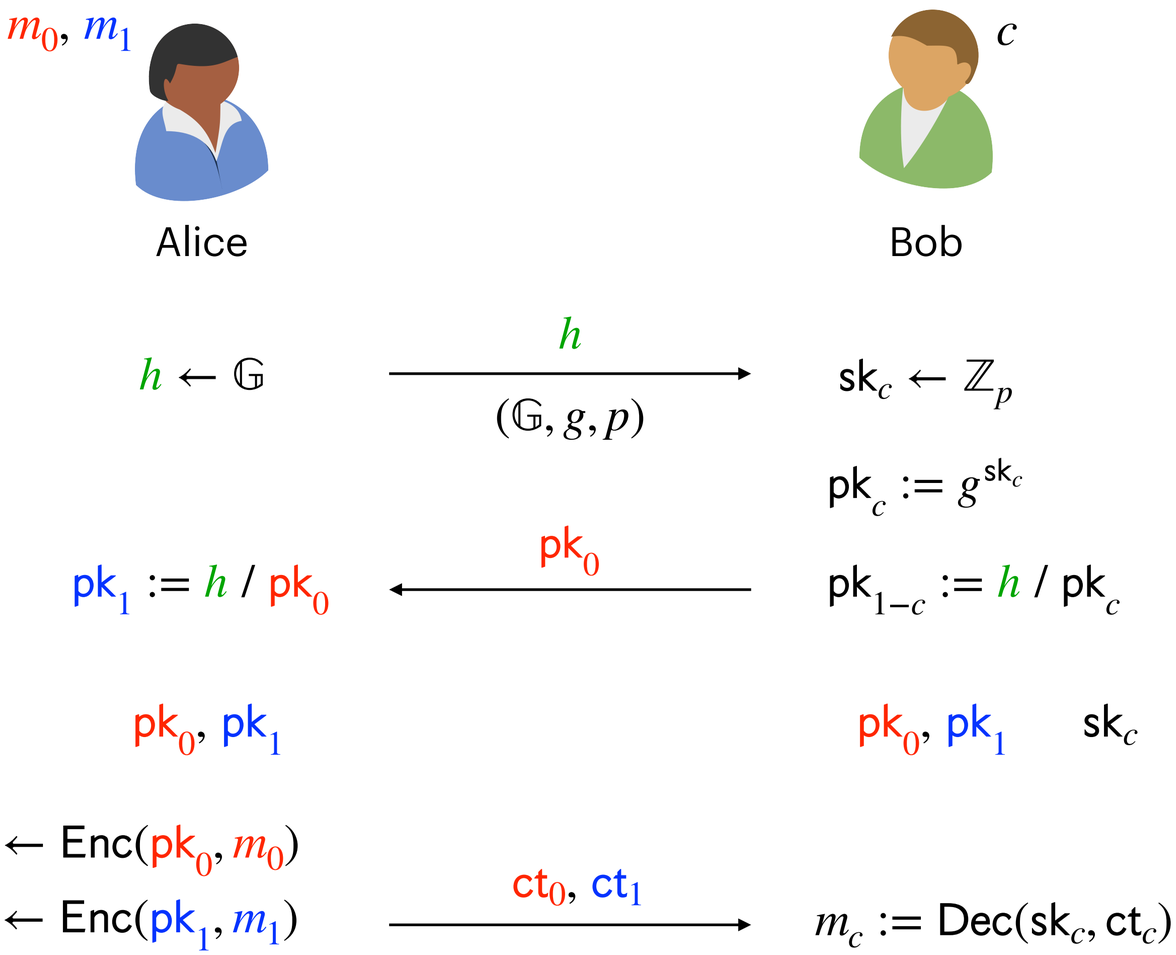
$m_c := \mathsf{Dec}(\mathsf{sk}_c, \mathsf{ct}_c)$

Constructing Oblivious Transfer Protocol



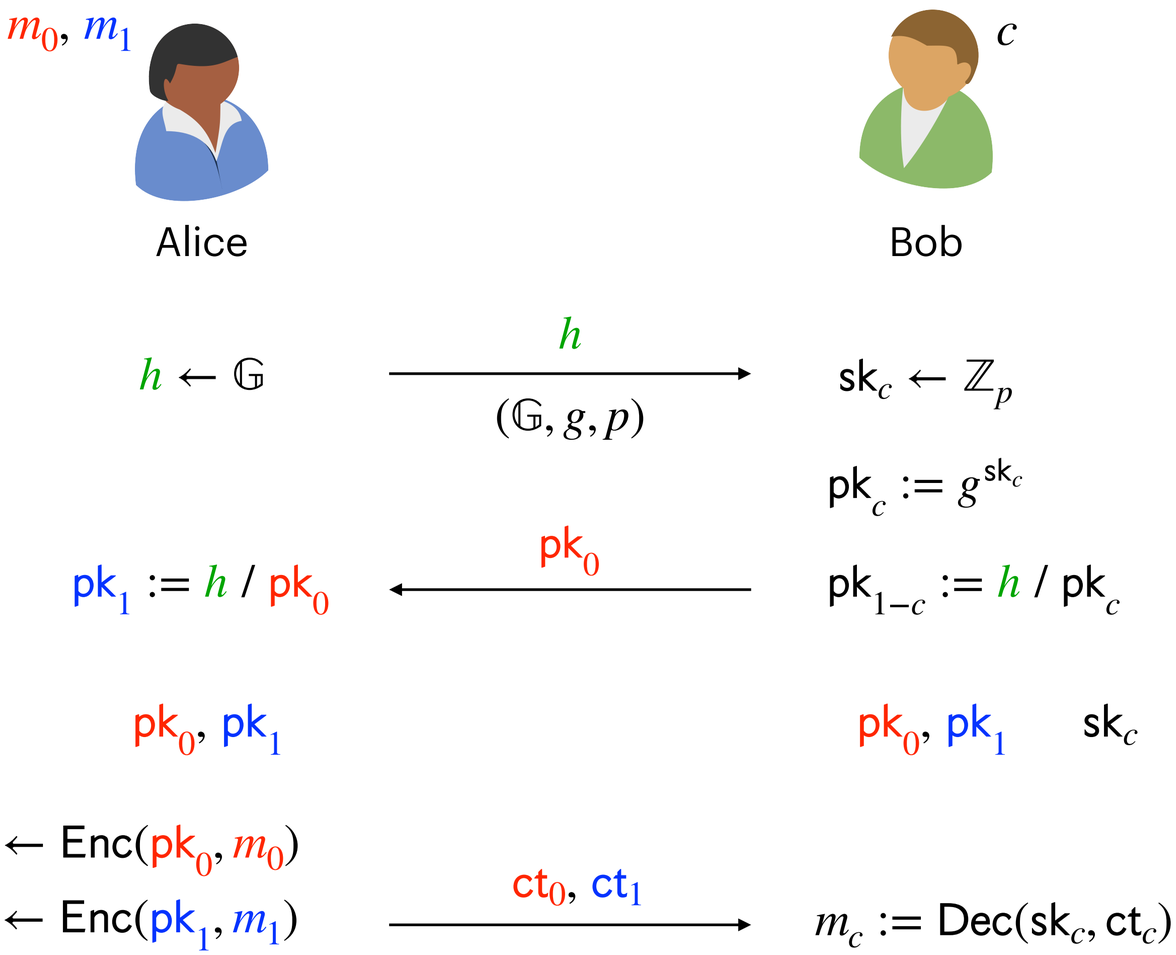
- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.

Constructing Oblivious Transfer Protocol



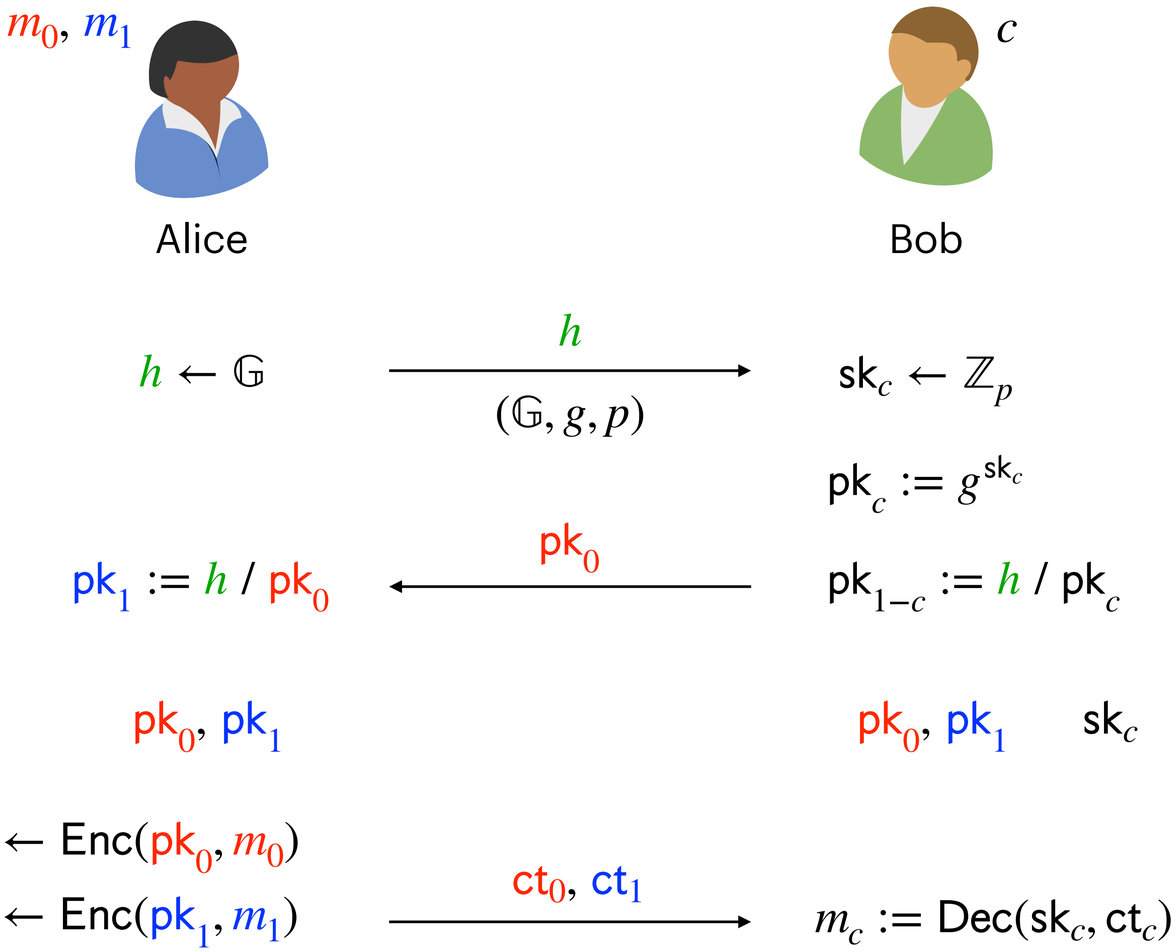
- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.

Constructing Oblivious Transfer Protocol



- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.

Constructing Oblivious Transfer Protocol

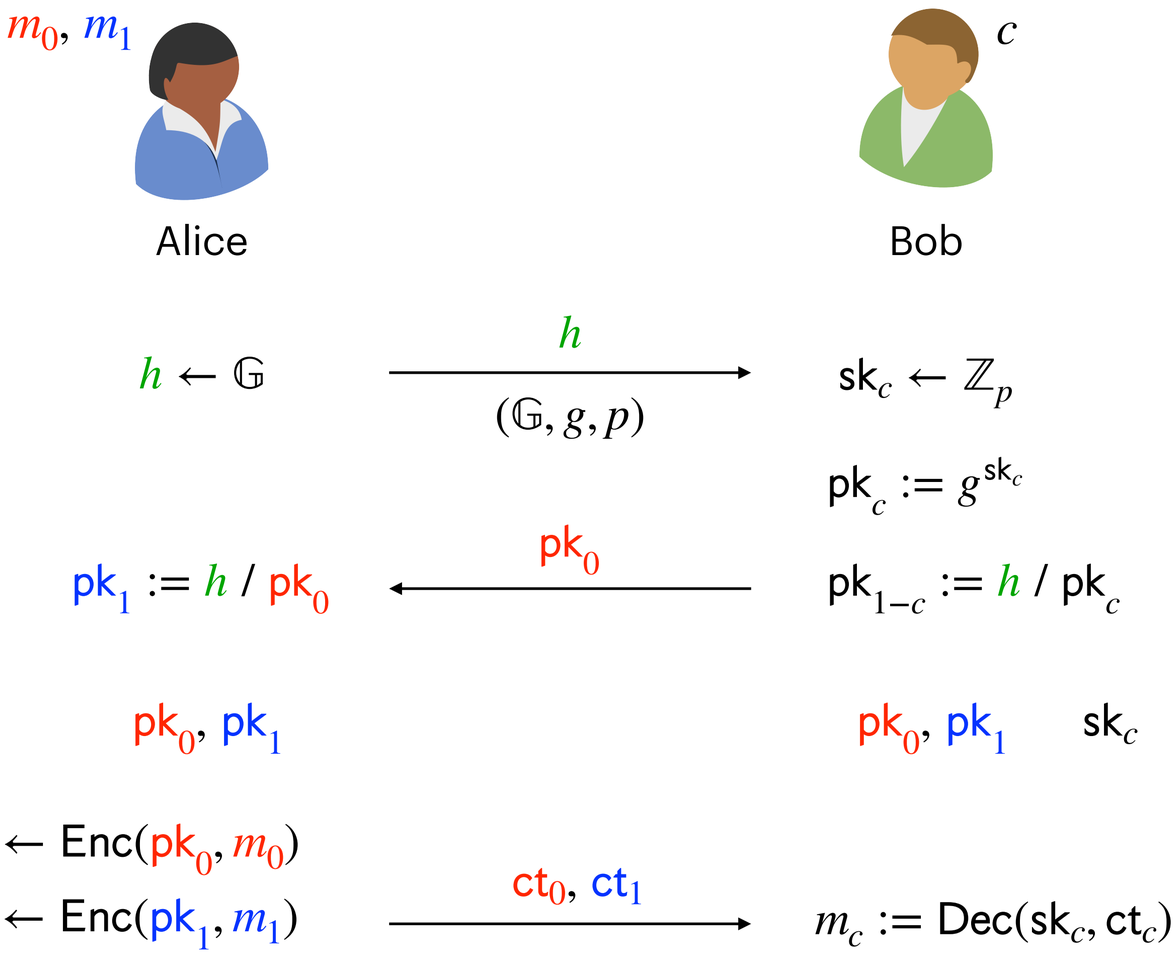


- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

$$\{S_A(m_0, m_1), m_c\} \overset{c}{\approx}$$

$$\{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)], \mathsf{Output}[A(m_0, m_1) \leftrightarrow B(c)]\}$$

Constructing Oblivious Transfer Protocol

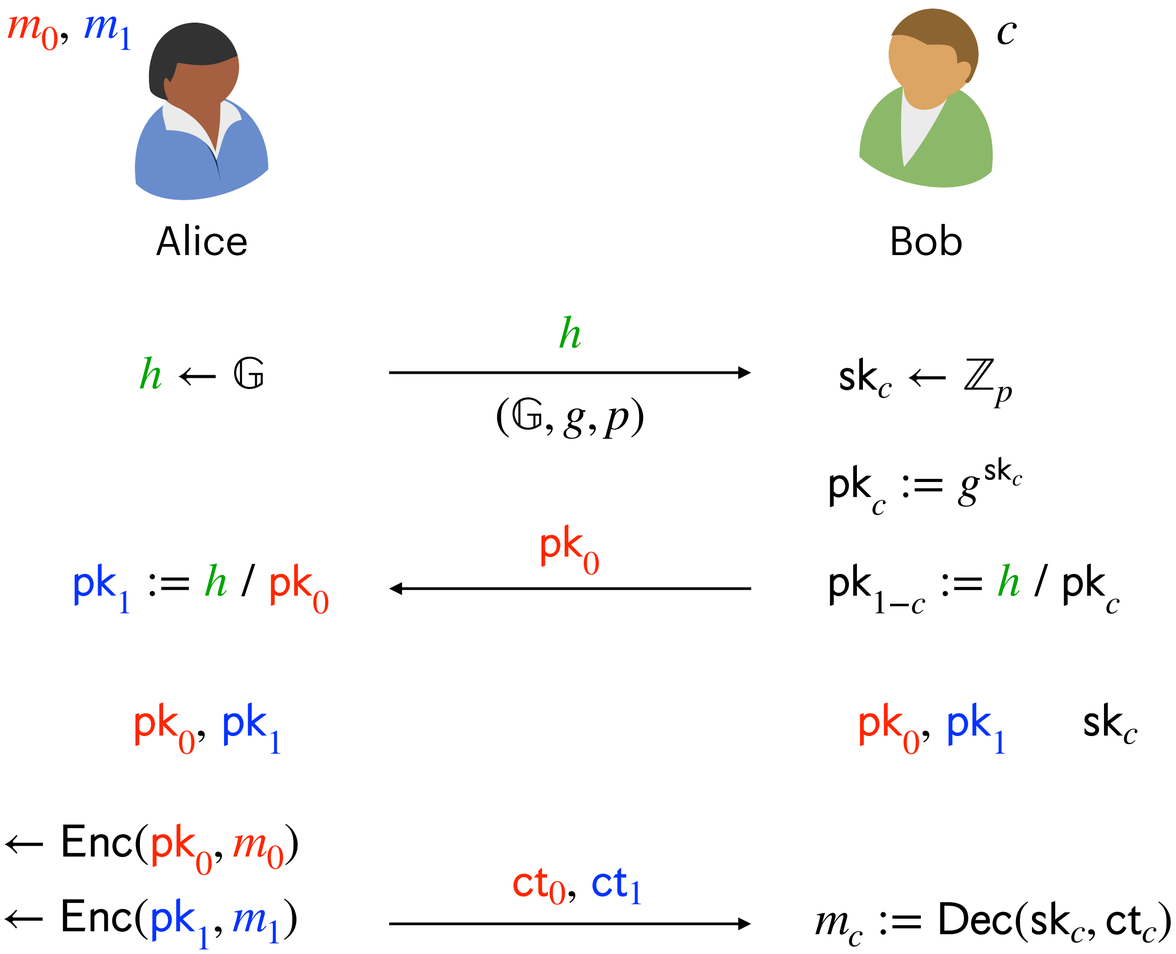


- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

$$\{S_A(m_0, m_1)\} \overset{c}{\approx} \{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

Equivalent definitions for deterministic programs.

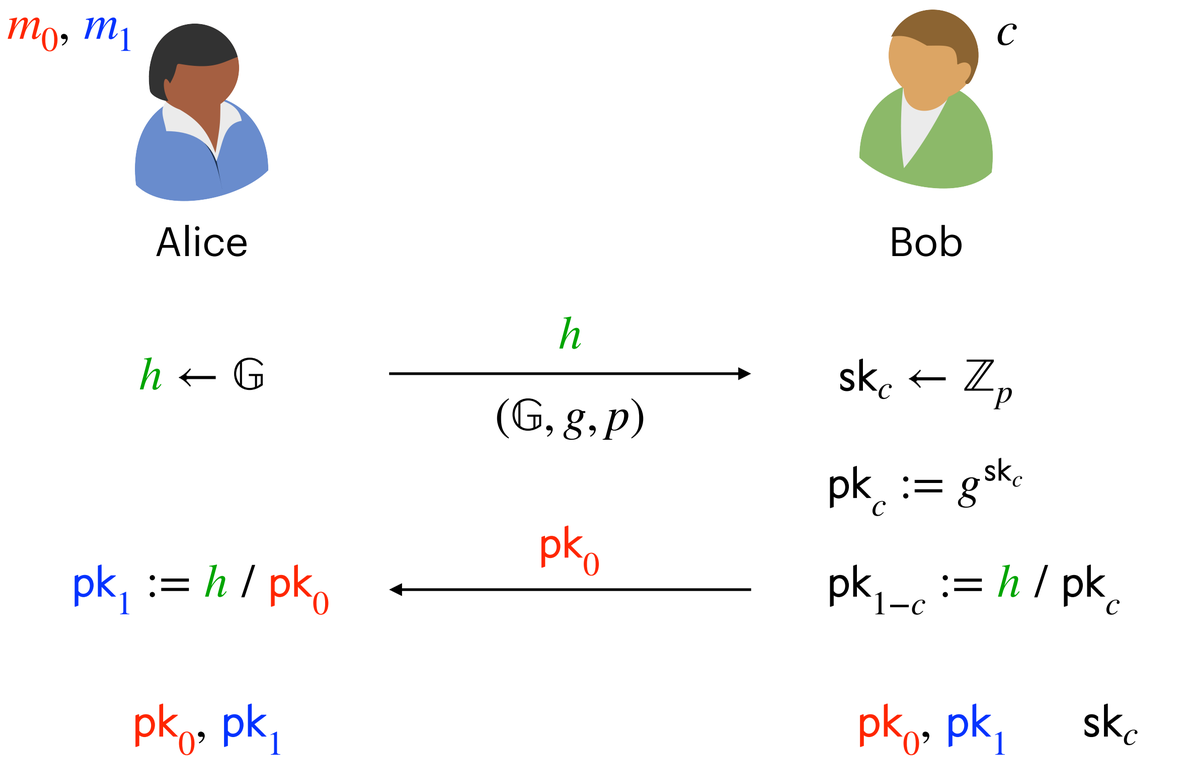
Constructing Oblivious Transfer Protocol



- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

$$\{S_A(m_0, m_1)\} \overset{c}{\approx} \{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

Constructing Oblivious Transfer Protocol

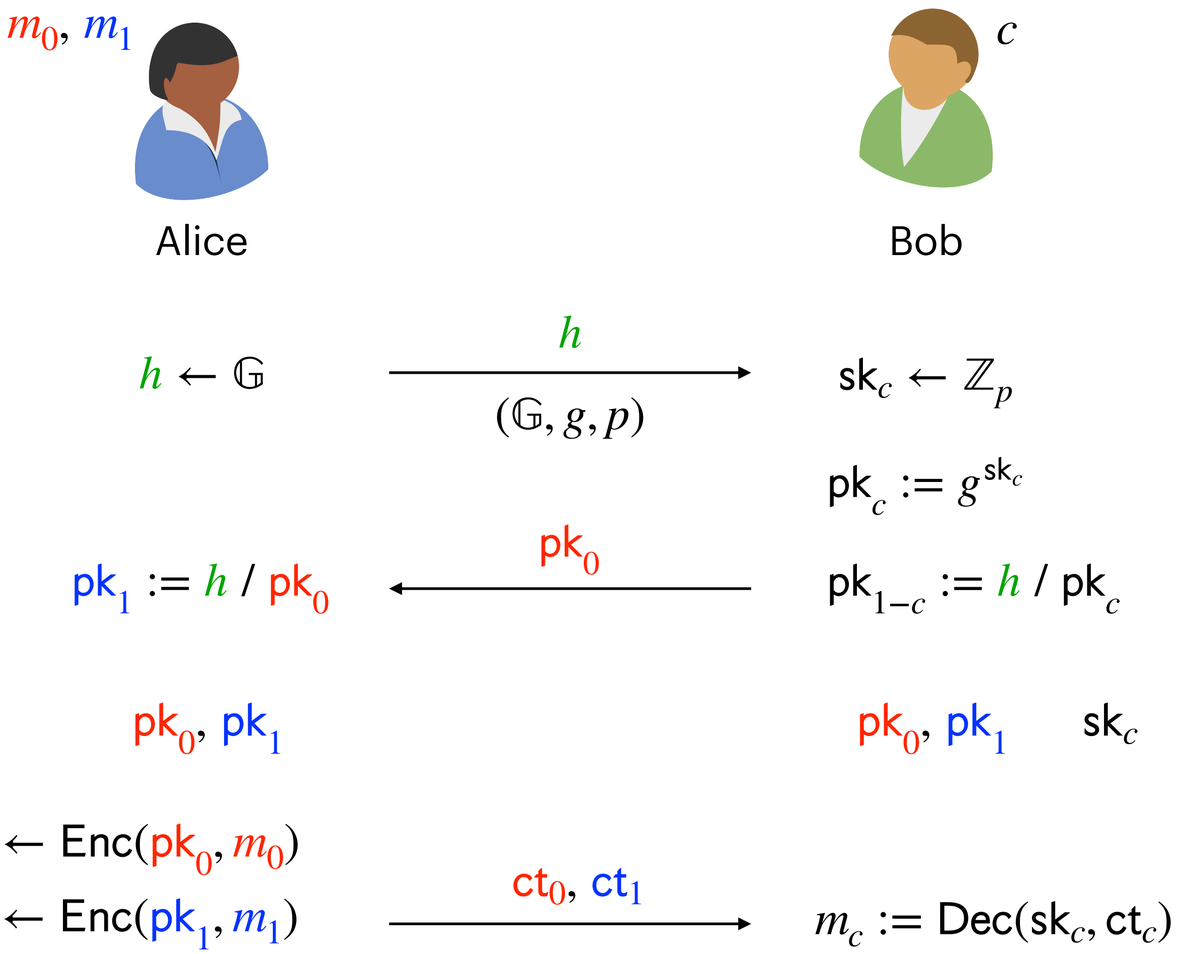


- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

  $\{S_A(m_0, m_1)\} \overset{c}{\approx} \{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)]\}.$

  - The only difference is in computation of pk_0.

Constructing Oblivious Transfer Protocol

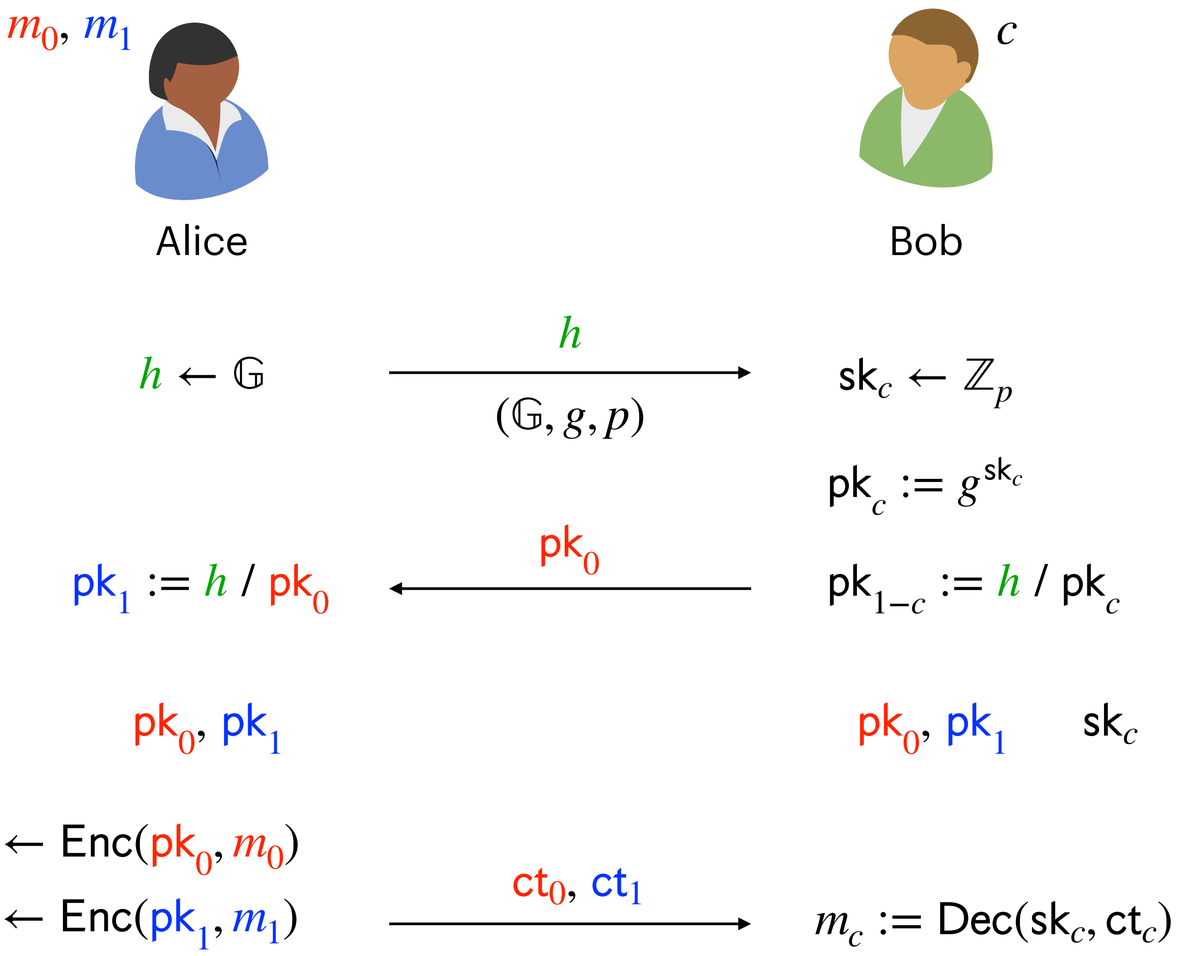


- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

  $\{S_A(m_0, m_1)\} \overset{c}{\approx} \{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)]\}.$

  - The only difference is in computation of pk_0.
  - If $c = 0$, then Bob sends uniformly random pk_0 since sk_c is uniformly random and $\mathsf{pk}_0 := g^{\mathsf{sk}_c}$.

Constructing Oblivious Transfer Protocol



- **Simulator** $S_A(m_0, m_1)$
  - Run $A(m_0, m_1)$ and receive the first message h.
  - Sample $\mathsf{pk}_0 \xleftarrow{\$} \mathbb{G}$ as B's message.
  - Receive ct_0, ct_1.
- **Claim:**

$$\{S_A(m_0, m_1)\} \overset{c}{\approx} \{\mathsf{View}_A[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

  - The only difference is in computation of pk_0.
  - If $c = 0$, then Bob sends uniformly random pk_0 since sk_c is uniformly random and $\mathsf{pk}_0 := g^{\mathsf{sk}_c}$.
  - If $c = 1$, pk_1 is uniformly random, which means that $\mathsf{pk}_0 := h \,/\, \mathsf{pk}_1$ is uniformly random.

Constructing Oblivious Transfer Protocol

m_0, m_1

Alice

c

Bob

$h \leftarrow \mathbb{G}$

h

$(\mathbb{G}, g, p)$

$\mathsf{sk}_c \leftarrow \mathbb{Z}_p$

$\mathsf{pk}_c := g^{\mathsf{sk}_c}$

$\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$

pk_0

$\mathsf{pk}_{1-c} := h \,/\, \mathsf{pk}_c$

$\mathsf{pk}_0, \mathsf{pk}_1$

$\mathsf{pk}_0, \mathsf{pk}_1 \quad \mathsf{sk}_c$

$\mathsf{ct}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}_0, m_0)$

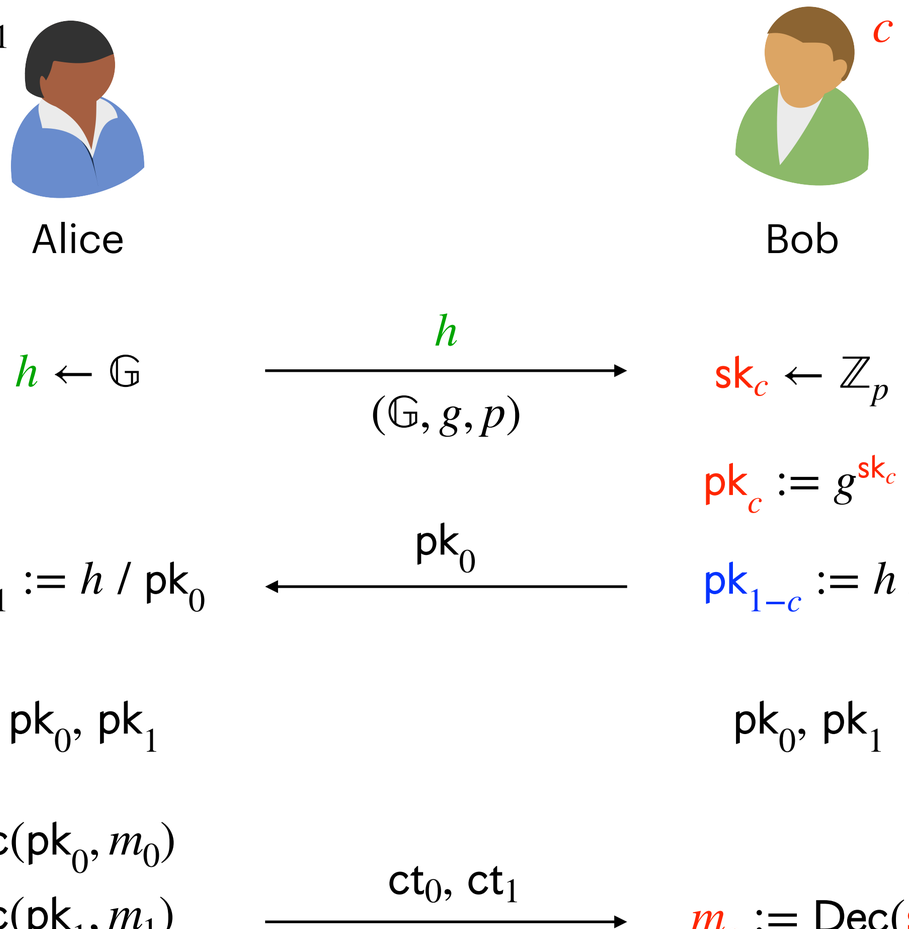
$\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}_1, m_1)$

$\mathsf{ct}_0, \mathsf{ct}_1$

$m_c := \mathsf{Dec}(\mathsf{sk}_c, \mathsf{ct}_c)$

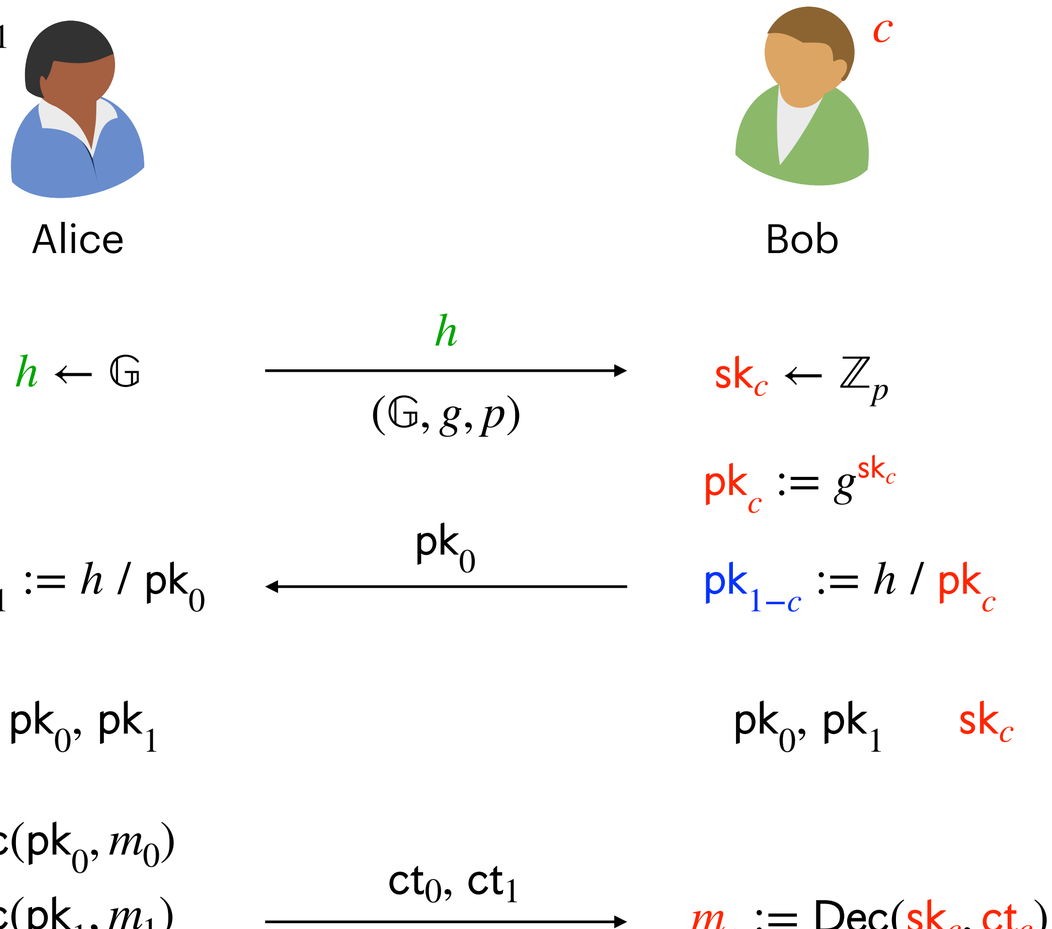
Constructing Oblivious Transfer Protocol

- **Simulator** $S_B(c, m_c)$

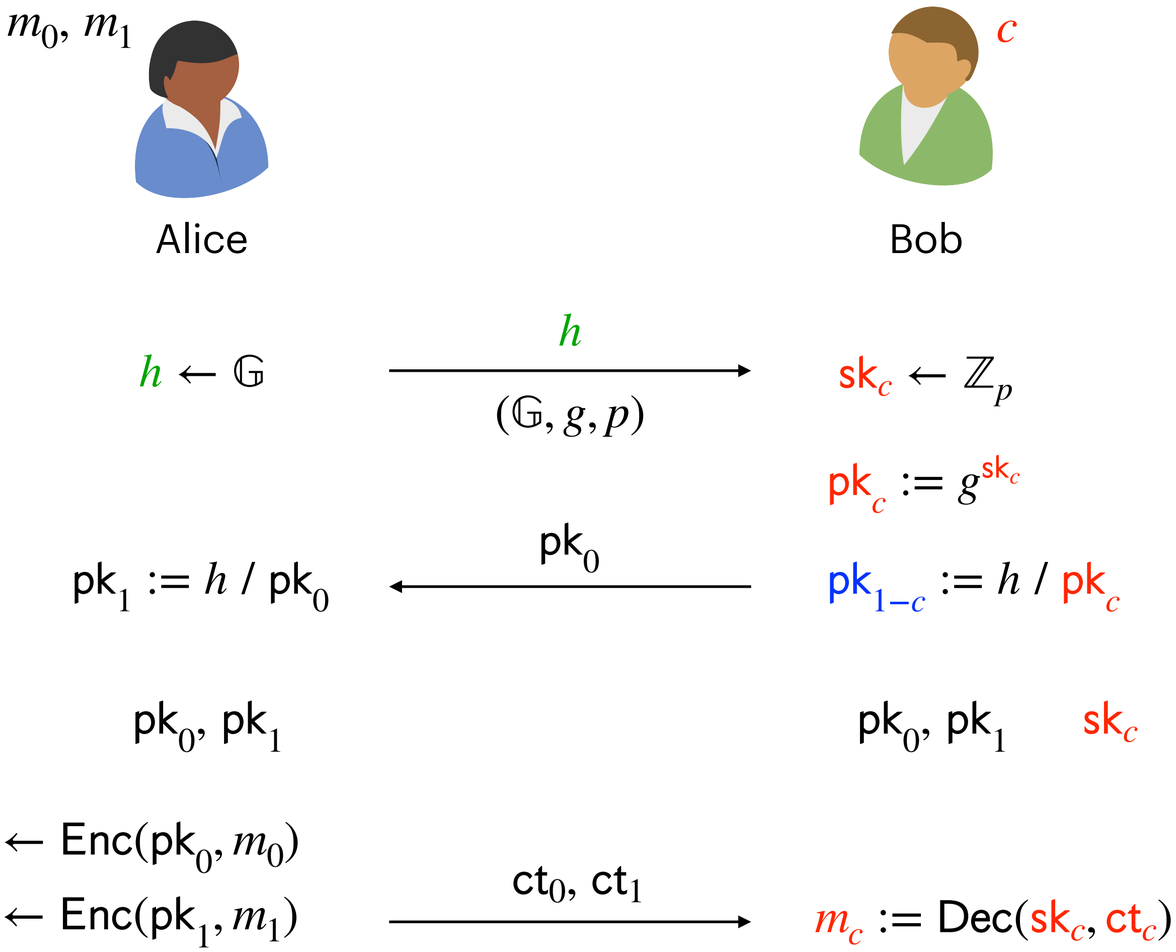


Constructing Oblivious Transfer Protocol

- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.

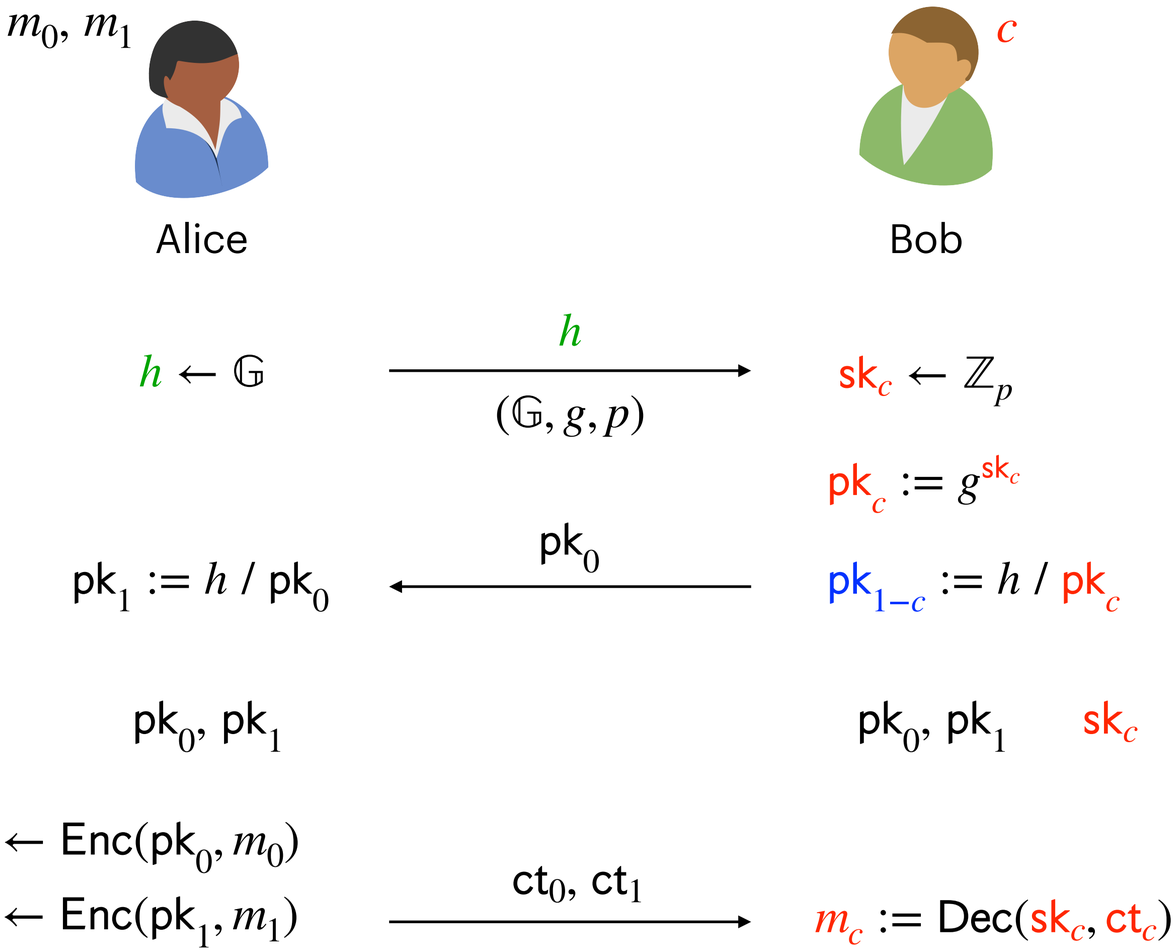


Constructing Oblivious Transfer Protocol



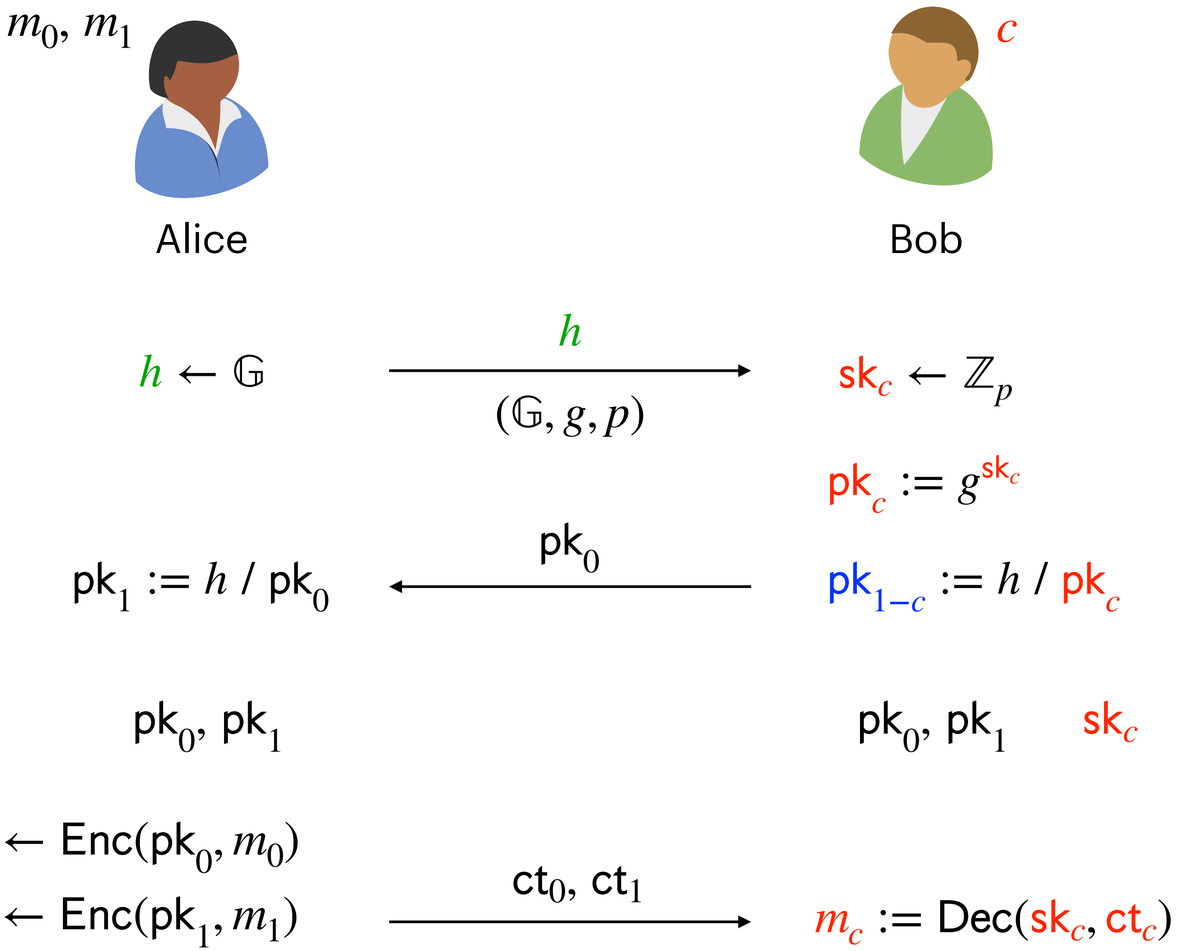
- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.

Constructing Oblivious Transfer Protocol



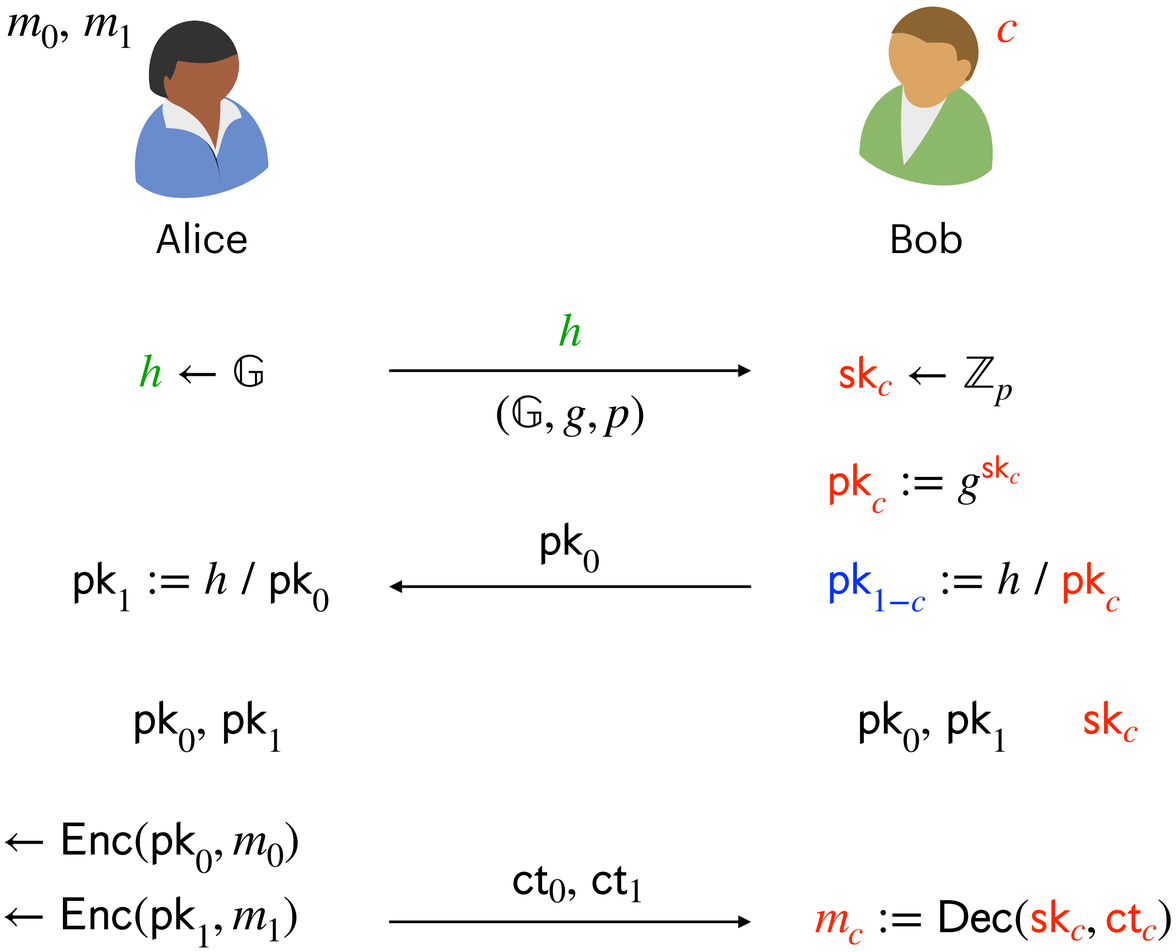
- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.

Constructing Oblivious Transfer Protocol



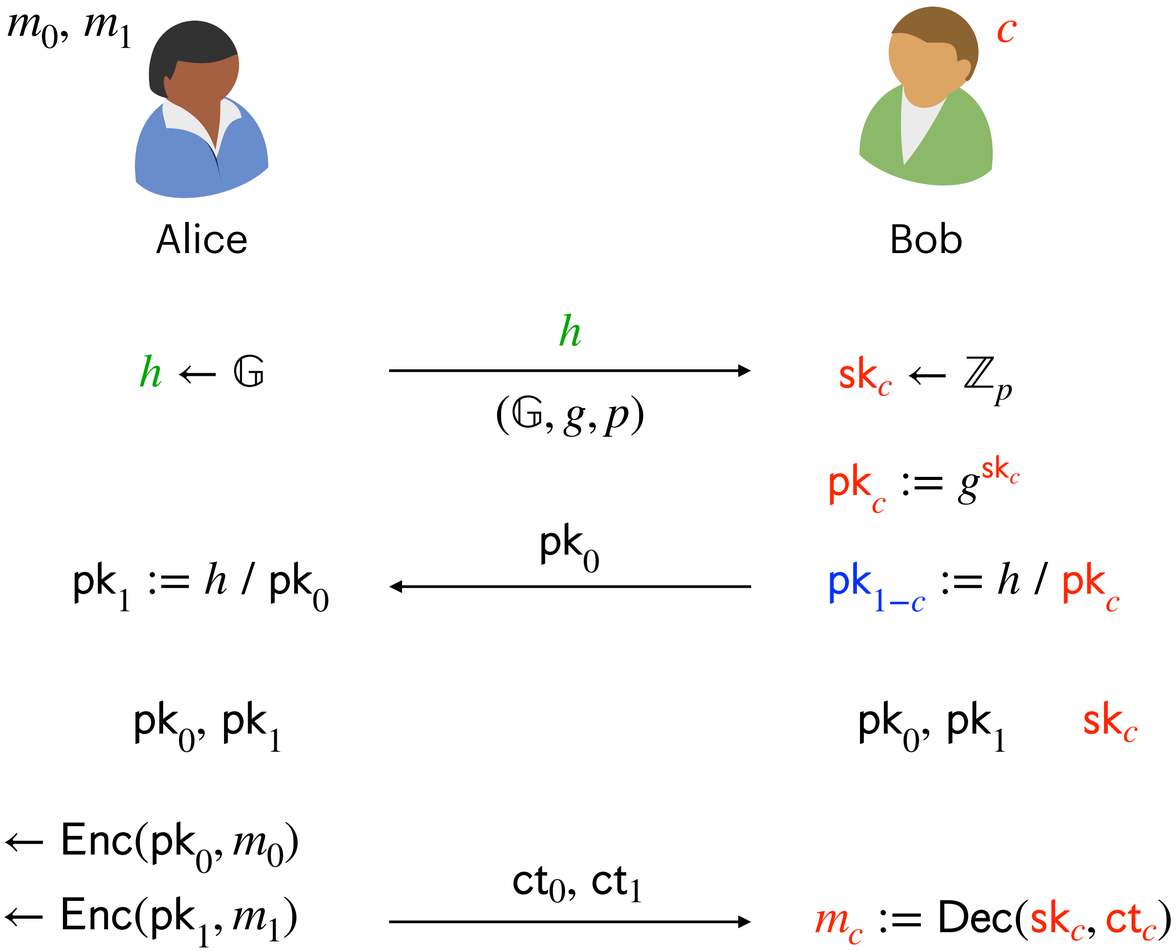
- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.

Constructing Oblivious Transfer Protocol



- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.
- **Claim:**

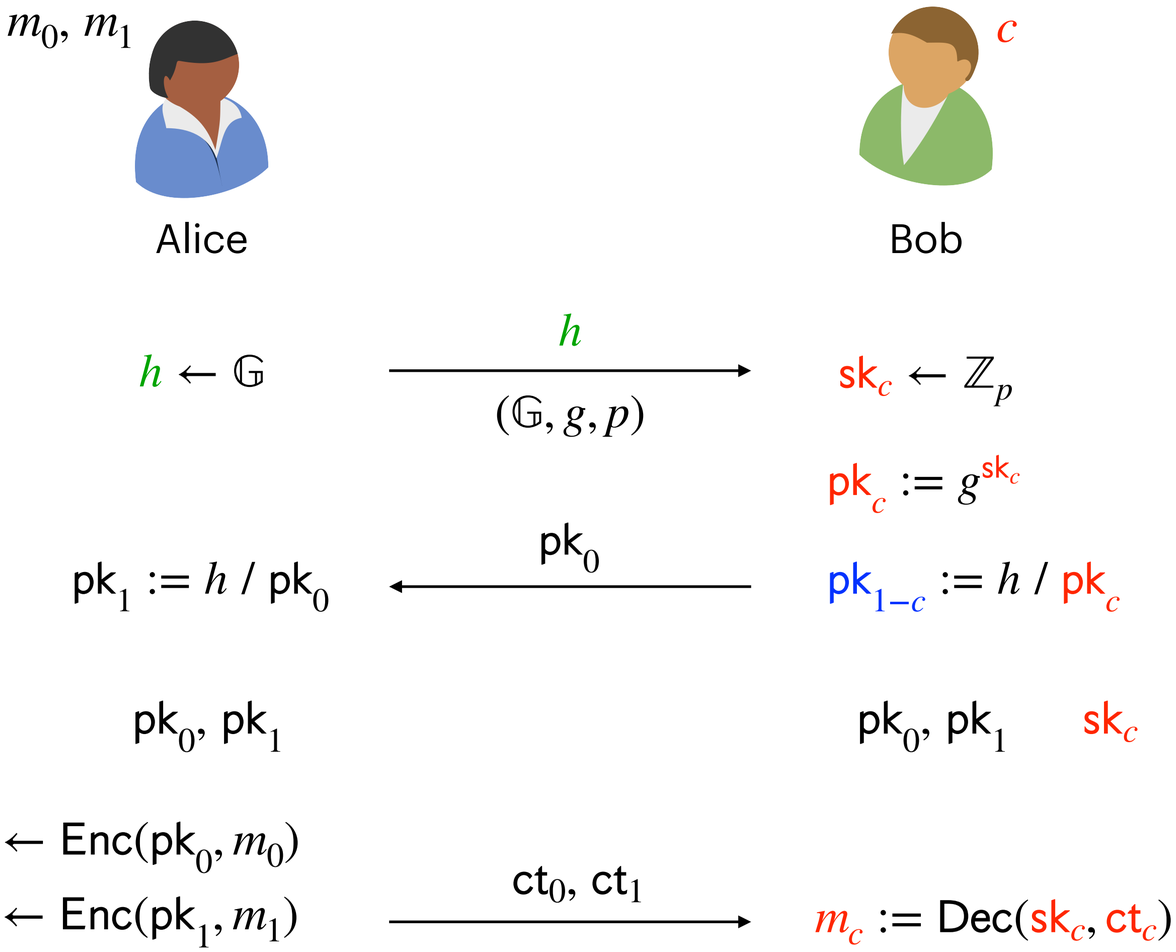
Constructing Oblivious Transfer Protocol



- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.
- **Claim:**

$$\{S_B(c, m_c)\} \overset{c}{\approx} \{\mathsf{View}_B[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

Constructing Oblivious Transfer Protocol

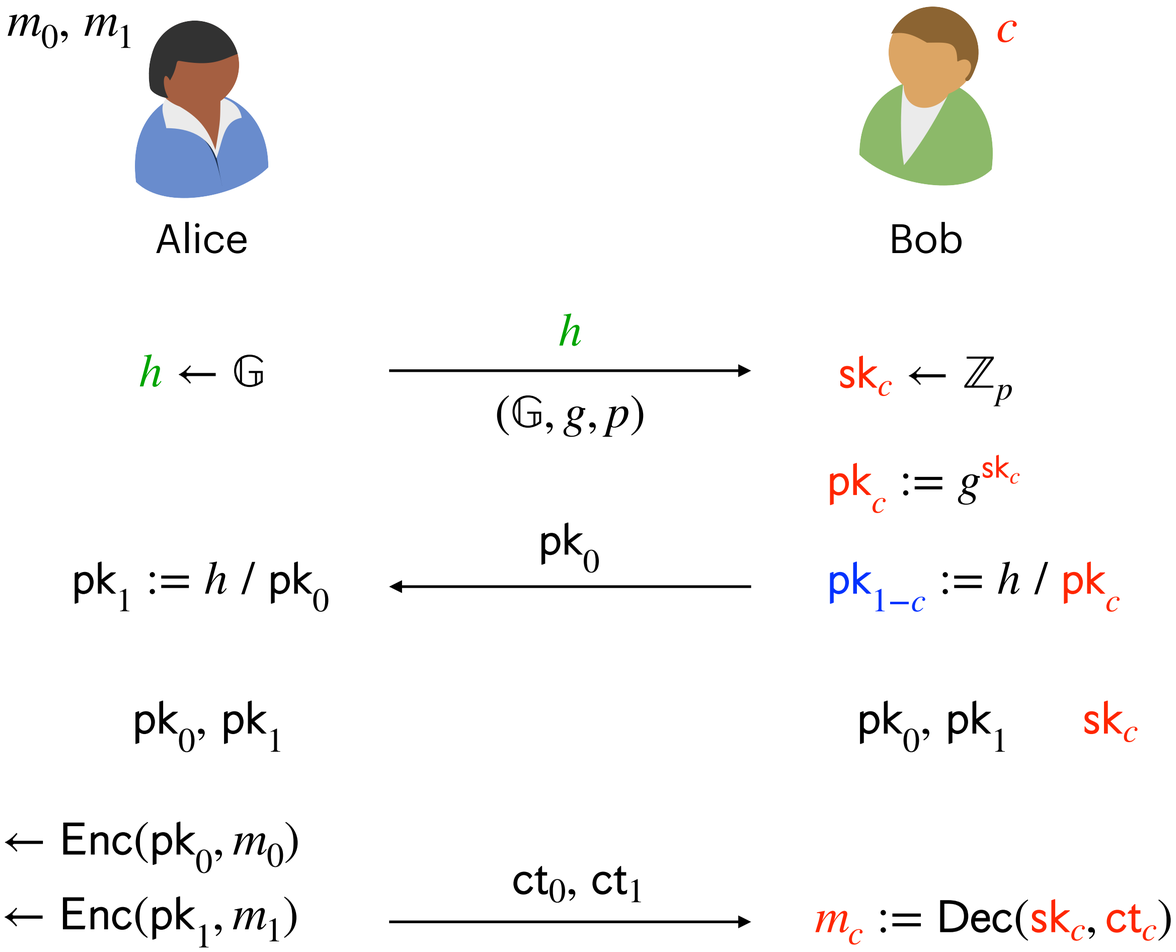


- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.
- **Claim:**

$$\{S_B(c, m_c)\} \overset{c}{\approx} \{\mathsf{View}_B[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

  - The only difference is in computation of ct_{1-c}.

Constructing Oblivious Transfer Protocol

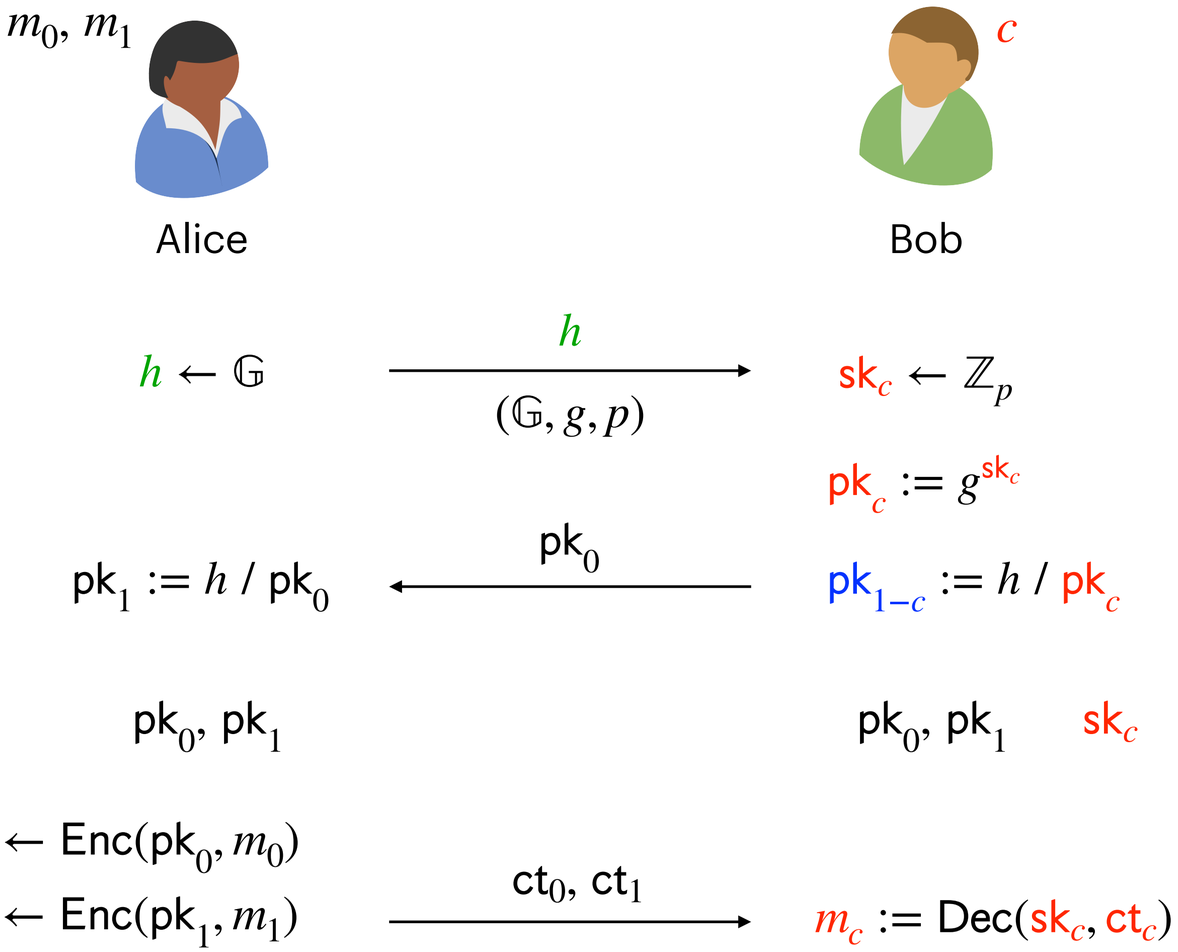


- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.
- **Claim:**

  $$\{S_B(c, m_c)\} \overset{c}{\approx} \{\mathsf{View}_B[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

  - The only difference is in computation of ct_{1-c}.
  - $\mathsf{ct}_{1-c} = \left(g^r, \mathsf{pk}_{1-c}{}^r\right) = \left(g^r, h^r \cdot \mathsf{pk}_c{}^{-r}\right)$.

Constructing Oblivious Transfer Protocol



- **Simulator** $S_B(c, m_c)$
  - Run $B(c)$.
  - Sample $h \xleftarrow{\$} \mathbb{G}$ uniformly at random as A's message.
  - Receive pk_0.
  - Compute $\mathsf{pk}_1 := h \,/\, \mathsf{pk}_0$, $\mathsf{ct}_c \leftarrow \mathsf{Enc}(\mathsf{pk}_c, m_c)$ and $\mathsf{ct}_{1-c} \leftarrow \mathsf{Enc}(\mathsf{pk}_{1-c}, 1)$.
- **Claim:**

  $$\{S_B(c, m_c)\} \overset{c}{\approx} \{\mathsf{View}_B[A(m_0, m_1) \leftrightarrow B(c)]\}.$$

  - The only difference is in computation of ct_{1-c}.
  - $\mathsf{ct}_{1-c} = \left(g^r, \mathsf{pk}_{1-c}{}^r\right) = \left(g^r, h^r \cdot \mathsf{pk}_c{}^{-r}\right)$.
  - In the first hybrid, switch $\mathsf{pk}_c{}^{-r}$ to a random group element and in the second hybrid use ElGamal security under public key h.

Remarks on Oblivious Transfer Construction

- The previous protocol easily generalizes to construct 1-out-of-k OT for $k > 2$.

Remarks on Oblivious Transfer Construction

- The previous protocol easily generalizes to construct 1-out-of-k OT for $k > 2$.
- Malicious Security
  - We showed that the protocol is semi-honest secure. In reality, the adversary might be malicious.

Remarks on Oblivious Transfer Construction

- The previous protocol easily generalizes to construct 1-out-of-k OT for $k > 2$.
- Malicious Security
  - We showed that the protocol is semi-honest secure. In reality, the adversary might be malicious.
  - Goldreich-Micali-Wigderson gave a compiler to transform any protocol secure against semi-honest adversaries to one secure against malicious adversary.

Remarks on Oblivious Transfer Construction

- The previous protocol easily generalizes to construct 1-out-of-k OT for $k > 2$.
- Malicious Security
  - We showed that the protocol is semi-honest secure. In reality, the adversary might be malicious.
  - Goldreich-Micali-Wigderson gave a compiler to transform any protocol secure against semi-honest adversaries to one secure against malicious adversary.
  - The transformation uses a coin-flipping protocol (to make sure that the adversary's random tape is truly random) and zero-knowledge proofs (to make sure the adversary is following the protocol description).