Data Privacy
Crypto-based Solutions

Erman Ayday

Secure Computation

- Sensitive data is divided among two or more different parties
- The aim being to run a data mining algorithm on the union of the parties ‘ databases without allowing any party to view another individual's private data
- Example: Medical data
  - Different hospitals wish to jointly mine their patient data for the purpose of medical research
  - It is necessary to find a solution that enables the hospitals to compute the desired data mining algorithm on the union of their databases
- Similar examples: intelligence agencies, governments, etc.

Possible Solutions

- Pool all of the data in one place and run the data mining algorithm on the pooled data?
- Not acceptable
  - Hospitals are not allowed to hand their raw data out
  - Security agencies cannot afford the risk
- Secure multiparty computation
  - A set of parties with private inputs wishes to jointly compute some function of their inputs
- Remaining problem: inference from the output of the algorithm using "background information"
  - Out-of-scope

Distributed Computing

Secure Multiparty Computation (SMC)

- **Goal:** to enable parties to carry out distributed computing tasks in a secure manner
- **Assumption:** a protocol execution may come under “attack” by an external entity, or even by a subset of the participating parties
  - To learn private information or cause the result of the computation to be incorrect
- **Key requirements:** privacy and correctness
- The setting of SMC can model almost every cryptographic problem

Examples

- Electronic voting, electronic auctions, electronic cash schemes, contract signing, anonymous transactions, private information retrieval, etc.
- In e-voting:
  - privacy requirement:
    - ensure that no parties learn anything about the individual votes of other parties
  - correctness requirement:
    - ensure that no coalition of parties has the ability to influence the outcome of the election
- In auctions:
  - privacy requirement:
    - ensure that only the winning bid is revealed
  - correctness requirement:
    - ensure that the highest bidder is indeed the winning party

Security in Multiparty Computation

- Set of requirements that should hold for any secure protocol:
  1) *Privacy*
     - No party should learn anything more than its prescribed output
  2) *Correctness*
     - Each party is guaranteed that the output that it receives is correct
  3) *Independence of Inputs*
     - Corrupted parties must choose their inputs independently of the honest parties' inputs
  4) *Guaranteed Output Delivery*
     - Corrupted parties should not be able to prevent honest parties from receiving their output
  5) *Fairness*
     - Corrupted parties should receive their outputs if and only if the honest parties also receive their outputs

Ideal World vs. Real World

- Just checking a set of requirements is not enough
- Need a definition that is general enough to capture all applications

- *Ideal World*: an external trusted (and incorruptible) party is willing to help the parties carry out their computation
  - Parties send their inputs to the trusted party
  - Trusted party computes the desired function and passes to each party its prescribed output
  - Only freedom given to the adversary is in choosing the corrupted parties' inputs
- *Real World*: no external party that can be trusted by all parties

Generalized Security Definition

- A real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if no adversary can do more harm in a real execution than in an execution that takes place in the ideal world
- The security of a protocol is established by comparing the outcome of a real protocol execution to the outcome of an ideal computation
  - A real protocol execution “emulates” the ideal world
- This formulation of security is called the *ideal/real simulation paradigm*
- Implies all 5 requirements in a general way

Adversarial Power (1)

- Key assumption for security definition (and proof) of an algorithm
- Adversary can be categorized based on its corruption strategy, allowed behavior, and computational power
- Corruption strategy:
  - Static corruption model
    - Honest parties remain honest and corrupted parties remain corrupted
  - Adaptive corruption model
    - Adversary has the capability of corrupting parties during the computation
  - Proactive model
    - Parties are corrupted only for a certain period of time

Adversarial Power (2)

- Allowed adversarial behavior
  - Semi-honest adversary
    - Corrupted parties correctly follow the protocol specification
    - “honest-but-curious” or “passive”
  - Malicious adversary
    - Corrupted parties can arbitrarily deviate from the protocol specification
- Complexity
  - Polynomial-time
    - Adversary is allowed to run in (probabilistic) polynomial-time
    - Any attack that cannot be carried out in polynomial-time is not a threat in real life (e.g., factoring large numbers)
    - Computational model for secure computation
  - Computationally unbounded
    - Information-theoretic model for secure computation

Feasibility of SMC

- Based on fraction of corrupted parties
- Let m denote the number of participating parties and let t denote a bound on the number of parties that may be corrupted
  - For $t < m/3$, SMC with fairness and guaranteed output delivery can be achieved for any function in a point-to-point network and without any setup assumptions
  - For $t < m/2$, SMC with fairness and guaranteed output delivery can be achieved for any function assuming that the parties have access to a broadcast channel
  - For $t \geq m/2$, SMC (without fairness or guaranteed output delivery) can be achieved assuming that the parties have access to a broadcast channel and that enhanced trapdoor permutations
    - Holds only in the computational setting

Definitions of Security

Preliminaries

- Assumptions:
  - Static corruptions and no honest majority
  - Polynomial-time adversaries
- Security parameter: n (length of the cryptographic key)
- A function $\mu(\cdot)$ is negligible in n if for every positive polynomial $p(\cdot)$ there exists an integer N such that for all $n > N$ it holds that $\mu(n) < 1/p(n)$
  - An event that happens with negligible probability can be dismissed

Definitions of Security

Computational Indistinguishability

- Let $X(n,a)$ and $Y(n,a)$ be random variables
- These two random variables are computationally indistinguishable if no algorithm running in polynomial-time can tell them apart (except with negligible probability)
- X and Y are computationally indistinguishable, denoted

$$X \overset{c}{\equiv} Y$$

if for every non-uniform polynomial-time distinguisher D there exists a function $\mu(\cdot)$ that is negligible in n, such that for *every* $a \in \{0,1\}^*$,

$$\left|\Pr\left[D\left(X(n,a)\right) = 1\right] - \Pr\left[D\left(Y(n,a)\right) = 1\right]\right| < \mu(n)$$

- Typically, the distributions X and Y will denote the output vectors of the parties in real and ideal executions,

Security in Semi-Honest Model

Two Party Computation

- **functionality** denoted as

$f: \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$

- The first party (with input *x*) wishes to obtain $f_1(x, y)$
- The second party (with input *y*) wishes to obtain $f_2(x, y)$

$$(x, y) \to (f_1(x, y), f_2(x, y))$$

Security in Semi-Honest Model

Highlevel Definition of Security

- A protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output only
- Formalized according to the simulation paradigm
  - A party's *view* in a protocol execution should be simulatable given only its input and output
- The parties learn nothing from the protocol execution itself, as desired

Security in Semi-Honest Model

Formal Definition of Security

- $f = (f_1, f_2)$: probabilistic polynomial-time functionality
- π: two-party protocol for computing f
- $\mathrm{view}_i^{\pi}(n, x, y)$: view of the i-th party during the execution of π
  - Includes contents of the party's internal random tape and messages it received
- $\mathrm{output}_i^{\pi}(n, x, y)$: output of the i-th party

➢ π securely computes f in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that for every $x, y \in \{0,1\}^*$ where $|x| = |y|$, we have

$$\{(S_1(1^n, x, f_1(x, y)), f(x, y))\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{\mathrm{view}_1^{\pi}(n, x, y), \mathrm{output}^{\pi}(n, x, y)\}_{n\in\mathbb{N}}$$

$$\{(S_2(1^n, y, f_2(x, y)), f(x, y))\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{\mathrm{view}_2^{\pi}(n, x, y), \mathrm{output}^{\pi}(n, x, y)\}_{n\in\mathbb{N}}$$

Security in Malicious Model

- Main differences: a malicious party may
  - refuse to participate in the protocol
  - substitute its local input (and instead use a different input)
  - abort the protocol prematurely
- Security definition is formalized according to the *ideal/real model paradigm*
- Execution in the real model: a real two-party protocol π is executed
  - No trusted third party

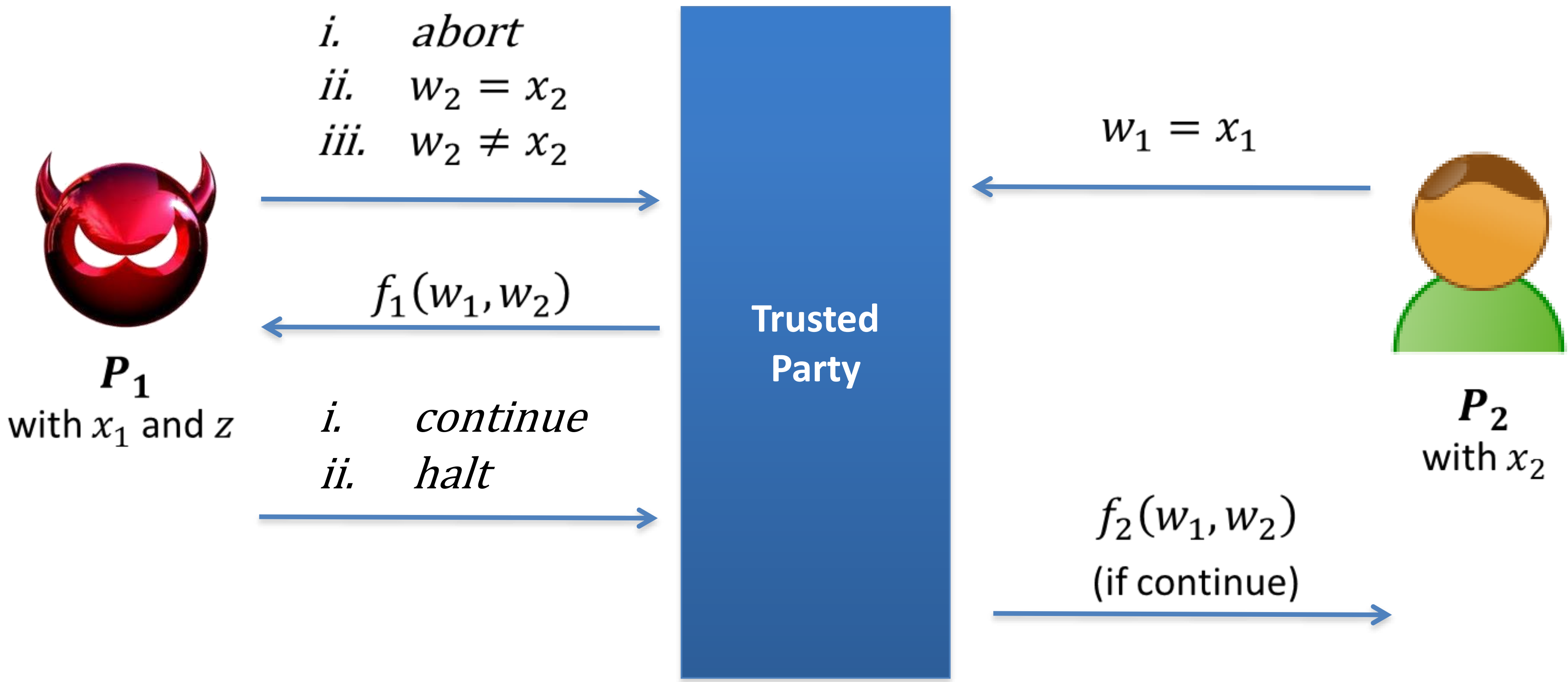
Security in Malicious Model

Ideal Execution

- Inputs
  - i-th party's input is denoted x_i
  - Adversary *A* receives an auxiliary input z
- Send inputs to the trusted party
  - The corrupted party may
    - abort by replacing the input x_i with a special abort message
    - send its input x_i
    - send some other input of the same length to the trusted party
  - Inputs sent to the trusted party: (w_1, w_2)
- Trusted party sends outputs to the adversary
  - Trusted party computes outputs and sends $f_i(w_1, w_2)$ to corrupted party P_i
- Adversary instructs trusted party to continue or halt
  - *A* sends either continue or abort to the trusted party
- Outputs
  - *A* outputs any arbitrary function of the initial input x_i, the auxiliary input z, and the output abort or $f_i(w_1, w_2)$

Security in Malicious Model

Ideal Execution



i. *abort*

ii. $w_2 = x_2$

iii. $w_2 \neq x_2$

$f_1(w_1, w_2)$

i. *continue*

ii. *halt*

P_1

with x_1 and z

Trusted Party

$w_1 = x_1$

P_2

with x_2

$f_2(w_1, w_2)$

(if continue)

z models side information of the adversary

Security in Malicious Model

Highlevel Definition of Security

- Assume a two-party functionality f on inputs (x_1, x_2), auxiliary input z to *A,* and security parameter n
- Let π be the two-party protocol for computing f
- Let I be the index of the corrupted party
- Output pairs of the honest party and the adversary *A* in ideal and real executions:
  - $\mathrm{IDEAL}_{f,A(z),I}(n, x_1, x_2)$
  - $\mathrm{REAL}_{\pi,A(z),I}(n, x_1, x_2)$

- ➢ A secure party protocol (in the real model) emulates the ideal model
  - Adversaries in the ideal model are able to simulate executions of the real-model protocol
  - Adversary's only possible attacks are to choose its input as it wishes and cause an early abort in the protocol

Security in Malicious Model

Formal Definition of Security

- Protocol π is said to securely compute f with abort in the presence of malicious adversaries if for every non-uniform probabilistic polynomial-time adversary *A* for the real model, there exists a non-uniform probabilistic expected polynomial-time adversary *S* for the ideal model, such that
  - For every I, every $x_1, x_2 \in \{0,1\}^*$ such that $|x_1| = |x_2|$, and every auxiliary input $\mathrm{z} \in \{0,1\}^*$:

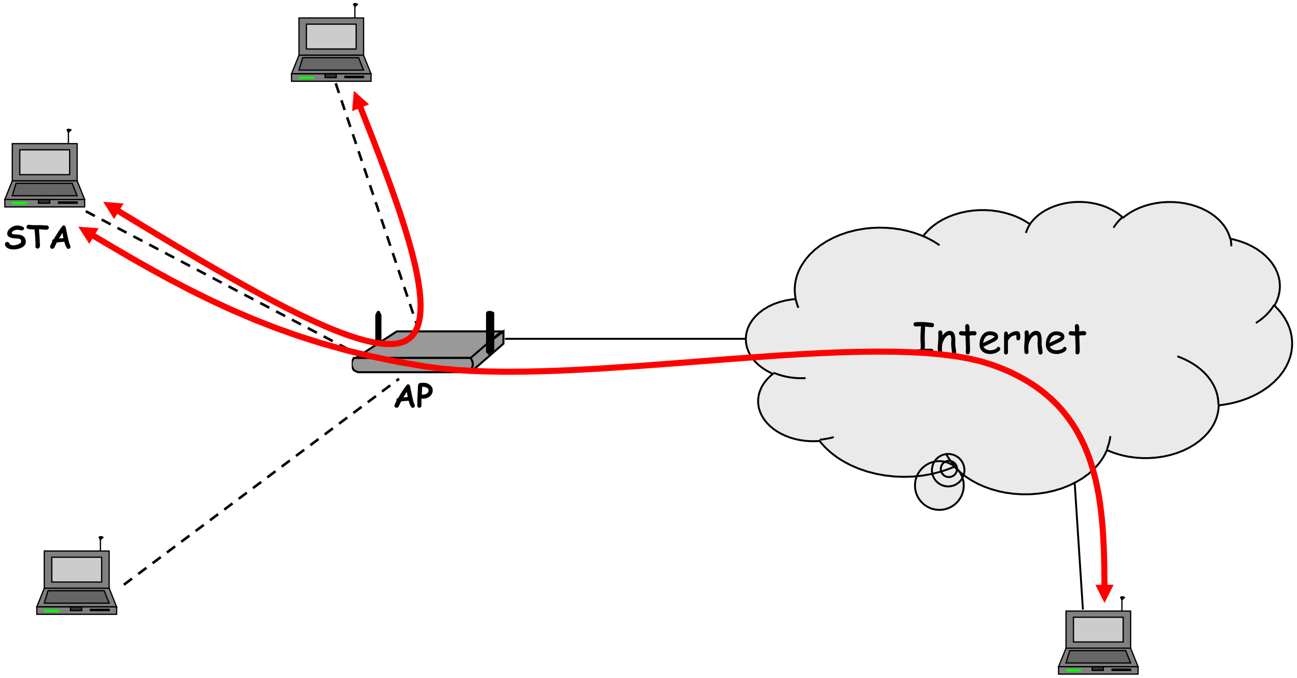
$$\{\mathrm{IDEAL}_{f,S(z),I}(n, x_1, x_2)\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{\mathrm{REAL}_{\pi,A(z),I}(n, x_1, x_2)\}_{n\in\mathbb{N}}$$

Security in Malicious Model

Modular Sequential Composition

- It is possible to design a protocol that uses an ideal functionality as a subroutine, then analyze the security of the protocol when a trusted party computes this functionality
  - First, construct a protocol for the functionality in question and prove its security
  - Next, prove the security of the larger protocol that uses the functionality as a subroutine in a model where the parties have access to a trusted party computing the functionality
- The composition theorem then states that when the “ideal calls” to the trusted party for the functionality are replaced by real executions of a secure protocol computing this functionality, the protocol remains secure

Example – Wired Equivalent Privacy (WEP)

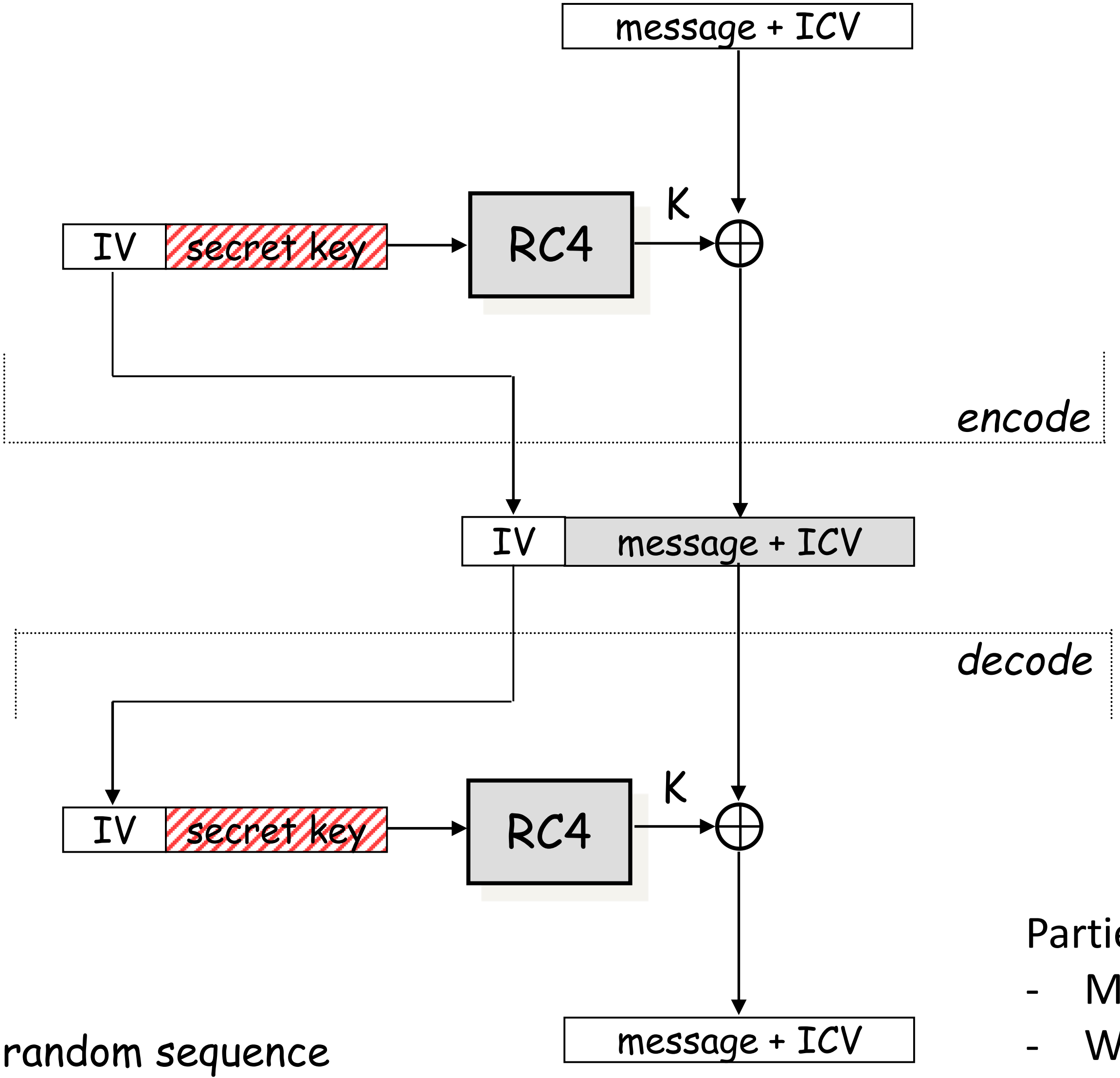


WEP – Message Confidentiality and Integrity

- WEP encryption is based on RC4 (a stream cipher developed in 1987 by Ron Rivest for RSA Data Security, Inc.)
  - operation:
    - for each message to be sent:
      - RC4 is initialized with the shared secret (between STA and AP)
      - RC4 produces a pseudo-random byte sequence (key stream)
      - this pseudo-random byte sequence is XORed to the message
    - reception is analogous

- WEP integrity protection is based on an encrypted CRC value
  - operation:
    - ICV (integrity check value) is computed and appended to the message
    - the message and the ICV are encrypted together

WEP – Message Confidentiality and Integrity



K: pseudo-random sequence

Parties:
- Message sender (honest)
- Wireless medium (malicious)

WEP Flaw – Integrity

- The attacker can manipulate messages despite the ICV mechanism and encryption
  - CRC is a linear function wrt to XOR:

$$\mathrm{CRC}(X \oplus Y) = \mathrm{CRC}(X) \oplus \mathrm{CRC}(Y)$$

  - attacker observes $(M \mid \mathrm{CRC}(M)) \oplus K$ where K is the RC4 output
  - for any ΔM, the attacker can compute $\mathrm{CRC}(\Delta M)$
  - hence, the attacker can compute:

$$\begin{aligned}
&((M \mid \mathrm{CRC}(M)) \oplus K) \oplus (\Delta M \mid \mathrm{CRC}(\Delta M)) = \\
&((M \oplus \Delta M) \mid (\mathrm{CRC}(M) \oplus \mathrm{CRC}(\Delta M))) \oplus K = \\
&((M \oplus \Delta M) \mid \mathrm{CRC}(M \oplus \Delta M)) \oplus K
\end{aligned}$$

WEP - Conclusion

- A malicious adversary can temper the message content
  - And hence, the output of the honest party
- “Correctness” property doe not hold anymore

- One can combine otherwise strong building blocks in a wrong way and obtain an insecure system at the end
- Example

– encrypting a message digest to obtain an ICV is a good principle

– but it doesn’t work if the message digest function is linear wrt to the encryption function

Discussion
Semi-Honest vs. Malicious Model

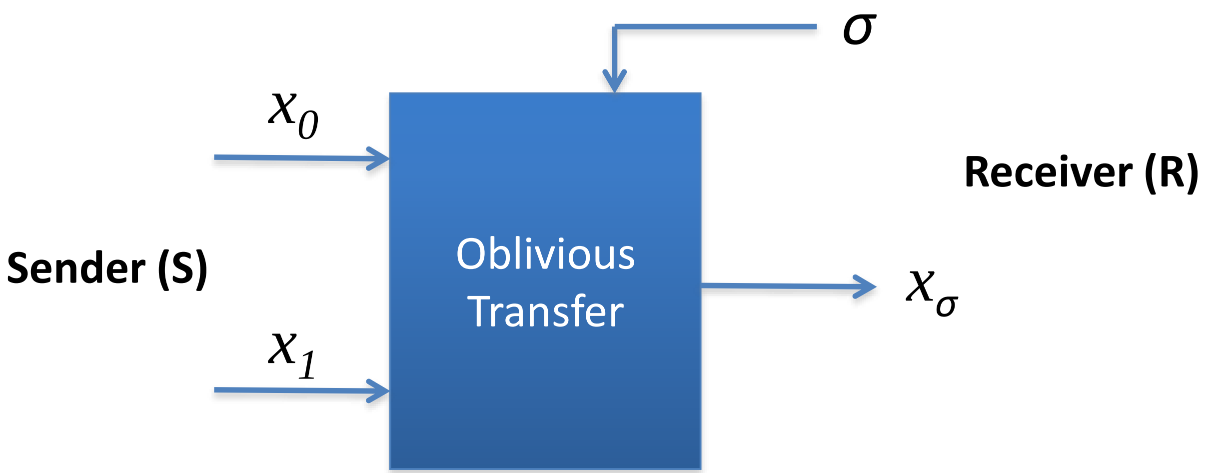
- Semi-honest: each party has to trust all other parties for not actively cheating
  - Hospitals who wish to carry out joint research on their confidential patient records.
  - This assumption is often too strong
- Malicious: leads to very heavy solutions
  - Performance issues
- Two possible avenues:
  - Reduce the level of guarantees (e.g., guaranteeing privacy only)
  - Intermediate adversary (e.g., covert adversary)

Security in the Presence of *Covert Adversaries*

- Covert adversary: willing to actively cheat, but only if they are not caught
  - It lies between the semi-honest and the malicious adversary
- Definition of security is based on the classical ideal/real simulation paradigm
- Additional ingredient: deterrence factor **ε**
- For a value $0 < \boldsymbol{\varepsilon} \leq 1$, the definition guarantees that any attempt to "cheat" by an adversary is detected by the honest parties with probability at least **ε**

Guaranteeing Privacy Only

- Definition of security that follows the ideal/real simulation paradigm provides strong security guarantees
  - Guarantees privacy, correctness, independence of inputs, and so on.
- In some settings, it may be sufficient to guarantee privacy only
- Toy example: two-message oblivious transfer



Two-Message Oblivious Transfer

- $view_S^n\big(S(a), R(b)\big)$: the view of *S* in an execution where it has input *a* and *R* has input *b*
- $S_n(a; q)$: the distribution over the message sent by *S* upon input *a* and message received *q*
  - Defines *R*'s view in the execution when the protocol has two messages only and the first message *q* is sent by *R*

➢ A two-message two-party probabilistic polynomial-time protocol (*S;R*) is said to be a *private oblivious transfer* if the following holds:

Two-Message Oblivious Transfer

Guaranteeing Privacy

- *Correctness:* If *S* and *R* follow the protocol, then the output of *R* is x_σ

- *Privacy for R:* For every non-uniform probabilistic polynomial-time S^* and every auxiliary input $z \in \{0,1\}^*$, it holds that

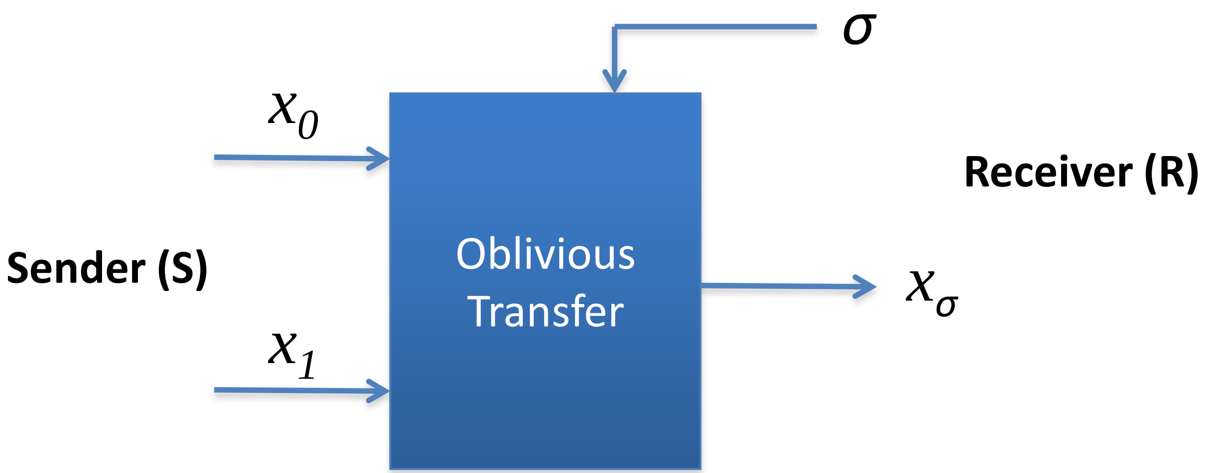
$$\{view_S^n(S^*(z), R(0))\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{view_S^n(S^*(z), R(1))\}_{n\in\mathbb{N}}$$

- *Privacy for S:* For every non-uniform deterministic polynomial-time receiver R^*, every auxiliary input $z \in \{0,1\}^*$, and every triple of inputs $x_0, x_1, x \in \{0,1\}^n$ one of the following should hold:

$$\{S_n((x_0, x_1); R^*(z))\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{S_n((x_0, x); R^*(z))\}_{n\in\mathbb{N}}$$
$$\{S_n((x_0, x_1); R^*(z))\}_{n\in\mathbb{N}} \overset{c}{\equiv} \{S_n((x, x_1); R^*(z))\}_{n\in\mathbb{N}}$$

SMC - Basic Building Blocks
Oblivious Transfer

- It was shown (by Kilian in 1988) that by using an implementation of oblivious transfer, and no other cryptographic primitive, it is possible to construct any secure computation protocol
- *1-out-of-2 oblivious transfer*: $((x_0, x_1), \sigma) \rightarrow (\lambda, x_\sigma)$



σ

x_0

Receiver (R)

Sender (S)

Oblivious Transfer

x_σ

x_1

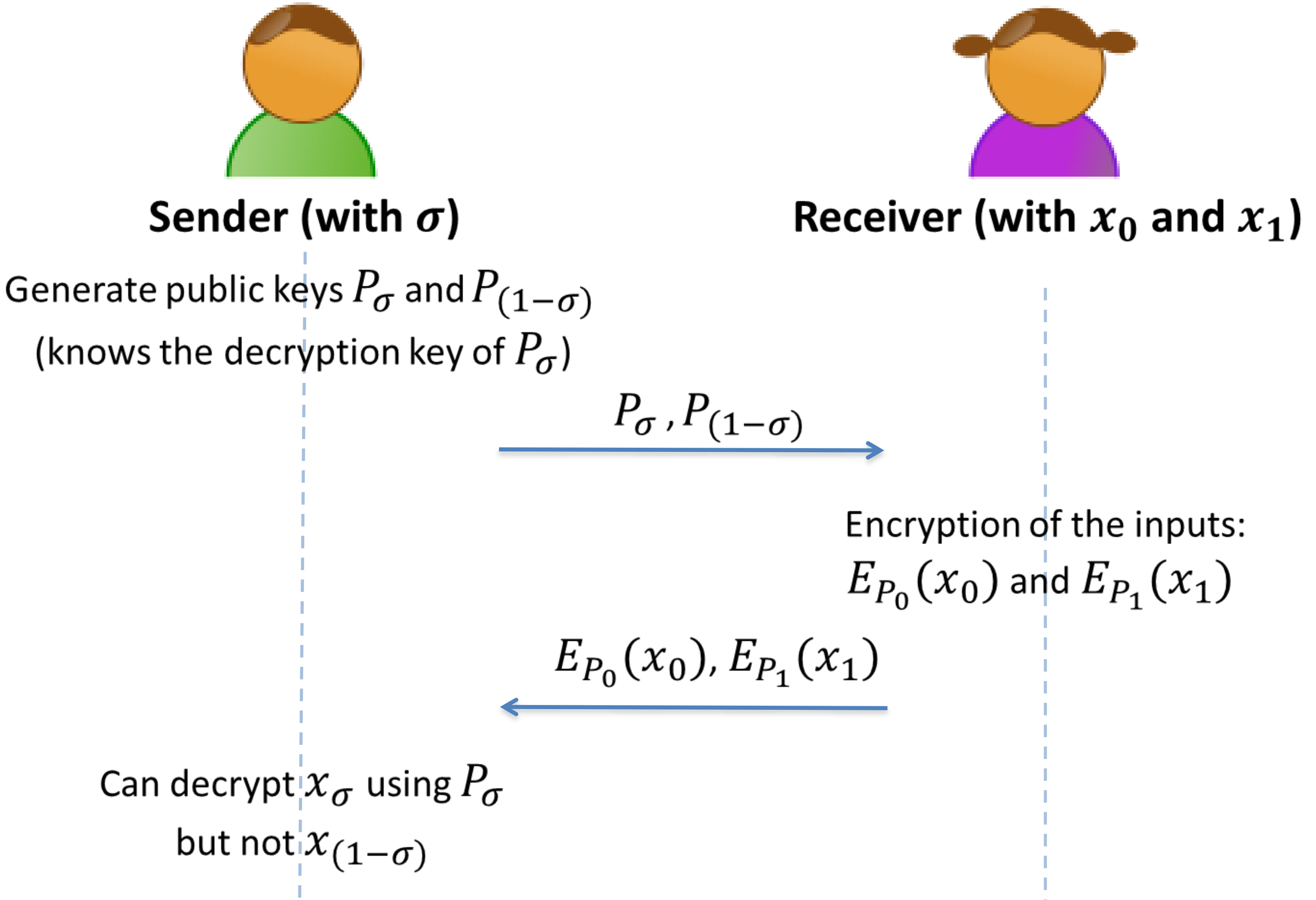
Oblivious Transfer - Example

- Receiver generates two random public keys, a key P_σ whose decryption key it knows, and a key $P_{(1-\sigma)}$ whose decryption key it does not know
- Receiver sends these two keys to the sender
- Sender encrypts x_0 with the key P_0 and encrypts x_1 with the key P_1
- Sender sends the two results to the receiver
- The receiver can then decrypt x_σ but not $x_{(1-\sigma)}$
  - If $\sigma = 0$, receiver knows the decryption key for P_0 only, and hence can only recover x_0, but not x_1

➢ Sender does not learn anything about σ, since its view in the protocol can be easily simulated:
  - The only message it receives includes two random public keys P_0 and P_1

➢ As for the sender's privacy, if the receiver follows the protocol, it only knows one private key and can therefore only decrypt one of the inputs
  - Assuming the encryption scheme to be semantically secure

Oblivious Transfer - Example



Sender (with σ)

Receiver (with x_0 and x_1)

Generate public keys P_σ and $P_{(1-\sigma)}$

(knows the decryption key of P_σ)

$P_\sigma \,, P_{(1-\sigma)}$

Encryption of the inputs:

$E_{P_0}(x_0)$ and $E_{P_1}(x_1)$

$E_{P_0}(x_0), E_{P_1}(x_1)$

Can decrypt x_σ using P_σ

but not $x_{(1-\sigma)}$

Oblivious Transfer - Discussion

- There are simple and efficient protocols for oblivious transfer which are secure only against semi-honest adversaries
- It is more challenging to construct oblivious transfer protocols which are secure against malicious adversaries
  - Can be achieved using zero-knowledge proofs that are used by the receiver

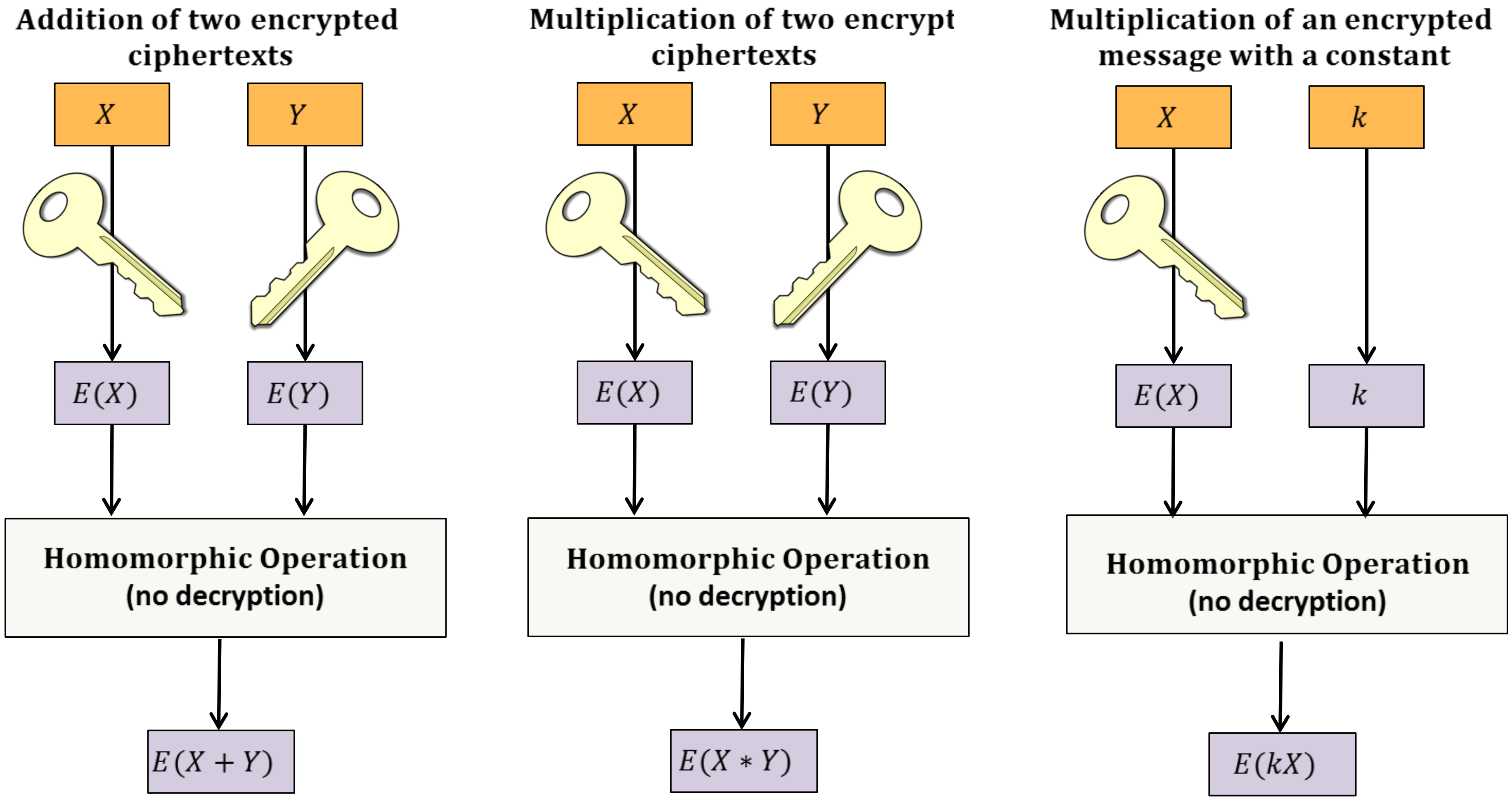
Reminder

- Secure multi-party computation
- Adversary models
    - Honest-but-curious adversary
    - Malicious adversary
- Security analysis of protocols
- Oblivious transfer

SMC - Basic Building Blocks

Homomorphic Encryption

- Allows specific types of computations to be carried out on ciphertext



Addition of two encrypted ciphertexts

X Y

$E(X)$ $E(Y)$

Homomorphic Operation
(no decryption)

$E(X+Y)$

Multiplication of two encrypt ciphertexts

X Y

$E(X)$ $E(Y)$

Homomorphic Operation
(no decryption)

$E(X*Y)$

Multiplication of an encrypted message with a constant

X k

$E(X)$ k

Homomorphic Operation
(no decryption)

$E(kX)$

Benaloh, Paillier

Unpadded RSA, ElGamal

Homomorphic Encryption

- Popular instantiations:
  - Paillier scheme
    - Encryption of a plaintext from $[1, N]$, where N is an RSA modulus, requires two exponentiations modulo N^2
    - Decryption requires a single exponentiation
    - Supports addition in ciphertext domain
  - Damgard-Jurik
    - Generalization of Paillier (to encrypt longer messages)
    - Encrypts messages from the range $[1, N^s]$

Problem: Ciphertext expansion and computational overhead

SMC - Basic Building Blocks
Oblivious Polynomial Evaluation (OPE)

- The input of the sender is a polynomial Q of degree k over some finite field $\mathcal{F}$

$$Q(z) = \sum_{i=0}^{k} a_i z^i$$

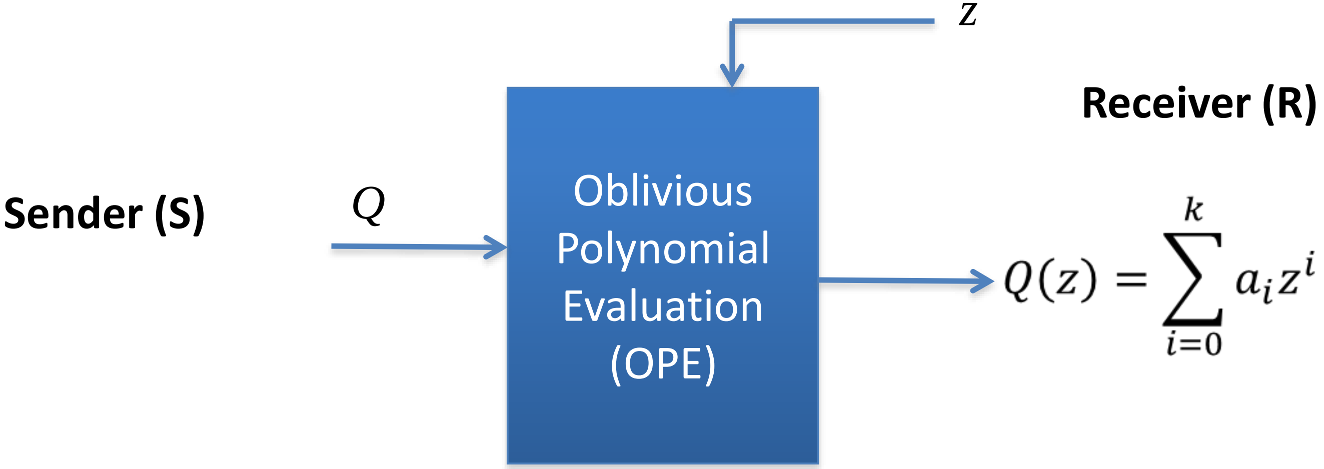
- The input of the receiver is an element $z \in \mathcal{F}$
- OPE implements the functionality $(Q, z) \rightarrow (\lambda, Q(z))$

SMC - Basic Building Blocks
Oblivious Polynomial Evaluation (OPE)

- The input of the sender is a polynomial Q of degree k over some finite field $\mathcal{F}$

$$Q(z) = \sum_{i=0}^{k} a_i z^i$$

- The input of the receiver is an element $z \in \mathcal{F}$
- OPE implements the functionality $(Q, z) \rightarrow (\lambda, Q(z))$



z

Receiver (R)

Sender (S)

Q

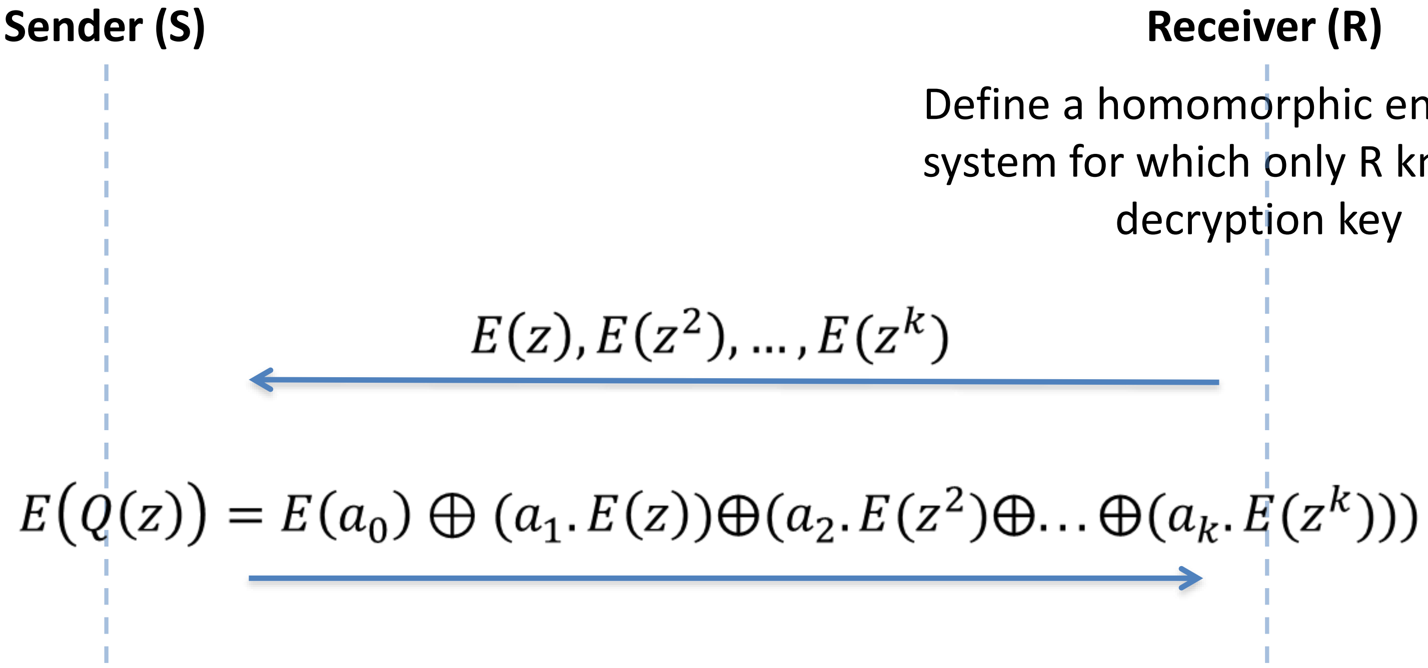
Oblivious Polynomial Evaluation (OPE)

$Q(z) = \sum_{i=0}^{k} a_i z^i$

Oblivious Polynomial Evaluation

Implementation

- Based on homomorphic encryption
- Secure in the semi-honest model and achieves privacy (but not simulatable security) in the face of a malicious adversary
    - Why not?



Sender (S)

Receiver (R)

Define a homomorphic encryption system for which only R knows the decryption key

$$E(z), E(z^2), \ldots, E(z^k)$$

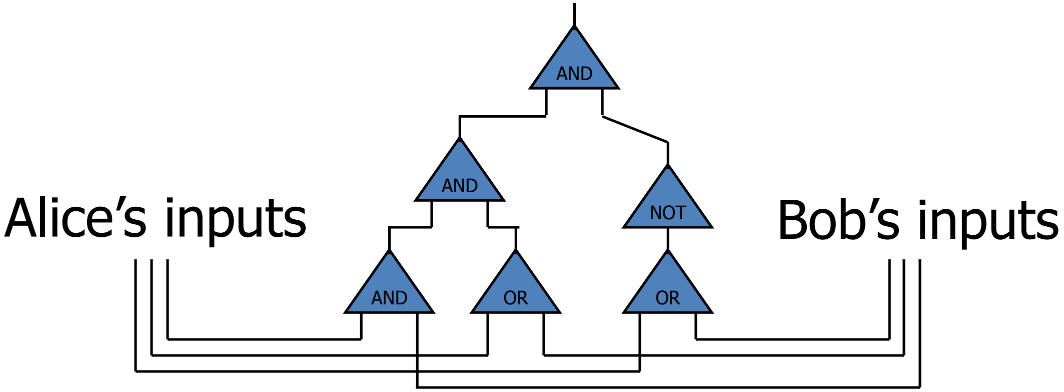
$$E\big(Q(z)\big) = E(a_0) \oplus (a_1.E(z))\oplus(a_2.E(z^2)\oplus\ldots\oplus(a_k.E(z^k)))$$

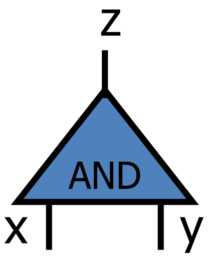
SMC – Generic Constructions
Yao's Garbled Circuit

- Implement secure computation for any probabilistic polynomial-time function
- Secure computation in the two-party case can be efficiently implemented by Yao's garbled circuit
- Proved to be secure against both semi-honest and malicious adversaries
- Next 10 slides from the lecture notes of Vitaly Shmatikov (UT Austin)

Yao’s Protocol

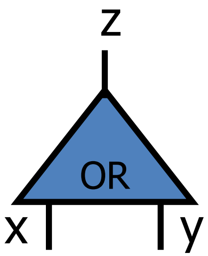
- Compute any function securely
  - … in the semi-honest model
- First, convert the function into a boolean circuit





Truth table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

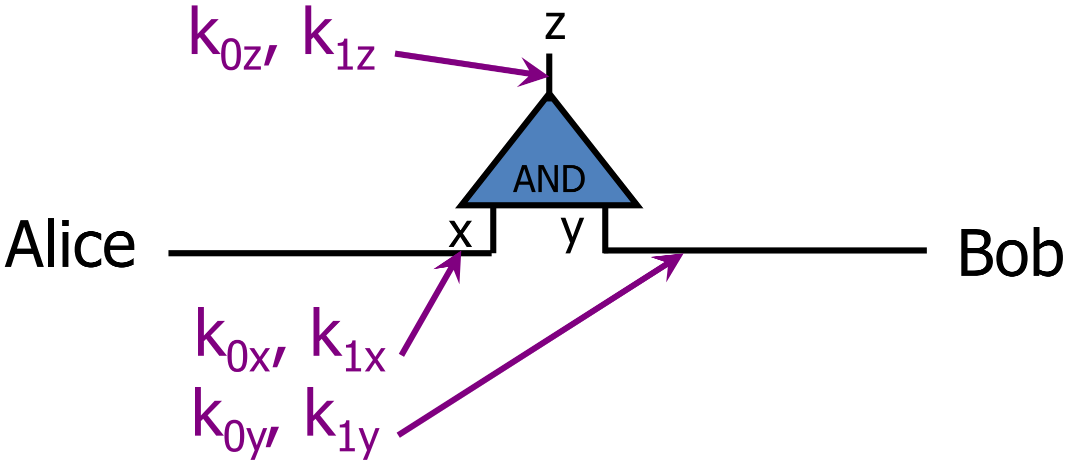


Truth table:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

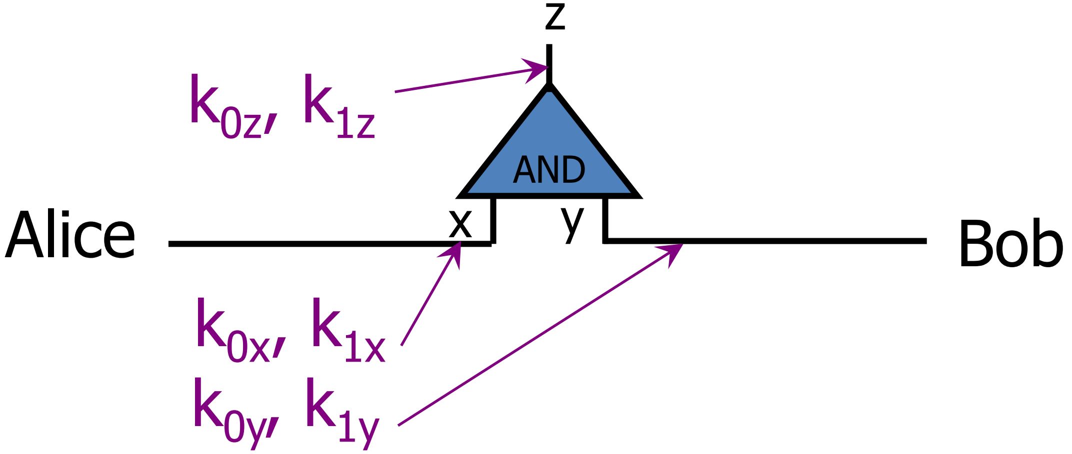
1: Pick Random Keys For Each Wire

- Next, evaluate <u>one gate</u> securely
  - Later, generalize to the entire circuit
- Alice picks two random keys for each wire
  - One key corresponds to “0”, the other to “1”
  - 6 keys in total for a gate with 2 input wires



2: Encrypt Truth Table

- Alice encrypts each row of the truth table by encrypting the output-wire key with the corresponding pair of input-wire keys



Original truth table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

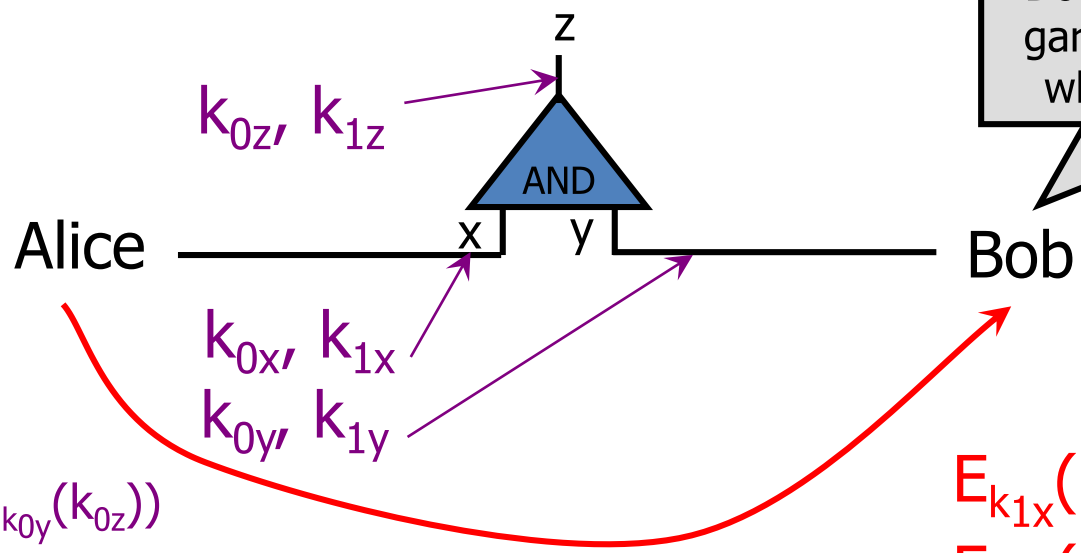
Encrypted truth table:

$$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$$
$$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$$
$$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$$
$$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$$

3: Send Garbled Truth Table

- Alice randomly permutes (“garbles”) encrypted truth table and sends it to Bob

Does not know which row of garbled table corresponds to which row of original table



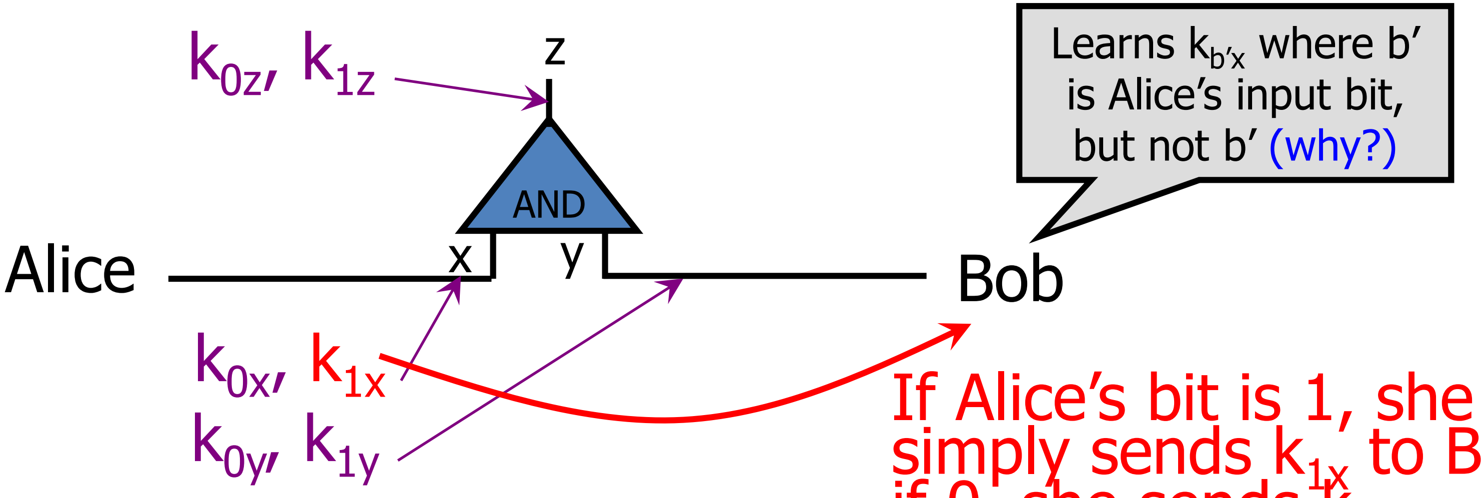
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

Garbled truth table:

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

4: Send Keys For Alice's Inputs

- Alice sends the key corresponding to her input bit
  - Keys are random, so Bob does not learn what this bit is



k_{0z}, k_{1z}

z

AND

x

y

Alice

Bob

k_{0x}, k_{1x}

k_{0y}, k_{1y}

Learns $k_{b'x}$ where b' is Alice's input bit, but not b' (why?)

If Alice's bit is 1, she simply sends k_{1x} to Bob; if 0, she sends k_{0x}

Garbled truth table:

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$

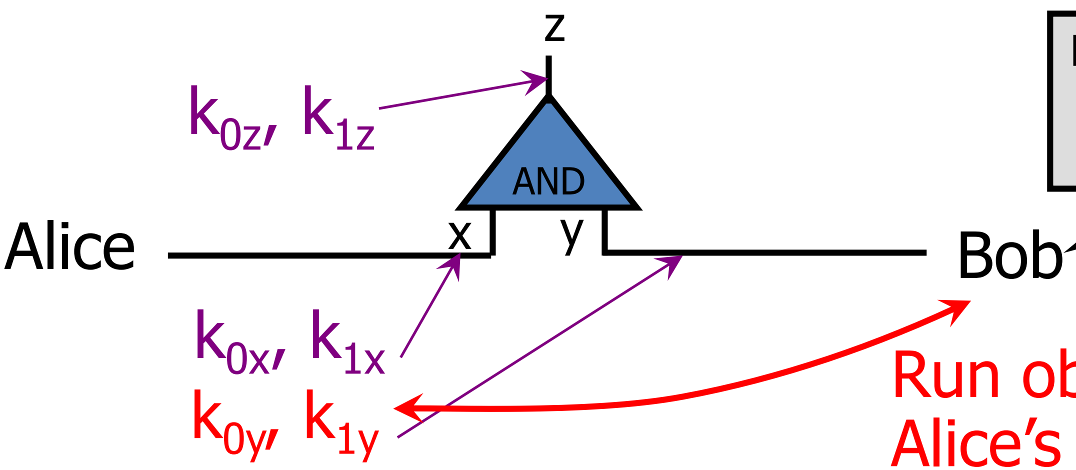
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$

$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

5: Use OT on Keys for Bob's Input

- Alice and Bob run oblivious transfer protocol
  - Alice's input is the two keys corresponding to Bob's wire
  - Bob's input into OT is simply his 1-bit input on that wire



Knows $k_{b'x}$ where b' is Alice's input bit and k_{by} where b is his own input bit

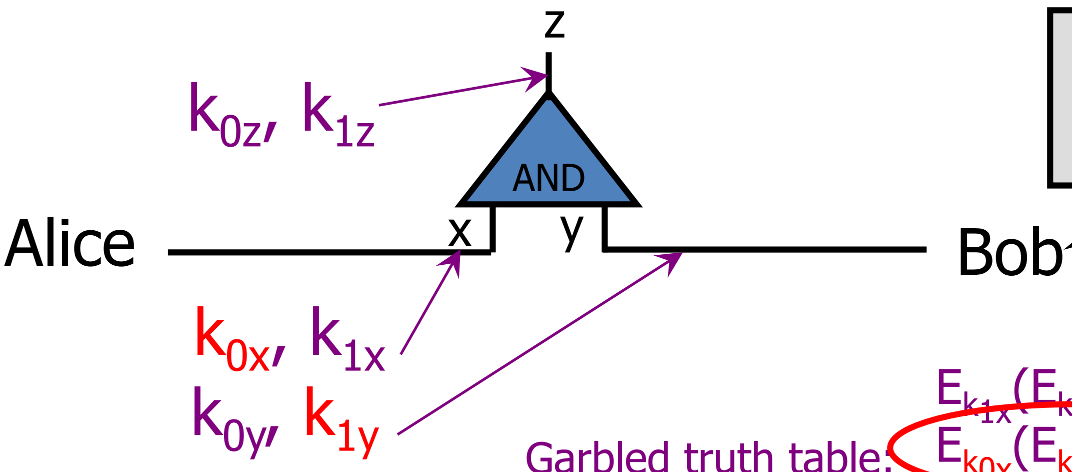
Run oblivious transfer
Alice's input: k_{0y}, k_{1y}
Bob's input: his bit b
Bob learns k_{by}

What does Alice learn?

Garbled truth table:
$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

6: Evaluate Garbled Gate

- Using the two keys that he learned, Bob decrypts exactly one of the output-wire keys
  - Bob does not learn if this key corresponds to 0 or 1
    - Why is this important?



Knows $k_{b'x}$ where b′ is Alice's input bit and k_{by} where b is his own input bit

Suppose b′=0, b=1

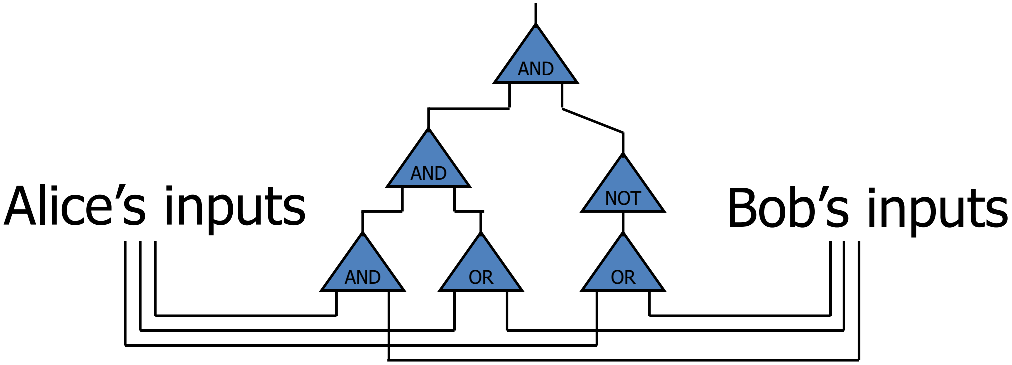
Garbled truth table:

$E_{k_{1x}}(E_{k_{0y}}(k_{0z}))$
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$
$E_{k_{1x}}(E_{k_{1y}}(k_{1z}))$
$E_{k_{0x}}(E_{k_{0y}}(k_{0z}))$

This is the only row Bob can decrypt. He learns K_{0z}

7: Evaluate Entire Circuit

- In this way, Bob evaluates entire garbled circuit
  - For each wire in the circuit, Bob learns only one key
  - It corresponds to 0 or 1 (Bob does not know which)
    - Therefore, Bob does not learn intermediate values (why?)



- Bob tells Alice the key for the final output wire and she tells him if it corresponds to 0 or 1
  - Bob does <u>not</u> tell her intermediate wire keys (why?)

Brief Discussion of Yao's Protocol

- Function must be converted into a circuit
  - For many functions, circuit will be huge
  - AES has around 30,000 gates
- If m gates in the circuit and n inputs, then need 4m encryptions and n oblivious transfers
  - Oblivious transfers for all inputs can be done in parallel
- Yao's construction gives a constant-round protocol for secure computation of any function in the semi-honest model
  - Two-round oblivious transfer protocol
  - Number of rounds does not depend on the number of inputs or the size of the circuit!

Garbled Circuits – Malicious Model

- Very difficult problem
- Several efficient protocols developed since 2004 (it should be possible to run AES under 1 second)
- Approach considered here: Yao's garbled circuit
- Problem: because the adversary is malicious, it could (if it is Party 1) deliver a deliberately false circuit
- Examples:
  - Replace some AND gates by XOR gates, or vice-versa
  - Organize the circuit in such a way that it leaks the input of Party 2

Possible Solution: Cut-and-Choose Protocol (1/2)

- Principle:
  - P1 constructs a high number of circuits and provides them all to P2
  - Then P2 chooses (say) half of them and asks P1 to “open” them (by providing all the keys)
  - If P1 had included one or several bogus circuits, P2 will detect it with high likelihood
- Problems with this solution
  - How to make sure that parties make use of the same inputs on all of them?
  - The circuits may be correct, but the garbled keys may be bogus
  - A sophisticated malicious P1 could construct a circuit with 2 sets of keys:
    - 1 opening to the correct circuit
    - 1 to a different circuit

Possible Solution: Cut-and Choose Protocol (2/2)

- Computation:
  - N: total number of circuits
  - Success of the adversary: if (at least) N/4 circuits are incorrect and none of them was chosen by P2 to be checked
  - If P2 selects circuits randomly, this happens with probability $2^{-N/4}$
  - For security of 2^{-40} (around 1 chance in 1 trillion), one needs N=160 circuits
  - Actually checking 60% of the circuits gives a better result, and in this case 125 circuits suffice
    - Note however that checking a circuit takes more time than “executing” the circuit (computation of all 4 possible values in the former case, and of a single value in the latter)

Yao's Protocol – Multiparty Case

- There are also constructions which enable a set of *m* > 2 parties to compute any function of their inputs without revealing any other information
- Have some drawbacks compared to the two-party protocol:
  - Require public-key operations for every gate of the circuit
  - Number of rounds is linear in the size of the circuit
  - Require communication between every pair of the *m* participating parties
  - Require the use of a broadcast channel

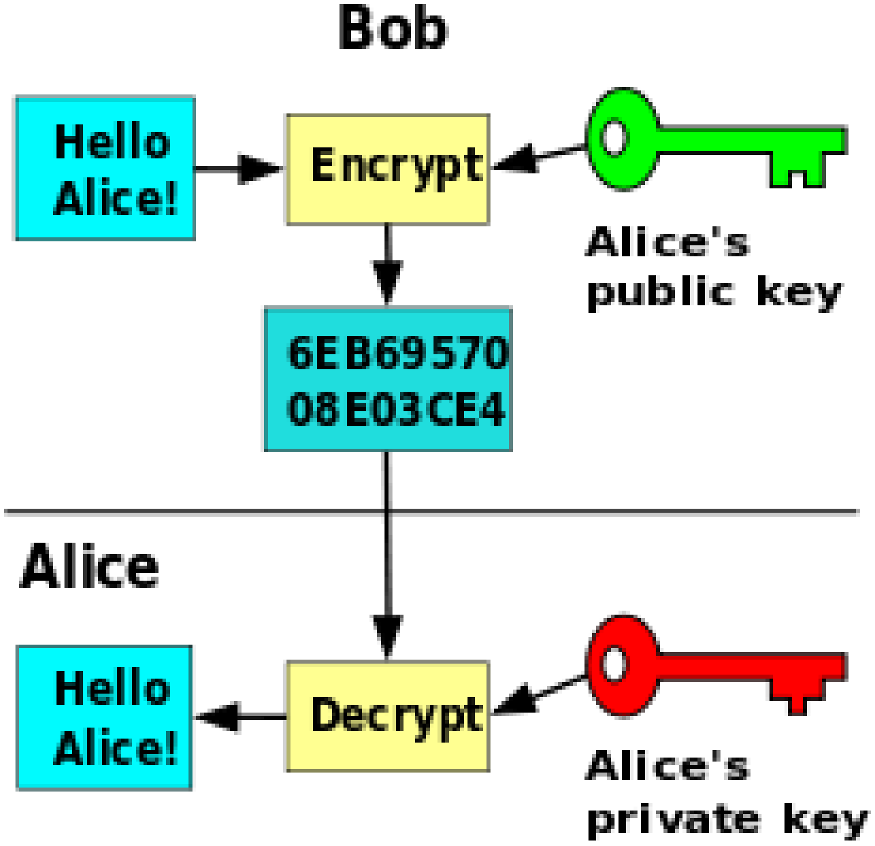
Other Crypto Tools for Privacy Protection

- Anonymous communication
  - TOR
- Anonymous credentials
- Blind signatures
- Searchable encryption
- Deterministic encryption
  - Order-preserving encryption
- Computing on encrypted data
  - Functional encryption
- Oblivious RAM
- Private information retrieval
- Zero-knowledge proofs
- Secret-sharing
- Etc.

In Class Exercise

- Goal: Design a system in which
  - Individuals have sensitive personal data – set of attributes (medical records)
  - Data is somehow encrypted by the individual and stored at the cloud
  - A third-party wants to do computation on the data (medical center)
  - The third party also has secret inputs and does not want to share those with the cloud
  - Ideally, user is not involved

Paillier Cryosystem



- The public key: $(n, g, h = g^x)$
- Secret key: $x \in [1, n^2/2]$
- Strong secret:

  Factorization of $n = zy$
  (z , y are safe primes)

Paillier Cryptosystem Encryption

- To encrypt a message $m \in Z\_n$
  - Select a random $r \in [1, n/4]$
  - Generate the ciphertext pair (C1,C2) such that
  - $C1 = g^r \bmod n^2$
  - $C2 = h^r(1 + mn) \bmod n^2$

  - $[m]=(C1,C2)$

The public key: $(n, g, h = g^x)$
Secret key: $x \in [1, n^2/2]$

Paillier Cryptosystem Decryption

- The message m can be recovered from $[m]=(C1,C2)$ as follows:
  - $m = \text{Delta}(C2 / C1^{x})$
  - $\text{Delta}(u) = [(u-1) \bmod n^{2}]/n$
    - For all $u \in \{u < n^{2} \mid u = 1 \bmod n\}$

The public key: $(n, g, h = g^{x})$
Secret key: $x \in [1, n^{2}/2]$

Paillier Cryptosystem Threshold Encryption

- Assume we randomly split the secret key in two shares x1 and x2 ,
  - x = x1 + x2
- The Paillier cryptosystem enables an encrypted message (C1,C2) to be partially decrypted to a ciphertext pair (C˜1,C˜2) using x1 as
  - C˜1 = C1
  - C˜2 = C2 /C1^(x1) mod n^2
- Then, (C˜1,C˜2) can be decrypted using x2

The public key: (n, g, h = g^x)
Secret key: x ∈ [1, n^2/2]

Homomorphism

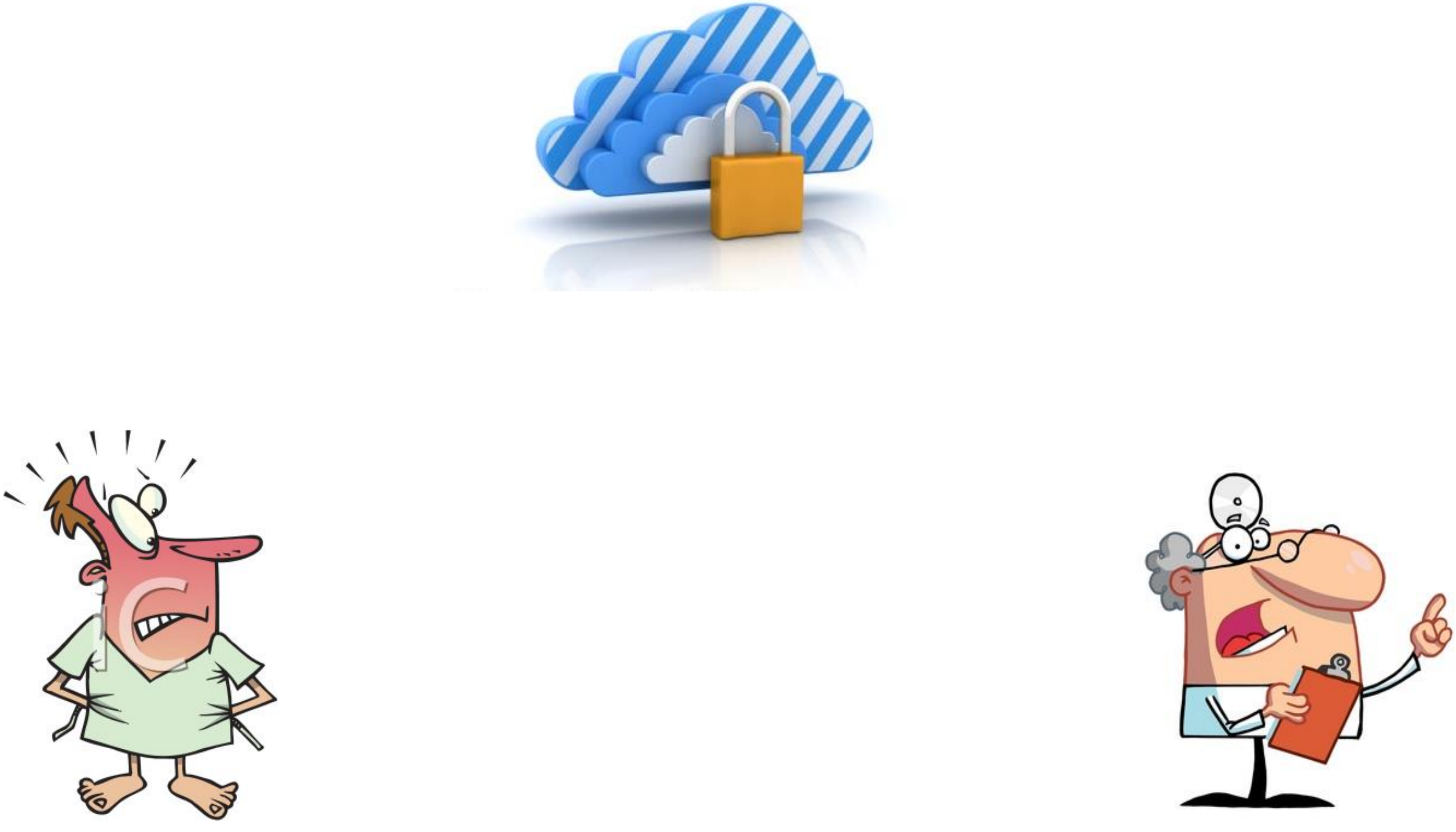
- The product of two ciphertexts is equal to the encryption of the sum of their corresponding plaintexts

- A ciphertext raised to a constant number is equal to the encryption of the product of the corresponding plaintext and the constant

Tasks

- Decide on the system model and parties involved
- Decide on the threat model for all parties involved
- Design the system
  - Initialization: Key generation, key management, encryption
  - Application: SMC
- Comment on the functions that can be supported
- Comment on the security/privacy of the system
- Comment on the performance
- Comment on the user-friendliness

System Model



Threat Model

- Semi-honest adversary vs. Malicious adversary

- Polynomial-time adversary vs. computationally unbounded adversary

- Collusion

Requirements

- Types of supported queries:
  - Weighted Average
  - Multiplication of ciphertexts
  - Division
  - Comparison/Classification
- Access Control
- Access Patterns

Design

- Initialization

- Application(s)