Chapter 5: More Approaches for Private Computation

Lecture PETs4DS:
Privacy Enhancing Technologies for Data Science

Parts of this slide set (slides 9 – 11, 28, 38 - 40) are based on slides from Lukas Prediger, RWTH Aachen University.

Parts of this slide set (slides 18 – 25, 29) are based on slides from Vitaly Shmatikov, Cornell University.

Dr. Benjamin Heitmann and Prof. Dr. Stefan Decker
Informatik 5
Lehrstuhl Prof. Decker

RWTH AACHEN UNIVERSITY

Overview of Chapter

- Review of Chapter 4
- Some more background on computational circuits
- Yao's Protocol for Garbled Circuits
- Oblivious Transfer
- Homomorphic Encryption
- Secure Multiplication Protocol using the Paillier Cryptosystem
  - Requires splitting secrets between two non-colluding cloud providers
- Performance Limits of Private Computation

Review of the Previous Chapter (Chapter 4)

Overview of Approaches for Private Computation

- **Cloud computing** provides shared resources for computation
- Different cloud deployment scenarios have different **confidentiality and integrity requirements**
- **Adversaries** can be honest but curious or malicious.
- Three promising **approaches to address threats** in cloud computing are:
  - Homomorphic encryption (HE)
  - Verifiable computation (VC)
  - Secure Multi-Party Compuation (SMPC)

- Each approach addresses different **requirements**, and provides different **guarantees**
- All approaches incur significant **performance overheads**
- SMPC is the most **promising** approach with the least overhead

Comparison of Approaches for Private Computation

Approach	Adversary Type	Confidentiality	Integrity	Requires Interaction
Homomorphic Encryption (HE)	Honest-but-curious (HBC)	YES	NO	NO
Verifiable Computation (VC)	Malicious	NO	YES	NO
Secure Multi-Party Computation (SMPC)	HBC or Malicious	YES	YES	YES

Clarification on adversary types against which SMPC is secure:
The two SMPC protocols from the lecture are **only** secure against HBC adversaries,
but there are other established SMPC protocols which are also secure against malicious adversaries.

Secure Multi-Party Computation (SMPC): Pre-conditions and Assumptions

- The parties or players are called $P_1, \dots, P_n$
- Each player P_i holds secret input x_i
- All players agree on a function f that takes n inputs

- Goal: compute $y = f(x_1, \dots, x_n)$ while satisfying the following two conditions:
  1. **Correctness:** the correct value of y is computed
  2. **Privacy:** y is the only new information that is released.
- **Computing f securely** means achieving correctness and privacy at the same time

- Other assumptions (for now):
  - All players follow the given protocol.
  - Any pair of players can communicate securely.

Comparison of SMPC Protocols as Presented in This Lecture

Two simple but different SMPC Protocols were discussed

1. **SMPC Protocol for exactly three parties**
   - Secrets are shared using addition and subtraction in a finite field
   - We looked at stand-alone addition and multiplication
   - Only three parties can use this protocol, no more and no less.
   - Guards against exactly one HBC adversary.

2. **SMPC Protocol with passive security (CEPS) for three or more parties**
   - Secrets are shared using polynomials and Lagrange Interpolation
   - We looked at evaluation of circuits with addition, multiplication and skalar multiplication
   - More than three parties are possible.
   - Guards against up to $t < n/2$ HBC adversaries.

Some Background on Arithmetic Circuits

Circuits

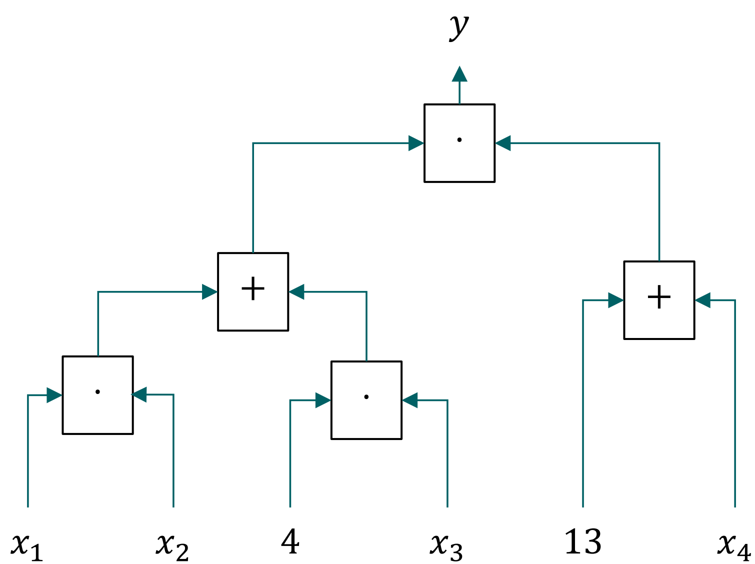
- Functions represented as arithmetic circuits
  - directed acyclic graphs where nodes are operations (= gates)
- Every computable function can be represented by a circuit family
  - not necessarily by a single circuit
- Limitations of a single circuit
  - no (direct) conditional branching or looping
  - no side-effects
  - limited size of inputs and output
- Every function evaluation can be represented by a circuit from the corresponding family

Important Properties of Circuits

- Complexity
  - the number of gates contained
- Depth
  - the depth of the DAG representing the circuit
- Multiplicative Complexity
  - the number of multiplication gates
- Multiplicative Depth
  - the number of layers containing multiplication gates
  - i.e., longest sequence of multiplications that depend on other multiplications (directly or indirectly)

Example of a Circuit

$$y = (x_1 \cdot x_2 + 4 \cdot x_3) \cdot (13 + x_4)$$



Complexity = 5

Depth = 3

Multiplicative Complexity = 3

Multiplicative Depth = 2

Private Computation for Exactly Two Parties using Yao’s Garbled Circuits

Motivation

- Short-comings of **SMPC**:
  - high complexity overhead
  - **only suitable for three parties or more**

- However, the **evaluation** of some functions also makes sense for **two parties**

- Other nice **properties** of Yao's Garbled Circuits:
  - Simple protocol
  - Requires only **two well established cryptographic primitives**
    - Symmetric encryption
    - Oblivious transfer

  - Efficient execution: **constant-round protocol**
    The number of protocol steps is not dependent on number of inputs or size of circuits

Yao's Millionaire Problem

- Two millionaires want to know who is richer without sharing the details of their wealth
- Example of a function where knowing the output does not allow one party to reconstruct the input of the other party

$$f(x,y) = \begin{cases} \text{Alice} & \text{if } x > y \\ \text{Bob} & \text{if } x < y \\ \text{same} & \text{if } x = y \end{cases}$$

- Yao's protocol allows solving this problem without any party learning anything new, except the value of f(x,y). This is independent of the number of times the protocol is executed.

Goals of Yao's Protocol

- Compute any function securely
- Guard against honest-but-curious adversaries

slide 15

Cryptographic Primitives Required for Yao's Garbled Circuits

1. Symmetric encryption
   – Also called private key encryption

2. Oblivious transfer
   – Will be explained after Yao's protocol

Symmetric Encryption

- A pair of functions, *E* and *D,* such that:

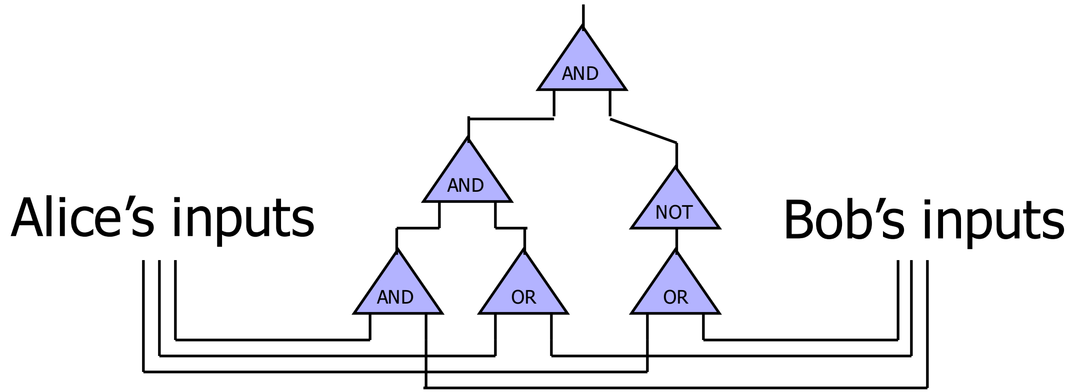
- $E_k(m) = c$ is the encryption of message m with key k
- $D_k(c) = m$ is the decryption of ciphertext c with key k
- Decrypting with the same secret key gives the original message:
$$D_k\left(E_k(m)\right) = m$$

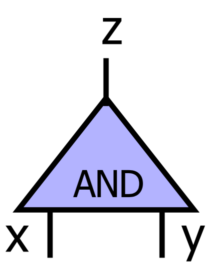
- Given a ciphertext c it is hard to find a key k and message m such that
$$E_k(m) = c$$
  - „hard“ means, it is not solvable in polynomial time.

- Assumption: Decrypting $E_k(m)$ with a different key k’ results in an error.
  - Many encryption implementations allow differentiating between random results and erroneous results. However, not all encryption implementations allow this.

Expressing a Function as a Boolean Circuit

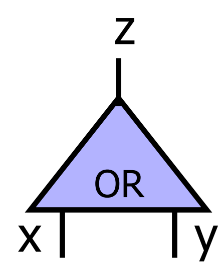
First, the function has to be converted into a boolean circuit





Truth table:

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



Truth table:

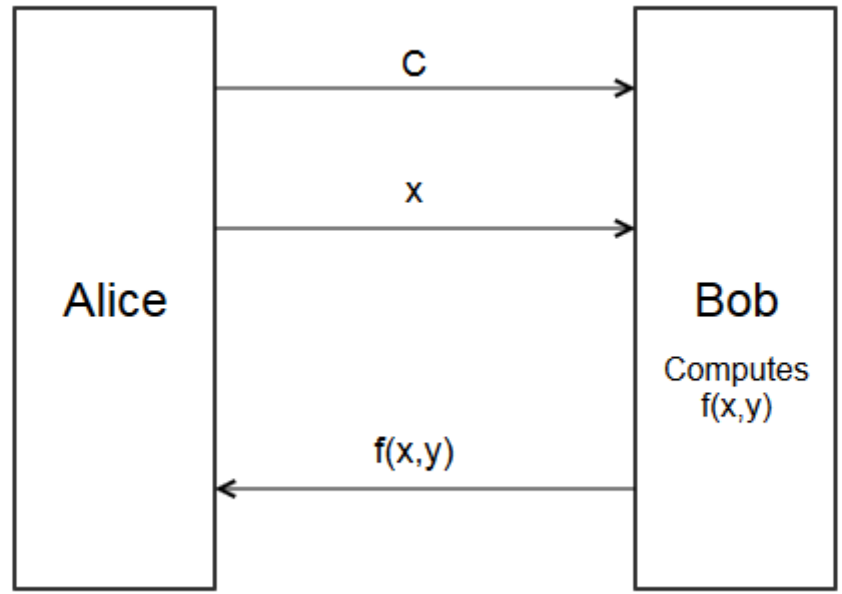
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Insecure Protocol for Evaluating a Circuit with Two Parties

- Alice sends circuit C to Bob.
- Alice sends her input x to Bob.
- Bob evaluates the circuit to get f(x,y)
- Bob sends f(x,y) back to Alice.

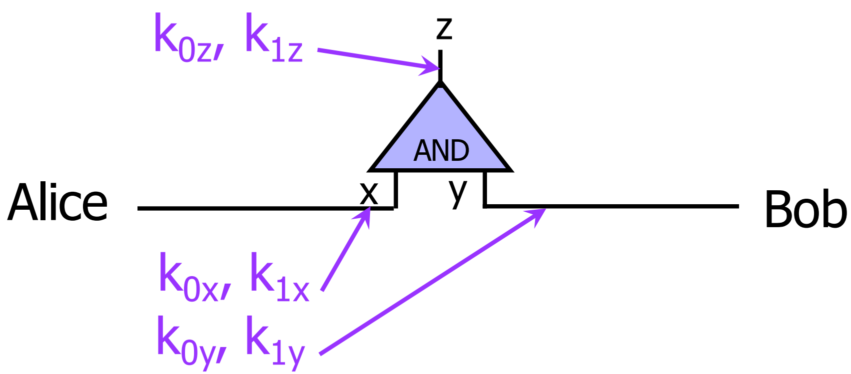
- This works, but Alice has to send x to Bob.

We don't want Bob to know anything besides f(x,y) and y after evaluating the circuit!



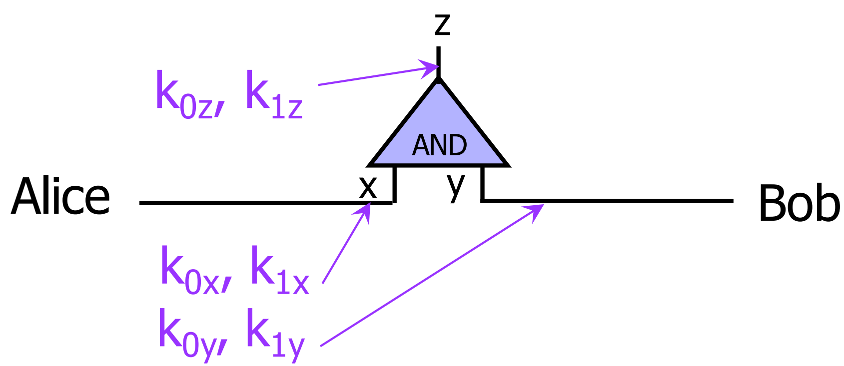
Step 1: Pick Random Keys for Each Wire

- We first show how to evaluate one gate securely
  - We later show how to generalise this to the entire circuit
- Alice picks two **random keys** for each wire
  - These are encryption keys for a symmetrical encryption function
  - One key corresponds to “0”, the other to “1”
  - 6 keys in total for a gate with 2 input wires



Step 2: Encrypt the Truth Table for every Gate

- 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

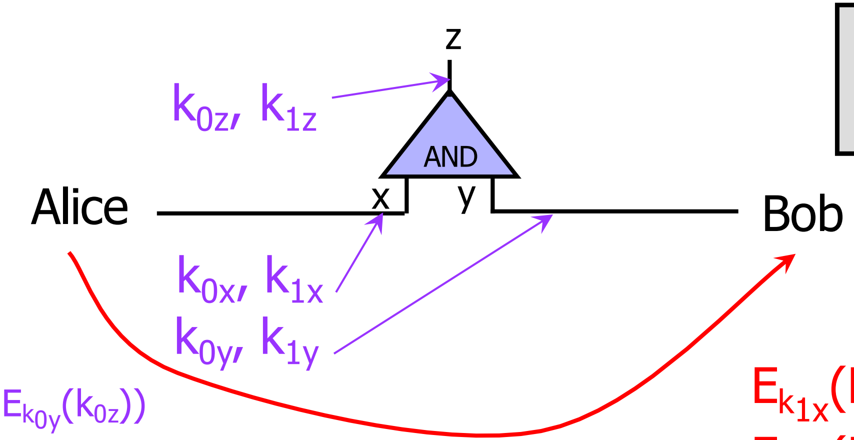
Encrypted truth table:

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

Lecture PETs4DS – Privacy Enhancing Technologies for Data Science
Dr. Benjamin Heitmann and Prof. Dr. Stefan Decker
Informatik 5, Lehrstuhl Prof. Decker

Step 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:

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

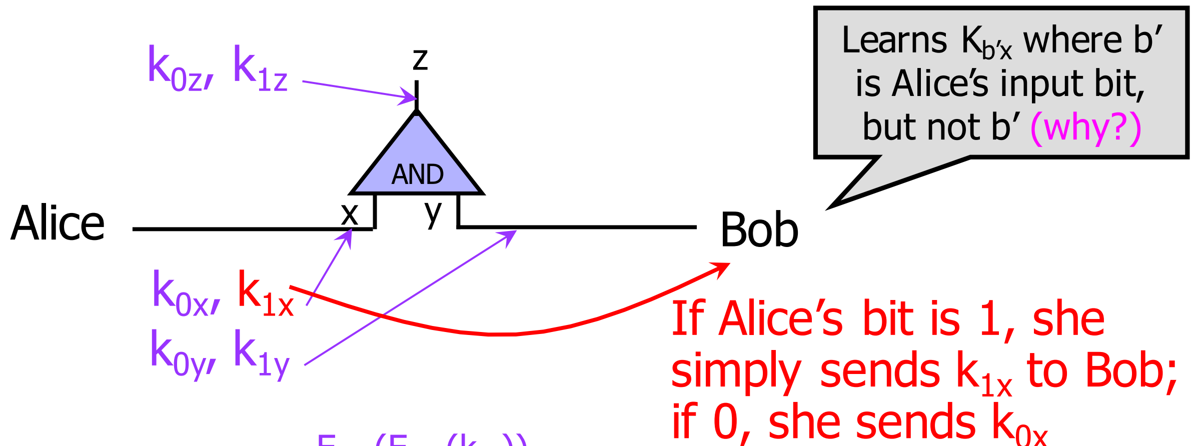
$E_{k_{0x}}(E_{k_{1y}}(k_{0z}))$

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

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

Step 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

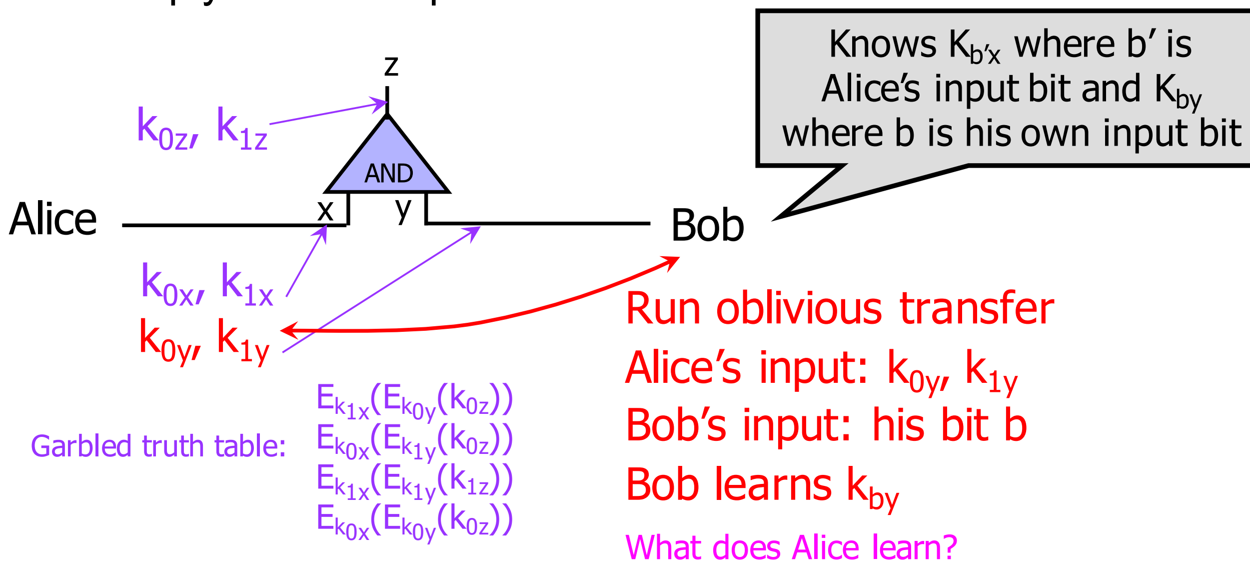


Garbled truth table:

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

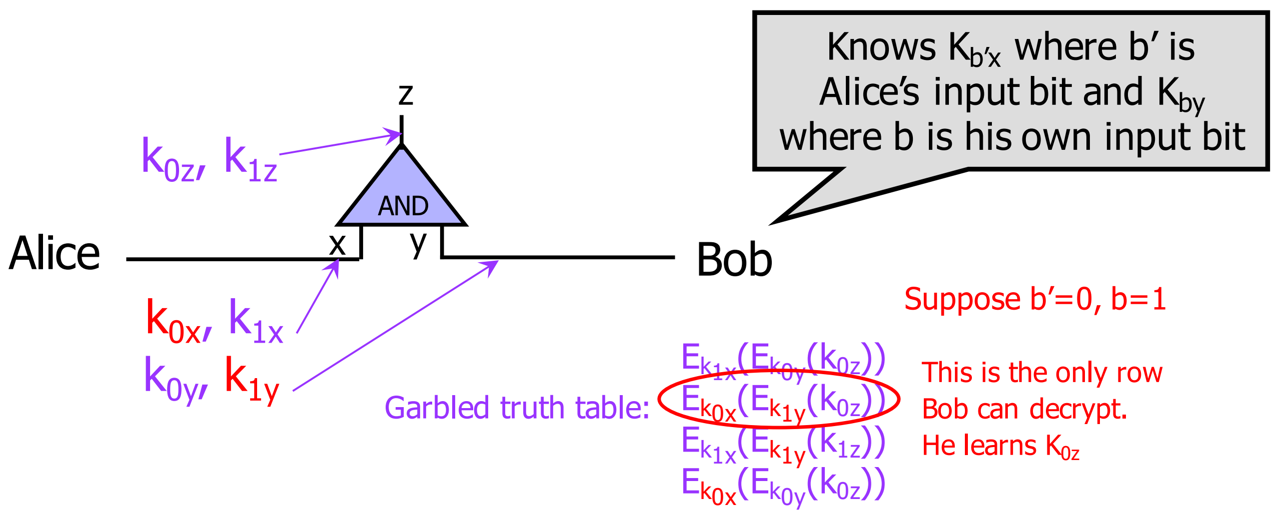
Step 5: Use Oblivious Transfer on Keys for Bob's Input

- 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



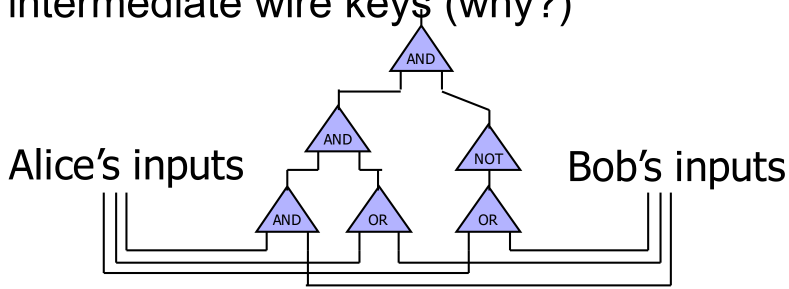
Step 6: Evaluate Garbled Gate

- 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?



Step 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 does not tell her intermediate wire keys (why?)



Step 8: Communicate the Result of the Circuit to Both Parties

- For the final garbled truth table, Alice has encrypted all the possible results
- After Bob evaluates the whole circuit he gets either E(final=1) or E(final=0)

- Bob sends the final result to Alice
  – However, Bob does not have the key to decrypt this final result

- Alice decrypts the result locally
  – Now she knows if final=1 or final=0

- Now Alice wants Bob to know the result without needing to trust her

- Alice sends Bob the key to decrypt the final result
  – Bob locally decrypts the result, and now he also knows if final=1 or final=0

- Why is there a problem if Alice just sends Bob the final result ?

Summary of Yao's Protocol for Evaluating Garbled Circuits for Two Parties

- Step 1: Pick Random Keys for Each Wire (input and output) of the gate
- Step 2: Encrypt the Truth Table for the gate
- Step 3: Send Garbled Truth Tables
- Step 4: Send Keys for Alice's Inputs
- Step 5: Use Oblivious Transfer on Keys for Bob's Input
- Step 6: Evaluate Garbled Gate
- Step 7: Evaluate Entire Circuit
- Step 8: Communicate the result of the Circuit

Yao's GC Protocol: Maturity and Complexity

- garbling linear in circuit complexity
  - 2 random values per wire (as in: 2 encryption keys per wire)
  - 4 masking operations per gate
- evaluation linear in circuit complexity
  - as many lookup and unmasking operations as gates
- communication linear in circuit complexity (and input length) times token length
  - communication of gate lookup tables and input tokens of P_1
  - as many oblivous transport instances as P_2's input length
- constant rounds of communication
- established frameworks for two-party case: e.g. Fairplay, FastGC

Brief Discussion of Yao's Protocol

- The function must be converted into a circuit
  - For many functions, circuit will be huge
- 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
  - Number of rounds does not depend on the number of inputs or the size of the circuit!
- However, every G(C) of a circuit C can only be used exactly once!

Extensions of Yao’s Garbled Circuits

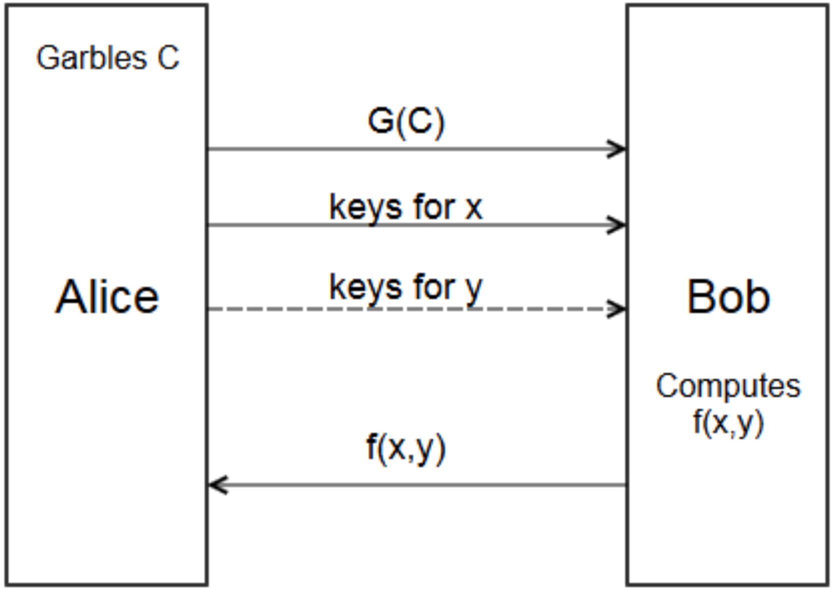
These extensions are outside of the scope of this lecture

- Protection against malicious adversaries
  - Extensive research on actively secure GC exists as well
- Efficiency improvements
  - Reduce e.g. the number of required encryptions/descriptions to evaluate a circuit
- Extensions to enable more than two parties
  - Still require two party protocol between all pairwise combinations of participants

Oblivious Transfer (OT): Why do we need it?

Without OT, we have the following protocol to evaluate a garbled circuit:

- Alice garbles circuit C to get garbled circuit G(C)
- Alice sends G(C) to Bob.
- Alice sends the keys for her input x to Bob.
- Bob combines them with the input keys for y, and evaluates G(C) to get f(x,y)
- Bob sends f(x,y) back to Alice.



Unsolved problem: How does Bob get the key which matches his input y?

Naïve solution: Send all possible keys for Bobs input to Bob.

Problem of Yao's GC protocol without OT

Unsolved problem:
How does Bob get the key which matches his input y?

Naïve solution:
Send all possible keys for Bobs input to Bob.

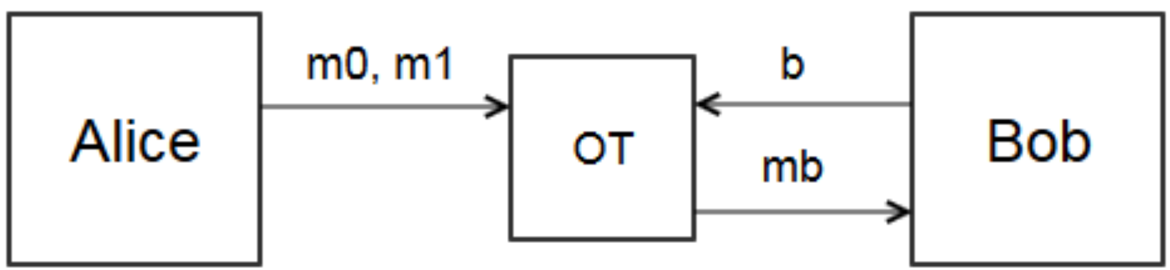
Big Problem:
This allows Bob to run the circuit two times (e.g. for y=0 and y=1).
This gives Bob more additional knowledge.

We want Bob to have exactly enough information to evaluate G(C) only once!

Goal of Oblivious Transfer

Alice has two messages m_0, m_1. Bob has a bit b.

We treat oblivious transfer as a black box method where:
- Alice gives m_0, m_1 into the black box.
- Bob gives bit b, which can have the value 0 or 1.
- If b=0 Bob gets m_0. Otherwise, he gets m_1. In both cases, Bob does not learn the other message.
- Alice does not learn which message Bob received. She only knows Bob got one of them.



Implementing OT using RSA – Part 1: The Setup

	Alice				**Bob**	
Secret	**Public**	**Explanation**		**Secret**	**Public**	**Explanation**
m_0, m_1		Messages to be sent				
d	N, e	Generate RSA key pair and send public portion to Bob	->		N, e	Receive public key
	x_0, x_1	Generate two random messages	->		x_0, x_1	Receive random messages

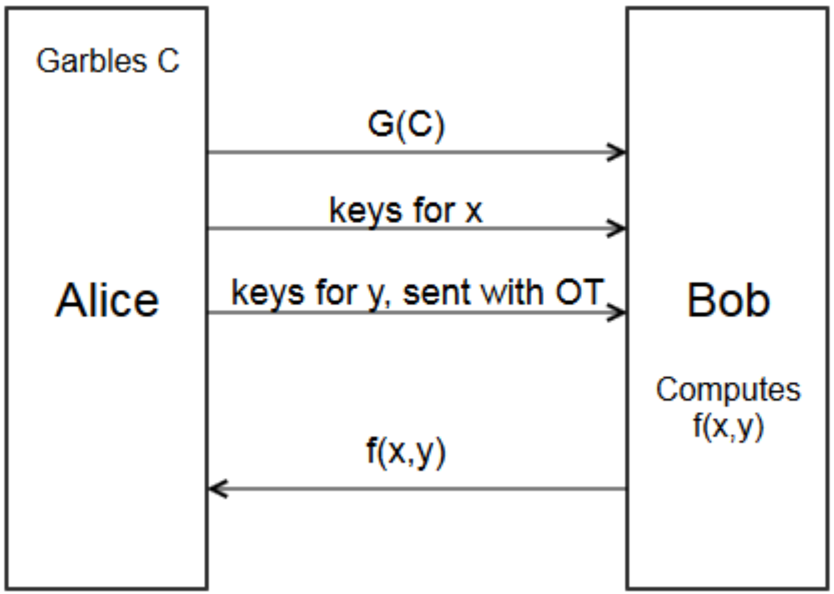
Implementing OT using RSA – Part 2: The OT protocol after the Setup

Alice | **Bob**

Secret	Public	Explanation		Secret	Public	Explanation
				k, b		Choose $b \in \{0, 1\}$ and generate random K
	v		<-		$v = (x_b + k^e) mod\ N$	Compute the encryption of k, blind with x_b and send to Alice
$k_0 = (v - x_0)^d mod\ n$ $k_1 = (v - x_1)^d mod\ N$		One of these will equal k, but Alice does not know which.				
	$m'_0 = m_0 + k_0$ $m'_1 = m_1 + k_1$	Send both messages to Bob.	->		m'_0, m'_1	Receive both messages
				$m_b = m'_b - k$		Bob decrypts the m'_b since he knows which x_b he selected earlier.

Summary of Yao's GC protocol with OT

- Alice garbles circuit C to get garbled circuit
- Alice sends G(C) to Bob.
- Alice sends the keys for her input x to Bob.
- Using oblivious transfer, for each of Bob's input wires, Alice sends k_{i,y_i} to Bob.
- With all input keys, Bob can evaluate the circuit to get f(x,y))
- Bob sends f(x,y) back to Alice.
- Note that f(x,y) is encrypted (see step 8).



Now Bob only learns the keys for his input.

Review of previous lecture

Clarifications for the slides from last week

- Clarifications for the comparison table
- SMPC:
  - SMPC with CEPS protocol protects against t < n/2
- Yao's garbled circuits:
  - 2 keys per wire
  - Clarification on communication of result of full circuit evaluation
- OT with RSA: $(x_b + k^e) mod\ N$ is correct, $\left(x_b + k\ \right)^e\ mod\ N$ is incorrect

Comparison of Approaches for Private Computation

Approach	Adversary Type	Confidentiality	Integrity	Requires Interaction
Homomorphic Encryption (HE)	Honest-but-curious (HBC)	YES	NO	NO
Verifiable Computation (VC)	Malicious	NO	YES	NO
Secure Multi-Party Computation (SMPC)	HBC or Malicious	YES	YES	YES

Clarification on adversary types against which SMPC is secure:
The two SMPC protocols from the lecture are **only** secure against HBC adversaries,
but there are other established SMPC protocols which are also secure against malicious adversaries.

Summary of Yao's Protocol for Evaluating Garbled Circuits for Two Parties

- Step 1: Pick Random Keys for Each Wire (input and output) of the gate
- Step 2: Encrypt the Truth Table for the gate
- Step 3: Send Garbled Truth Tables
- Step 4: Send Keys for Alice's Inputs
- Step 5: Use Oblivious Transfer on Keys for Bob's Input
- Step 6: Evaluate Garbled Gate
- Step 7: Evaluate Entire Circuit
- Step 8: Communicate the result of the Circuit

Step 8: Communicate the Result of the Circuit to Both Parties

- For the final garbled truth table, Alice has encrypted all the possible results
- After Bob evaluates the whole circuit he gets either E(final=1) or E(final=0)

- Bob sends the encrypted final result to Alice
  - However, Bob does not have the key to decrypt this final result

- Alice decrypts the result locally
  - Now she knows if final=1 or final=0

- Now Alice wants Bob to know the result without needing to trust her

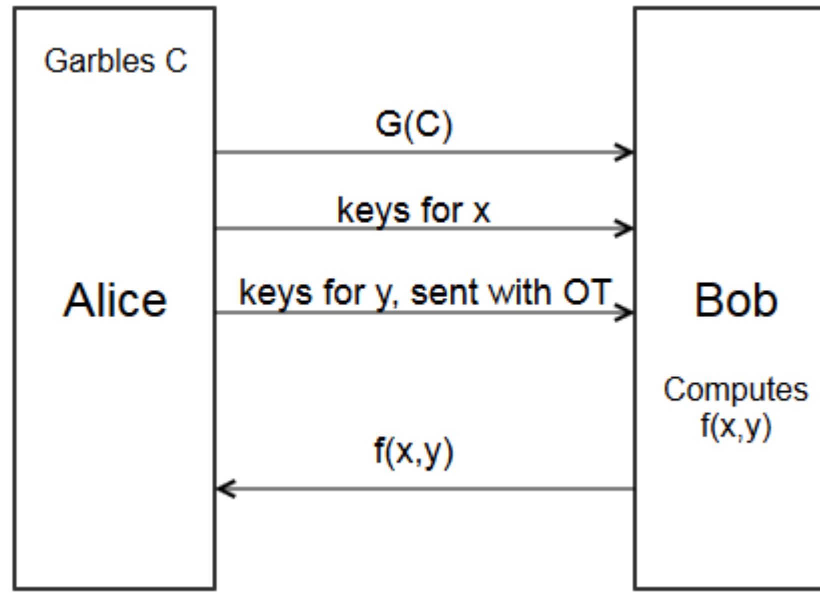
- Alice sends Bob the key to decrypt the final result
  - Bob locally decrypts the result, and now he also knows if final=1 or final=0

- Why is there a problem if Alice just sends Bob the final result ?

Summary of Yao's GC protocol with OT

- Alice garbles circuit C to get garbled circuit
- Alice sends G(C) to Bob.
- Alice sends the keys for her input x to Bob.
- Using oblivious transfer, for each of Bob's input wires, Alice sends k_{i,y_i} to Bob.
- With all input keys, Bob can evaluate the circuit to get f(x,y))
- Bob sends f(x,y) back to Alice.
- Note that f(x,y) is encrypted (see step 8).

Now Bob only learns the keys for his input.



Homomorphic Encryption

The Secure Multiplication Protocol (SMP) for the Paillier SC is described in:
Samanthula, Bharath Kumar, Wei Jiang, and Elisa Bertino. "Privacy-preserving complex query evaluation over semantically secure encrypted data." *European Symposium on Research in Computer Security*. Springer International Publishing, 2014.

Homomorphic Encryption

- Homomorphism:
  - Given: groups $(P,\oplus)$ and $(Q,\otimes)$, relation $f: P \rightarrow Q$
  - f homomorphic w.r.t. $\oplus$ iff

$$\forall a, b \in P: f(a \oplus b) = f(a) \otimes f(b)$$
$$\Leftrightarrow$$
$$\forall a, b \in P: a \oplus b = f^{-1}(f(a) \otimes f(b))$$

- Homomorphic Cryptosystem:
  - P: Message space, Q: Ciphertext space, f: encryption fct., f^{-1}: decryption fct.
  - encryption fct. homomorphic w.r.t. at least one operation in message space

- Fully Homomorphic Cryptosystem:
  - encryption function homomorphic w.r.t addition **and** multiplication in message space

The Challenge for Homomorphic Encryption

- All cryptosystems which are fast enough to be practically used are not homomorphic regarding addition and multiplication at the same time
- On the other hand, all cryptosystems which are fully homomorphic, are to slow to be used practically.
- **Many “tricks” are required to approximate FHE with realistic performance**
- We will look at one such trick now!

Example for Multiplicative Homomorphic Cryptosystem: RSA

- Homomorphic with respect to multiplication

- Public Key: (e, n) , Private Key: d
  with $n = pq, d = e^{-1} \, mod \, \varphi(n), \;\; p, q \; prime$
- Plaintexts: $m_1, m_2 \; mod \; n$
- Ciphertexts: $\varsigma_{m_1} = m_1^e \; mod \; n, \varsigma_{m_2} = m_2^e \; mod \; n$

$$\varsigma_{m_1} \cdot \varsigma_{m_2} = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = \varsigma_{m_1 \cdot m_2} \quad (mod \; n)$$

- But: Not homomorphic with respect to addition

Example for Additive Homomorphic Cryptosystem: Paillier Cryptosystem

- The Paillier Cryptosystem is asymmetric, so pk is the public key used for encryption, and sk is the secret key used for decryption.

a. **Homomorphic Addition:**

$$E_{pk}(x + y) \leftarrow E_{pk}(x) * E_{pk}(y) \bmod N^2;$$

b. **Homomorphic Multiplication:**

$$E_{pk}(x * y) \leftarrow E_{pk}(x)^y \bmod N^2;$$

- Note that for multiplication, the exponent y is required as **plain text!**

Can the Paillier Cryptosystem also be used for private multiplication?

- Multiplication requires one of the factors to be available as plaintext.
- Homomorphic multiplication allows both factors to be encrypted.
- So Paillier is not homomorphic with regards to multiplication.

- Can we multiply numbers in a private way for less strict security guarantees?

SMP using Paillier CS: Adversary Model

- Yes, it is possible to develop a protocol for Secure Multiplication (SMP) using the Paillier cryptosystem.

- However, we need to make two strong assumptions:

1. Adversaries are honest or semi-honest, not malicious
2. Adversaries do NOT collude

SMP using Paillier CS: Idea and Intuition

- Assume Bob wants to query his data in the cloud.
- However, he wants to use a device with constrained resources
  - Examples: smart phone, fitness tracker, IoT device.
- So the processing of the data is off-loaded to more powerful servers in the cloud.

- If we assume: the cloud == one server
  - Without homomorphic multiplication, the cloud can not process the data

- However, if we use two servers, maybe we can split the computation between the servers ?

SMP using Paillier CS: Use Case

- **Use case: Query processing over encrypted data**
  - Can Bob send queries to the his data in the cloud and receive answers, if his data is encrypted, and if

- Instead of using one cloud provider, Bob uses two providers.
  - Bob generates a keypair *(pk, sk)* for the Paillier CS.
  - Bob uses *pk* to encrypt his data.
  - Bob uploads his encrypted data T' to C_1.
  - Bob sends C_1 the public key *pk.*
  - Bob sends C_2 the secret key *sk*.
  - Note that C_1 has the encrypted data without the decryption key, and C_2 has the decryption key without the data.

- **Justification:** Big companies like Amazon, Google, Microsoft could loose a lot of money if corporate customers find out they are colluding.

SMP using Paillier CS: How to mask numbers in a finite field ?

- SMP is based on this property which holds for any $a, b \in Z_N$

$$a * b = (a + r_a) * (b + r_b) - a * r_b - b * r_a - r_a * r_b$$

- $r_a, r_b \in Z_N$ are random numbers which are only known to C_1

- This allows C_1 to mask a and b, even if they are encrypted.
- After masking a and b, C_1 can send them to C_2.

SMP using Paillier CS: The Protocol

a. **Homomorphic Addition:**

$$E_{pk}(x + y) \leftarrow E_{pk}(x) * E_{pk}(y) \bmod N^2;$$

b. **Homomorphic Multiplication:**

$$E_{pk}(x * y) \leftarrow E_{pk}(x)^y \bmod N^2;$$

Step 1

Algorithm 1 SMP$(E_{pk}(a), E_{pk}(b)) \rightarrow E_{pk}(a * b)$

Require: C_1 has $E_{pk}(a)$ and $E_{pk}(b)$; C_2 has sk

1: C_1:

(a). Pick two random numbers $r_a, r_b \in \mathbb{Z}_N$

(b). $a' \leftarrow E_{pk}(a) * E_{pk}(r_a)$

(c). $b' \leftarrow E_{pk}(b) * E_{pk}(r_b)$; send a', b' to C_2

SMP using Paillier CS: The Protocol

a. **Homomorphic Addition:**

$$E_{pk}(x+y) \leftarrow E_{pk}(x) * E_{pk}(y) \bmod N^2;$$

b. **Homomorphic Multiplication:**

$$E_{pk}(x*y) \leftarrow E_{pk}(x)^y \bmod N^2;$$

Step 2

2: C_2:

(a). Receive a' and b' from C_1

(b). $h_a \leftarrow D_{sk}(a')$

(c). $h_b \leftarrow D_{sk}(b')$

(d). $h \leftarrow h_a * h_b \bmod N$

(e). $h' \leftarrow E_{pk}(h)$; send h' to C_1

SMP using Paillier CS: The Protocol

a. **Homomorphic Addition:**

$$E_{pk}(x+y) \leftarrow E_{pk}(x) * E_{pk}(y) \bmod N^2;$$

b. **Homomorphic Multiplication:**

$$E_{pk}(x*y) \leftarrow E_{pk}(x)^y \bmod N^2;$$

Step 3

3: C_1:

(a). Receive h' from C_2

(b). $s \leftarrow h' * E_{pk}(a)^{N-r_b}$

(c). $s' \leftarrow s * E_{pk}(b)^{N-r_a}$

(d). $E_{pk}(a*b) \leftarrow s' * E_{pk}(N - r_a * r_b)$

- Note that N-x is equivalent to –x under Z_N

SMP using Paillier CS: Summary

- Given the following three parties:
  - Bob
  - Cloud C_1
  - Cloud C_2
- The described protocol allows these three parties to perform a multiplication:
  - C_1 only gets encrypted numbers from Bob
  - C_2 only gets the secret key from Bob
  - C_1 sends the encrypted result of the multiplication to Bob
  - Bob decrypts the result

- The main assumption is that C_1 and C_2 are not colluding

SMP using Paillier CS: Discussion

- The main assumption is that C_1 and C_2 are not colluding
- **Discuss: Is this a safe assumption today?**
- How could C_1 and C_2 collude to share their data:
  - C_1 sends Bob's encrypted data to C_2: this allows C_2 to decrypt the data
  - C_2 sends the secret key to C_1: this allows C_1 to to decrypt the data
- Today this assumption does not hold:
  - There are state-actors who can force cloud providers to collude
  - However, this requires pressure on cloud providers using strong "*extrinsic factors*".
  - Example for such extrinsic factors: The US government parking a small army in Silicone Valley in front of e.g. the Google HQ.

SMP using Paillier CS: What can we do with it?

- The presented secure multiplication protocol (SMP) forms the foundation for:
  - Secure BIT-OR protocol
  - Secure Comparison protocol
  - Secure Evaluation of Individual Predicates
  - Protocol for Query Processing over Encrypted Data

- All of these protocols are fast enough to be queried from a device with constrained resources.

- Full describtion of all protocols in:

  Samanthula, Bharath Kumar, Wei Jiang, and Elisa Bertino. "Privacy-preserving complex query evaluation over semantically secure encrypted data." *European Symposium on Research in Computer Security*. Springer International Publishing, 2014.

Summary of Challenges for Achieving Fully Homomorphic Cryptosystems

- Problem: Supporting addition **and** multiplication is difficult
  - operations on ciphertexts typically introduce noise
  - too many operations: ciphertexts don‘t decrypt correctly

  → Somewhat homomorphic encryption (bounded by multiplicative depth)

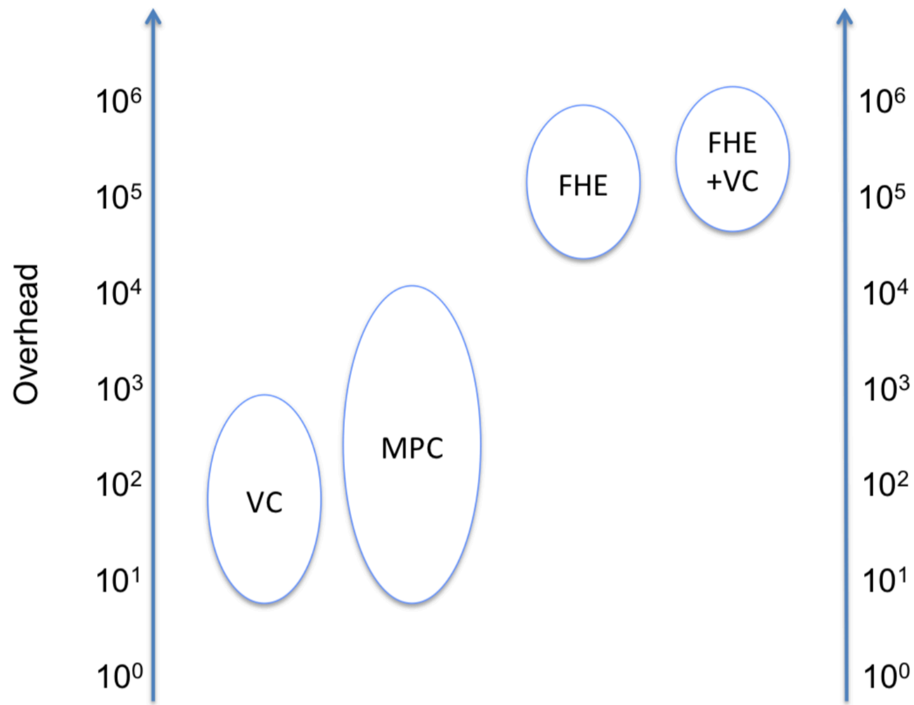
- First fully homomorphic cryptosystem presented by Gentry in 2009
  - basis: somewhat homomorphic scheme
  - bootstrappable: large enough bound to perform some operations and „recryption“

    → produce a new ciphertext without decrypting

- However, FHE implementations remain impractically slow

Limitations of Private Computation Regarding Performance

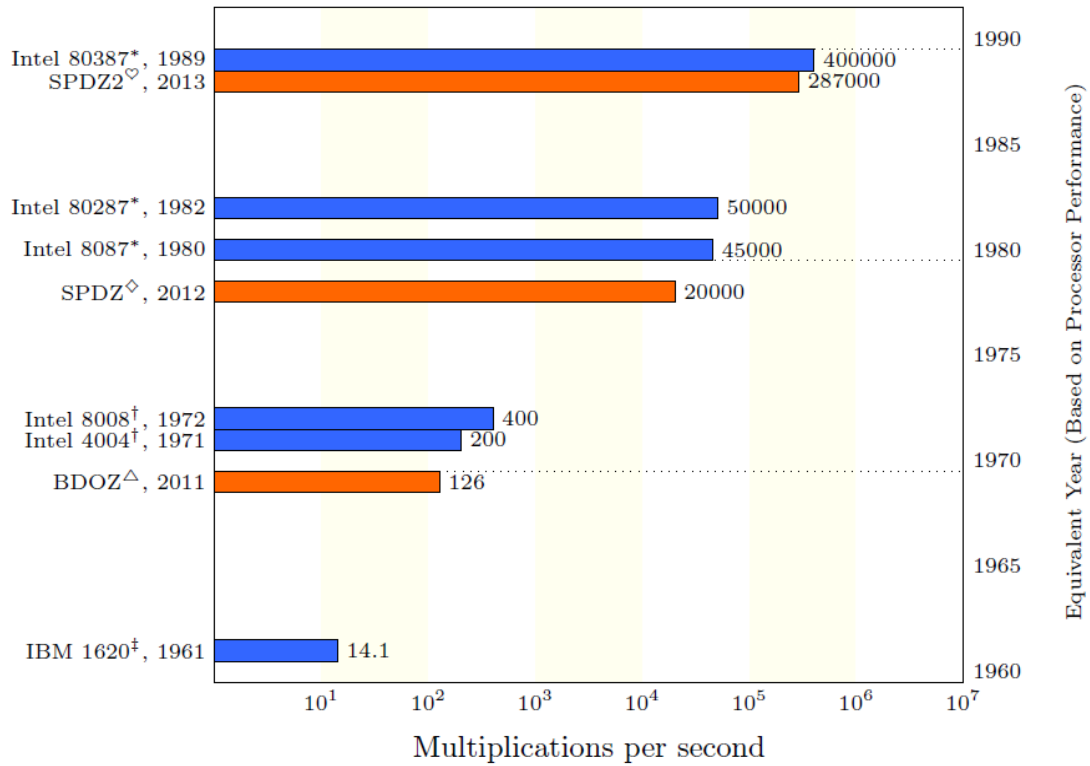
Comparison of Performance Overhead Incurred by Approaches



Graphical depiction of the multiplicative performance overheads over unsecured computation incurred by HE, VC and multi-party computation (MPC).

MPC and SMPC are the same.

CPU Computation Equivalence of SPDZ Protocol for SMPC



Source of the diagram:
Ivan Damgard et al. Practical covertly secure MPC for dishonest majority–or: Breaking the SPDZ limits. presentation slides, 2013. url: https://www.cs.bris.ac. uk/home/ps7830/spdz2.pdf.

Performance of Private Computation

- All discussed approaches incure a significant performance overhead.

- Examples for SMPC performance:
  - SPDZ protocol in 2011: 126 multiplications -> equivalent to Intel 4004 CPU from 1971
  - SPDZ protocol in 2013: 287000 multiplications -> equivalent to Intel 80387 CPU from 1989
  - More performance improvements can be expected.

- SMPC incures the least overhead.
  - But SMPC does not scale well for more than 3 parties!