Crypto-Based Solutions

Asst. Prof. Sinem Sav

Topics:

Crypto-based solutions

- Adversarial Models
- Homomorphic encryption
- Attribute-based credentials
- Zero-knowledge proofs
- Secure Multiparty Computation

Some slides/ideas adapted from: Jean-Pierre Hubaux, Erman Ayday, Carmela Troncoso, Bryan Ford, Vitaly Shmatikov, Wouter Lueks, David Froelicher, Christian Mouchet, Sylvain Chatel and Florian Tramer

Secure computation

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

Example

- Privacy and correctness property in “anonymous evaluation system for lecturers”
  - Privacy:
  - Correctness:

Example

- Privacy and correctness property in “anonymous evaluation system for lecturers”
  - Privacy:
    - The lecturer does not learn about the writer of the commment
    - The students do not learn about the writer of a specific comment
    - Overall, no parties learn more that their inputs (comments)
  - Correctness:
    - The lecturer learns about the correct comments, i.e., an adversary cannot corrupt a set of students to cheat on their comments to influence the outcome

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

Secure Multiparty Computation

Problem statement:

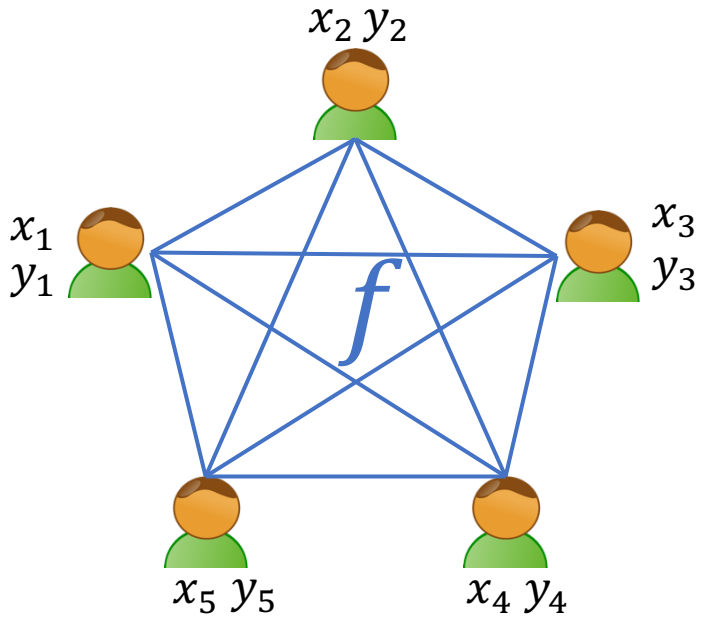
A set of players $\mathcal{P} = \{P_1, P_2, \dots, P_N\}$ would like to compute a function $f(x_1, x_2, \dots, x_N) = (y_1, y_2, \dots, y_N)$ of their joint inputs.

Requirements:

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

Realization:

A multiparty cryptographic protocol



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 models and security definitions

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 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

Definitions of Security
Preliminaries

- Assumptions:
  - Static corruptions and no honest majority
  - Polynomial-time adversaries
- Security parameter: *n* (length of the cryptographic key)

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

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

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

$f: \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, 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 $f_2(x, y)$

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

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
- $\text{view}_i^{\pi}(n, x, y)$: view of the i-th party during the execution of π
  - Includes contents of the party's internal random tape and messages it received
- $\text{output}_i^{\pi}(n, x, y)$: 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

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

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

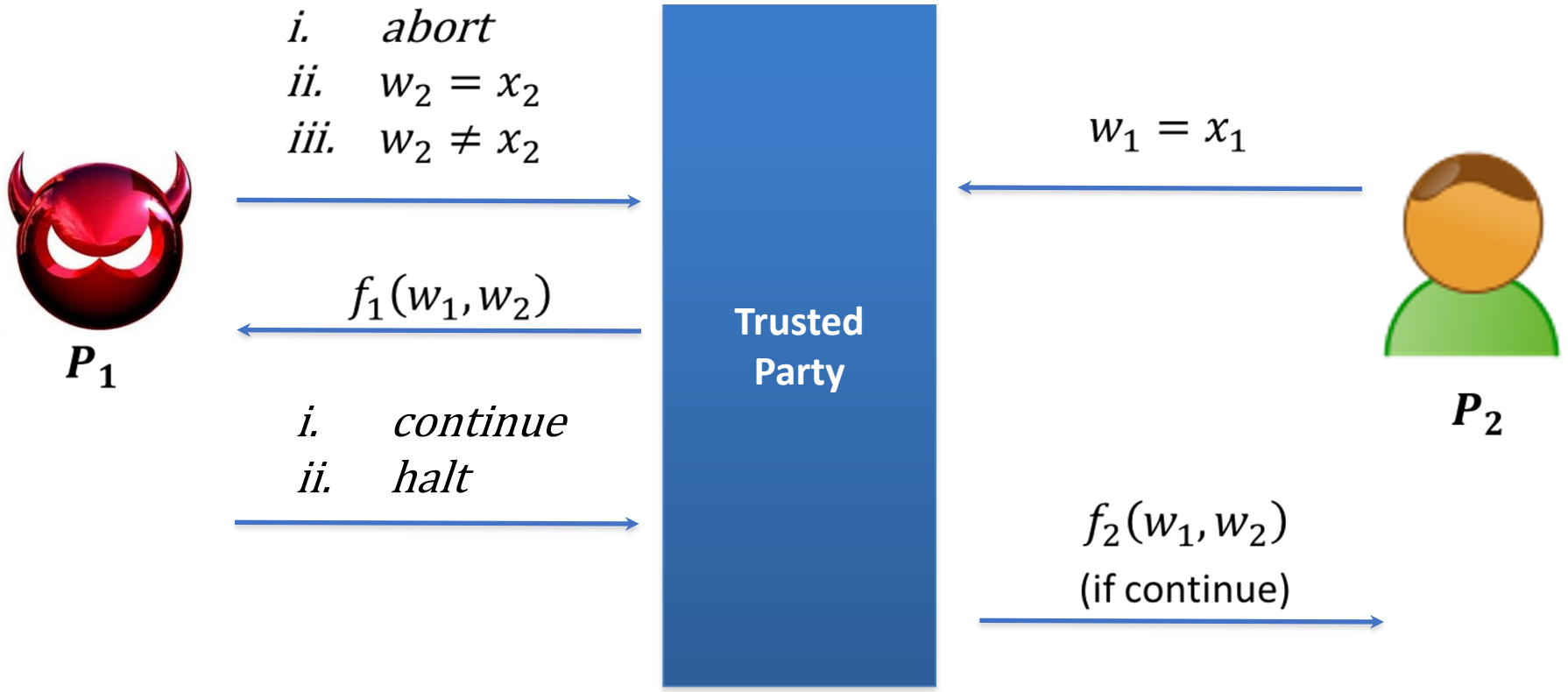
Security in Malicious Model

- 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

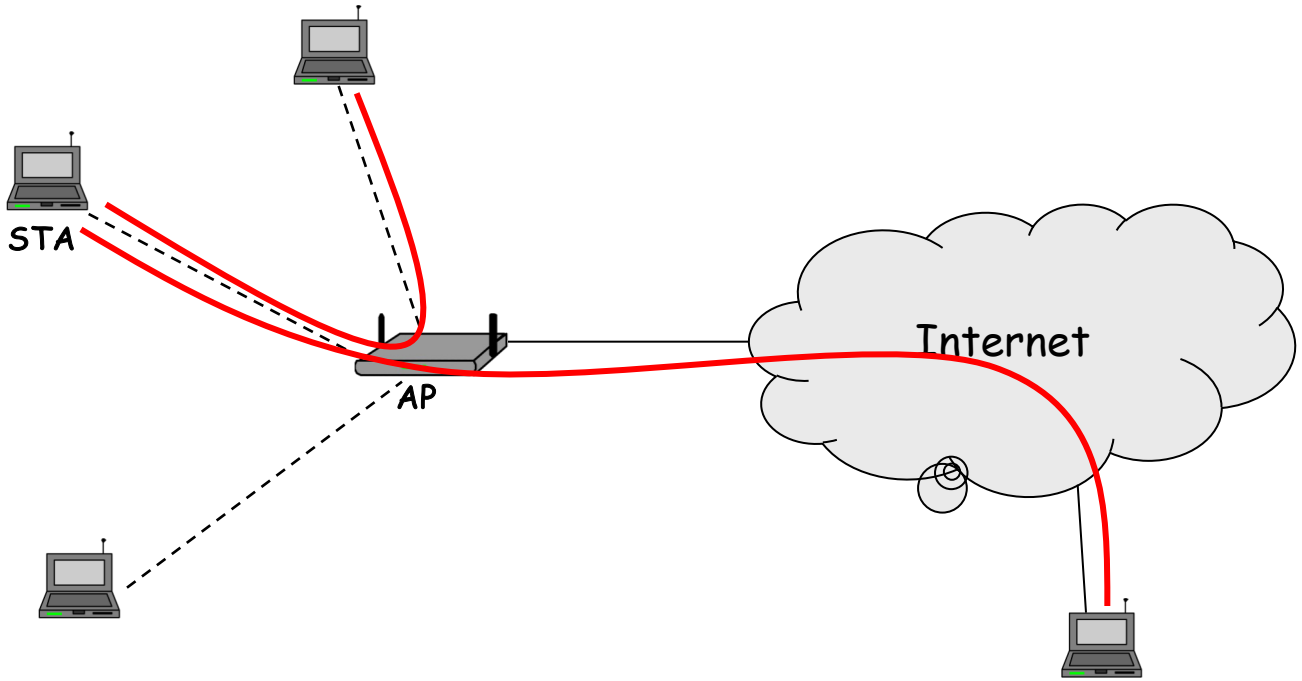


z models side information of the adversary

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)

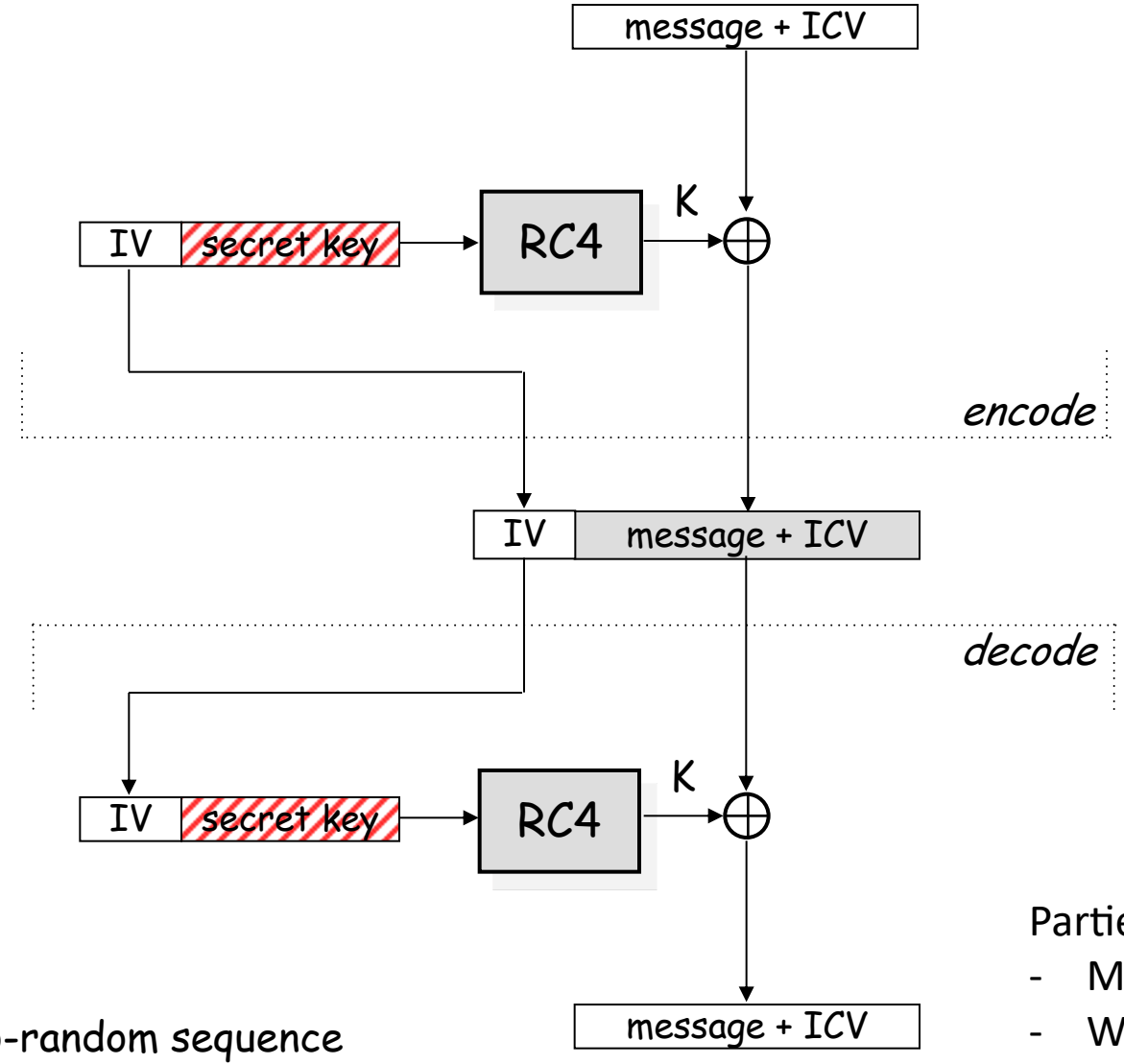


WEP – Message Confidentiality and Integrity

- 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:

$$CRC(X \oplus Y) = CRC(X) \oplus CRC(Y)$$

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

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

WEP- Conclusion

- A malicious adversary can temper the message content
  - And hence, the output of the honest party
- “Correctness” property does 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

Technologies for Privacy and Security Protection

Traditional Encryption

- Protects data at rest and in transit
- Cannot protect computation

Homomorphic Encryption

- Protects computation in untrusted environments
- Limited versatility vs efficiency

Secure Multiparty Computation

- Protects computation in distributed environments
- High communication overhead

Trusted Execution Environments

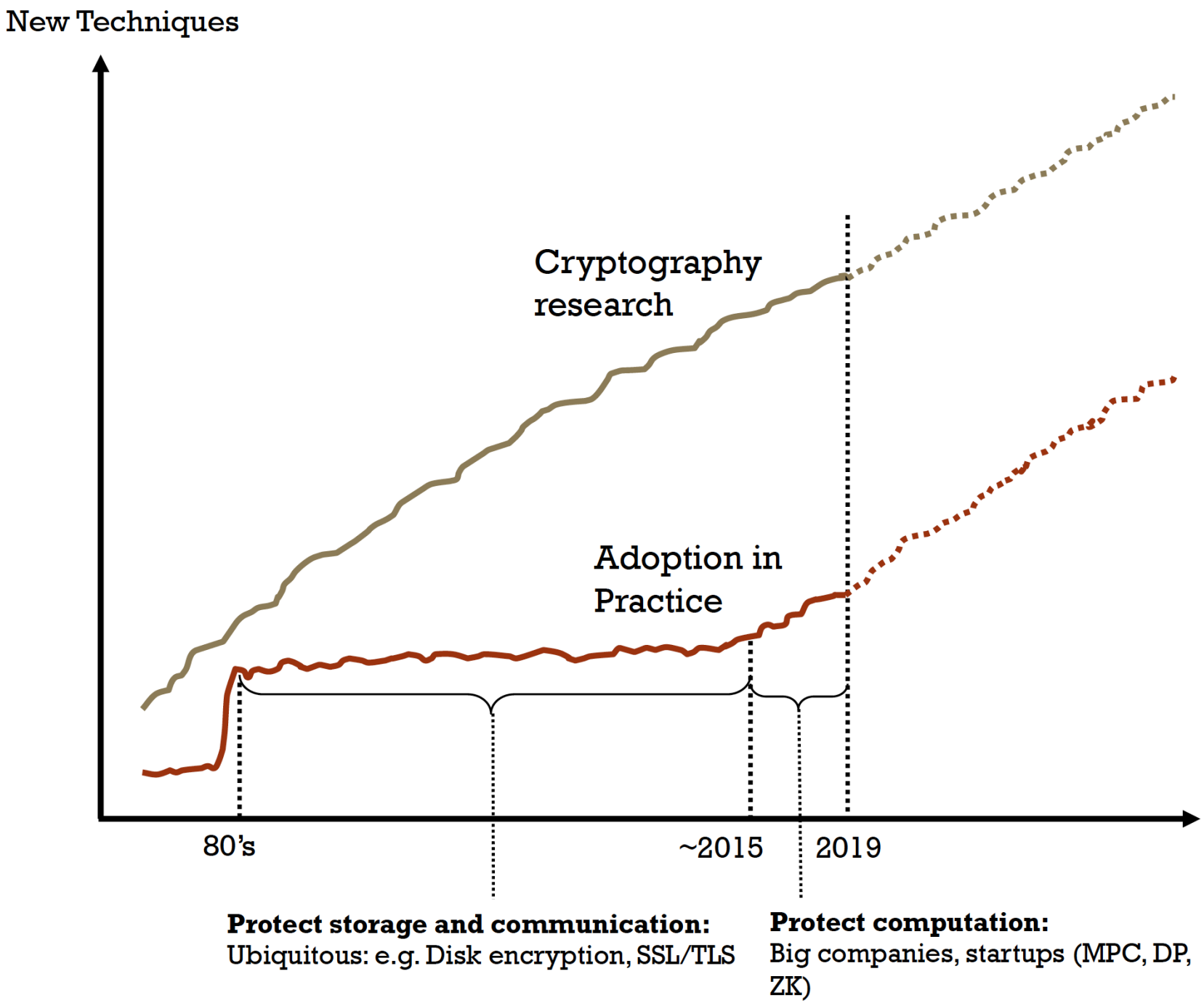
- Protects computation with Hardware Trusted Element
- Requires trust in the manufacturer, vulnerable to side-channels

Differential Privacy

- Protects released data from inferences
- Degrades data utility (privacy-utility tradeoff)

Distributed Ledger Technologies (Blockchains)

- Strong accountability and traceability in distributed environments
- Usually no data privacy



Qualitative diagram, courtesy Mariana Raykova

1.1 Homomorphic Encryption

Terminology

Plaintext space

Given a cryptosystem, we denote $\mathcal{P}$ its plaintext space: the set of all possible messages m

Ciphertext space

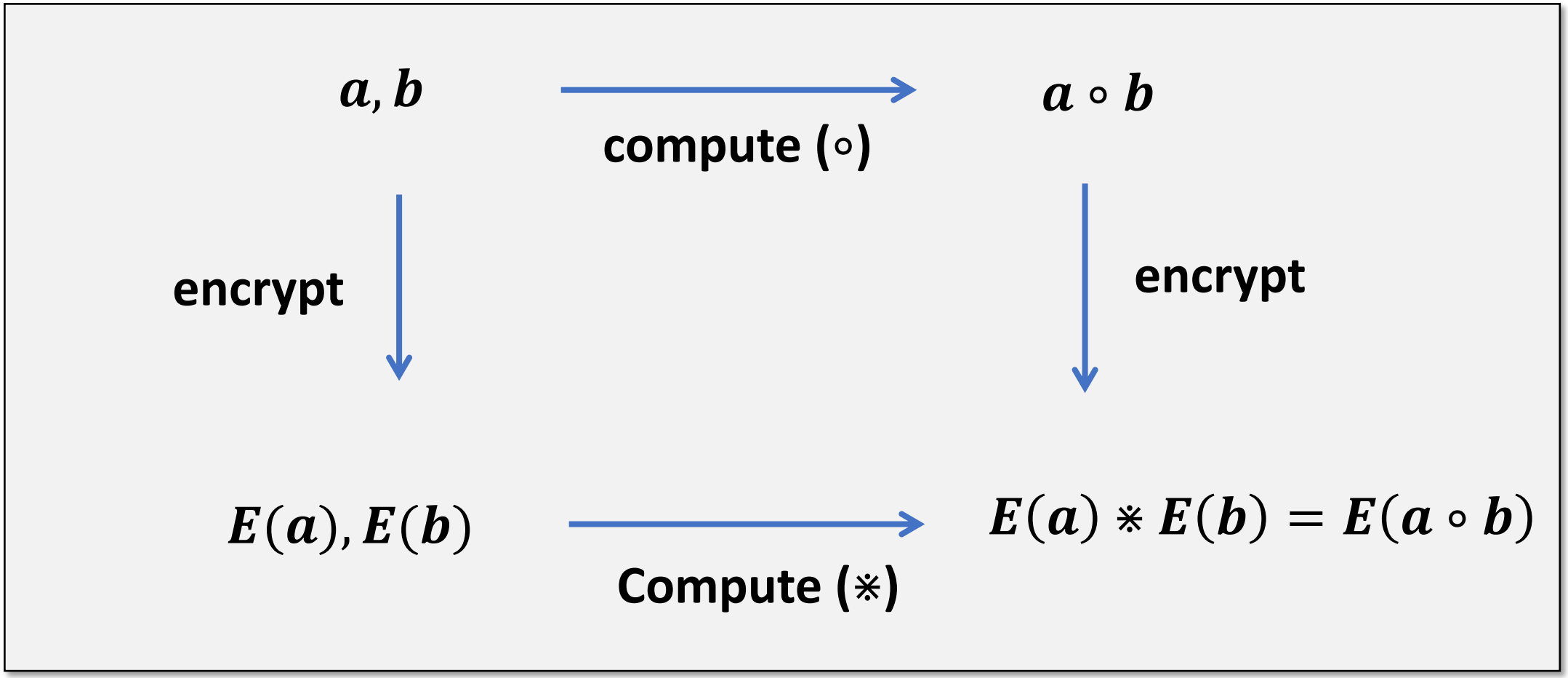
Given a cryptosystem, we denote C its ciphertext space: the set of all possible ciphertexts

Group homomorphism

Given $(G,\boxplus)$, $(H,\boxtimes)$ two groups, the function $h: G \mapsto H$ is a group homomorphism if

$$\forall u, v \in G, \qquad h(u \boxplus v) = h(u) \boxtimes h(v)$$

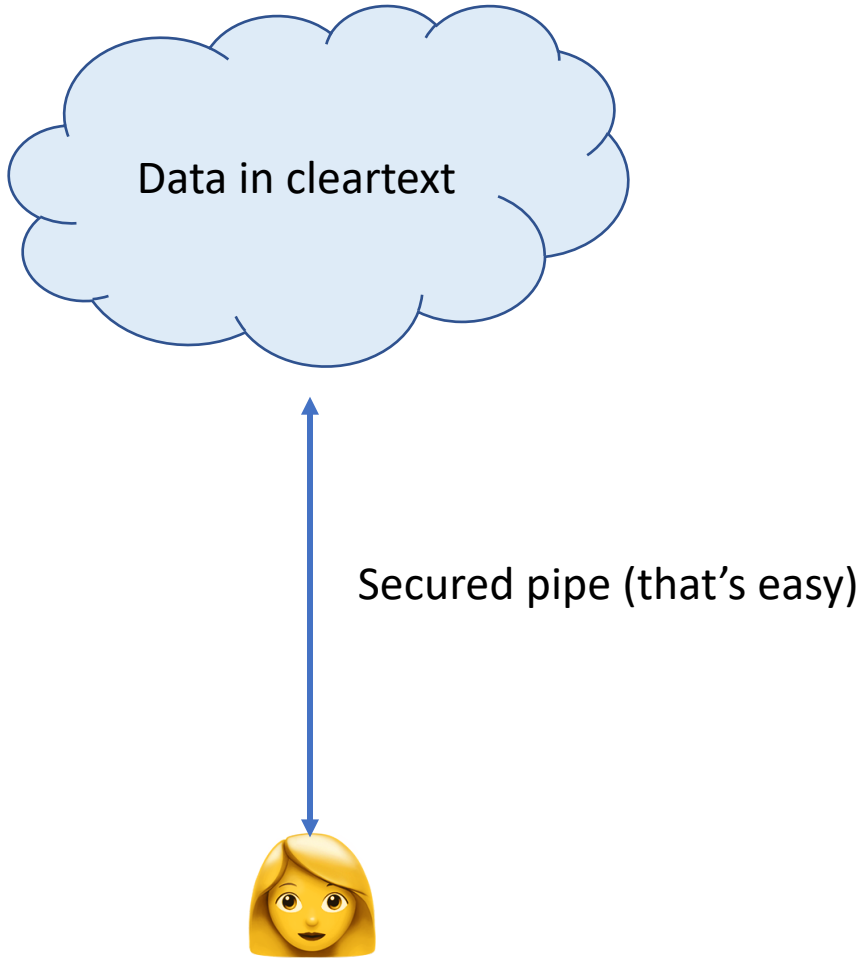
Homomorphic Encryption



Homomorphic encryption enables computations directly on encrypted data.

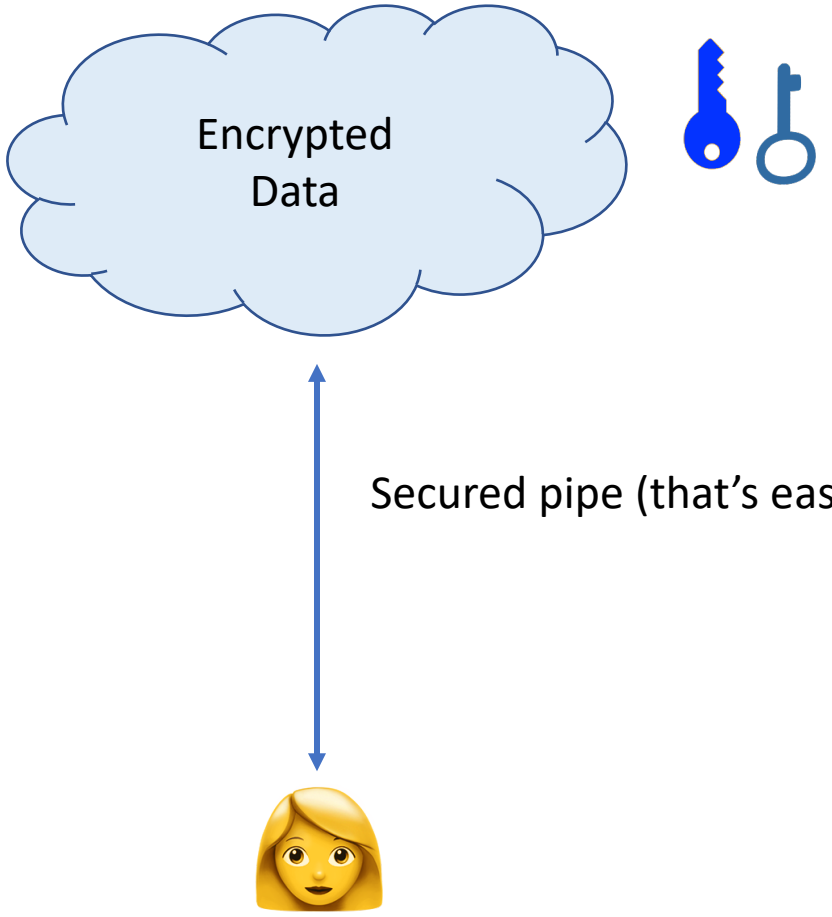
Storing data in the (un)trusted cloud

Case 0: The Cloud is Fully Trusted – Storage in clear text (never happens in practice)



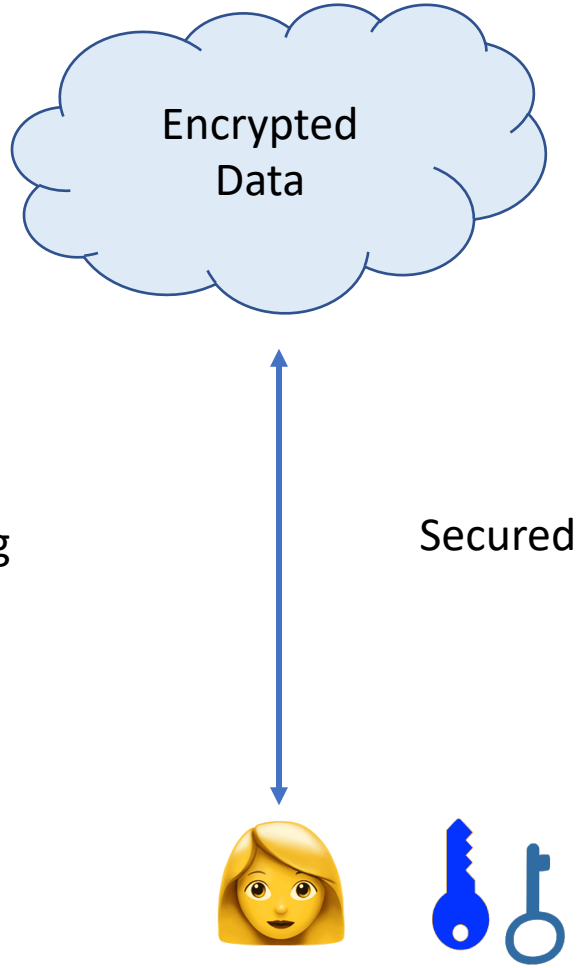
- Data sharing is easy
- Computation in the cloud is easy

Case 1: The Cloud is Fully Trusted – It encrypts with keys it controls



- Data sharing is easy
- Computation in the cloud is easy

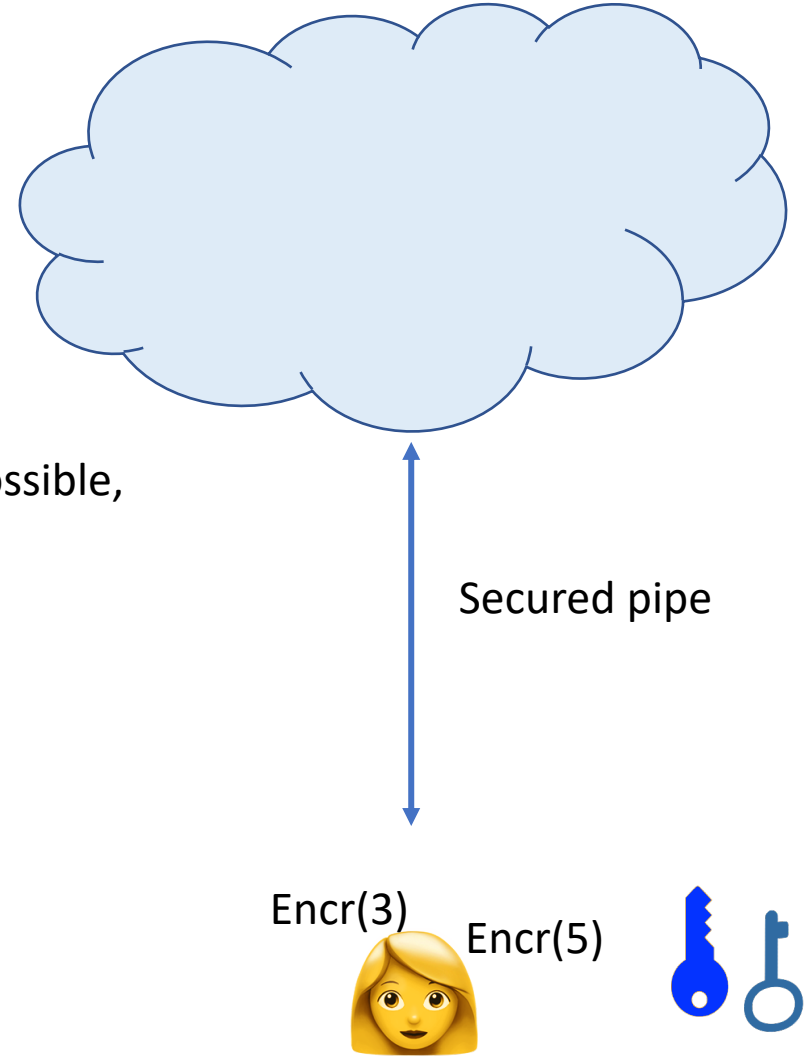
Case 2: The Cloud Is Untrusted – The user encrypts under their own keys



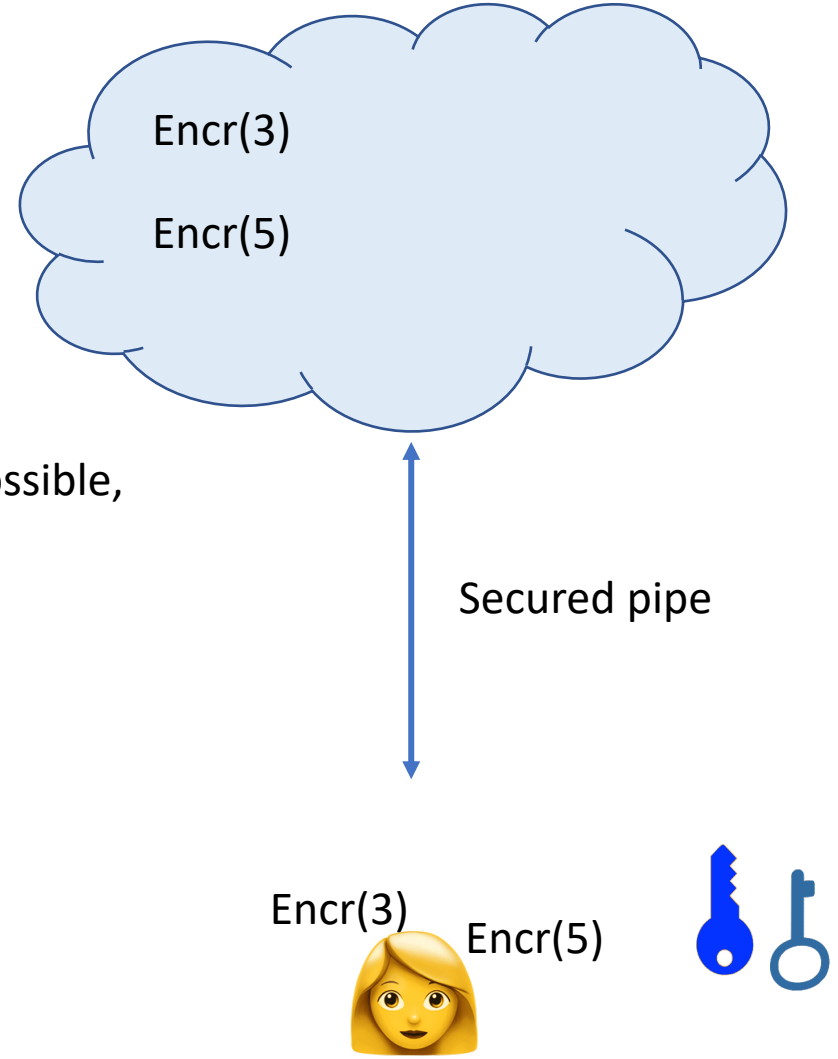
- Data sharing is tricky (key management)
- Computation in the cloud is impossible
- Some of the benefits of cloud computing are thus lost
- If the user loses their keys, they lose all their data

Case 3: The Cloud is Untrusted – The user homomorphically encrypts with keys it controls (1/3)

- Data sharing is doable
- Computation in the cloud is possible, but expensive

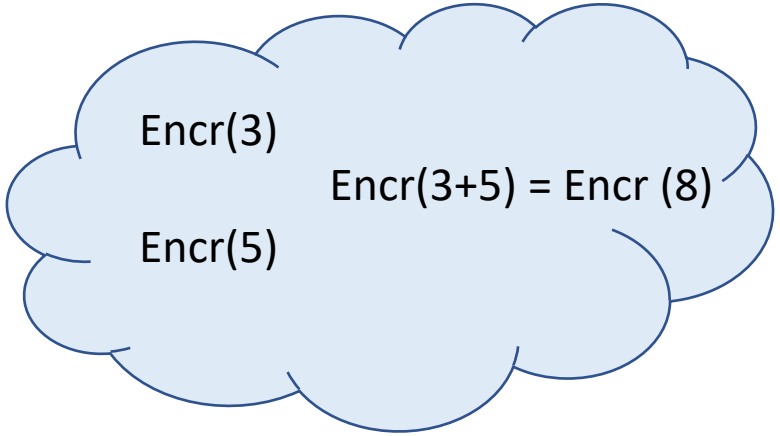


Case 3: The Cloud is Untrusted – The user homomorphically encrypts with keys it controls (2/3)

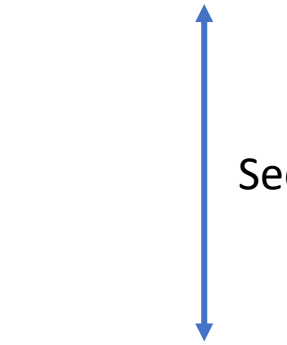


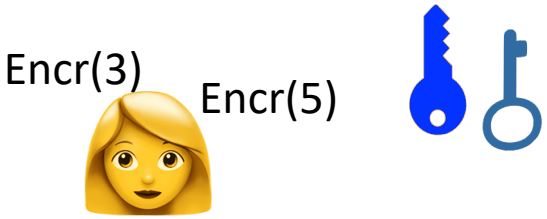
- Data sharing is doable
- Computation in the cloud is possible, but expensive

Case 3: The Cloud is Untrusted – The user homomorphically encrypts with keys it controls (3/3)



- The cloud can make computations on encrypted data, **for which it does not know the crypto keys**
- Hence computation in the cloud is possible (albeit expensive)
- Data sharing is doable





Homomorphic Encryption

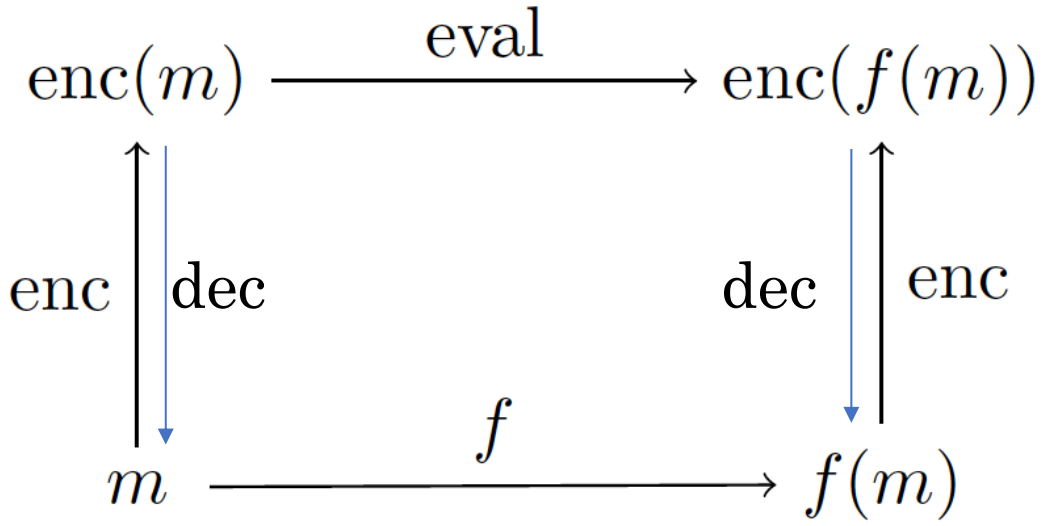
- Homomorphic cryptosystems are malleable by nature
- **There exists an homomorphism between their ciphertext and plaintext space**

  $\mathrm{h}(\mathrm{x}) = Dec(x)$ is an homomorphism between $\mathcal{C}$ and $\mathcal{P}$

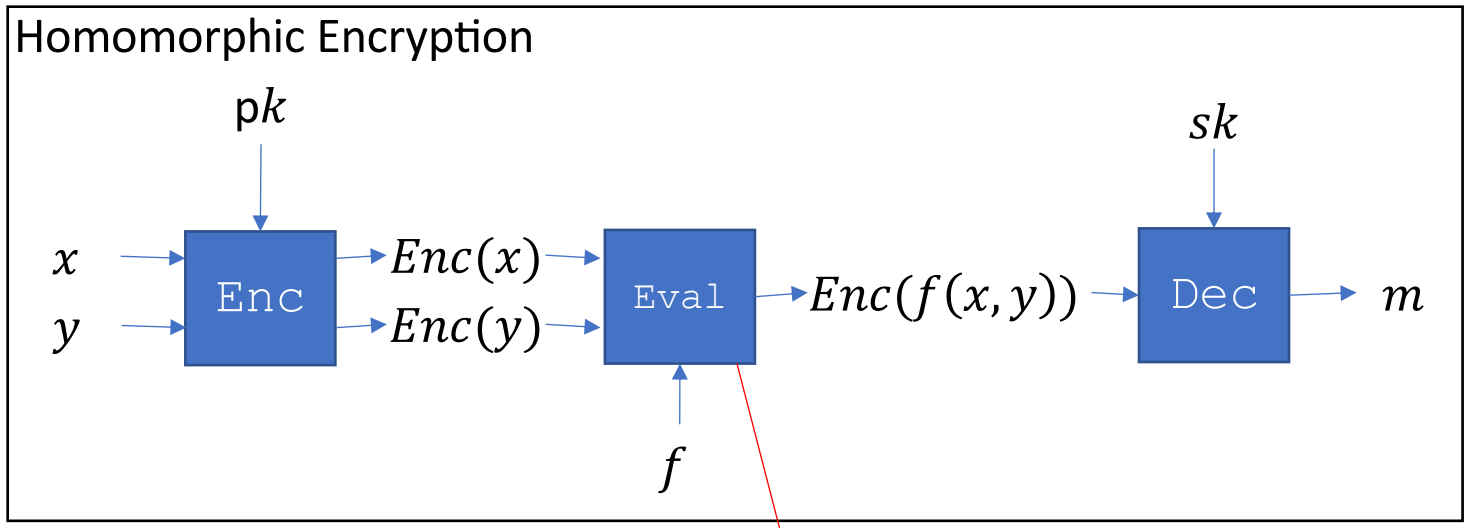
Some cryptosystems are **partially** homomorphic
→ they support **one** arithmetic operation

$Enc : (\mathcal{P}, \boxplus) \mapsto (\mathcal{C}, \boxtimes)$

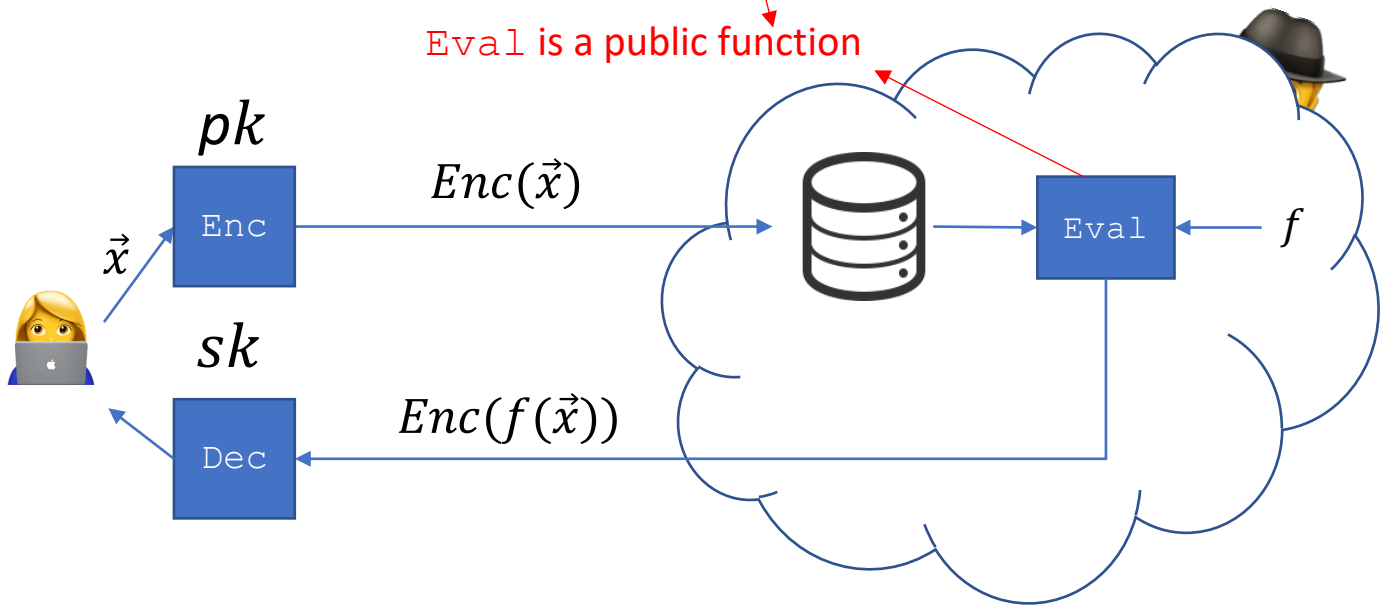
$\forall m_1, m_2 \in \mathcal{P}, Enc(m_1) \boxtimes Enc(m_2) = Enc(m_1 \boxplus m_2)$



Homomorphic Encryption



In Processing



Types of Homomorphic Encryption

- **Partially Homomorphic Encryption (PHE):** allows only a limited set of operations, e.g., only addition or only multiplication, can be performed on the encrypted data
- **Somewhat Homomorphic Encryption (SHE):** allows more than one operation, but still in fixed and finite combinations
- **Fully Homomorphic Encryption (FHE):** allows arbitrary operations to be applied to encrypted data in unconstrained combinations
  - bootstrapping?

Homomorphic Encryption - Paillier

<u>Additively Homomorphic</u>: Paillier Cryptosystem

Let p, q be two independent primes st. $\gcd\big(pq, (p-1)(q-1)\big) = 1$.
We define $n = pq$ and $\lambda = \varphi(n) = \varphi(p,q) = lcm((p-1)(q-1))$.
Let $\mu = \varphi(n)^{-1} \ mod \ n$.
Let (λ, μ) be the private key and n the public key.

For message $m \in P$, and $r \leftarrow \mathbb{Z}_n^*$

$$\boldsymbol{Enc}(m) = (1+n)^m r^n \ \ mod \ n^2$$

$$\boldsymbol{Dec}(c) = \frac{\left[c^{\varphi(n)} \ mod \ n^2\right] - 1}{n} (\varphi(n))^{-1} \ mod \ n$$

There exists a group homomorphism $\mathrm{Enc} : (P, +) \longrightarrow (C, \times)$:
For messages $m_1, m_2 \in P$, and $r_1, r_2 \leftarrow \mathbb{Z}_n^*$

$$Dec(Enc(m_1) \cdot Enc(m_2) \ mod \ n^2) \equiv Dec((1+n)^{m_1} r_1^n (1+n)^{m_2} r_2^n \ \ mod \ n^2)$$
$$\equiv \ Dec((1+n)^{m_1+m_2} (r_1 r_2)^n \ \ mod \ n^2)$$
$$\equiv m_1 + m_2 \ \ mod \ n$$

$' \leftarrow '$ means sampled from

Homomorphic Encryption- Paillier

Homomorphic properties of this encryption scheme:

Addition of two ciphertexts:

When two ciphertexts are multiplied, the result decrypts to the sum of their plaintexts

Enc(m1)* Enc(m2)= Enc(m1+m2)

Multiplication of a ciphertext by a constant (plaintext):

When a ciphertext is raised to the power of a plaintext, the result decrypts to the product of the two plaintexts:

Enc(m)^**k**= Enc(m*k)

Homomorphic Encryption- Paillier

<u>Additively Homomorphic</u>: Paillier Cryptosystem

Let p, q be two independent primes st. $\gcd\big(pq, (p-1)(q-1)\big) = 1$.
We define $n = pq$ and $\lambda = \varphi(n) = \ \varphi(p,q) = lcm((p-1)(q-1))$.
Let $\mu = \ \varphi(n)^{-1} \ mod \ n$.
Let (λ, μ) be the private key and n the public key.

For message $m \in P$, and $r \leftarrow \mathbb{Z}_n^*$

$$\boldsymbol{Enc}(m) = (1+n)^m r^n \ \ mod \ n^2$$

$$\boldsymbol{Dec}(c) = \frac{\left[c^{\varphi(n)} \ mod \ n^2\right] - 1}{n} (\varphi(n))^{-1} \ mod \ n$$

There exists a group homomorphism $\mathrm{Enc} : (P, +) \longrightarrow (C, \times)$:
For messages $m_1, m_2 \in P$, and $r_1, \ r_2 \leftarrow \mathbb{Z}_n^*$

$$\begin{aligned} Dec(Enc(m_1) \cdot Enc(m_2) \ mod \ n^2) &\equiv Dec((1+n)^{m_1} r_1^n (1+n)^{m_2} r_2^n \ \ mod \ n^2) \\ &\equiv \ Dec((1+n)^{m_1+m_2} (r_1 r_2)^n \ \ mod \ n^2) \\ &\equiv m_1 + m_2 \ \ mod \ n \end{aligned}$$

$' \leftarrow '$means sampled from

Homomorphic Encryption - Paillier

Paillier Cryptosystem : Example

Let p = 11, q= 17.
Thus, $n = 11 * 17 = 187$ and $n^2 = 34969$.
$\varphi(n) = 80$.

$m = 175, r = 83 \in \mathbb{Z}^*_{187}$.

The ciphertext is $c = (1 + 187)^{175}. 83^{187} mod\ 34969 = 23911$

The decryption is then :

$$\hat{m} = \left[\frac{(23911^{80}\ mod\ 34969) - 1}{187}\right] . [80^{-1}\ mod\ 187\ ]\ mod\ 187$$

$$= 175 = m$$

Example from J. Katz, Y.Lindell, "Introduction to Modern Cryptography: Principles and Protocols," Chapman & Hall/CRC, 2007 p.414

Paillier Cryptosystem

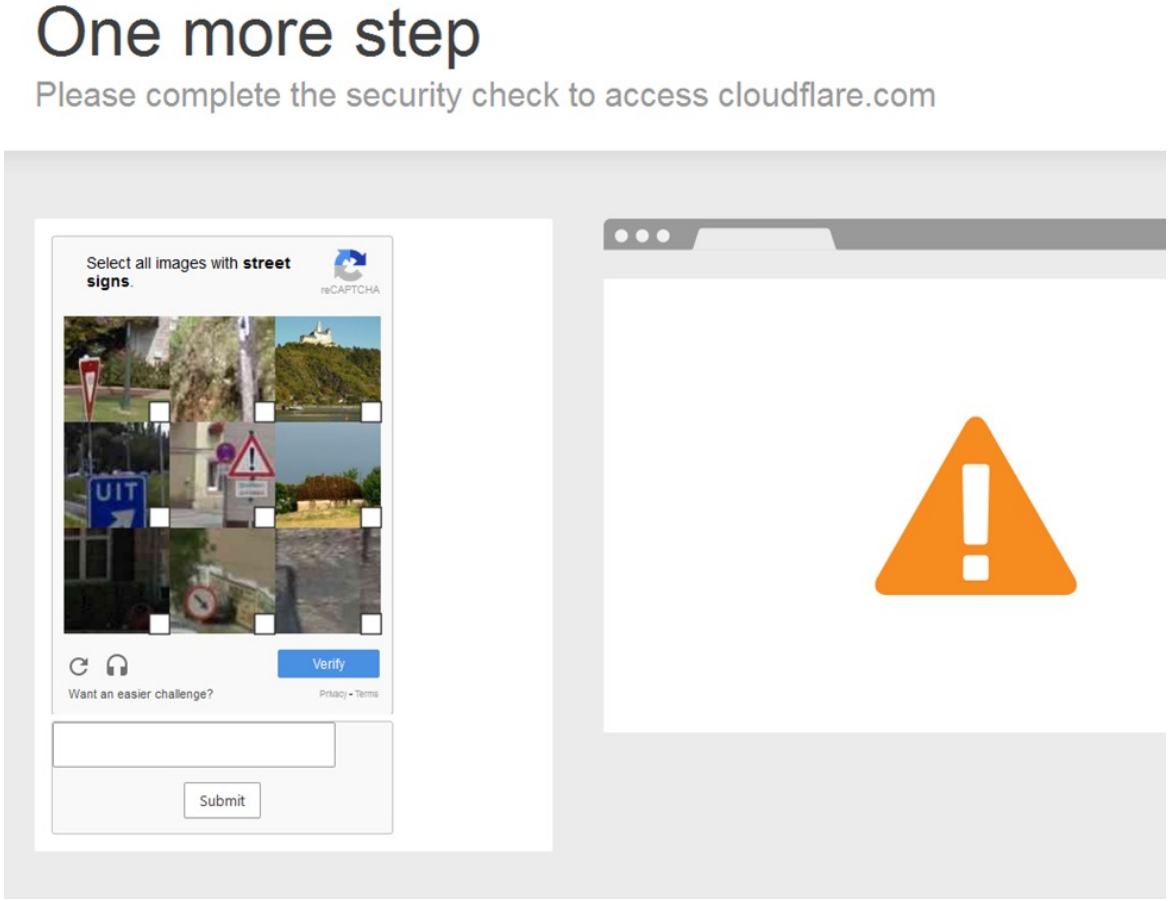
Exercise!
Check homomorphic properties, e.g., addition of two encrypted numbers

1.2 Attribute-Based Credentials

Browsing website using Tor

When browsing websites using Tor, users frequently have to solve captcha's?

Why?

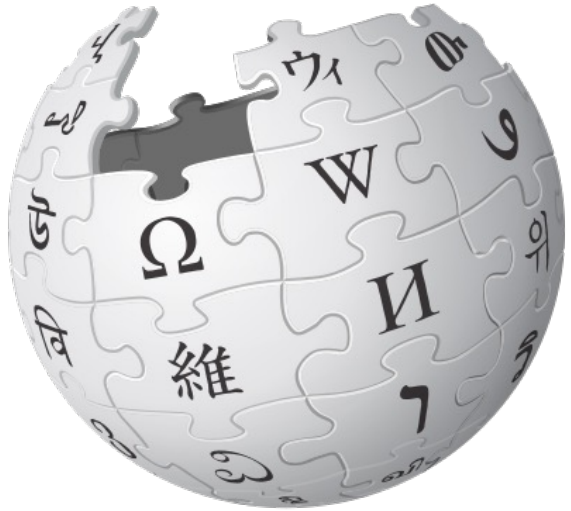


Editing Wikipedia via Tor

Wikipedia does not allow:

- Editing pages when using the Tor Browser
- Create accounts when using the Tor Browser

Why?



WIKIPEDIA
The Free Encyclopedia

How do you solve this normally

Authentication

- Username and password (everybody)
- Biometrics
- Client certificates (some rare websites)
- Challenge response with public key cryptography (ssh)

All of these *identify* the user. Is this always necessary?

What you really want: authorization

Check that this user

- is a real person and not a bot
- is an honest editor (Wikipedia)
- paid for this service (video streaming, music, games, etc.)
- is old enough to access this service

None of these require *identification*.

Passports: issuers and verifiers

Why do you trust the data in a passport?

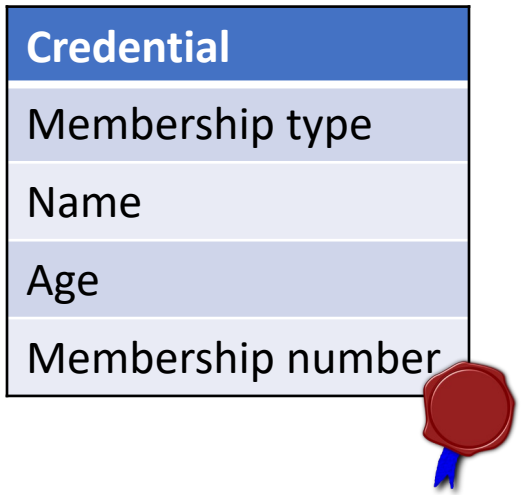
- Issued by a reliable country
- The passport looks authentic (i.e., it is not fake)

Issuer or Identity Provider (IdP): entity that validates the information in the documents

Verifier or Service Provider (SP): entity that verifies that the information is correct and provides a service

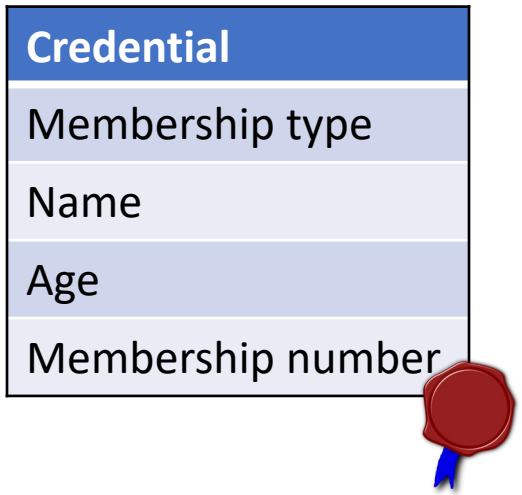
Attribute-based credentials (ABC)

- Digital variant of passports, driver's license, etc.
- Also known as anonymous credentials
- As opposed to tokens, can contain other attributes
- Attributes are encoded as numbers, may represent, e.g.,
  - Membership status (normal user, premium user)
  - Name
  - Age
  - Social security number
  - Random identifiers
  - Application-specific identifiers

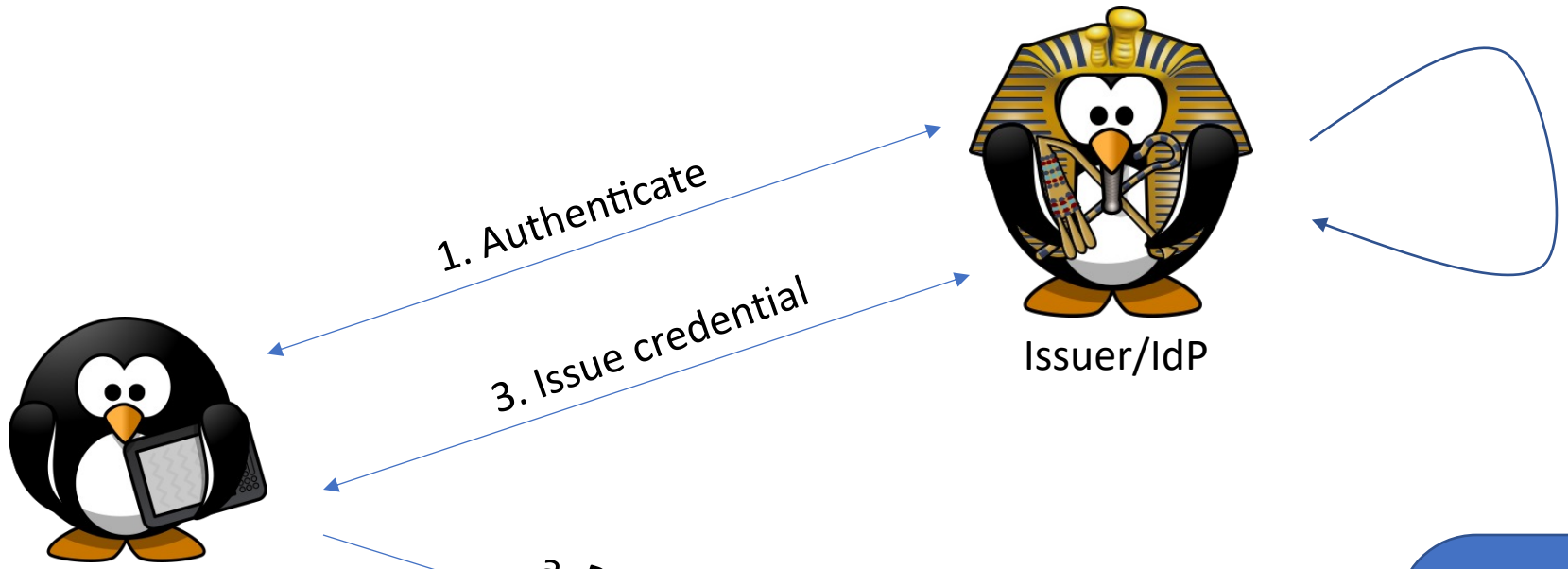


Attribute-based credentials (ABC)

- An authentication mechanism that allows to **selectively** authenticate different attributes of an entity
  - without revealing additional information about the entity (zero-knowledge)



Obtaining credentials and showing credentials



Credential
Secret key
Age
Membership number

What properties should these credentials satisfy?

Properties of attribute-based credentials

Unforgeability: only the issuer should be able to produce valid credentials.

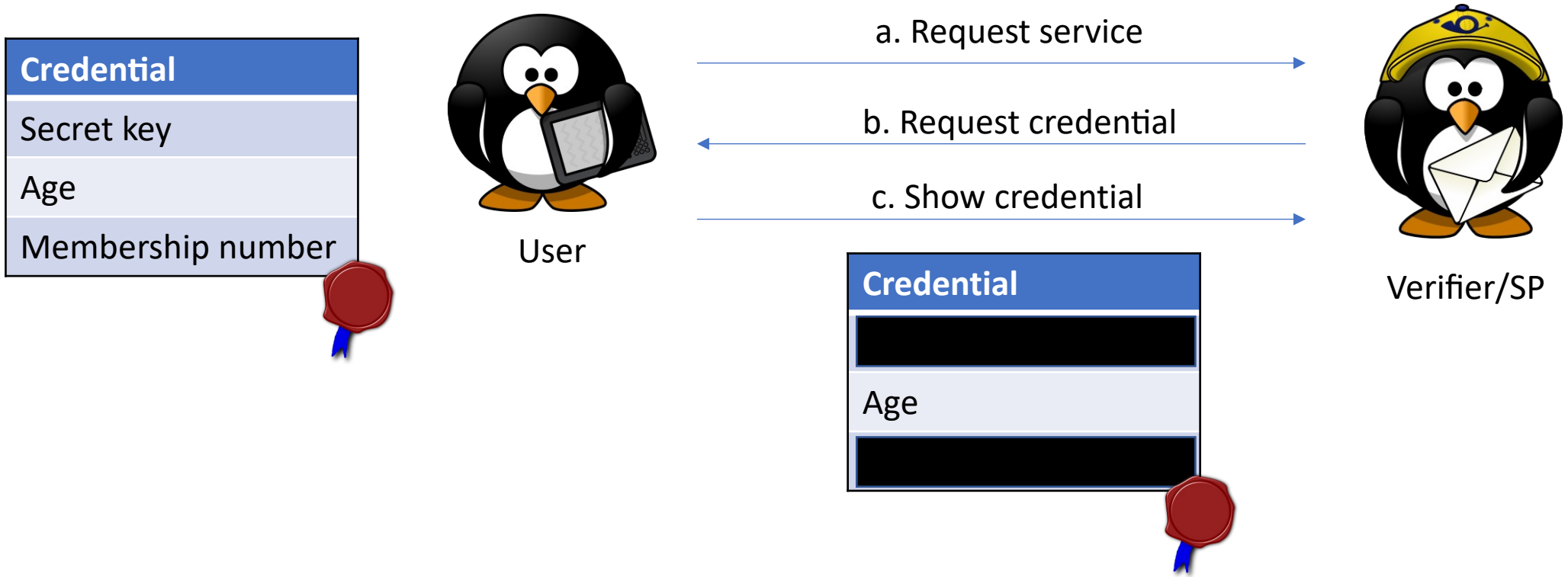
Selective disclosure: the user can hide irrelevant attributes

Issuer unlinkability: the issuer should not be able to recognize a credential that it previously issued

Verifier unlinkability: the verifier should not be able to link two consecutive showings of the same credential

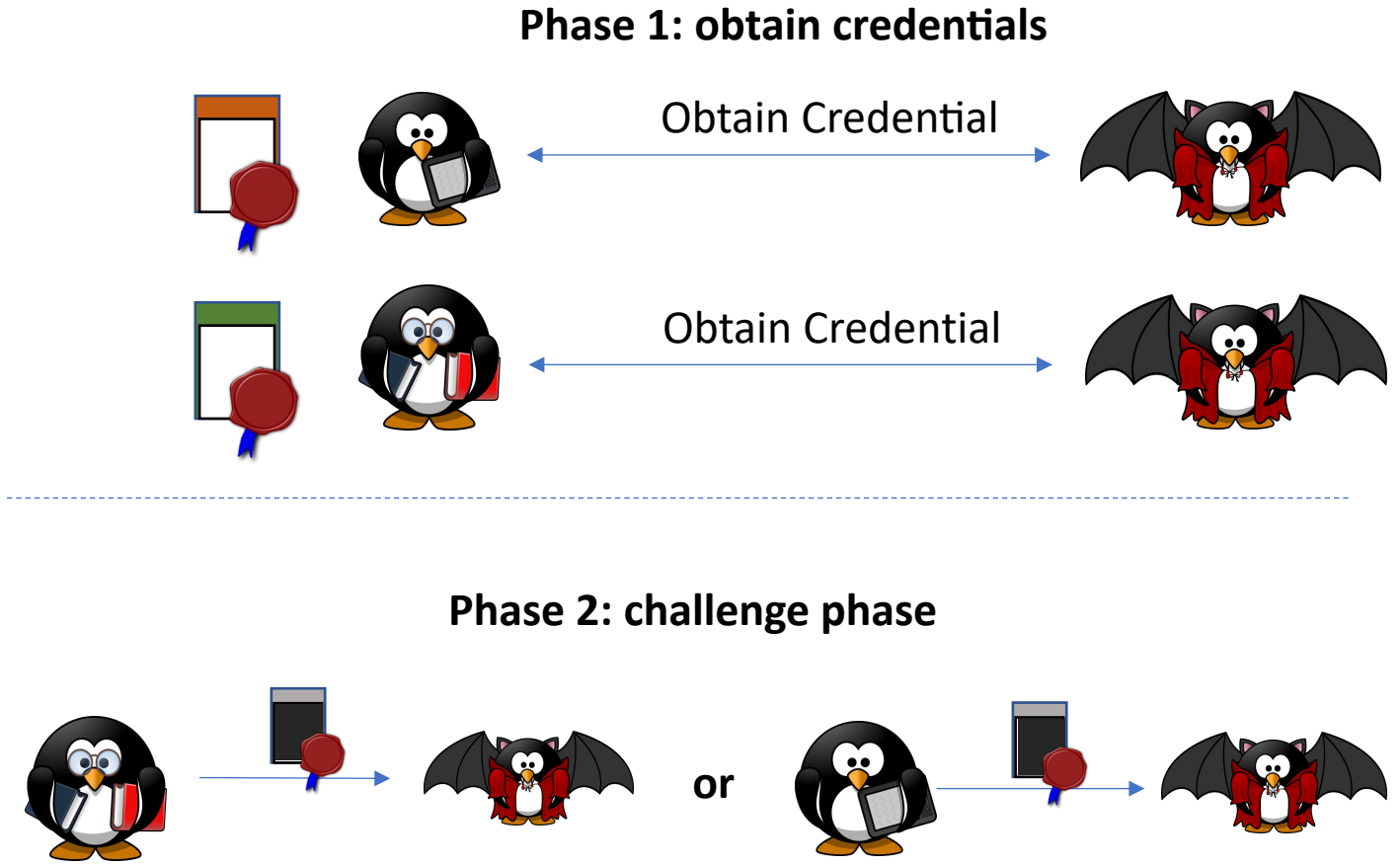
Selective disclosure

The user can hide irrelevant attributes. But the verifier can still check the validity of the credential.



Property: issuer unlinkability

- The issuer should not be able to recognize a credential that it previously issued
- Modelled using an indistinguishability game.
- Phase 1: obtain credentials
- Phase 2: challenge phase, try to distinguish users

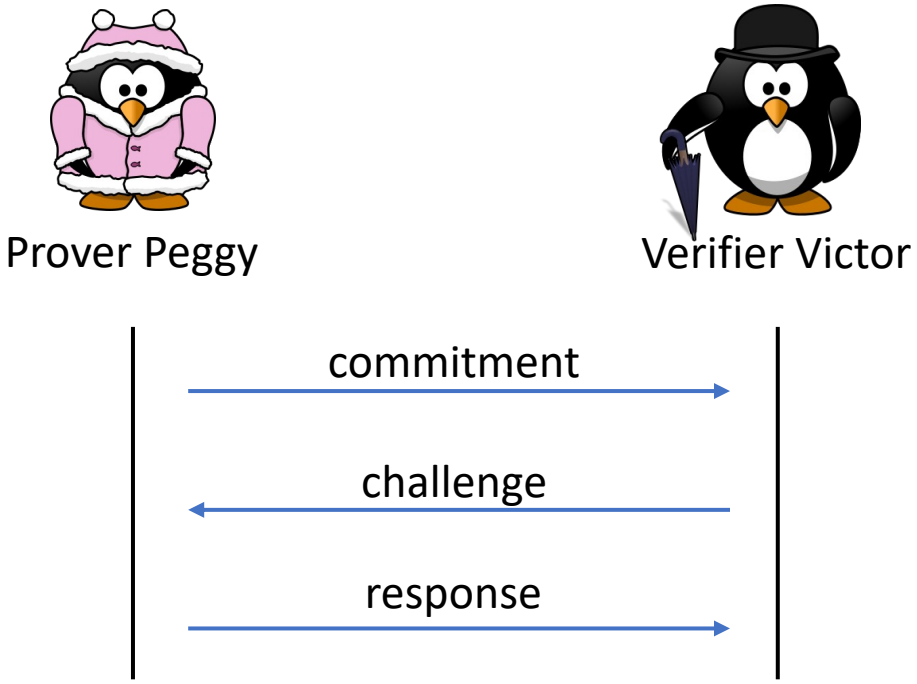


Note: both credentials should "look" the same, they should disclose the same attributes.

1.3 Zero-knowledge proofs

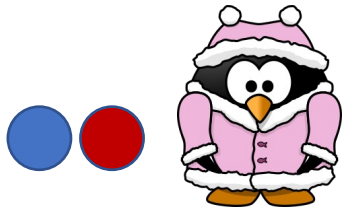
Zero-knowledge proof- definition

- Broadly speaking, a zero-knowledge proof allows a ***prover*** to convince a ***verifier*** of some facts on a private input without revealing this input.
- An interactive zero-knowledge proof protocol operates as a *challenge-and-response* protocol.



Zero-knowledge proof- example

- Consider Victor to be color-blind. Peggy wants to prove to Victor she is not color-blind.
- Peggy takes two balls of different colors indistinguishable in shape.
- She engages in an interactive zero-knowledge protocol with Victor.



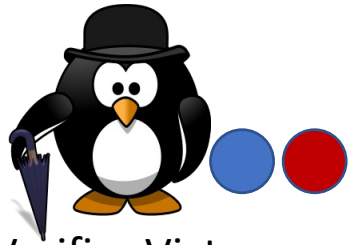
Prover Peggy

Verifier Victor

Zero-knowledge proof - example

- 1-Peggy sends the balls to Victor

Prover Peggy



Verifier Victor

- 2-Victor hides the balls and selects one at random, say the red one, and shows it to Peggy

Prover Peggy

Verifier Victor

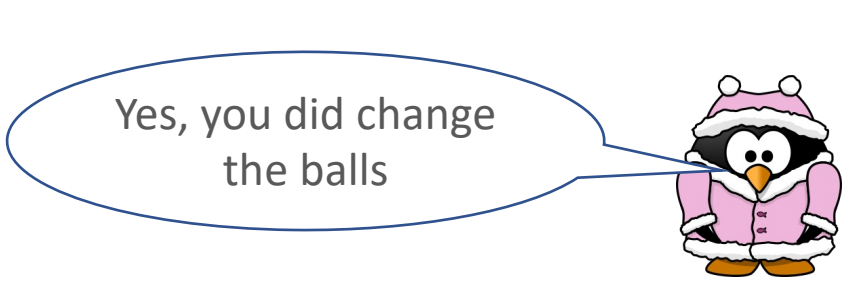
Zero-knowledge proof- example

- 3-Victor hides back the ball and picks a ball uniformly at random again

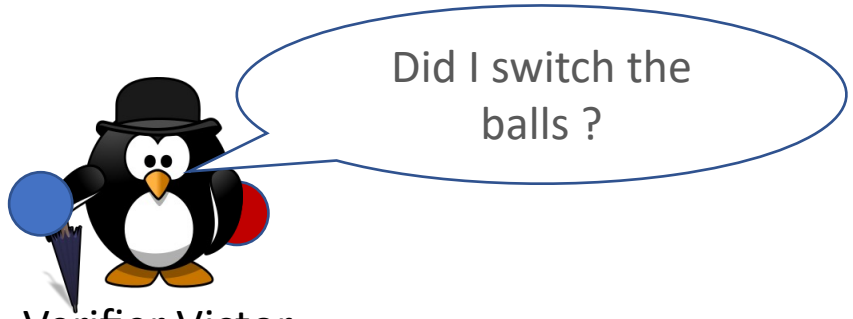
Prover Peggy

Verifier Victor

- 4-Victor shows this ball to Peggy and asks: “did I switch the balls ?”



Prover Peggy



Verifier Victor

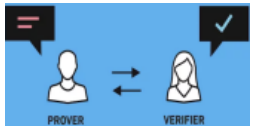
- 5 – Peggy answers truthfully to the challenge without stating the color

Zero-knowledge proof - example

- 6-Victor proceeds to steps 3 to 5 again picking a new ball at random

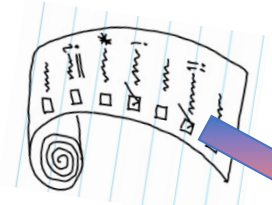
By iterating steps 3 to 5 multiple times, Victor gains confidence in Peggy's knowledge of the difference between blue and red without ever learning which ball is blue and which ball is red.

Zero-Knowledge Example



- DNA match against a database (zk-STARK, [BBHR, 2018])

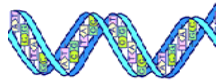
Police has a forensic DNA database



Public commitment 0x3b2a108a

"the sample whose hash is 0xe677d398 does not match anything in the database whose hash is 0x3b2a108a"

Presidential candidate has a DNA sample



Public commitment 0xe677d398

- Size-100,000 DB, proving in ~1 hour, verifying in milliseconds

Courtesy Shai Halevi

Properties of zero-knowledge proofs

Completeness: If the statement is true, an honest prover can convince an honest verifier that the statement is true.

Soundness: If the statement is false, a cheating prover cannot convince an honest verifier with very high probability (i.e., very close to 1).

Zero-knowledge: If the statement is true, no verifier learns anything other than the fact that the statement is true.

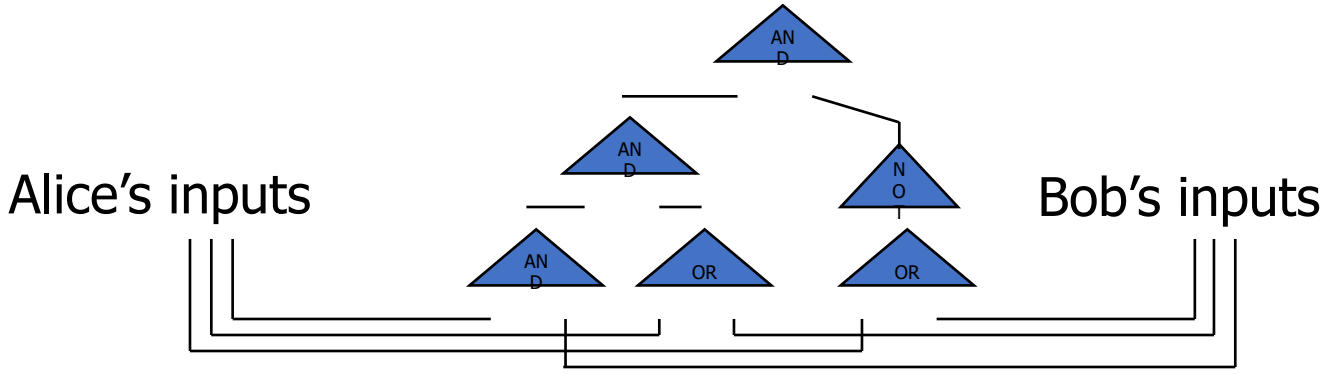
1.3 Secure Multiparty Computation

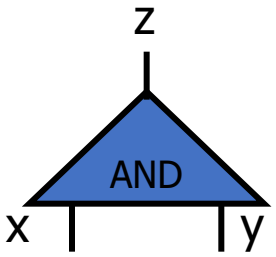
Garbled circuits

…

Garbled Circuits: Yao's Protocol

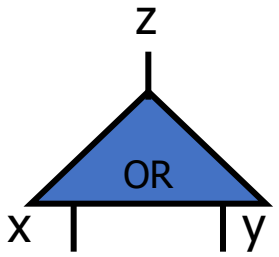
- Compute any function securely
  - … in the semi-honest model
- First, convert the function into a boolean circuit





Truth table:

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

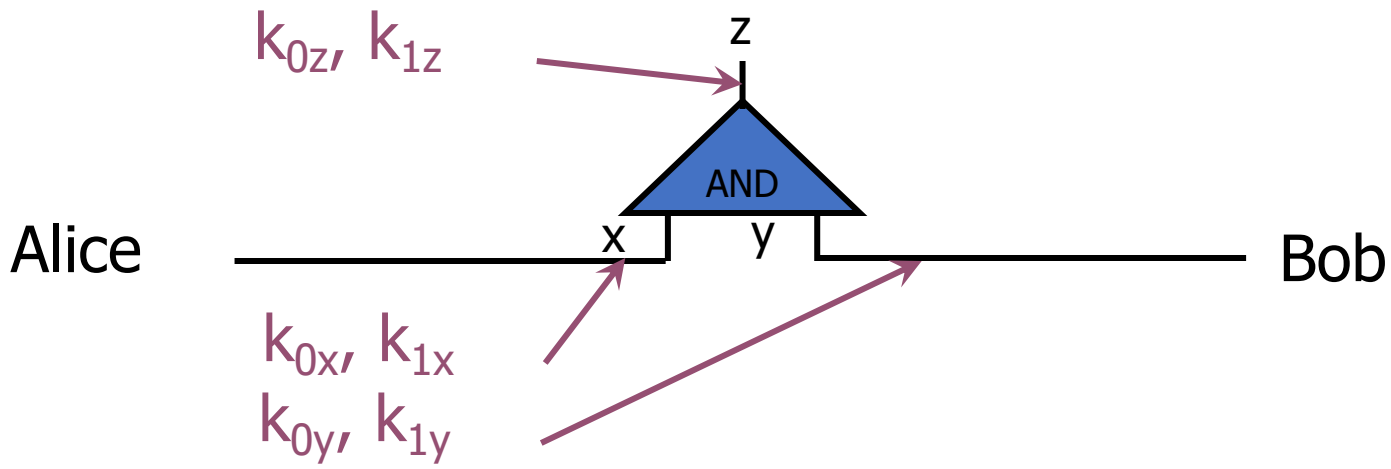


Truth table:

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

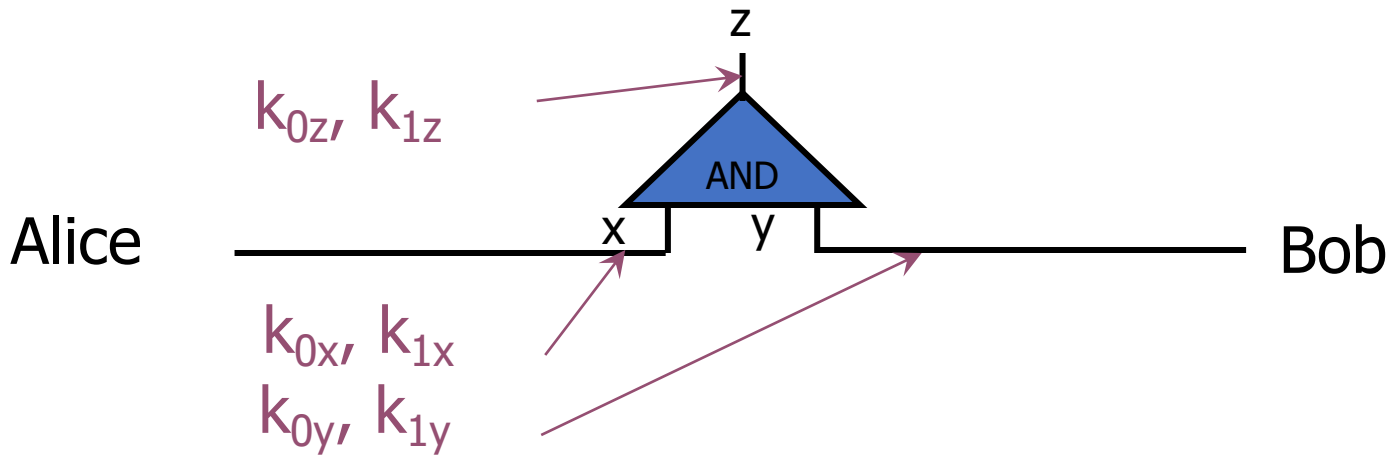
1: Pick Random Keys For Each Wire

- Next, evaluate one gate 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

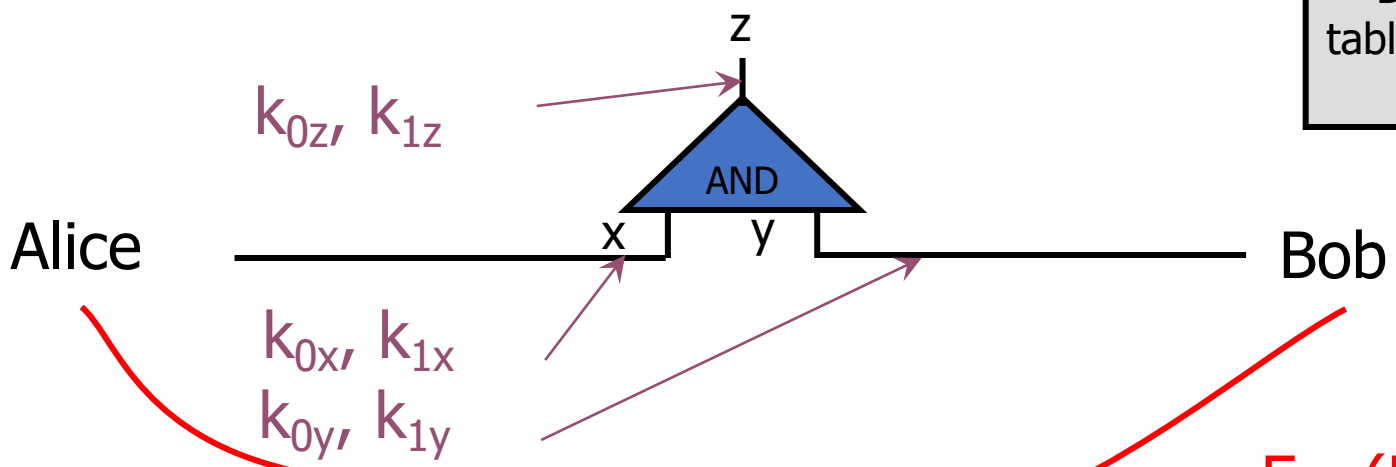
Encrypted truth table:

$E_{k0x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$

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



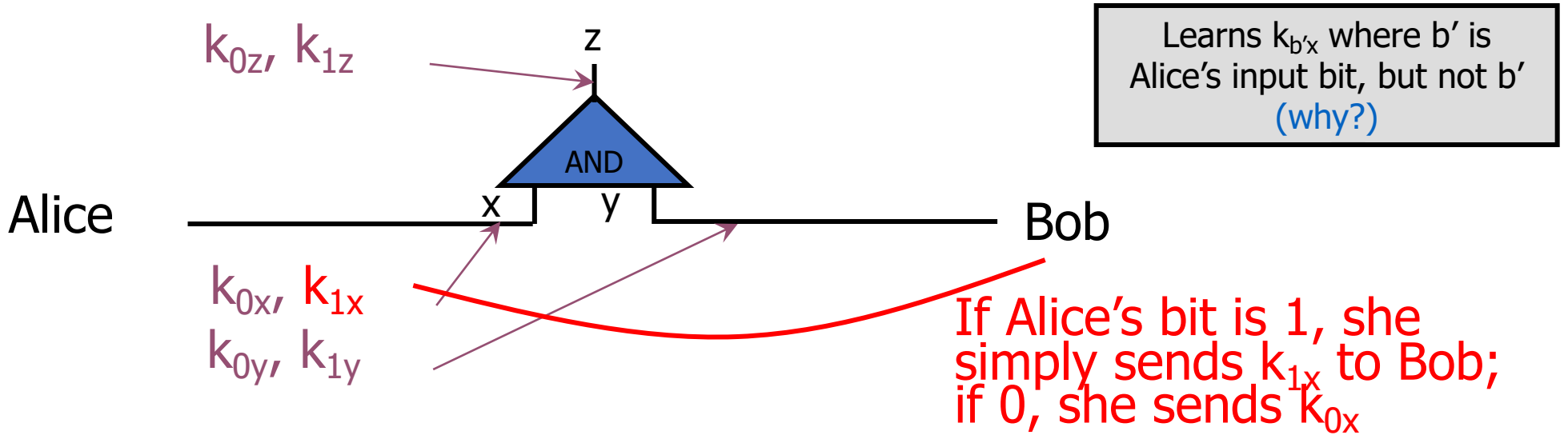
$E_{k0x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$

Garbled truth table:

$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$
$E_{k0x}(E_{k0y}(k_{0z}))$

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

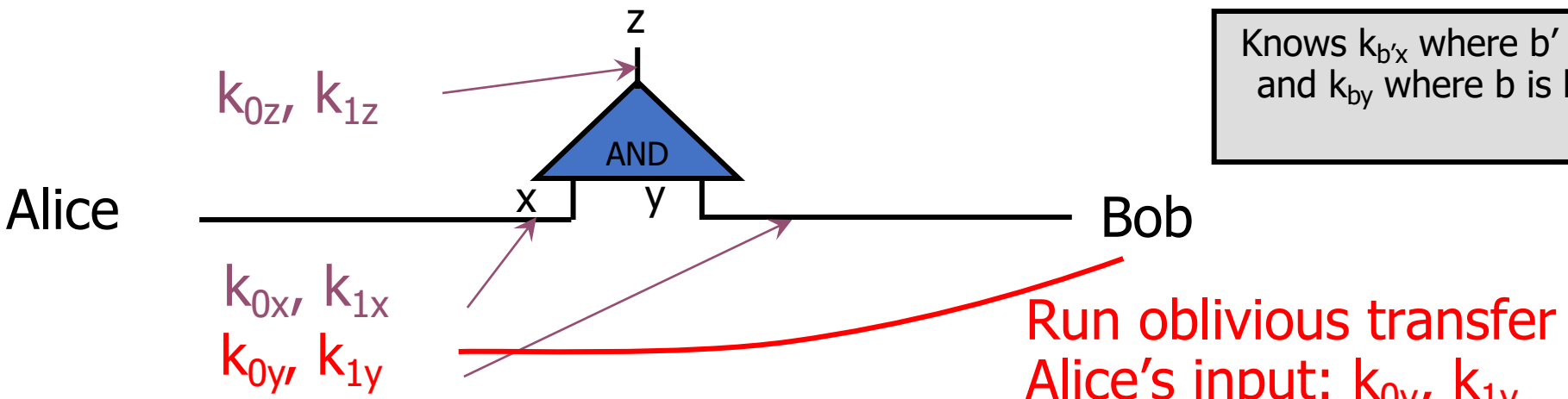


Garbled truth table:

$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$
$E_{k0x}(E_{k0y}(k_{0z}))$

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

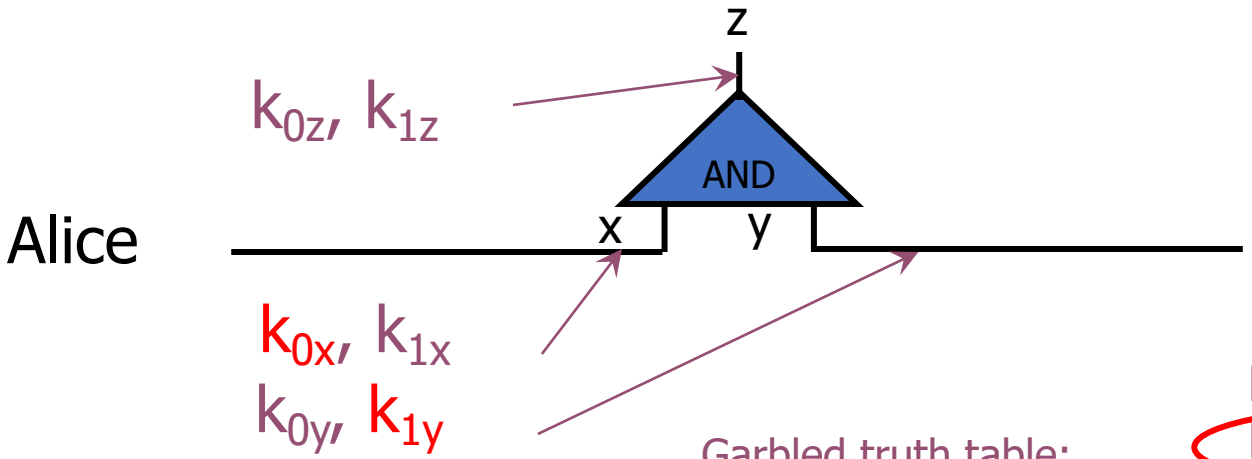


Knows $k_{b'x}$ where b' is Alice's input bit and k_{by} where b is his own input bit

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

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

Garbled truth table:

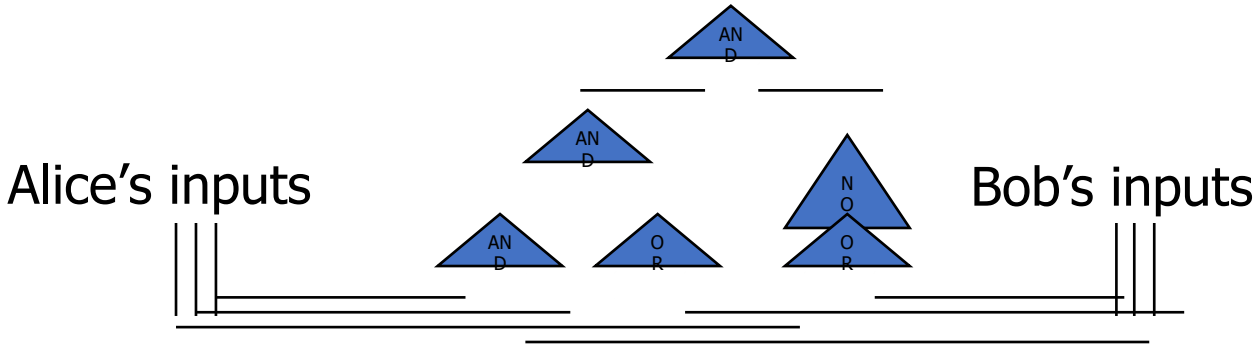
$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$
$E_{k0x}(E_{k0y}(k_{0z}))$

Suppose b'=0, b=1

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 not 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!

Cryptographic Mechanisms for Privacy Protection

- Anonymous communication
  - Tor
- Anonymous credentials
- Blind signatures
- Secure multiparty computation
  - Garbled circuits
  - Secret sharing
- Deterministic encryption
  - Order-preserving encryption
- Computing on encrypted data
  - Homomorphic encryption
- Oblivious RAM
- Private information retrieval
- Zero-knowledge proofs
- Etc.

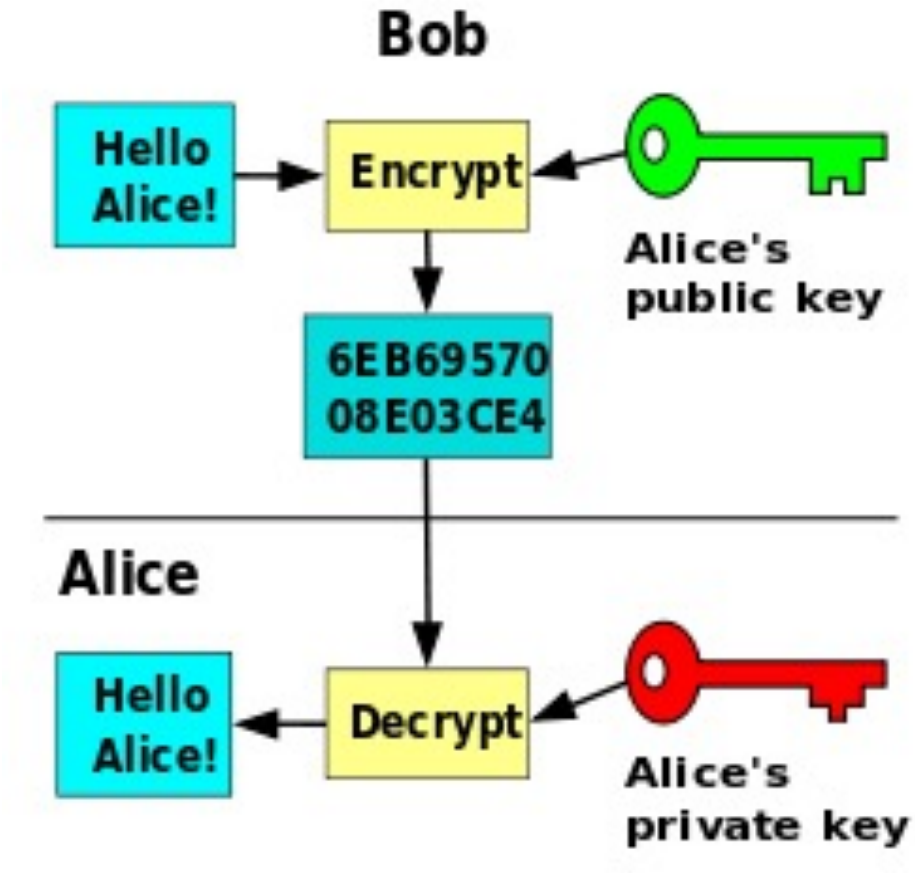
Steps to design a privacy-preserving system

To synthesize and apply acquired knowledge

Steps to Design A Privacy-Preserving System

- 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 ∈ [1, n^2/2]
- Strong secret:

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

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

Steps

- 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, requires security proofs?
- 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
- Collusions?

Requirements

- Decide what types of queries will be supported:
  - Weighted Average
  - Multiplication of ciphertexts
  - Division
  - Comparison/Classification
  - Etc..
- Access Control
- Access Patterns

Design

- Initialization

- Application(s)

Exercise: Let's Design A Privacy-Preserving System

For the above example, assume you have the following threat model and requirements:

- The parties are semi-honest, could be a malicious party in the wireless medium
- Supported functionality is addition and multiplication
- Collusions are not allowed
- Performance (in terms of computation time) is not an issue but the communication should be optimized
- The data on the cloud should be encrypted

What types of technologies you would use? Why?

More references

- Nigel Smart. Cryptography Made Simple. Springer 2016
- HE: http://homomorphicencryption.org/
- Zero-Knowledge: https://zkp.science/
- Secure-MPC: https://github.com/rdragos/awesome-mpc and http://www.multipartycomputation.com