# Secure computing: computing on encrypted or private data

## Introduction and case studies

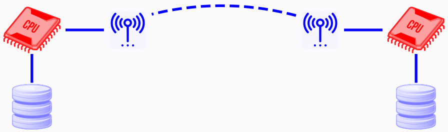
Xavier Leroy

2025-11-06

Collège de France, chair of Software sciences
xavier.leroy@college-de-france.fr

## Introduction

## Securing data using cryptography

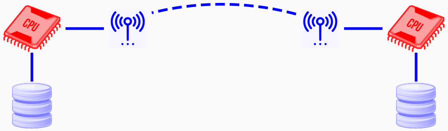


Strong cryptographic techniques to protect data

- at rest (file systems, databases);
- in transit (on communication networks).

Guarantees: confidentiality (encryption); integrity (signatures).

## Securing computations?



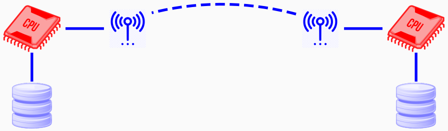
Usually, data is decrypted before computation.

Example: executing a query on an encrypted database.

Risk of information leaks during computation.

Need to give decryption keys to a third party that is not always trustworthy (cloud computing).

## Securing computations using cryptography!



"It is necessary to decrypt before computing" …Is it, really?

Recent cryptographic techniques:

- Homomorphic encryption: computing on encrypted data, without knowing the decryption key.
- Secure multi-party computation: more generally, computing on private data, without revealing it to other computers.

## Secure computing: system-based and language-based approaches (2022 course)

A few principles and many mechanisms:

- Hardware and software isolation of computations.
  (Virtualization, sandboxing, type and memory safety, ...)
- Access control.
  (Permissions, capabilities, passwords, ...)
- Information flow control.

Works well for integrity, not so well for confidentiality.

Vulnerable to low-level attacks
(leaking information via execution times, circumventing virtualization, breaking sandboxes, etc.)

## Secure computing: cryptographic approaches (this course)

Encrypted or masked data reveal no information to an attacker who does not have the corresponding key or mask.

If we are able to compute on encrypted or masked data, they can safely leak during computation, as long as the key remains secret.

## Secure computing: cryptographic approaches (this course)

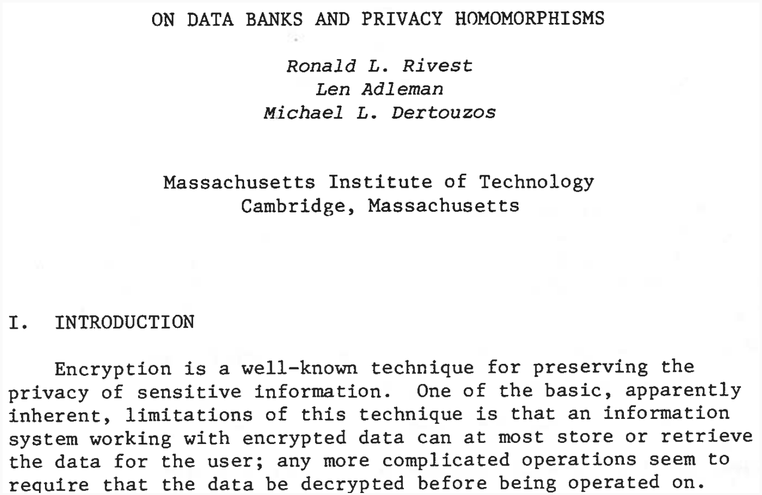
Encrypted or masked data reveal no information to an attacker who does not have the corresponding key or mask.

If we are able to compute on encrypted or masked data, they can safely leak during computation, as long as the key remains secret.

$\rightarrow$ The security of a small piece of data (key, mask, ...) guarantees the security of a large computation.

(An extension of Kerckhoff's principle: "the security of a cryptosystem must lie in the choice of its keys only".)

## *On data banks and privacy homomorphisms* (1978)



ON DATA BANKS AND PRIVACY HOMOMORPHISMS

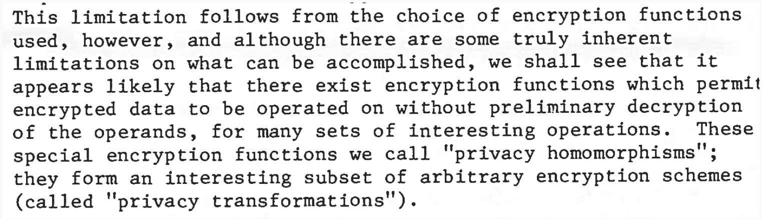
*Ronald L. Rivest*
*Len Adleman*
*Michael L. Dertouzos*

Massachusetts Institute of Technology
Cambridge, Massachusetts

I. INTRODUCTION

Encryption is a well-known technique for preserving the privacy of sensitive information. One of the basic, apparently inherent, limitations of this technique is that an information system working with encrypted data can at most store or retrieve the data for the user; any more complicated operations seem to require that the data be decrypted before being operated on.

## *On data banks and privacy homomorphisms* (1978)


This limitation follows from the choice of encryption functions used, however, and although there are some truly inherent limitations on what can be accomplished, we shall see that it appears likely that there exist encryption functions which permi encrypted data to be operated on without preliminary decryption of the operands, for many sets of interesting operations. These special encryption functions we call "privacy homomorphisms"; they form an interesting subset of arbitrary encryption schemes (called "privacy transformations").

## Homomorphic encryption

$$\begin{tikzcd}
m_1, \dots, m_n \arrow[r, "\mathcal{E}"] \arrow[d, "F"'] & c_1, \dots, c_n \arrow[d, "\hat{F}"] \\
m \arrow[r, shift left, "\mathcal{E}"] & c \arrow[l, shift left, "\mathcal{D}"]
\end{tikzcd}$$

Consider $F(m_1, \dots, m_n)$ a function over cleartext data.

Can we find a cipher $(\mathcal{E}, \mathcal{D})$ and a function $\hat{F}(c_1, \dots, c_n)$ over ciphertexts such that

$$\hat{F}(\mathcal{E}(m_1), \dots, \mathcal{E}(m_n)) = \mathcal{E}(F(m_1, \dots, m_n)) \qquad ?$$

If so, we can ask a third-party to compute $F(m_1, \dots, m_n)$ while keeping the $m_i$ private:

$$F(m_1, \dots, m_n) = \mathcal{D}(\hat{F}(\mathcal{E}(m_1), \dots, \mathcal{E}(m_n)))$$

## RSA is homomorphic for multiplication

RSA cipher: ($e, N$ public key; $d$ private key)

$$\mathcal{E}(m) \overset{def}{=} m^e \bmod N \qquad \mathcal{D}(c) \overset{def}{=} c^d \bmod N$$

If $m_1, m_2$ are two plaintext messages,

$$\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = m_1^e \cdot m_2^e = (m_1 \cdot m_2)^e = \mathcal{E}(m_1 \cdot m_2) \pmod{N}$$

Multiplication modulo $N$ can be performed homomorphically.

($\rightarrow$ lecture #2 for more examples)

## Secure Multi-Party Computation (MPC)

Distributed computing on secret data coming from *n* participants:

- Each participant $i$ has a secret $x_i$.
- The participants cooperate to compute $y = F(x_1, \dots, x_n)$.
- The result $y$ is revealed to all.
- Each participant $i$ learns nothing about $x_j$ ($j \neq i$) that it cannot deduce from $y$ and $x_i$.

## Example: evaluating bids for a call for tenders

Using a trusted third party:

- Each participant $i$ sends their bid $x_i$ to the third party.
- The third party determines $j$ such that $x_j = \min(x_1, \ldots, x_n)$ and announces $j$.
- The participants learn that $j$ is the lowest bidder.
- The participants learn nothing else about the bids of the other participants.

## Example: evaluating bids for a call for tenders

Using a trusted third party:

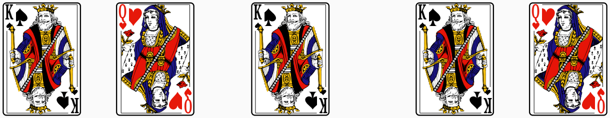
- Each participant $i$ sends their bid $x_i$ to the third party.
- The third party determines $j$ such that $x_j = \min(x_1, \ldots, x_n)$ and announces $j$.
- The participants learn that $j$ is the lowest bidder.
- The participants learn nothing else about the bids of the other participants.

Can we distribute this computation among the participants, without involving a trusted third party and without revealing the secrets $x_i$?

## Example: two-party computation of a Boolean “and”

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says “yes” then the other says “no”.

A protocol with 5 cards: 3 Kings, 2 Queens.

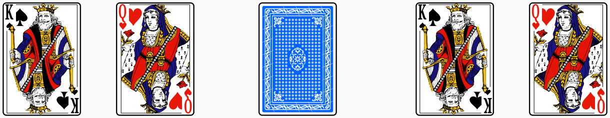


A King is played (face down).

## Example: two-party computation of a Boolean "and"

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says "yes" then the other says "no".

A protocol with 5 cards: 3 Kings, 2 Queens.

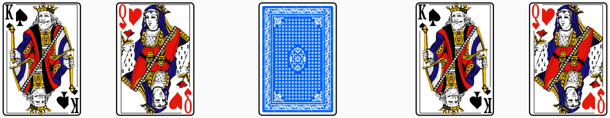


A King is played (face down).

## Example: two-party computation of a Boolean “and”

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says “yes” then the other says “no”.

A protocol with 5 cards: 3 Kings, 2 Queens.



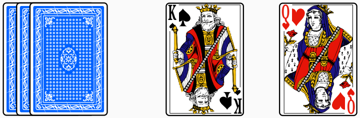
A King is played (face down).

Alice plays King then Queen if “yes”, Queen then King if “no”.

## Example: two-party computation of a Boolean "and"

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says "yes" then the other says "no".

A protocol with 5 cards: 3 Kings, 2 Queens.



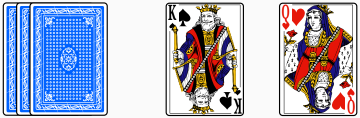
A King is played (face down).

Alice plays King then Queen if "yes", Queen then King if "no".

## Example: two-party computation of a Boolean "and"

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says "yes" then the other says "no".

A protocol with 5 cards: 3 Kings, 2 Queens.



A King is played (face down).

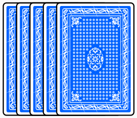
Alice plays King then Queen if "yes", Queen then King if "no".

Bob plays Queen then King if "yes", King then Queen if "no".

## Example: two-party computation of a Boolean “and”

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says “yes” then the other says “no”.

A protocol with 5 cards: 3 Kings, 2 Queens.



A King is played (face down).

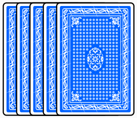
Alice plays King then Queen if “yes”, Queen then King if “no”.

Bob plays Queen then King if “yes”, King then Queen if “no”.

## Example: two-party computation of a Boolean “and”

Alice and Bob discuss having a second date, and would like to avoid the embarrassing situation where one says “yes” then the other says “no”.

A protocol with 5 cards: 3 Kings, 2 Queens.



A King is played (face down).

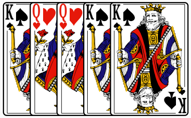
Alice plays King then Queen if “yes”, Queen then King if “no”.

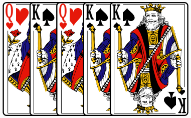
Bob plays Queen then King if “yes”, King then Queen if “no”.

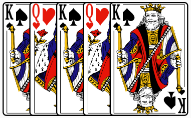
They cut the deck of cards and turn it over.

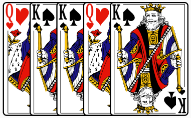
## Two-party computation of a Boolean “and”

Up to a rotation (circular permutation) of cards, we have 4 possible configurations:


yes / yes


yes / no


no / yes


no / no

The result is “yes” if and only if the two Queens are adjacent up to rotation.

The three configurations yes/no, no/yes, no/no are equal up to rotation. The “no” result does not reveal who voted “no”.

## Course outline

## Lectures 2 and 3: Homomorphic encryption: computing on encrypted data

Weakly homomorphic encryption (for one operation).

Somewhat homomorphic encryption (for $+$ and $\times$ but limited).

Fully homomorphic encryption via bootstrapping.

The LWE and RLWE problems and their uses.

Some techniques from the BGV and TFHE scheme.

For more information:

CKKS $\rightarrow$ Damien Stehlé's seminar, 20 nov. 2025

TFHE $\rightarrow$ Ilaria Chillotti's seminar, 27 nov. 2025.

## Lectures 4 and 5: Secure multi-party computation: computing on private data

Secret sharing: additive, Shamir's, linear.

Distributed computing on shared secrets.

Resistance to active attacks.

Yao's garbled circuits.

Oblivious transfer.

For more information: Geoffroy Couteau's seminar, 4 déc. 2025.

## Lecture 6: Zero-knowledge proofs

Interactive proofs. “Sigma” protocols.

Examples of ZK proofs for specific problems.

Non-interactive proofs.

A generic construction of SNARK proofs
(succint non-interactive argument of knowledge).

For more information: Michele Orrù’s seminar, 11 dec. 2025.

## Lecture 7:
## Secure computing: new directions and conclusions

Random-access memory: oblivious RAM, homomorphic RAM.

Indistinguishable obfuscation.

Conclusions.

See also: David Pointcheval’s seminar on functional encryption, 18 dec. 2025.

## Seminar talks

13/11 Pierrick Gaudry (CNRS)
*Outils cryptographiques pour le vote électronique.*

20/11 Damien Stehlé (CryptoLab)
*Chiffrement totalement homomorphe CKKS.*

27/11 Ilaria Chillotti (DESILO Inc)
*Chiffrement totalement homomorphe : panorama, applications et nouvelles directions.*

04/12 Geoffroy Couteau (CNRS)
*Calcul sécurisé et aléa corrélé, de la théorie à la pratique.*

11/12 Michele Orrù (CNRS)
*Des preuves zero-knowledge à l'anonymat en ligne.*

18/12 David Pointcheval (Cosmian)
*Le chiffrement fonctionnel : agréger des données sensibles.*

## Case study: secure multi-party computation of an average

## Computing the average salary

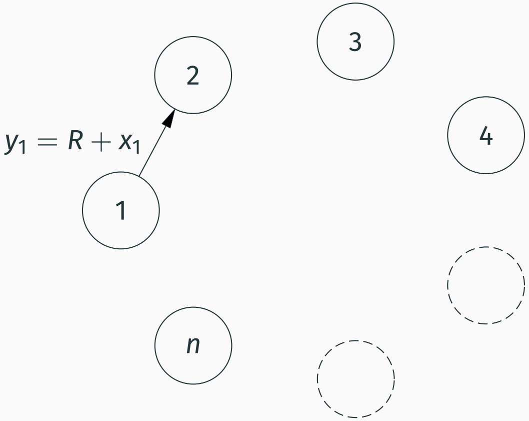
$n$ persons wish to compute the average of their salaries,
and share this average between all participants,
without revealing their salaries.

A solution involving a trusted third party (TTP):

- Each participant sends (confidentially) their salary to the TTP.
- The TTP computes the average and announces it.
- The TTP forgets all the information it received.

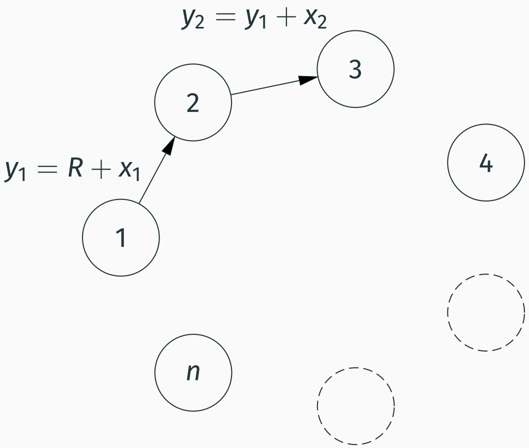
Can we obtain the same result with the same security guarantees without a trusted third party?

## Multi-party computation of the average of $x_1, \ldots, x_n$



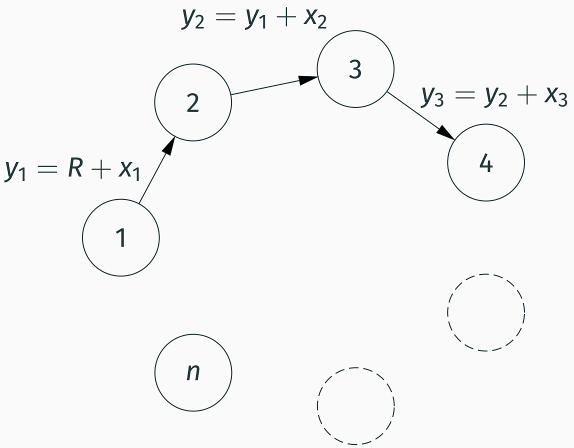
Participant 1 draws a random number $R \gg x_1, \ldots, x_n$ and sends $R + x_1$ to participant 2. (Blinding $x_1$ by $R$.)

## Multi-party computation of the average of $x_1, \ldots, x_n$



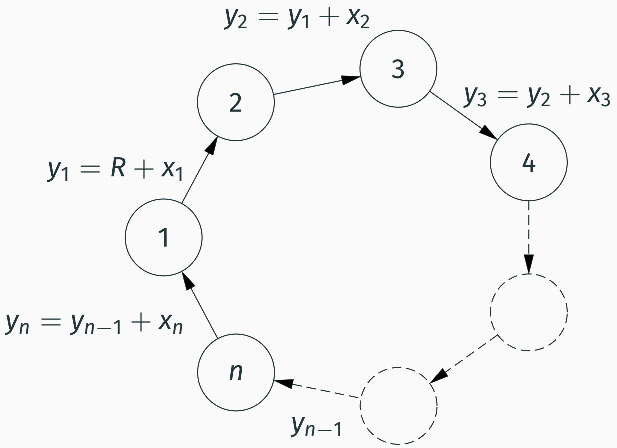
Each participant $i = 2, \ldots, n$ receives $y_{i-1}$, adds $x_i$, and sends this sum to the next participant $i + 1$.

## Multi-party computation of the average of $x_1, \ldots, x_n$



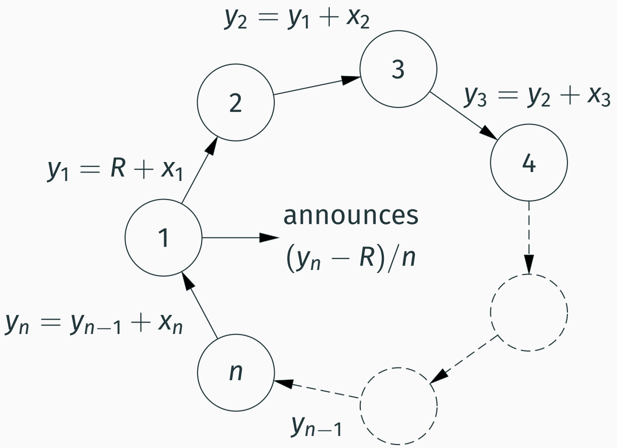
Each participant $i = 2, \ldots, n$ receives $y_{i-1}$, adds $x_i$, and sends this sum to the next participant $i + 1$.

## Multi-party computation of the average of $x_1, \dots, x_n$



Each participant $i = 2, \dots, n$ receives $y_{i-1}$, adds $x_i$, and sends this sum to the next participant $i + 1$.

## Multi-party computation of the average of $x_1, \dots, x_n$



Participant $n$ receives $y_n$, computes $(y_n - R)/n$, (un-blinding) and announces this result.

## Multi-party computation of the average of $x_1, \ldots, x_n$

$$y_1 = R + x_1 \qquad R \text{ random}, R \gg x_1, \ldots, x_n$$
$$y_i = y_{i-1} + x_i \qquad \text{for } i = 2, \ldots, n$$

**Correctness:**

$$y_i = R + x_1 + \cdots + x_i$$

therefore

$$\frac{y_n - R}{n} = \frac{x_1 + \cdots + x_n}{n}$$

## Multi-party computation of the average of $x_1, \ldots, x_n$

$$\begin{aligned} y_1 &= R + x_1 && R \text{ random, } R \gg x_1, \ldots, x_n \\ y_i &= y_{i-1} + x_i && \text{for } i = 2, \ldots, n \end{aligned}$$

**Confidentiality:** the number $y_i$ received by participant $i + 1$ has the shape "big random number + small number" (blinding), thus reveals nothing on the "small number" $x_1 + \ldots + x_i$, *a fortiori* nothing on each number $x_1, \ldots, x_i$.

The number $y_n$ received by participant 1 reveals $x_1 + \cdots + x_n$ but nothing more on each $x_i$.

## Some possible attacks

Passive attacks by listening to messages:

- An attacker who can intercept the messages $y_{i-1}$ and $y_i$ received / sent by participant $i$ can deduce $x_i = y_i - y_{i-1}$.

Counter-measure: use encrypted communication channels.

## Some possible attacks

Passive attacks by listening to messages:

- An attacker who can intercept the messages $y_{i-1}$ and $y_i$ received / sent by participant $i$ can deduce $x_i = y_i - y_{i-1}$.

Counter-measure: use encrypted communication channels.

Active attacks by malicious participants:

- Collusion between participants $i$ and $i + 2$:
  $P(i)$ send 0 to $P(i + 1)$; $\quad P(i + 2)$ receives $x_{i+1}$.
- Malicious choice of $R$ by participant 1.
  E.g. $R$ big enough to trigger overflow and a crash of $P(2)$.

Counter-measure: defensive programming of each participant.

## Extensions

This algorithm can be used for secure multi-party computation of $\sum f(x_i)$ and $\prod f(x_i)$ for any function $f$. This includes:

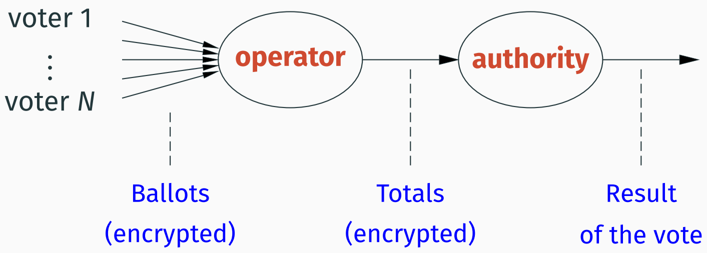
- Averages: arithmetic, harmonic, geometric, of order $p$, with or without weights. $\sqrt[p]{\frac{1}{n}\sum x_i^p}$
- Variance and standard deviation. $\sum x_i^2 - (\sum x_i)^2$
- An approximation of the maximum: order-$p$ average when $p$ is large.

Unable to compute the index $i$ of the largest $x_i$.

(Yao's "millionaire problem", see lecture #5)

## Case study:
## counting electronic votes

## An example of electronic voting



The vote authority has the decryption key but no access to individual ballots.

The vote operator collects the ballots but does not have the decryption key

$\rightarrow$ the totals are computed using homomorphic encryption.

## A naive implementation of this protocol

Using the RSA cipher (homomorphic for multiplication).

Voter $i$ encodes their vote $v_i$ as

$$v_i = 2 \text{ for Alice} \quad v_i = 3 \text{ for Bob} \quad v_i = 1 \text{ for blank}$$

and encrypts their ballot: $b_i = \mathcal{E}_{pk}(v_i)$.

The operator collects the ballots $b_i$ and computes the product
$T = b_1 \cdots b_n \pmod{N}$. ($N$ = modulus of the RSA key)

The authority decrypts $T$ and factorizes:

$$\mathcal{D}_{sk}(T) = 2^A \cdot 3^B$$

$A$ is the number of votes for Alice, $B$ the number of votes for Bob.

## Many problems in this implementation

A voter can send multiple ballots.

The operator can add ballots. (“Ballot box stuffing”)

## Many problems in this implementation

A voter can send multiple ballots.
The operator can add ballots. (“Ballot box stuffing”)

Ballots are not secret: RSA is deterministic, hence the operator recovers $v_i$ by comparing $b_i$ with $\mathcal{E}_{pk}(1)$, $\mathcal{E}_{pk}(2)$ and $\mathcal{E}_{pk}(3)$.

## Many problems in this implementation

A voter can send multiple ballots.
The operator can add ballots. (“Ballot box stuffing”)

Ballots are not secret: RSA is deterministic, hence the operator recovers $v_i$ by comparing $b_i$ with $\mathcal{E}_{pk}(1)$, $\mathcal{E}_{pk}(2)$ and $\mathcal{E}_{pk}(3)$.

A voter can “stuff” their ballot, e.g.
$v_i = 4$ (two votes for Alice) or $v_i = 27$ (three votes for Bob).

## Many problems in this implementation

A voter can send multiple ballots.
The operator can add ballots. (“Ballot box stuffing”)

Ballots are not secret: RSA is deterministic, hence the operator recovers $v_i$ by comparing $b_i$ with $\mathcal{E}_{pk}(1)$, $\mathcal{E}_{pk}(2)$ and $\mathcal{E}_{pk}(3)$.

A voter can “stuff” their ballot, e.g.
$v_i = 4$ (two votes for Alice) or $v_i = 27$ (three votes for Bob).

The counting of votes can easily overflow:

$$\mathcal{D}_{sk}(T) = v_1 \cdots v_n \bmod N \neq v_1 \cdots v_n \quad \text{if } v_1 \cdots v_n \geq N$$

## Some solutions

*A voter can send multiple ballots.*
*The operator can add ballots.*

Authentify the voters.
Have them sign their ballots cryptographically.
Make the list of voters public.

## Some solutions

*A voter can send multiple ballots.*
*The operator can add ballots.*

Authentify the voters.
Have them sign their ballots cryptographically.
Make the list of voters public.

*Ballots are not secret: RSA is deterministic, hence the operator recovers $v_i$ by comparing $b_i$ with $\mathcal{E}_{pk}(1)$, $\mathcal{E}_{pk}(2)$ and $\mathcal{E}_{pk}(3)$.*

Use a cipher with randomized encryption, so that encrypting the same plaintext multiple times produces different ciphertexts.

## Quelques solutions

*Arithmetic overflow as soon as $v_1 \cdots v_n \geq N$.*

Use a cipher that is homomorphic for addition (of integers or tuples of integers) instead of multiplication.

$$\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \boxplus \mathcal{E}(m_2) \pmod N$$

Encode the votes as $(1, 0)$ = Alice, $(0, 1)$ = Bob, $(0, 0)$ = blank.

## The exponential ElGamal cipher (EEG)

A finite group $G$ of order $q$ generated by $g$.

Private key: $x \in \{1, \ldots, q-1\}$. Public key: $h \overset{def}{=} g^x$.

Randomized encryption:

$$\mathcal{E}(m) = (g^r, h^r \cdot g^m) \quad \text{with } r \in \{1, \ldots, q-1\} \text{ random}$$

Homomorphic for addition:

$$\begin{aligned}
\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{r_1} \cdot g^{r_2}, (h^{r_1} \cdot g^{m_1}) \cdot (h^{r_2} \cdot g^{m_2})) \\
&= (g^r, h^r \cdot g^{m_1+m_2}) \qquad (\text{with } r = r_1 + r_2 \bmod q) \\
&= \text{an encryption of } m_1 + m_2
\end{aligned}$$

($\rightarrow$ lecture #2)

## Some solutions

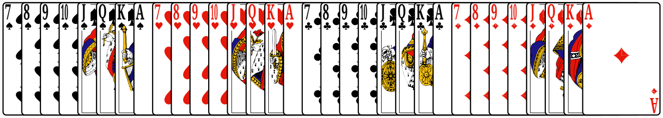
*A voter can stuff their ballot by encrypting an "impossible" value, e.g. 4 (two votes for Alice).*

Use zero-knowledge proofs (ZKP).

The voter provides a proof that their ballot is $\mathcal{E}_{pk}(v)$ for a value $v$ that is permitted by the election rules, without revealing the value $v$.

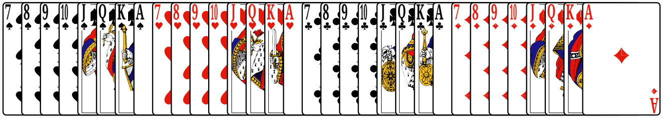
## An example of a zero-knowledge proof

Peggy (the prover) draws a card from a deck and wishes to convince Victor (the verifier) that it is a red card, without showing him the card.



## An example of a zero-knowledge proof

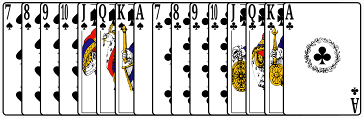
Peggy (the prover) draws a card from a deck and wishes to convince Victor (the verifier) that it is a red card, without showing him the card.

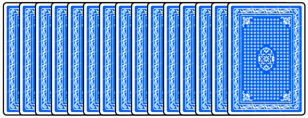


1. Peggy and Victor check the deck of cards:
16 red cards, 16 black cards.

## An example of a zero-knowledge proof

Peggy (the prover) draws a card from a deck and wishes to convince Victor (the verifier) that it is a red card, without showing him the card.

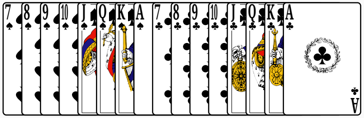


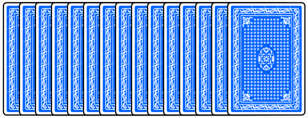


2. Peggy takes the cards, keeps one, puts the remaining cards face down, and turns 16 black cards over.

## An example of a zero-knowledge proof

Peggy (the prover) draws a card from a deck and wishes to convince Victor (the verifier) that it is a red card, without showing him the card.





3. Victor is convinced that the card kept by Peggy is red, but knowns nothing else about this card.

## Zero-knowledge proofs for EEG

**Proof that $(A, B)$ is an encryption of 0:** (Chaum-Pedersen)

i.e. that there exists $r$ such that $(A, B) = (g^r, h^r)$,
without revealing $r$.

Commitment: Peggy publishes another encryption of 0
$(A', B') = (g^u, h^u)$, while keeping $u$ secret.

Challenge: Victor sends a random integer $c$.

Response: Peggy sends $t = u + cr \pmod{q}$.

Verification: Victor checks that $g^t = A' \cdot A^c$ and $h^t = B' \cdot B^c$.

## Zero-knowledge proofs for EEG

**Proof that $(A, B)$ is an encryption of 0:** (Chaum-Pedersen)

i.e. that there exists $r$ such that $(A, B) = (g^r, h^r)$, without revealing $r$.

**Proof that $(A, B)$ is an encryption of 1:**

i.e. that there exists $r$ such that $(A, B) = (g^r, h^r \cdot g)$, without revealing $r$.

Use Chaum-Pedersen for $(A, B/g)$.

## Zero-knowledge proofs for EEG

**Proof that $(A, B)$ is an encryption of 0:** (Chaum-Pedersen)

i.e. that there exists $r$ such that $(A, B) = (g^r, h^r)$, without revealing $r$.

**Proof that $(A, B)$ is an encryption of 1:**

i.e. that there exists $r$ such that $(A, B) = (g^r, h^r \cdot g)$, without revealing $r$.

**Proof that $(A, B)$ is an encryption of 0 or 1:**

A generic construction: given two “Sigma” protocols, one for $P$ and one for $Q$, we obtain a “Sigma” protocol for $P \vee Q$.

($\rightarrow$ lecture #6)

## Case study:
## private set intersection

## The private set intersection problem (PSI)

The client has a set of names $C$.
The server has a set of names $S$.

The client wants to compute $C \cap S$
*i.e.* to learn which of its names are known to the server.

The server must learn nothing about $C$:

- neither the new names ($C \setminus S$)
- nor the shared names ($C \cap S$).

The client must learn nothing about $S \setminus C$.

## Applications of PSI

**Contact discovery on social networks:**

The client sends its contact list (phone numbers or emails).

The server responds with the list of contacts who are already members of the social network (and who accepted to be discoverable).

## Applications of PSI

**Contact discovery on social networks:**

**Detecting pirated accounts and passwords:**

The client sends its identifiers and passwords (hashed).

The server computes the intersection with lists of pirated accounts found on the Web.

## Applications of PSI

**Contact discovery on social networks:**

**Detecting pirated accounts and passwords:**

**Detecting illegal images:**

The client sends hashes of new photos and images.

The server finds those that appear in police databases of illegal images.

The client blocks illegal images.

## Unsatisfactory solutions

**The client sends $C$ in the clear.**

- The server learns $C$.

## Unsatisfactory solutions

**The client sends $C$ in the clear.**

- The server learns $C$.

**The client sends $C$ after hashing.**

- I.e. the client sends $\{H(c) \mid c \in C\}$ and the server computes the intersection with $\{H(s) \mid s \in S\}$.
- The server learns $C \cap S$.
- If the name space is small (e.g. phone numbers), the server can precompute $H(x)$ for all $x$, and learn all of $C$.

## Unsatisfactory solutions

### The client sends $C$ in the clear.

- The server learns $C$.

### The client sends $C$ after hashing.

- I.e. the client sends $\{H(c) \mid c \in C\}$ and the server computes the intersection with $\{H(s) \mid s \in S\}$.
- The server learns $C \cap S$.
- If the name space is small (e.g. phone numbers), the server can precompute $H(x)$ for all $x$, and learn all of $C$.

### The serveur sends a Bloom filter for $S$.

- Reveals too much information on $S$ to the client.
- Encrypted Bloom filters exist, but are costly.

## Homomorphic evaluation of a polynomial

The client constructs a polynomial $P$ whose roots are the elements $c_i$ of its set $C$:

$$P = (X - c_1)(X - c_2)\cdots(X - c_n) = \sum_{i=0}^{n} a_i X^i$$

Consider a cipher that is homomorphic for addition and multiplication by a constant (such as EEG or Paillier's cipher):

$$\mathcal{E}(x + x') = \mathcal{E}(x) \boxplus \mathcal{E}(x') \qquad \mathcal{E}(n \cdot x) = n \boxdot \mathcal{E}(x)$$

We can, then, evaluate the polynomial $P$ homomorphically:

$$\begin{aligned} & x^n \boxdot \mathcal{E}(a_n) \boxplus \cdots \boxplus x \boxdot \mathcal{E}(a_1) \boxplus \mathcal{E}(a_0) \\ = \;\; & \mathcal{E}(x^n \cdot a_n + \cdots + x \cdot a_1 + a_0) \;=\; \mathcal{E}(P(x)) \end{aligned}$$

## The protocol of Freedman, Nissim, and Pinkas (2004)

(*Efficient Private Matching and Set Intersection*, Eurocrypt 2004, LNCS 3027.)

1. The client draws a key pair $(pk, sk)$,
   constructs the polynomial $P = (X - c_1) \cdots (X - c_n) = \sum a_i X^i$,
   and sends $pk$ and the encrypted coefficients $\mathcal{E}(a_i)$ to the server.

## The protocol of Freedman, Nissim, and Pinkas (2004)

(*Efficient Private Matching and Set Intersection*, Eurocrypt 2004, LNCS 3027.)

1. The client draws a key pair $(pk, sk)$,
   constructs the polynomial $P = (X - c_1) \cdots (X - c_n) = \sum a_i X^i$,
   and sends $pk$ and the encrypted coefficients $\mathcal{E}(a_i)$ to the server.

2. For each $s_i \in S$, the server evaluates homomorphically

   $$y_i = \mathcal{E}(r_i \cdot P(s_i) + s_i) \qquad \text{with } r_i \text{ a large random integer}$$

   and sends all these ciphertexts $y_i$ to the client.

## The protocol of Freedman, Nissim, and Pinkas (2004)

(*Efficient Private Matching and Set Intersection*, Eurocrypt 2004, LNCS 3027.)

1. The client draws a key pair $(pk, sk)$,
   constructs the polynomial $P = (X - c_1) \cdots (X - c_n) = \sum a_i X^i$,
   and sends $pk$ and the encrypted coefficients $\mathcal{E}(a_i)$ to the server.

2. For each $s_i \in S$, the server evaluates homomorphically

   $$y_i = \mathcal{E}(r_i \cdot P(s_i) + s_i) \qquad \text{with } r_i \text{ a large random integer}$$

   and sends all these ciphertexts $y_i$ to the client.

3. The client decrypts the $y_i$ and keeps those that belong to $C$:

   $$C \cap S = \{\mathcal{D}(y_i) \mid \mathcal{D}(y_i) \in C\}$$

## The protocol of Freedman, Nissim, Pinkas (2004)

**Correctness.** The decryption $x_i = \mathcal{D}(y_i)$ has the form

$x_i = r_i \cdot P(s_i) + s_i \quad$ with $s_i \in S$ (unknown) and $r_i$ random (unknown)

If $s_i \in C$, we have $P(s_i) = 0$, therefore $x_i = s_i$ and the test $x_i \in C$ succeeds.

If $s_i \notin C$, we have $P(s_i) \neq 0$, therefore $x_i$ is random and the test $x_i \in C$ fails with very high probability.

## The protocol of Freedman, Nissim, Pinkas (2004)

**Correctness.** The decryption $x_i = \mathcal{D}(y_i)$ has the form

$x_i = r_i \cdot P(s_i) + s_i$ with $s_i \in S$ (unknown) and $r_i$ random (unknown)

If $s_i \in C$, we have $P(s_i) = 0$, therefore $x_i = s_i$ and the test $x_i \in C$ succeeds.

If $s_i \notin C$, we have $P(s_i) \neq 0$, therefore $x_i$ is random and the test $x_i \in C$ fails with very high probability.

**Confidentiality.** The server learns nothing about the $c_i$, since the coefficients of the polynomial $P$ are encrypted.

The client learns nothing on the $s_i \notin C$, since the corresponding messages $y_i$ decrypt to noise.

## Reducing the protocol complexity

The polynomial $P$ has degree $|C|$, hence its evaluation takes time $\mathcal{O}(|C|)$, and phase 2 of the protocol takes time $\mathcal{O}(|C| \cdot |S|)$.

The client can partition $C$ in $B$ buckets of size $\approx |C|/B$ using a (non-cryptographic) hash function of its choice.

The server receives the hash function and $B$ polynomials of degree $|C|/B$, hence phase 2 runs in time $\mathcal{O}(\frac{|C| \cdot |S|}{B})$.

Freedman *et al.* outline a hashing schema that remains secure with $B = |C| / \ln \ln |C|$, resulting in $\mathcal{O}(|S| \ln \ln |C|)$ complexity.

## References

## References

General background on cryptography:

- *Serious cryptography: A practical introduction to modern encryption*, Jean-Philippe Aumasson, No Starch Press, 2nd ed, 2024.
- *Cryptography made simple*, Nigel P. Smart, Springer, 2016.

On electronic voting:

- *Le vote électronique: les défis du secret et de la transparence*, Véronique Cortier et Pierrick Gaudry, Odile Jacob, 2022.

The PSI protocol described in this lecture:

- Michael J. Freedman, Kobbi Nissim and Benny Pinkas, “Efficient Private Matching and Set Intersection”, *Advances in cryptology – Eurocrypt 2004*, LNCS 3027, Springer, 2004.
  `https://doi.org/10.1007/978-3-540-24676-3_1`