Language-based software security, seventh lecture

Computing over encrypted or private data

Xavier Leroy

2022-04-21

Collège de France, chair of software sciences

`xavier.leroy@college-de-france.fr`

The cryptographic approach

A number of cryptographic primitives

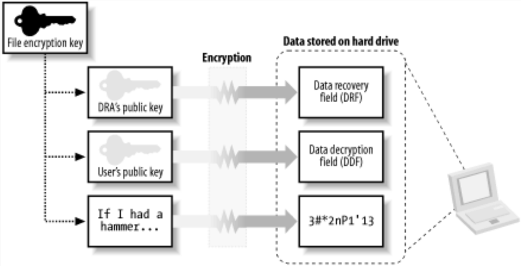
- encryption (symmetric or public-key)
- signature (public-key)
- hashes
- etc.

that we combine and apply

to guarantee confidentiality and integrity of information

at rest (storage) and in transit (networks).

Example: an encrypted file system



Encryption with a secret key, randomly-generated, itself encrypted with the passwords of authorized users.

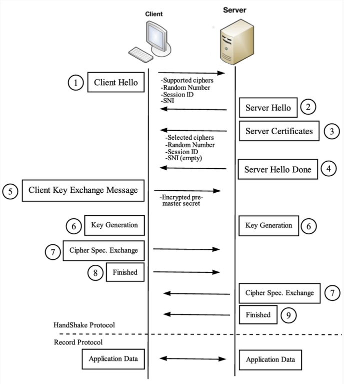
The best known protection against low-level attacks (stealing the computer, stealing the disk, booting another OS).

The best known way to erase instantaneously a lot of data.

Example: network protocols: TLS, SSH, Signal, …

Point-to-point communication over the Internet, with

- encryption and authentication of the messages (no eavesdropping, no packet injection, no packet replay);
- authentication of the server (no impersonation, no man-in-the-middle attack).



The only known protection against an attacker who controls parts of the network.

What about data during computation?

Programs usually operate over data in the clear.

Consider for example a database management system:
to perform queries, it "obviously needs" access to cleartext data.

However, it is difficult to guarantee confidentiality of data during computation:

- complex flows of information;
- indirect flows: time, caches, speculative execution, ...

In this lecture

Cryptographic solutions to the problem of computing over data while preserving its confidentiality.

Two detailed examples:

- Homomorphic encryption: to computer directly over encrypted data, without decrypting it.
- Secure multiparty computation: several participants compute jointly a function of their private data, without revealing their data to the others.

Homomorphic encryption

Homomorphic encryption

$$\begin{tikzcd}
x_1, \ldots, x_n \arrow[r, shift left, "\mathcal{E}"] \arrow[d, "F"'] & c_1, \ldots, c_n \arrow[l, shift left, "\mathcal{D}"] \arrow[d, "\hat{F}"] \\
y & c \arrow[l, "\mathcal{D}"]
\end{tikzcd}$$

Let F be a n-argument function: $y = F(x_1, \ldots, x_n)$.

An encryption $\mathcal{E}, \mathcal{D}$ is homomorphic for function F if there exists a function $\hat{F}$ such that

$$\mathcal{D}(\hat{F}(c_1, \ldots, c_n)) = F(\mathcal{D}(c_1), \ldots, \mathcal{D}(c_n))$$

for all encrypted arguments $c_1, \ldots, c_n$.

As a corollary, for all cleartext arguments $x_1, \ldots, x_n$, we have

$$\mathcal{D}(\hat{F}(\mathcal{E}(x_1), \ldots, \mathcal{E}(x_n)) = F(x_1, \ldots, x_n)$$

Example: RSA is homomorphic for multiplication

RSA encryption: (public key is e, N; secret key is d)

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

If c_1, c_2 are two encrypted messages,

$$\mathcal{D}(c_1) \cdot \mathcal{D}(c_2) = c_1^d \cdot c_2^d = (c_1 \cdot c_2)^d = \mathcal{D}(c_1 \cdot c_2) \pmod{N}$$

If F is multiplication modulo N, its homomorphic function $\hat{F}$ is multiplication modulo N.

Application: tallying a vote

Votes v_i: 1 for blank vote, 2 for Alice, 3 for Bob.

Each voter i encrypts their vote v_i with the public key of the voting authority.

The voting operator collects the votes $\mathcal{E}(v_i)$ and computes their product

$$P \stackrel{def}{=} \mathcal{E}(v_1) \cdots \mathcal{E}(v_n) \pmod{N}$$

The product P (still encrypted) is sent to the voting authority, which decrypts it:

$$\mathcal{D}(P) = \mathcal{D}(\mathcal{E}(v_1)) \cdots \mathcal{D}(\mathcal{E}(v_n)) = v_1 \cdots v_n \pmod{N}$$

If $v_1 \cdots v_n < N$, this result $\mathcal{D}(P)$ is $2^a \cdot 3^b$
where a is the number of Alice votes and b that of Bob votes.

(Warning: terrible protocol, do not use!)

El Gamal's encryption

A finite group G of order q.

Secret key: $x \in \{1, \ldots, q-1\}$.
Public key: a generator g of G, and $h \stackrel{def}{=} g^x$.

Encryption: $\mathcal{E}(m) = (c_1, c_2)$ with

- $y \in \{1, \ldots, q-1\}$ randomly generated
- $s = h^y$ the shared secret
- $c_1 = g^y$ and $c_2 = g^m \cdot s$.

Decryption: $\mathcal{D}(c_1, c_2) = m$ where

- we recover the shared secret s by computing c_1^x
- we recover g^m by computing $c_2 \cdot s^{-1}$
- we recover m by discrete logarithm in time $O(\sqrt{m})$.

El Gamal is homomorphic for addition

Homomorphism:

$$\begin{aligned}
\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) &= (g^{y_1} \cdot g^{y_2}, (g^{m_1} \cdot h^{y_1}) \cdot (g^{m_2} \cdot h^{y_2})) \\
&= (g^{y}, g^{m_1+m_2} \cdot h^{y}) \qquad \text{(with } y = y_1 + y_2 \bmod q\text{)} \\
&= \text{an encryption of } m_1 + m_2
\end{aligned}$$

Therefore, the homomorphic operation for addition of cleartexts is multiplication of ciphertexts.

This property is used in the Belenios electronic voting system.

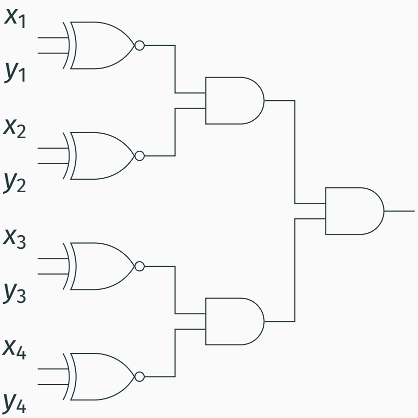
Fully-homomorphic encryption and Boolean circuits

A crucial step: find an encryption schema that is homomorphic both for addition (modulo 2) and for multiplication (modulo 2).

Such an encryption is said to be fully homomorphic (FHE, *Fully Homomorphic Encryption*) because it makes it possible to evaluate any Boolean circuit.

Logic gate		Arithmetic computation (mod 2)
exclusive or		$(a+b) \bmod 2$
and		$a \cdot b$
not		$1-a=(1+a) \bmod 2$
or	 = 	$1-(1-a)\cdot(1-b)$

Example: a comparator



Evaulated homomorphically, this circuit can be used for searching in an encrypted database.

Encryption = adding noise

We'll use encryption algorithm that rely on the idea that

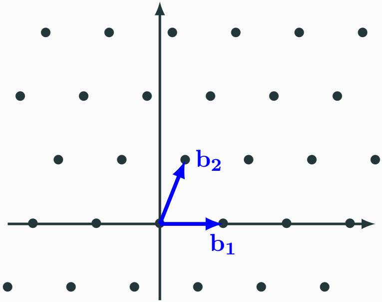
$$\text{encrypting a message } m \;=\; \text{drown out } m \text{ in (random) noise}$$

in such a way that

$$\text{decrypting the ciphertext} \;=\; \text{removing the noise to recover } m$$

is easy if we have the secret key, and infeasible otherwise.

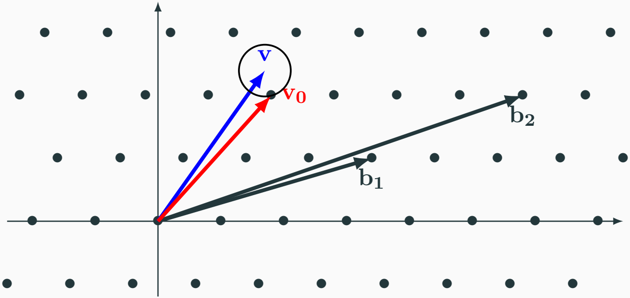
An example based on Euclidean lattices



A lattice = the set of vectors with integer coordinates in a given base $B = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$.

$$\left\{ \sum_{i=1}^{n} p_i \, \mathbf{b}_i \;\middle|\; p_i \in \mathbb{Z} \right\}$$

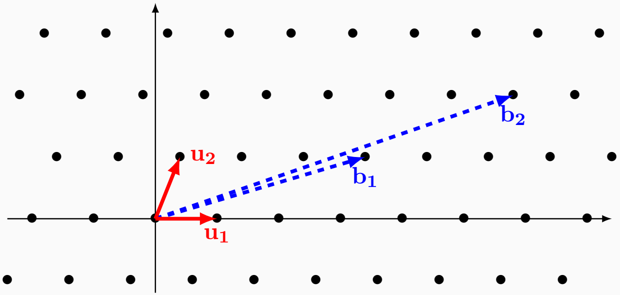
The closest vector problem (CVP)



Given a vector $\mathbf{v}$, find the coordinates of a vector $\mathbf{v}_0$ belonging to the lattice and closest to $\mathbf{v}$.

A computationally hard problem, even in the average case, even for approximate solutions, even with a quantum computer.

Good bases



We can change base while preserving the set of lattice vectors: $B \mapsto U.B$ where U is a unimodular matrix.

The CVP is easily solved if we have a good base, whose vectors are short and nearly orthogonal.

Encrypting with noise (Goldreich–Goldwasser–Halevi)

Secret key: a good base B_0, a unimodular matrix U.

Public key: the bad base $B = U.B_0$.

To encrypt a message $(m_1, \ldots, m_n)$ (a n-tuple of integers):

$$\mathcal{E}(m_1, \ldots, m_n) = \sum_{i=1}^{n} m_i \, \mathbf{b}_i + \mathbf{e}$$

where $\mathbf{e}$ is a small error, randomly generated.

To decrypt $\mathbf{c}$

Find the vector of the lattice closest to c

(using U, switch to base B_0, solve the CVP, switch to base B)

It has the shape $\sum_{i=1}^{n} m_i \, \mathbf{b}_i$, and $(m_1, \ldots, m_n)$ is the cleartext.

Simple encryption based on integers

(Craig Gentry, *Computing arbitrary functions of encrypted data*, 2010.)

Key: an odd integer p of P bits.

Encrypting one bit $m \in \{0, 1\}$:

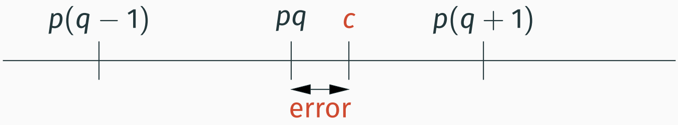
$$\mathcal{E}_p(m) = pq + 2r + m$$

q random Q-bit integer
$r \ll p$ random N-bit integer

In other words: a multiple of p plus an error $2r + m$.
The message is the least significant bit of the error.

(Typical parameters: $N = \lambda$, $P = \lambda^2$, $Q = \lambda^5$.)

Simple encryption based on integers



Decryption:

$$\mathcal{D}_p(c) = (c \bmod p) \bmod 2$$

Intuition: the multiple of p immediately below c is $p\lfloor c/p \rfloor$.
The error $2r + m$ is $c - p\lfloor c/p \rfloor = c \bmod p$.
The message m is $(2r + m) \bmod 2$.

An attacker cannot recover p from ciphertexts $c_1, \ldots, c_n$
(this is the approximate GCD problem, believed to be hard).

Simple encryption based on integers

To turn this schema into a public-key encryption schema, we can keep p as the secret key and publish as public key a set Z of random encryptions of zero:

$pk = \{2r_i + pq_i \mid r_i \text{ random } N\text{-bit integer}, q_i \text{ random } Q\text{-bit integer}\}$

To encrypt one bit $m \in \{0, 1\}$:

$$\mathcal{E}_{pk}(m) = m + \sum_{z \in Z} z$$

where Z is a random subset of pk of appropriate cardinal.

Somewhat homomorphic encryption (SHE)

$$\begin{aligned}\mathcal{E}_p(m_1) + \mathcal{E}_p(m_2) &= (m_1 + 2r_1 + pq_1) + (m_2 + 2r_2 + pq_2) \\ &= m_1 \oplus m_2 + 2(m_1 m_2 + r_1 + r_2) + p(q_1 + q_2)\end{aligned}$$

Decrypts to $m_1 \oplus m_2$ as long as the noise $2(m_1 m_2 + r_1 + r_2)$ remains less than p.

$$\begin{aligned}\mathcal{E}_p(m_1) \cdot \mathcal{E}_p(m_2) &= (m_1 + 2r_1 + pq_1)\,(m_2 + 2r_2 + pq_2) \\ &= m_1 m_2 + 2(r_1 m_2 + r_2 m_1 + 2r_1 r_2) + p(\ldots)\end{aligned}$$

Decrypts to $m_1 m_2$ as long as the noise $2(r_1 m_2 + r_2 m_1 + 2r_1 r_2)$ remains less than p.

Somewhat homomorphic encryption (SHE)

Noise increases

- slowly (+ 1 bit) at each addition;
- quickly ($\times$ 2 bits) at each multiplication.

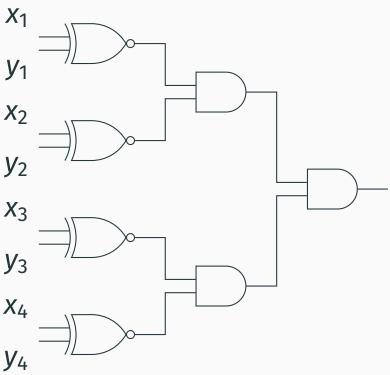
When the noise gets larger than p, encrypted results become false.

This limits strongly the multiplicative depth (and weakly the additive depth) of the computations that we can perform homomorphically.

Limited circuits

The multiplicative depth of a circuit is the maximal number of "and" / "or" gates between an input and an output.

Example: a n-bit comparator has multiplicative depth $\lceil \log_2 n \rceil$.



From SHE to FHE: how to reduce noise?

To homomorphically evaluate circuits with arbitrary depth, we must reduce noise of some of the encrypted intermediate results so that the noise never exceeds P bits.

Naive idea: we decrypt, then encrypt again!

$$c \mapsto \mathcal{E}_{pk}(\mathcal{D}_{sk}(c))$$

The noise that was reaching P bits drops to N bits.

Problems: (1) the intermediate result $\mathcal{D}_{sk}(c)$ is in the clear;
(2) we do not have the secret key sk to begin with.

From SHE to FHE: Gentry's bootstrap

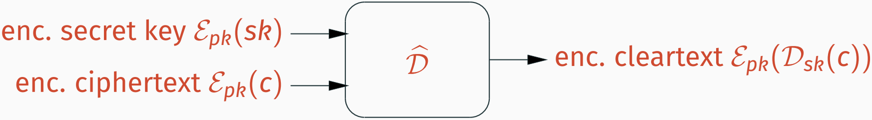
(Craig Gentry, *A fully homomorphic encryption scheme*, PhD, Stanford, 2009.)

$$\text{secret key } sk \longrightarrow \boxed{\quad \mathcal{D} \quad} \longrightarrow \text{cleartext } \mathcal{D}_{sk}(c)$$
$$\text{ciphertext } c \longrightarrow$$

The decryption algorithm $\mathcal{D}$ can be implemented by a Boolean circuit.

From SHE to FHE: Gentry's bootstrap

(Craig Gentry, *A fully homomorphic encryption scheme*, PhD, Stanford, 2009.)

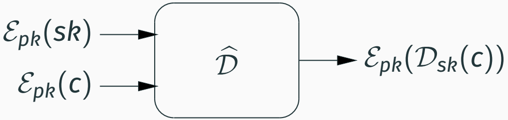


The decryption algorithm $\mathcal{D}$ can be implemented by a Boolean circuit.

If its multiplicative depth is low enough, this circuit can be evaluated homomorphically using our somewhat homomorphic encryption schema (SHE).

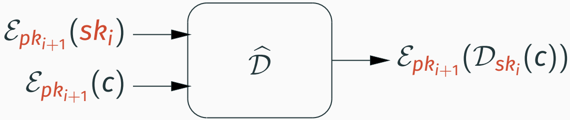
The result is a ciphertext equivalent to c, but whose noise depends only on the multiplicative depth of the decryption circuit.

Bootstrapping problems



Cryptographers hate the idea of encrypting a secret key *sk* with its public key *pk*.

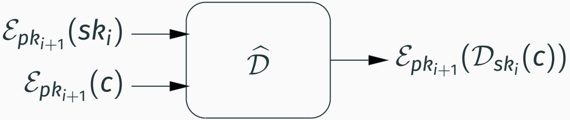
Bootstrapping problems



Cryptographers hate the idea of encrypting a secret key sk with its public key pk.
$\rightarrow$ We can change keys at bootstrap points.

To bootstrap as many times as necessary, the homomorphic evaluation receives a sequence of public keys pk_i
and encryptions of the corresponding secret keys $\mathcal{E}_{pk_{i+1}}(sk_i)$.

Bootstrapping problems



The multiple encryption $\mathcal{E}_{pk_{i+1}}(c)$ greatly increases the size of the encrypted intermediate result c.

The decryption circuit is often too “deep”
$\rightarrow$ Favor SHEs with simple decryption algorithms
(e.g. based on lattices).
$\rightarrow$ Tweak encryption to leave hints that simplify decryption
(without weakening encryption too much).

Summary on homomorphic encryption

Much research work since Gentry's smashing result:

- other somewhat homomorphic encryption scheme;
- multiplication algorithms that increase noise less;
- more efficient bootstrap.

Several implementations with almost reasonable performance, such as TFHE (`https://github.com/tfhe/tfhe`):

- evaluating a logic gate $\approx$ 20 ms
- bootstrap $\approx$ 100 ms.

New direction: approximate homomorphic encryption, for machine learning from confidential data.

Secure multiparty computation

The millionaires problem (A. Yao, 1982)

Alice and Bob want to know who is the wealthier, without revealing their wealth to the other.

With a trusted third-party (Charlie):
Alice tells her wealth to Charlie.
Bob does likewise.
Charlie announces who is the wealthier, and reveals nothing else.

Without a trusted third-party: which distributed algorithm, executed by Alice and by Bob, would give the same result and the same privacy guarantees?

Secure multiparty computation

n participants, having a secret datum x_i each,
cooperate to compute a function $y = F(x_1, \ldots, x_n)$
without revealing anything about the x_i that is not implied by the result y.

Example: a public tender

$$F(x_1, \ldots, x_n) = (i, x_i) \quad \text{where } x_i = \min(x_1, \ldots, x_n)$$

Reveals the identity of the lowest bidder and their bid, but not the other bids.

Other examples: statistical indicators over the x_i
(average, median, histogram for deciles, etc.)

Using homomorphic encryption

A secret key sk cut in n shares $(sk_1, \ldots, sk_n)$
+ the corresponding public key pk.

1. Each participant encrypts their data and publishes it: $\mathcal{E}(x_i)$.
2. Someone computes F homomorphically from the $\mathcal{E}(x_i)$.
3. The participants collaborate to decrypt the result.

This is a correct solution. However, it is much more efficient to distribute the computation between the participants.

Bit sharing

How can we share a secret bit b between two participants?

- Draw a random bit r.
- Send $b_1 = r$ to one participant and $b_2 = b \oplus r$ to the other.

None of the participants can recover b by itself.

If both participants publish their bits b_1 and b_2, they recover b by computing $b_1 \oplus b_2 = r \oplus b \oplus r = b$.

We write $[b]$ for a sharing of a bit b: a pair of bits (b_1, b_2) such that $b = b_1 \oplus b_2$.

Sharing private bits

Bit sharing also enables two participants A, B to share two private bits, a provided by A and b provided by B:

- A draws a sharing $[a] = (a_1, a_2)$ and sends a_2 to B.
- B draws a sharing $[b] = (b_1, b_2)$ and sends b_1 to A.

$$\begin{tikzcd}
a \arrow[r] \arrow[dr] & a_1, b_1 \\
b \arrow[r] \arrow[ur] & a_2, b_2
\end{tikzcd}$$

Adding two shared bits

We have two shared bits, $[x] = (x_1, x_2)$ and $[y] = (y_1, y_2)$.

Participant 1 knows x_1 and y_1, and computes $z_1 \stackrel{def}{=} x_1 \oplus y_1$.

Participant 2 knows x_2 and y_2, and computes $z_2 \stackrel{def}{=} x_2 \oplus y_2$.

The pair (z_1, z_2) is a sharing of $x \oplus y$:

$$z_1 \oplus z_2 = (x_1 \oplus y_1) \oplus (x_2 \oplus y_2) = (x_1 \oplus x_2) \oplus (y_1 \oplus y_2) = x \oplus y$$

The computation is local (no communication between the participants).

Multiplying two shared bits

We have two shared bits, $[x] = (x_1, x_2)$ and $[y] = (y_1, y_2)$, and we wish to compute a sharing (z_1, z_2) of $x \wedge y$.

No purely local computation suffices. In particular,

$$(x_1 \wedge y_1) \oplus (x_2 \wedge y_2) \neq (x_1 \oplus x_2) \wedge (y_1 \oplus y_2)$$

An expensive solution based on 1-in-4 oblivious transfer.

Multiplication by oblivious transfer

P1 chooses z_1 randomly and tabulates the correct value of z_2 ($z_2 = z_1 \oplus ((x_1 \oplus x_2) \wedge (y_1 \oplus y_2))$) for the unknowns x_2 and y_2:

line	x_2	y_2	z_2
0	0	0	$z_1 \oplus (x_1 \wedge y_1)$
1	0	1	$z_1 \oplus (x_1 \wedge \neg y_1)$
2	1	0	$z_1 \oplus (\neg x_1 \wedge y_1)$
3	1	1	$z_1 \oplus (\neg x_1 \wedge \neg y_1)$

P2 chooses the line (0 to 3) corresponding to its values of x_2 and y_2 and receives the corresponding z_2.

P1 does not know which line P2 chose. (“Oblivious”.)

P2 learns nothing about the other lines.

Multiplication using Beaver triples

Ahead of time, we can prepare multiplicative triples also called Beaver triples:
a number of shared bits $[a], [b], [c]$ such that $c = a \wedge b$.

(By oblivious transfer, or other zero-knowledge protocols, or via a trusted third-party.)

P1 has the (a_1, b_1, c_1) parts of the triples and P2 the (a_2, b_2, c_2) parts.

Multiplication using Beaver triples

To compute a sharing (z_1, z_2) of $x \wedge y$:

P1 and P2 pick the next Beaver triple (a, b, c) on their list.

P1 publishes $a_1 \oplus x_1$ and $b_1 \oplus y_1$.

(i.e. its shares of x, y blinded by a, b)

P2 publishes $a_2 \oplus x_2$ and $b_2 \oplus y_2$ likewise.

P1 and P2 now know $d = a \oplus x$ and $e = b \oplus y$.

P1 computes z_1 and P2 computes z_2 as follows:

$$z_i = d \wedge e \oplus d \wedge b_i \oplus a_i \wedge e \oplus c_i$$

It's a sharing of $x \wedge y$ since

$$x \wedge y = (d \oplus a) \wedge (e \oplus b) = d \wedge e \oplus d \wedge b \oplus a \wedge e \oplus \underbrace{a \wedge b}_{=c}$$

Generalization to $n > 2$ participants

We can share a bit b between $n > 2$ participants:

$$[b] = (b_1, \ldots, b_n) \quad \text{with} \quad b = b_1 \oplus \cdots \oplus b_n$$

If $b_1, \ldots, b_{n-1}$ are chosen randomly, none of the participants has any information about b.

The n participants must share their knowledge to recover b.

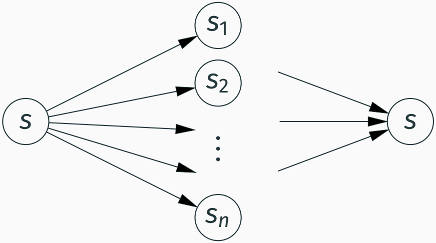
A collusion of $t < n$ participants cannot recover b.

Problem: as soon as one participant crashes or produces a wrong result, the multiparty computation fails or produces a wrong result.

Secret sharing: the general case

Divide a secret s into n shares $s_1, \ldots, s_n$ so that

- t shares suffice to recover s;
- fewer than t shares reveal nothing about s.



Distribution of n shares

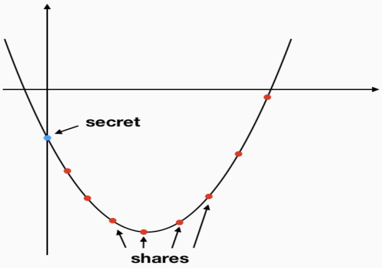
Reconstruction from t parts

Shamir's secret sharing

The secret s is an element of a finite field such as $\mathbb{Z}/p\mathbb{Z}$.

Sharing the secret:

- Choose a polynomial P of degree $t-1$ whose constant coefficient is s and the other coefficients are random.
- The shares are $s_i = P(i)$ for $i = 1, \ldots, n$.



Shamir's secret sharing

The secret s is an element of a finite field such as $\mathbb{Z}/p\mathbb{Z}$.

Recovering the secret:

To know t shares = to know t points $(x_1, y_1), \ldots, (x_t, y_t)$ on the curve of P.

Since P has degree $t - 1$, these t points determine P entirely.

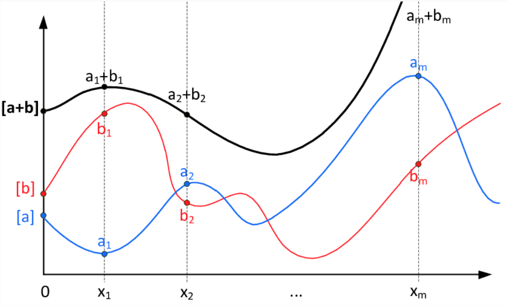
The secret s is $P(0)$.

More directly, using Lagrange's interpolation formula:

$$s = P(0) = \sum_{j=1}^{t} y_j \prod_{k=1, k \neq j}^{t} \frac{x_k}{x_k - x_j}$$

(Note: if more than t shares are revealed, we can not only recover s but also check that the shares are consistent.)

Computing with Shamir sharings: addition



Let $[a] = (a_1, \ldots, a_n)$ and $[b] = (b_1, \ldots, b_n)$ be Shamir sharings of the secrets a, b.

Then, $(a_1 + b_1, \ldots, a_n + b_n)$ is a Shamir sharing of $a + b$.

It can be computed locally by each of the n participants.

Computing with Shamir sharings: multiplication

Let $[a] = (a_1, \ldots, a_n)$ and $[b] = (b_1, \ldots, b_n)$ be Shamir sharings for the secrets a, b:

$$a = A(0) \quad a_i = A(i) \quad b = B(0) \quad b_i = B(i)$$

where A and B are polynomials with degree $t - 1$.

The points $(i, a_i b_i)$ belong to the curve of polynomial AB.

But AB has degree $2t - 2$, hence $t - 1$ points are not enough to determine $AB(0) = ab$.

Therefore, $(a_1 b_1, \ldots, a_n b_n)$ is not a sharing of ab.

Computing with Shamir sharings: multiplication

Each of the first $2t - 1$ participants prepares a sharing of its coefficient $a_i b_i$, that is, a random polynomial P_i of degree $t-1$ such that $P_i(0) = a_i b_i$.

They publish these sharings:
participant i sends $P_i(j)$ to participant j.

Then, the n participants reconstruct a sharing $(c_1, \ldots, c_n)$ using Lagrange's interpolation formula:

$$c_j = \sum_{i=1}^{2t-1} P_i(j)\lambda_i \quad \text{where} \quad \lambda_i = \prod_{k=1, k\neq i}^{2t-1} \frac{k}{k-i}$$

It's a sharing of ab, since $P = \sum_{i=1}^{2t-1} P_i \lambda_i$ is a polynomial of degree $t-1$ that has value ab at 0: $P(0) = \sum_{i=1}^{2t-1} a_i b_i \lambda_i = AB(0) = ab$.

Summary

Computing over encrypted or blinded data

Three approaches that are now well understood:

	Multiparty computation	Homomorphic encryption	Functional encryption
Inputs	blinded	encrypted	encrypted
Outputs	in the clear	encrypted	in the clear
Communications	yes	no	no
Efficiency	decent	low	decent in special cases

Other approaches remain theoretical, such as indistinguishable obfuscation.

Protecting data during computation

The cryptographic approach:

- high security that can be characterized mathematically;
- expensive computations;
- limited expressiveness
  (circuits only, no conditionals, no loops).

Already usable in practice for simple but highly confidential computations: electronic voting, secret auctions, …