Secure computing, fourth lecture

Secure multi-party computation: secret sharing

Xavier Leroy

2025-11-27

Collège de France, chair of Software sciences

xavier.leroy@college-de-france.fr

Secure multi-party computation

Secure multi-party computation (MPC) (reminder)

Computing over secret data provided by n participants:

- Each participant i has a secret x_i.
- The participants work together to compute $y = F(x_1, \ldots, x_n)$.
- 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 (reminder)

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 (reminder)

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?

Homomorphic encryption *vs.* secure multi-party computation

	Homomorphic encryption	Secure multi-party computation
Paradigm	delegated computation (*cloud*)	distributed computation
Participants	1 client, 1 computer	n participants
Secrets	held by the client only	each participant has some secrets
Results	known to the client only	known to all participants
Computing power	computer $\gg$ client	$\approx$ the same for all
Communications	few	many
Protocols	non-interactive	interactive

Correctness and security criteria

Correctness: the distributed computation produces the correct result if all participants follow the protocol.

Passive security ("honest but curious" participants): if all participants follow the protocol, a collusion of $\leq A$ participants cannot learn anything about the secrets of the other participants.

Active security (malicious participants): if $\leq A$ participants do not follow the protocol, the other participants can detect it and abort the computation.

(Note: we assume that communications between participants are encrypted and authenticated $\Rightarrow$ the only possible attackers are the participants.)

Additive sharing of bits: the GMW protocol

Sharing a secret bit

How to share a secret bit b between two participants?

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

Each participant learns nothing about the bit b.
(One-time pad principe: $b \oplus r$ is as random as r).

When both participants agree to reveal the bit b, they exchange their bits b_1, b_2 and recover b by computing

$$b_1 \oplus b_2 = r \oplus b \oplus r = (r \oplus r) \oplus b = b$$

We write $[b]$ for a sharing of the bit b:

$$[b] = (b_1, b_2) \quad \text{such that} \quad b = b_1 \oplus b_2$$

Sharing then revealing

Sharing a secret (of one participant, or of a third party):

Alice: $[x] = (x_1, x_2)$ ⇢ Alice: x_1
→ Bob: x_2

Trent: $[x] = (x_1, x_2)$ → Alice: x_1
→ Bob: x_2

Revealing (opening) a shared secret:

Alice: x_1 ⇢ Alice: $x = x_1 \oplus x_2$

Bob: x_2 ⇢ Bob: $x = x_1 \oplus x_2$

Secure two-party evaluation of

$$z = F(\mathbf{x}, \mathbf{y})$$

F: a Boolean circuit
$\mathbf{x}$: Alice's secrets
$\mathbf{y}$: Bob's secrets

- Inputs: Alice draws sharings $[\mathbf{x}]$, Bob draws sharings $[\mathbf{y}]$, they exchange the shares.
- Two-party computation: Alice computes z_1 and Bob computes z_2, where (z_1, z_2) is a sharing of z.
  (They may need to communicate during this computation.)
- Output: Alice and Bob reveal z_1, z_2 and recover $z = z_1 \oplus z_2$.

Adding two shared bits (XOR gate)

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

Alice knows x_1 and y_1. She computes $z_1 = x_1 \oplus y_1$.

Bob knows x_2 and y_2. He computes $z_2 = x_2 \oplus y_2$.

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

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

Purely local computation: no communication between the participants.

Negation of a shared bit (NOT gate)

We have a shared bit $[x] = (x_1, x_2)$.

Alice knows x_1 and computes $z_1 = x_1 \oplus 1 = \neg x_1$.

Bob knows x_2 and sets $z_2 = x_2$.

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

$$z_1 \oplus z_2 = (x_1 \oplus 1) \oplus x_2 = (x_1 \oplus x_2) \oplus 1 = x \oplus 1 = \neg x$$

Multiplying two shared bits (AND gates, OR gates)

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

We want to compute a sharing of $x \wedge y = x \cdot y$
or $x \vee y = \neg(\neg x \cdot \neg y)$.

This cannot be done by a purely local computation. In particular, $(x_1 \cdot y_1, x_2 \cdot y_2)$ is not a sharing of $x \cdot y$:

$$(x_1 \oplus x_2) \cdot (y_1 \oplus y_2) = x_1 \cdot y_1 \oplus x_1 \cdot y_2 \oplus x_2 \cdot y_1 \oplus x_2 \cdot y_2 \neq x_1 \cdot y_1 \oplus x_2 \cdot y_2$$

Goldreich, Micali, Wigderson (STOC 1987) propose to use a
1 out of 4 oblivious transfer.

Oblivious transfer (OT)

A protocol between two participants:

- Alice knows n values $v_1, \ldots, v_n$.
- Bob chooses $i \in \{1, \ldots, n\}$.

At the end of the protocol,

- Bob knows the value v_i.
- Alice does not know Bob's choice i.
- Bob learnt nothing about the other values v_j for $j \neq i$.

(More details in lecture #5.)

Multiplication by oblivious transfer

Computation of a sharing (z_1, z_2) of $x \cdot y$:

Alice picks z_1 randomly and tabulates the value of z_2
as a function of the possible values of the unknowns x_2, y_2.

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

Presented as a table:

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

Multiplication by oblivious transfer

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

Oblivious transfer: Bob requests the line number $2x_2 + y_2$ corresponding to his shares x_2, y_2, and receives the corresponding z_2.

We have $z_1 \oplus z_2 = x \cdot y$.

Alice does not know Bob's choice $\Rightarrow$ learns nothing about x_2, y_2.

Bob does not see the other lines $\Rightarrow$ learns nothing about x_1, y_1.

Multiplication using Beaver triples

(D. Beaver, *Efficient Multiparty Protocols Using Circuit Randomization*, CRYPTO 1991.)

We prepare beforehand a list of Beaver triples:
random shared bits $[a], [b], [c]$ such that $c = a \cdot b$.

Alice knows the shares a_1, b_1, c_1 of these triples.
Bob knows the shares a_2, b_2, c_2.

We can produce these triples in advance by oblivious transfer between the two participants, or by using a trusted third-party.

("Offline" communications before the actual computation starts, instead of "online" communications during the computation, as with the OT protocol used for GMW.)

Multiplication using Beaver triples

Computation of a sharing of (z_1, z_2) of $x \cdot y$:

Alice and Bob take the next triple a, b, c on their lists.

Alice sends Bob $a_1 \oplus x_1$ and $b_1 \oplus y_1$

(her shares of x et y masked by a and b)

Bob sends Alice $a_2 \oplus x_2$ and $b_2 \oplus y_2$ (similar masking)

Alice and Bob now know $d = a \oplus x$ and $e = b \oplus y$.

Alice computes z_1 and Bob computes z_2 as follows:

$$z_i = d \cdot y_i \oplus a_i \cdot e \oplus c_i$$

(z_1, z_2) is a sharing of $x \cdot y$ because

$$\begin{aligned} z_1 \oplus z_2 &= a \cdot y \oplus x \cdot y \oplus a \cdot b \oplus a \cdot y \oplus c \\ &= x \cdot y \oplus (a \cdot b \oplus c) = x \cdot y \quad \text{since } c = a \cdot b \end{aligned}$$

Extension to n > 2 participants

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

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

If participant 1 wishes to share the secret x, it draws $b_2, \dots, b_n$ randomly, sends b_i to participant i, and keeps $b_1 = x \oplus b_2 \oplus \cdots \oplus b_n$.

To reveal the sharing $[b] = (b_1, \dots, b_n)$, each participant i sends its share b_i to the $n - 1$ other participants.
All participants, then, obtain $b = b_1 \oplus \cdots \oplus b_n$.

Security of the GMW protocol

Assuming the OT protocol used is secure.

- Passive security: the only way to recover a shared bit b is that the n participants reveal their shares $b_1, \ldots, b_n$. A collusion of $A < n$ participants learns nothing about b.
- Active security: none. If one participant produces the wrong share b_i, the result of the computation is wrong, and this cannot be detected.
- Fault tolerance: none. If one participant fails or is cut off the network, the result of the computation is lost.

Sharings k among n

Replicated sharing 2 among 3

An example of a redundant sharing between 3 participants.

Each participant has 2 shares out of the 3 shares of the secret.

$$b = b_1 \oplus b_2 \oplus b_3$$

Alice has b_1 and b_2

Bob has b_2 and b_3

Charlie has b_3 and b_1

Any two participants can exchange their shares and recover the secret.

Fault tolerance: resists failure of one of the 3 participants.

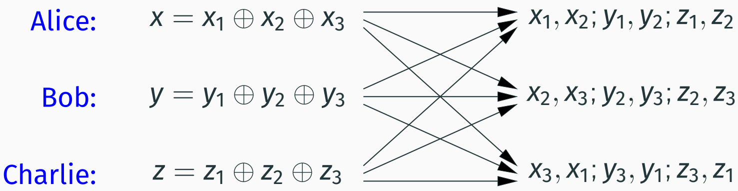
Passive security: one participant learns nothing about the secret.

Active security: if one participant produces wrong results, the other two can detect it.

Secret sharing

To share her secret x, Alice draws x_2, x_3 randomly, takes $x_1 = x \oplus x_2 \oplus x_3$, and sends the correct x_i to Bob and Charlie.

Bob can do likewise with y and Charlie with z.



Addition

The participants add their shares pointwise:

$$\begin{array}{llll} \text{Alice} & x_1, x_2 & y_1, y_2 & \rightarrow & x_1 \oplus y_1, x_2 \oplus y_2 \\ \text{Bob} & x_2, x_3 & y_2, y_3 & \rightarrow & x_2 \oplus y_2, x_3 \oplus y_3 \\ \text{Charlie} & x_3, x_1 & y_3, y_1 & \rightarrow & x_3 \oplus y_3, x_1 \oplus y_1 \end{array}$$

If $x = x_1 \oplus x_2 \oplus x_3$ and $y = y_1 \oplus y_2 \oplus y_3$, the result is a redundant sharing of $x \oplus y$.

Multiplication

The participants combine their shares as follows:

$$\begin{array}{llll} \text{Alice} & x_1, x_2 & y_1, y_2 & \rightarrow & p = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \\ \text{Bob} & x_2, x_3 & y_2, y_3 & \rightarrow & q = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \\ \text{Charlie} & x_3, x_1 & y_3, y_1 & \rightarrow & r = x_3y_3 \oplus x_3y_1 \oplus x_1y_3 \end{array}$$

Alice draws a sharing $[p]$ of p, sends it to Bob and Charlie.
Bob draws a sharing $[q]$ of q, sends it to Alice and Charlie.
Charlie draws a sharing $[r]$ of r, sends it to Alice and Bob.

The 3 participants compute a sharing of $p \oplus q \oplus r$ by local addition. It is a sharing of xy, because

$$\begin{aligned} xy &= (x_1 \oplus x_2 \oplus x_3)(y_1 \oplus y_2 \oplus y_3) \\ &= x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \oplus x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_3y_3 \oplus x_3y_1 \oplus x_1y_3 \\ &= p \oplus q \oplus r \end{aligned}$$

Shamir's secret sharing

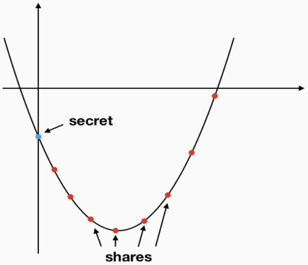
(A. Shamir, *How to share a secret*, CACM 22(11), 1979.)

Secrets and shares are elements of a finite field $\mathbb{F}_q$ of order $q > n$ (n is the number of participants).

Sharing the secret x:

Pick a polynomial P of degree $t < n$ with the constant coefficient equal to x and the other coefficients randomly chosen in $\mathbb{F}_q$.

The shares are $x_i = P(i)$ for $i = 1, \ldots, n$.



(Like Reed-Solomon codes.)

Shamir's secret sharing

$$[x] = (P(1), \ldots, P(n)) \quad \text{with } \deg(P) = t \text{ and } P(0) = x$$

Recovering the secret x from $t+1$ shares:

Knowing $t+1$ shares is knowing $t+1$ points $(x_0, y_0), \ldots, (x_t, y_t)$ on the curve of P.

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

The secret x is $P(0)$.

Lagrange interpolation formula:

$$x = P(0) = \sum_{j=0}^{t} y_j \lambda_j \quad \text{where} \quad \lambda_j = \prod_{k=0, k \neq j}^{t} \frac{x_k}{x_k - x_j}$$

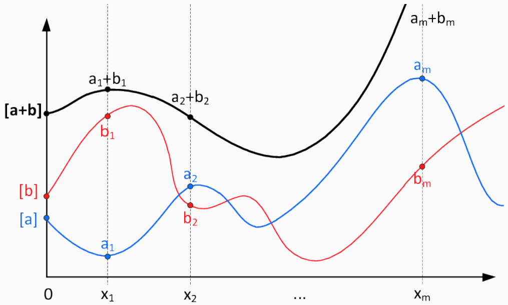
Shamir's secret sharing

Fault tolerance: $t + 1$ shares among n suffice to recover the secret.

Passive security: a collusion of at most t participants learns nothing about the secret.
(If $P(i)$ is known for t points $i \neq 0$, $P(0)$ can still take any value.)

Active security: as with Reed-Solomon codes, we can detect up to $n - t - 1$ errors and correct up to $(n - t - 1)/2$ errors.

Addition of two Shamir sharings



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

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

It can be computed locally by each participant.

Addition and multiplication by a constant

If $[a] = (a_1, \ldots, a_n)$ is a sharing of a (with polynomial P):

- $(a_1 + k, \ldots, a_n + k)$ is a sharing of $a + k$ (polynomial $P + k$).
- $(ka_1, \ldots, ka_n)$ is a sharing of ka (polynomial kP).

(Local computation.)

Multiplication of two Shamir sharings

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

$$a = P(0) \quad a_i = P(i) \quad b = Q(0) \quad b_i = Q(i)$$

where P and Q are degree-t polynomials.

The points $(i, a_i b_i)$ lie on the curve of the polynomial PQ.

However, PQ has degree $2t$, hence t points do not determine $PQ(0) = ab$.

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

Multiplication of two Shamir sharings

Assume $t < n/2$. Each of the first $2t$ participants prepares a sharing $[a_i b_i]$ of the product of its two shares a_i and b_i, and sends it to the other participants.

Thus, we have random polynomials $R_1, \dots, R_{2t}$ of degree t such that

$$R_i(0) = a_i b_i \qquad \text{participant } j \text{ knows } R_i(j)$$

The n participants reconstruct (locally) a sharing $(c_1, \dots, c_n)$ using Lagrange's interpolation formula:

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

Multiplication of two Shamir sharings

$$R_i(0) = a_i b_i \qquad\qquad \deg(R_i) = t$$

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

Consider $R = \sum_{i=1}^{2t} R_i \lambda_i$. We have

$$\deg R = t \qquad\qquad c_j = R(j)$$

$$R(0) = \sum_{i=1}^{2t} a_i b_i \lambda_i = \sum_{i=1}^{2t} PQ(i)\lambda_i = PQ(0) = ab$$

Therefore, $(c_1, \ldots, c_n)$ is a sharing of ab, using the polynomial R.

Alternative: multiplication using Beaver triples

We assume the participants received beforehand three sharings $[u], [v], [w]$ with u, v random and $w = uv$.

To compute the product ab:

The participants locally compute $[a + u]$ and $[b + v]$, and reveal these sharings.

All participants, then, know $\alpha = a + u$ and $\beta = b + v$ (the secret operands a, b masked by random u, v).

The participants locally compute

$$[c] = [w] + \alpha[b] - \beta[u]$$

It is a sharing of the product ab, since

$$c = uv + ab + ub - bu - vu = ab$$

Generalization:
linear secret sharing scheme

Linear secret sharing scheme (LSSS)

Defined by

- a matrix $\mathbf{M}$ of dimensions $m \times d$
- a vector $\mathbf{v}$ of dimension d
- a surjective function $\varphi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ (row $\mapsto$ participant).

To share the secret s, we draw a random vector k such that

$$s = \langle \mathbf{v}, \mathbf{k} \rangle$$

Then, we compute m parts $s_1, \ldots, s_m$ by applying the matrix $\mathbf{M}$

$$\mathbf{M} \cdot \mathbf{k} = (s_1, \ldots, s_m)^\intercal$$

We give the share s_i to the participant number $\varphi(i)$.

Full additive sharing viewed as a trivial LSSS

Dimensions $m = d = n$ (number of participants).

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \qquad \begin{aligned} \mathbf{v} &= (1, 1, \dots, 1) \\ \varphi(i) &= i \end{aligned}$$

To share s, we choose $\mathbf{k} = (k_1, \dots, k_n)$ such that

$$s = \langle \mathbf{v}, \mathbf{k} \rangle = k_1 + \cdots + k_n$$

We give the share $s_i = k_i$ to the participant number $\varphi(i) = i$.

Shamir's sharing viewed as a LSSS

Dimensions $m = t + 1$ and $d = n$
(t degree of the polynomials, n number of participants).

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1^2 & \cdots & 1^t \\ 1 & 2 & 2^2 & \cdots & 2^t \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & n & n^2 & \cdots & n^t \end{pmatrix} \qquad \mathbf{v} = (1, 0, \ldots, 0) \qquad \varphi(i) = i$$

To share s, we choose $\mathbf{k} = (s, k_1, \ldots, k_t)$ with random k_i.
$\mathbf{k}$ are the coefficients of a polynomial P. We have

$$s = \langle \mathbf{v}, \mathbf{k} \rangle \qquad \mathbf{M} \cdot \mathbf{k} = (P(1), \ldots, P(n))^\intercal$$

We give the i-th share $P(i)$ to the participant number $\varphi(i) = i$.

Replicated sharing viewed as a LSSS

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{v} = (1, 1, 1)$$

$$\varphi(i) = \lceil i/2 \rceil$$

To share s, we write $s = \langle \mathbf{v}, \mathbf{k} \rangle = k_1 + k_2 + k_3$ with random k_i.

The shares are $\mathbf{M} \cdot \mathbf{k} = (k_2, k_3, k_1, k_3, k_1, k_2)^\intercal$.

We give the first two shares to P1, the next two to P2, the last two to P3.

Revealing the secret from a LSSS sharing

$$s = \langle \mathbf{v}, \mathbf{k} \rangle = \mathbf{v}^\intercal \cdot \mathbf{k} \qquad \mathbf{M} \cdot \mathbf{k} = (s_1, \ldots, s_m)^\intercal$$

We can recover the secret s from the shares s_i if there exists a linear combination of the lines of $\mathbf{M}$ that is equal to **v**,
i.e. a vector **x** of dimension d such that

$$\mathbf{M}^\intercal \cdot \mathbf{x} = \mathbf{v}$$

Then,

$$\langle \mathbf{x}, s_1, \ldots, s_m \rangle = \mathbf{x}^\intercal \cdot \mathbf{M} \cdot \mathbf{k} = (\mathbf{M}^\intercal \cdot \mathbf{x})^\intercal \cdot \mathbf{k} = \mathbf{v}^\intercal \cdot \mathbf{k} = s$$

Revealing the secret from a LSSS sharing

$$\mathbf{M}^{\mathsf{T}} \cdot \mathbf{x} = \mathbf{v}$$

If the sharing is redundant, multiple vectors $\mathbf{x}$ are possible. The zeros in $\mathbf{x}$ correspond to the shares that are not needed to recover s.

Example: for Shamir's sharing with $t = 2$ and $n = 4$, there are 4 possible $\mathbf{x}$ with one 0 coefficient:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 2 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{pmatrix}$$

$$\mathbf{v} = (1, 0, 0)$$

$$\mathbf{x}_1 = (0, 6, -8, 3)$$
$$\mathbf{x}_2 = (2, 0, -2, 1)$$
$$\mathbf{x}_3 = (8/3, -2, 0, 1/3)$$
$$\mathbf{x}_4 = (3, -3, 1, 0)$$

Arithmetic operations on LSSS sharings

Addition:

- local addition of each share.

Multiplication by a constant:

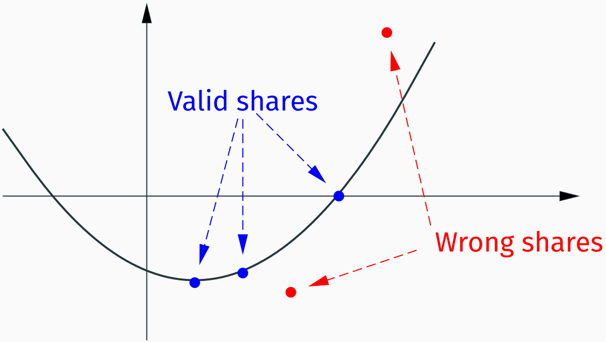
- local multiplication of each share.

General multiplication:

- Schur product (linear combination of local products) if the LSSS supports it (like Shamir with $2t < n$, or 2-3 replication).
- Beaver triples.
- Damgård-Nielsen product (not treated here).

Resistance against active attacks

Error detection in Shamir sharings



If there is no polynomial P of degree t that passes through the points $(1, a_1), \ldots, (n, a_n)$, the sharing $(a_1, \ldots, a_n)$ is wrong.

One or several shares a_i were corrupted, probably by malicious participants (active attack).

Error detection, error correction

A (n, t) Reed-Solomon code / a (n, t) Shamir sharing can:

- Detect up to $n - t - 1$ errors.

  Simply by applying the parity matrix H to the sharing:

  $$H \cdot (a_1, \ldots, a_n)^\top \neq \mathbf{0} \implies \text{the sharing } (a_1, \ldots, a_n) \text{ is wrong}$$

- Correct up to $(n - t - 1)/2$ errors.

  Naively: find a subset of $n - (n - t - 1)/2$ shares that is not wrong.

  Efficiently: use the Berlekamp-Welch algorithm.

Active security

A malicious participants, who do not follow the protocol.

Not numerous enough to reveal the secrets: $A \leq t$

Hypothesis: the malicious participants can modify their shares of the sharings, but not those of the honest participants.

Detecting the attack:

If $A \leq n - t - 1$, the attack can be detected when the n participants reveal their shares $(a_1, \ldots, a_n)$.

The honest participants notice that $H \cdot (a_1, \ldots, a_n)^\intercal \neq \mathbf{0}$.

Can be done as long as $A < n/2$ (by taking $t \approx n/2$).

Active security

A malicious participants, who do not follow the protocol.

Not numerous enough to reveal the secrets: $A \leq t$

Hypothesis: the malicious participants can modify their shares of the sharings, but not those of the honest participants.

Neutralizing the attack:

If $A \leq (n - t - 1)/2$, the attack can be detected as before. Moreover, the honest participants can identify the attackers, ignore them, and continue the computation.

Can be done as long as $A < n/3$ (by taking $t \approx n/3$).

Active security of the operations over sharings

The hypothesis

> *The malicious participants can modify their shares of the sharings, but not those of the honest participants.*

holds for local operations (addition, multiplication by a constant), where each participant only modifies its share.

It does not hold when a participant prepares and sends a sharing $[x]$ of a value x of its choice.

Active security of multiplication

Example: multiplication of two Shamir sharings using the Schur product.

> *Each of the first $2t$ participants prepares a sharing $[a_i b_i]$ of the product of its two shares a_i and b_i, and sends it to the other participants.*

If one of these $2t$ participants lies about its value of $a_i b_i$, the result of the multiplication is wrong, but there are no coding errors.

Active security of Beaver multiplication

Multiplication using Beaver triples only involves “verifiable” operations (addition, multiplication by a constant, revelation).

Locally compute $[a + u]$ and $[b + v]$
Reveal these sharings to give $\alpha = a + u$ and $\beta = b + v$ to all
Locally compute $[c] = [w] + \alpha[b] - \beta[u]$

However, the attackers could have corrupted the Beaver triple: the sharings $[a], [b], [c]$ are well formed but $c \neq ab$.

This can happen if the triples were generated beforehand using Schur product between the participants.

Verification of a Beaver triple by "sacrificing" another triple

Consider two Beaver triples $([u], [v], [w])$ and $([x], [y], [z])$.

We want to confirm that $w = uv$.

Let t be a random integer known to all participants
(but not controlled by any participant).

Locally compute $[\rho] = t[u] - [x]$ and $[\sigma] = [v] - [y]$
Reveal ρ and σ
Locally compute $[e] = t[w] - [z] - \sigma[x] - \rho[y] - \sigma\rho$
Reveal e
If $e \neq 0$, reject the triple $([u], [v], [w])$.

Fails with high probability if $w \neq uv$ or $z \neq xy$.
We can sacrifice several $([x], [y], [z])$ to increase confidence.

SPDZ: authenticating shares using a MAC

Another way to authenticate shares. It applies to all LSSS, including the full additive sharing.

An authenticated sharing $\langle s \rangle$ is a pair of sharings $([s], [\alpha \cdot s])$:

$$\langle s \rangle = (s_1, \ldots, s_n, m_1, \ldots, m_n) \text{ where } s = \sum s_i \text{ and } \alpha \cdot s = \sum m_i$$

α is a global, shared secret.
Participant i only knows s_i, m_i, and α_i.

We can check $\alpha \sum s_i = \sum m_i$ when we reveal $\langle s \rangle$,
and compute homomorphically on the sharings $\langle s \rangle$.

This provides active security even for $n - 1$ attackers.

(See section 6.6 of Evans and al, *A pragmatic introduction to secure multi-party computation*, 2018.)

Summary

Linear secret sharing

A way for multiple parties to compute over private data while controlling which results are revealed to all parties.

Nice properties:

- A good match for many practical problems (of the "let's work together but not trust each other" kind).
- In addition to confidentiality of private data, can ensure fault tolerance and resistance to active attacks.
- No hairy cryptography involved!

Two main limitations:

- The protocols are interactive by nature.
- Communications between participants severely limit the speed of computation.

References

References

The main source for this lecture:

- *Cryptography made simple*, Nigel P. Smart, Springer, 2016.
  Chapter 19 and section 22.3.

For further reading:

- *A pragmatic introduction to secure multi-party computation*,
  David Evans, Vladimir Kolsnikov, Mike Rosulek,
  NOW Publishers, 2018.