Secure Multiparty Computation (part 2)

Unconditionally secure MPC

- A week ago we considered secure multiparty computation.
  - The security was computational.
  - Good thing — with semi-honest adversary, the number of corrupted parties did not matter.
- Today we take a look what is possible if we want to remain unconditionally secure.

Semi-honest adversary

- Computed function f represented as a circuit consisting of
  - binary addition and multiplication gates;
  - unary gates for adding or multiplying with a constant.

  - Values on wires — elements of $\mathbb{Z}_p$.
- n players, where at most $t-1$ may be adversarial.
- All values on wires are shared using Shamir's (n, t)-secret sharing scheme.
- The protocol starts by each party sharing his inputs.
- Binary addition and unary operations — each party performs the same operation with his own respective shares only.
- Binary multiplication — next slides.
- Protocol ends by parties sending the shares of outputs to each other.

Multiplying shared secrets

- Let n parties hold shares $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_n$ for two secrets $v, v' \in \mathbb{Z}_p$.
- We want them to learn shares $s''_1, \ldots, s''_n$ for $v'' = v \cdot v'$, such that these shares are uniformly distributed and independent from anything else.

Multiplying shared secrets

- Let n parties hold shares $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_n$ for two secrets $v, v' \in \mathbb{Z}_p$.
- We want them to learn shares $s''_1, \ldots, s''_n$ for $v'' = v \cdot v'$, such that these shares are uniformly distributed and independent from anything else.
- Ideal protocol:
  - There is a trusted dealer $D \notin \{P_1, \ldots, P_n\}$.
  - D is sent the shares $s_1, \ldots, s_n, s'_1, \ldots, s'_n$.
  - D recovers v and v', computes $v'' = v \cdot v'$.
  - D constructs the shares for v'', sends them to $P_1, \ldots, P_n$.
- We want the real protocol to cause the same distribution of $s_1, \ldots, s_n, s'_1, \ldots, s'_n, s''_1, \ldots, s''_n$.
  - Each party P_i will see some more random values, but their distribution must be constructible from s_i, s'_i, s''_i.

Gennaro-Rabin-Rabin multiplication protocol

- Assume $t - 1 < n/2$. (in other words, $t - 1 \leq (n-1)/2$)
- Let f, f' be polynomials of degree $\leq t - 1$ used to share v, v'.
- $f(0) = v$. $f'(0) = v$. Let $f'' = f \cdot f'$. Then $f''(0) = v \cdot v''$.
- The degree of f'' is $\leq 2(t-1) \leq n - 1$.
- The values of f'' on n points suffice to reconstruct f''.
  - Party P_i can compute $f''(i)$ as $s_i \cdot s_i'$.
  - But we don't want to use f'' to share v''.
- There exist (public) $r_1, \ldots, r_n$, such that $f''(0) = \sum_{i=1}^{n} r_i(s_i \cdot s_i')$.
  - By Lagrange interpolation formula $r_i = \prod_{1 \leq j \leq n, j \neq i} j/(j-i)$.
- At least t of $r_1, \ldots, r_n$ are non-zero.
  - If only $r_{i_1}, \ldots, r_{i_{t-1}}$ were non-zero, then

$$v = (f \cdot \mathbf{1})(0) = \sum_{i=1}^{n} r_i f(i) \mathbf{1}(i) = \sum_{j=1}^{t-1} r_{i_j} s_{i_j},$$

allowing $P_{i_1}, \ldots, P_{i_{t-1}}$ to determine v.

Gennaro-Rabin-Rabin multiplication protocol

- Each party P_i randomly generates a polynomial f_i of degree at most $t-1$, such that $f_i(0) = s_i \cdot s'_i$.
- Party P_i sends to party P_j the value $u_{ij} = f_i(j)$.
  - Party P_i receives the values $u_{1i}, \ldots, u_{ni}$.
- P_i defines $s''_i = \sum_{j=1}^{n} r_j u_{ji}$.
- The shares $s''_1, \ldots, s''_n$ correspond to the polynomial $\hat{f} = \sum_{j=1}^{n} r_j f_j$.
  - It is a random polynomial because f_i-s were randomly generated.
  - It is independent from any $f_{i_1}, \ldots, f_{i_{t-1}}$, because at least t of the values $r_1, \ldots, r_n$ are non-zero.
- This polynomial shares the value

$$\hat{f}(0) = \sum_{j=1}^{n} r_j \cdot f_j(0) = \sum_{j=1}^{n} r_j s_j s'_j = f''(0) = v'' \ .$$

Over half of the parties must be honest

- Consider a two-party protocol Π for computing the AND of two bits.
- Let $\Pi(b_1, r_1, b_2, r_2)$ be the sequence of messages exchanged for party P_i's bit b_i and random coins r_i.

$$\forall r_1, r_2^0 \ \exists r_2^1 : \Pi(0, r_1, 0, r_2^0) = \Pi(0, r_1, 1, r_2^1)$$
$$\forall r_1, r_2^1 \ \exists r_2^0 : \Pi(0, r_1, 0, r_2^0) = \Pi(0, r_1, 1, r_2^1)$$
$$\forall r_1, r_2^0, r_2^1 : \Pi(1, r_1, 0, r_2^0) \neq \Pi(1, r_1, 1, r_2^1)$$

- Party P_2 whose input is $b_2 = 0$ and random coins r_2^0 can find b_1 as follows:
  - Let $\mathcal{T}$ be the exchanged sequence of messages.
  - Try to find such (b', r', r_2^1), that $\Pi(b', r', 1, r_2^1) = \mathcal{T}$.
  - If such triple exists then $b_1 = 0$. If not, then $b_1 = 1$.

Exercise. Generalize this result to more than 2 parties.

Exercise

Repeat the previous MPC construction, but using a verifiable secret sharing scheme.

- For example, Feldman's VSS.

Exercise

Repeat the previous MPC construction, but using a verifiable secret sharing scheme.

- For example, Feldman's VSS.

This exercise shows the possiblity of MPC, where

- security is computational;
- the number of corrupted parties is strictly less than $n/2$;
- the adversary is malicious;
- there is a broadcast channel;
- the adversary can shut down the computation.

The security can be made unconditional and shutdown possibilities can be eliminated.

Exercise

Consider Feldman's VSS:

- n parties, the share of i-th party is P_i.
- A group G with hard discrete logarithm. An element $g \in G$ of order p.
- The secret $v = a_0$ is shared using a polynomial of degree at most $t - 1$.
- The values $y_i = g^{a_i}$ for $0 \leq i \leq t - 1$ have been published.

Suppose that during the secret reconstruction time, one of the parties P_z refuses to produce a valid s_z. How can the honest parties find s_z?

Exercise

Consider Feldman's VSS:

- n parties, the share of i-th party is P_i.
- A group G with hard discrete logarithm. An element $g \in G$ of order p.
- The secret $v = a_0$ is shared using a polynomial of degree at most $t - 1$.
- The values $y_i = g^{a_i}$ for $0 \leq i \leq t - 1$ have been published.

Suppose that during the secret reconstruction time, one of the parties P_z refuses to produce a valid s_z. How can the honest parties find s_z?

This method allows us to kick out parties who behave maliciously.

What have we seen so far?

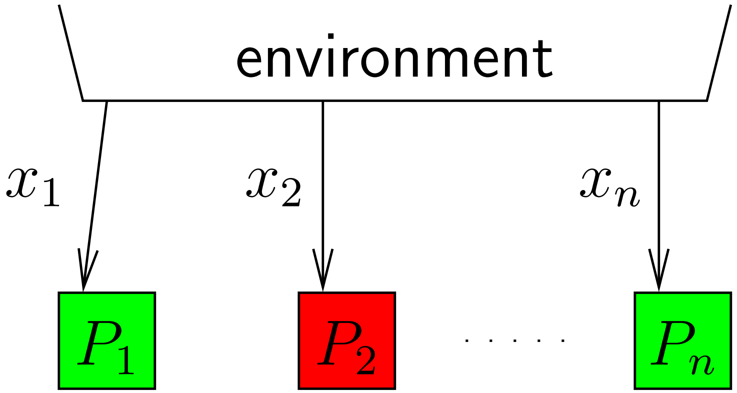
- 2-party, computational, semi-honest, constant-round.
- 2- or n-party, computational, semi-honest($< n$), linear-round.
- n-party, unconditional, semi-honest($< n/2$), linear-round.
- n-party, computational, malicious($< n/2$), constant-round.

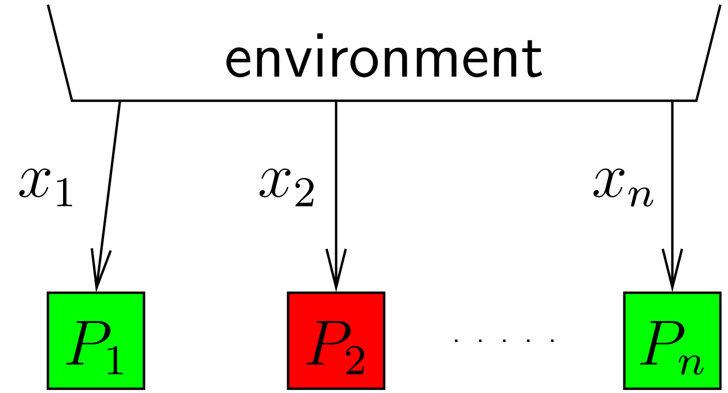
Coming up next: n-party, unconditional, broadcast, malicious($< n/3$), linear-round.

Malicious model — security definition

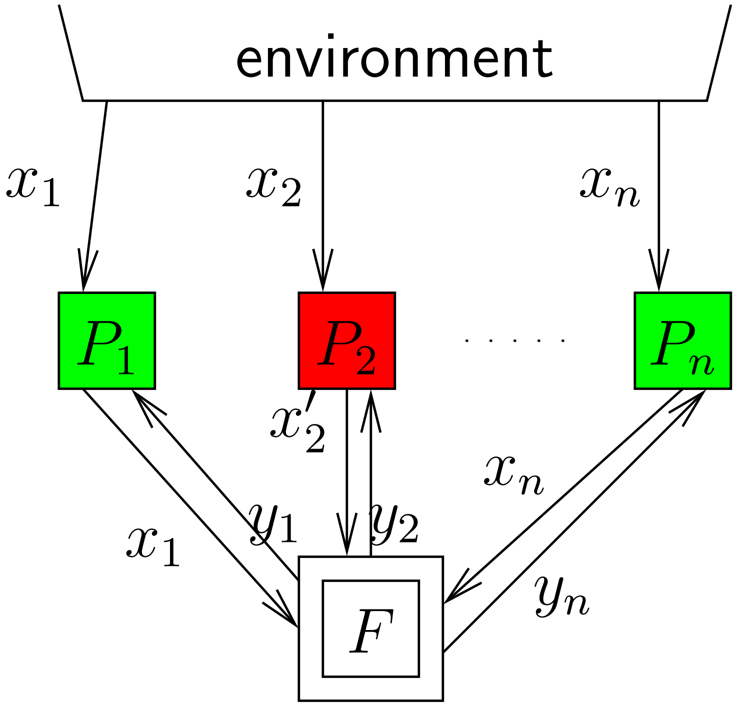
- Simulatability — turn real adversary into an ideal one.
- In the Ideal model, the computation proceeds as follows:
  - The parties receive the inputs.
  - Parties send their inputs to the ideal functionality F.
    - Malicious parties do not have to send it.
  - If everybody sent something to F, it will compute the function f and send the outputs to the parties. Otherwise sends $\perp$ to everybody.
  - Honest parties output what they got. Malicious parties output whatever they like.
- In the Real model, two middle steps are replaced by the execution of the actual protocol.
- Real must be simulatable by ideal.

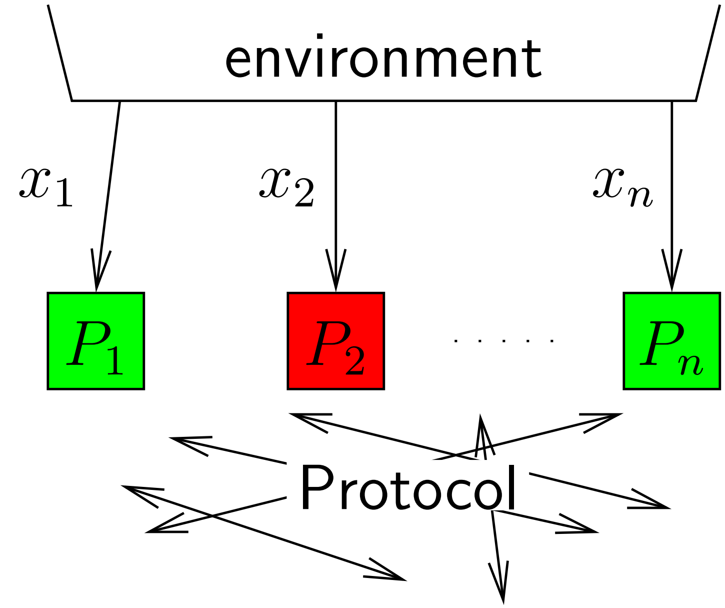
Malicious model — security definition



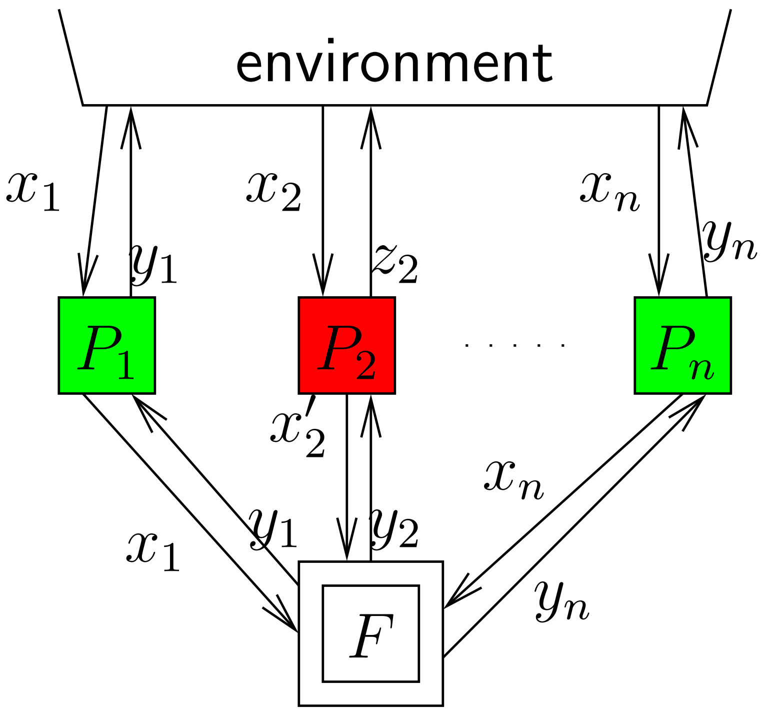


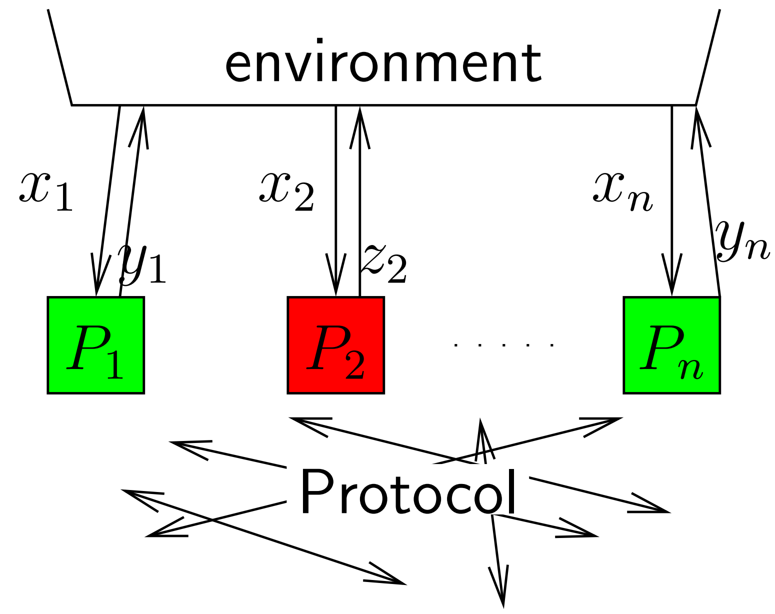
Malicious model — security definition





Malicious model — security definition





Malicious model — security definition

- There must exist a simulator rtoi that turns real parties to ideal parties.
  - $\mathsf{rtoi}(i, P_i^{\mathsf{real}})$ must equal P_i^{ideal}.
- For all $Q_1, \ldots, Q_n$, where $Q_i = P_i^{\mathsf{real}}$ for at least $n - t$ different values of i
- For all environments $\mathcal{Z}$: its views in the following two runs must be indistinguishable:
  - $\mathcal{Z} \mid Q_1 \mid \cdots \mid Q_n$
  - $\mathcal{Z} \mid \mathsf{rtoi}(1, Q_1) \mid \cdots \mid \mathsf{rtoi}(n, Q_n) \mid F$

Error-correcting codes

- An (n, t, d)-code over a set X is a mapping $\mathbf{C} : X^t \to X^n$, such that for all $x_1, x_2 \in X^t$, $x_1 \neq x_2$ implies that $\mathbf{C}(x_1)$ and $\mathbf{C}(x_2)$ differ in at least d positions.
- An element $x \in X^t$ is encoded as $y = \mathbf{C}(x) \in X^n$ and transmitted. During transmission, errors may occur in some positions of y.
- A (n, t, d)-code can detect at most $d - 1$ errors.
- A (n, t, d)-code can correct at most $(d - 1)/2$ errors.
- Efficiency is another question, though.

Error-correcting codes

- An (n, t, d)-code over a set X is a mapping $\mathbf{C} : X^t \to X^n$, such that for all $x_1, x_2 \in X^t$, $x_1 \neq x_2$ implies that $\mathbf{C}(x_1)$ and $\mathbf{C}(x_2)$ differ in at least d positions.
- An element $x \in X^t$ is encoded as $y = \mathbf{C}(x) \in X^n$ and transmitted. During transmission, errors may occur in some positions of y.
- A (n, t, d)-code can detect at most $d - 1$ errors.
- A (n, t, d)-code can correct at most $(d - 1)/2$ errors.
- Efficiency is another question, though.
- In a linear code, X is a field and $\mathbf{C}$ is a linear mapping between vector spaces X^t and X^n.
- For linear codes, $d \leq n - t + 1$.

Reed-Solomon codes

- Reed-Solomon codes are linear codes over some finite field $\mathbb{F}$.
- To encode t elements of $\mathbb{F}$ as n elements of $\mathbb{F}$, fix n different elements $c_1, \ldots, c_n \in \mathbb{F}$.
- Interpret the source word $(f_0, \ldots, f_{t-1})$ as a polynomial $p(x) = \sum_{i=1}^{t-1} f_i x^i$.
- Encode it as $(p(c_1), \ldots, p(c_n))$.
- For Reed-Solomon codes, $d = n - t + 1$.
- Hence they can correct up to $(n - t)/2$ errors.

Decoding Reed-Solomon codes

- Suppose that the original codeword was $(s_1, \ldots, s_n)$, corresponding to the polynomial p.
- But we received $(\tilde{s}_1, \ldots, \tilde{s}_n)$.
  - We assume it has at most $(n-t)/2$ errors.
- Find the coefficients for polynomials q_0 and q_1, such that
  - Degree of q_0 is at most $(n+t-2)/2$. Degree of q_1 is at most $(n-t)/2$.
  - For all $i \in \{1, \ldots, n\}$: $q_0(c_i) - q_1(c_i) \cdot \tilde{s}_i = 0$.
  - q_0 and q_1 are not both equal to 0.
- Then $p = q_0/q_1$.
- In general, there are more equations than variables, but $\tilde{s}_i$ are not arbitrary.

Correctness of decoding

Such polynomials q_0, q_1 exist:

- $(s_1, \ldots, s_n)$, $(\tilde{s}_1, \ldots, \tilde{s}_n)$ — original and received codewords. Let E be the set of i, where $s_i \neq \tilde{s}_i$. Then $|E| \leq (n-t)/2$.
- Let $k(x) = \prod_{i \in E}(x - c_i)$. Then $\deg k \leq (n-t)/2$.
- Take $q_1 = k$ and $q_0 = p \cdot k$. Then $\deg q_0 \leq (n+t-2)/2$.
- For all $i \in \{1, \ldots, n\}$ we have

$$q_0(c_i) - q_1(c_i) \cdot \tilde{s}_i = k(c_i)(p(c_i) - \tilde{s}_i) = k(c_i)(s_i - \tilde{s}_i) = \begin{cases} k(c_i)(s_i - s_i) = 0, & i \notin E \\ 0 \cdot (s_i - \tilde{s}_i) = 0, & i \in E \end{cases}$$

Correctness of decoding

If q_0 and q_1 satisfy the equalities and upper bounds on degrees, then $p = q_0/q_1$:

- Let $q'(x) = q_0(x) - q_1(x)p(x)$. Degree of q' is at most $(n + t - 2)/2$.
- For each $i \notin E$, $q'(c_i) = q_0(c_i) - q_1(c_i)p(c_i) = q_0(c_i) - q_1(c_i)\tilde{s}_i = 0$.
    - $1 \leq i \leq n$.
- The number of such i is at least $n - (n - t)/2 = (n + t)/2$.
- Thus the number of roots of q' is larger than its degree. Hence $q' = 0$.
- $q_0 - q_1 \cdot p = 0$.

MPC with no errors

- The number of corrupted players is at most $t - 1 < n/3$.
- To distribute inputs, each party first commits to his input and then shares the commitment.
- Shamir's scheme is used for both committing and sharing.
  - Hence the commitments are homomorphic.
  - For a value a, let $[a]_i$ denote the commitment of P_i to a. The commitment is distributed, hence $[a]_i = ([a]_i^1, \ldots, [a]_i^n)$, with P_j holding the piece $[a]_i^j$.

Commitments

We need the following functionalities:

- Commit: P_i commits to a value a.
  - $[a]_i$ is a sharing of a using (n, t)-secret sharing.
  - Followed by a proof that the degree of the polynomial is $\leq (t - 1)$.
- Open and OpenPrivate: opens a commitment.
  - Everybody broadcasts his share or sends it privately to the party that is supposed to open it.
  - Errors can be corrected.
- Linear Combination: several commitments of the same party (or different parties) are linearly combined.
  - Everybody performs the same combination on the shares he's holding.

Commitments

- Transfer: turns P_i's commitment $[a]_i$ into P_j's commitment $[a]_j$. Party P_j learns a.
  - OpenPrivate a for P_j.
  - P_j Commits a, giving $[a]_j$.
  - Find the Linear Combination $[a]_i - [a]_j$ and Open it; check that it is 0.
- Share: applies Shamir's secret sharing to a committed value $[a]_i$.
  - P_i generates the values $a_1, \ldots, a_{t-1}$ and Commits to them.
  - $s_i = a + \sum_{j=1}^{t-1} a_j i^j$. These Linear Combinations of $[a]_i$ and $[a_1]_i, \ldots, [a_{t-1}]_i$ are computed, resulting in commitments $[s_1]_i, \ldots, [s_n]_i$.
  - Commitment $[s_j]_i$ is Transfered to $[s_j]_j$.

Commitments

- Multiply. Given $[a]_i$ and $[b]_i$, the party P_i causes the computation of $[c]_i$, where $c = a \cdot b$.
  - Compute c and Commit to it.
  - Share $[a]_i$ and $[b]_i$, giving $[s_1^a]_1, \ldots, [s_n^a]_n$ and $[s_1^b]_1, \ldots, [s_n^b]_n$.
    - Let the polynomials be f^a and f^b.
  - Let $f^c(x) = f^a(x) \cdot f^b(x) = c + \sum_{j=1}^{2t-2} c_j x^j$. Party P_i Commits to $c_1, \ldots, c_{2t-2}$.
  - Compute $[f^c(1)]_i, \ldots, [f^c(n)]_i$ as Linear Combinations of $[c]_i$ and $[c_1]_i, \ldots, [c_{2t-2}]_i$.
  - OpenPrivate $[f^c(j)]_i$ to P_j. He checks that $s_j^a \cdot s_j^b = f^c(j)$. If not, broadcast complaint and Open $[s_j^a]_j$, $[s_j^b]_j$.
  - If P_j complains then P_i Opens $[f^c(j)]_i$. Either P_i or P_j is disqualified.

Exercise. Show that if P_i cheats then there will be a complaint.

MPC

- For each wire, the value on it is shared and the parties have commitments to those shares.
- Start: each party Commits to his input and then Shares it.
- Addition gates: Linear Combination is used to add the shares of values on incoming wires.
- Multiplication gates: the shares of the values on incoming wires are Multiplied together. These products are Shared and those shares are recombined into the shares of the product, using Linear Combination.
  - i.e. Gennaro-Rabin-Rabin multiplication is performed on committed shares.
- End: the shares of a value that a party is supposed to learn are Opened Privately to this party.

Commit: proving the degree of a polynomial

- P_i wants to commit to a value a using a random polynomial f, where $\deg f \leq t - 1$ and $f(0) = a$. A party P_j learns $[a]_i^j = f(j)$.
- P_i has to convince others that f has a degree at most $t - 1$.

Commit: proving the degree of a polynomial

- P_i wants to commit to a value a using a random polynomial f, where $\deg f \leq t-1$ and $f(0) = a$. A party P_j learns $[a]_i^j = f(j)$.
- P_i has to convince others that f has a degree at most $t-1$.
- P_i randomly generates a two-variable symmetric polynomial F, such that $F(x,0) = f(x)$ and the degrees of F with respect to x and y are $\leq (t-1)$. I.e.
  - randomly generate coefficients $c_{kl} \in \mathbb{F}$, where $1 \leq l \leq k \leq t-1$;
  - Let $c_{00} = a$. Let c_{i0} be the coefficient of x^i in f.
  - Let $c_{lk} = c_{kl}$ for $l \geq k$.
  - Let $F(x,y) = \sum_{k=0}^{t-1} \sum_{l=0}^{t-1} c_{kl} x^k y^l$.
- P_i sends to P_j the polynomial $F(x,j)$ (i.e. its coefficients). The share $[a]_i^j$ of P_j is $F(0,j) = F(j,0) = f(j)$.

Commit: proving the degree of a polynomial

- P_j and P_k compare the values $F(k, j)$ and $F(j, k)$. If they differ, they broadcast a complaint$\{j, k\}$.
- P_i answers to "complaint$\{j, k\}$" by publishing the value $F(j, k)$ (which is the same as $F(k, j)$).
- If P_j (or P_k) has a different value then he broadcasts "disqualify P_i".
- P_i responds to that by broadcasting $F(x, j)$.
- All parties P_l check that $F(l, j) = F(j, l)$. If not, broadcast "disqualify P_i". Again P_i responds by broadcasting $F(x, l)$, etc.
- If there are at least t disqualification calls then P_i is disqualified.
- Otherwise the commitment is accepted and parties update their shares with the values that P_i had broadcast.

Exercise. Show that if P_i is honest then the adversary does not learn anything beyond the polynomials $F(x, j)$, where P_j is corrupt.
Exercise. Show that if the commitment is accepted then the shares $[a]_i^j$ of honest parties are lay on a polynomial of degree $\leq (t-1)$.

Consistency of shares

- Let $\mathbf{B} \subseteq \{1, \ldots, n\}$ be the set of indices of honest parties. We must show that there exists a polynomial g of degree at most $t-1$, such that $g(j) = [a]_i^j = F(0, j)$ for all $j \in \mathbf{B}$.
- Let $\mathbf{C} \subseteq \mathbf{B}$ be the indices of honest parties that did not accuse the dealer. **Exercise.** How large must $\mathbf{C}$ be?
- **Exercise.** Show that for all $j \in \mathbf{B}$ and $k \in \mathbf{C}$ we have $F(j, k) = F(k, j)$ at the end of the protocol.
- Let r_k, where $k \in \mathbf{C}$ be the Lagrange interpolation coefficients for polynomials of degree $\leq t-1$. I.e. $h(0) = \sum_{k \in \mathbf{C}} r_k h(k)$ for all such polynomials h. **Exercise.** Why do such r_k exist?
- **Exercise.** Show that $g(x) = \sum_{k \in \mathbf{C}} r_k \cdot F(x, k)$ is the polynomial we're looking for.

Consistent broadcast

- There are n parties $P_1, \ldots, P_n$.
- A party P_i has a message m to broadcast.
- There are secure channels between each pair of parties.
- t of the parties $(t < n/3)$ are malicious.
- All honest parties must eventually agree on a broadcast message and the sender.
  - If P_i is honest then all honest parties must eventually agree that the message m was sent by P_i.
  - If P_i was malicious then all honest parties must eventually agree on the same message and a dishonest sender, or that there was no message.

Protocol for consistent broadcast

- Assume that a party never sends the same message twice.
- If P_i wants to broadcast m, it sends $(\textsc{Init}, P_i, m)$ to all other parties.
- If a party P_j receives $(\textsc{Init}, P_i, m)$ from party P_i then it sends $(\textsc{Echo}, P_i, m)$ to all parties (including himself).
- If a party P_j receives $(\textsc{Echo}, P_i, m)$ from at least $t + 1$ different parties, then it sends $(\textsc{Echo}, P_i, m)$ to all parties himself, too.
- If a party P_j receives $(\textsc{Echo}, P_i, m)$ from at least $2t + 1$ different parties then it *accepts* that P_i broadcast m.

Exercise. Show that if an honest P_i wants to broadcast m, then all honest parties have accepted it after two rounds.
Exercise. Show that if the honest party P_i has not broadcast m then no honest party will accept that P_i has broadcast m.
Exercise. Show that if an honest party accepts that P_i broadcast m, then all other honest parties will accept that at most one round later.

What have we seen so far?

- 2-party, computational, semi-honest, constant-round.
- 2- or n-party, computational, semi-honest($< n$), linear-round.
- n-party, unconditional, semi-honest($< n/2$), linear-round.
- n-party, computational, malicious($< n/2$), constant-round.
- n-party, unconditional (with $2^{-\eta}$ chance of failing), broadcast, malicious($< n/2$), linear-round.
- n-party, unconditional, malicious($< n/3$), linear-round.

Not covered yet:

- 2-party, computational, malicious.
- n-party, computational, malicious($< n$).

What have we seen so far?

- 2-party, computational, semi-honest, constant-round.
- 2- or n-party, computational, semi-honest($< n$), linear-round.
  - Linear in . . . of the circuit computing f.
  - **Exercise.** Fill the blank.
- n-party, unconditional, semi-honest($< n/2$), linear-round.
- n-party, computational, broadcast, malicious($< n/2$), linear-round.

What have we seen so far?

- 2-party, computational, semi-honest, constant-round.
- 2- or n-party, computational, semi-honest($< n$), linear-round.
  - Linear in . . . of the circuit computing f.
  - **Exercise.** Fill the blank.
- n-party, unconditional, semi-honest($< n/2$), linear-round.
- n-party, computational, broadcast, malicious($< n/2$), linear-round.

Exercise. How to implement a broadcast channel using only point-to-point channels in the computational setting, assuming a malicious adversary that has corrupted less than half of the parties?

What have we seen so far?

- 2-party, computational, semi-honest, constant-round.
- 2- or n-party, computational, semi-honest($< n$), linear-round.
  - Linear in . . . of the circuit computing f.
  - **Exercise.** Fill the blank.
- n-party, unconditional, semi-honest($< n/2$), linear-round.
- n-party, computational, broadcast, malicious($< n/2$), linear-round.

Exercise. How to implement a broadcast channel using only point-to-point channels in the computational setting, assuming a malicious adversary that has corrupted less than half of the parties?

Coming up: n-party, computational, malicious($< n/2$), constant-round.

Beaver-Micali-Rogaway's MPC

- Recall Yao's garbled circuits:
  - P_1 coverts the circuit evaluating f to a garbled circuit.
  - P_1 sends to P_2 the garbled circuit and keys corresponding to his(P_1) input bits.
  - P_2 obtains the keys corresponding to his input bits using oblivious transfer.
  - P_2 evaluates the circuit and reports back (to P_1) the result.
- In Micali-Rogaway's MPC, the garbled circuit and keys corresponding to all parties' inputs are produced cooperatively.
  - All gates can be garbled in parallel — need only constant rounds.
- After that, all parties evaluate that circuit by themselves.

Rabin's and Ben-Or's VSS

(MPC: n-party, unconditional (with small chance of failing), broadcast, malicious($< n/2$), linear-round)

- An interactive VSS.
  - Sharing and recovery protocols involve more communication between parties.
- Unconditionally secure.
- Has a small error probability (of the order $2^{-\eta}$), where η is the security parameter.
  - Has a flavor of zero-knowledge proofs.

Rabin's and Ben-Or's VSS

(MPC: n-party, unconditional (with small chance of failing), broadcast, malicious($< n/2$), linear-round)

- An interactive VSS.
  - Sharing and recovery protocols involve more communication between parties.
- Unconditionally secure.
- Has a small error probability (of the order $2^{-\eta}$), where η is the security parameter.
  - Has a flavor of zero-knowledge proofs.
- Let $p \in \mathbb{P} \cap \{n+1, \ldots, 2n\}$. Let $p' \geq 2^{\eta}$ be a large prime, such that $p \mid (p'-1)$.

Check vectors

- A bit like signatures. . .
- Three parties — Dealer, Intermediary, Recipient.
- D gives to I the $v \in \mathbb{Z}_{p'}$. I may later want to pass v to R.
- D is honest.
- R wants to be sure that the value he received is really v.

Check vectors

- A bit like signatures...
- Three parties — Dealer, Intermediary, Recipient.
- D gives to I the $v \in \mathbb{Z}_{p'}$. I may later want to pass v to R.
- D is honest.
- R wants to be sure that the value he received is really v.
- D generates random values $b \in \mathbb{Z}_{p'}^*$ and $y \in \mathbb{Z}_{p'}$. Let $c = v + by$.
- D sends (v, y) to I and (b, c) to R.
- Later, I sends (v, y) to R who verifies that $c = v + by$.

Exercise. Security? Can R learn v too soon? Can I send a wrong value to R? What if there are several R-s (the check vectors are different)?

Honest-dealer VSS

- D generates random $f(x) = v + \sum_{i=1}^{t-1} a_i x^i$ and sends $s_i = f(i)$ to P_i.
- For each s_i and P_j, the dealer sends the check vector (b_{ij}, c_{ij}) to P_j and the corresponding y_{ij} to P_i.
- To recover v, P_i sends (s_i, y_{ij}) to P_j (for all i and j). The parties verify the check vectors. To reconstruct v, they use those shares that passed verification.

Check vectors with malicious dealer

- If D is dishonest then the proof y sent to I might not match the check vector (b, c) sent to R.
- I, when receiving (v, y), wants to be sure that R will accept his (v, y) afterwards.

Check vectors with malicious dealer

- If D is dishonest then the proof y sent to I might not match the check vector (b, c) sent to R.
- I, when receiving (v, y), wants to be sure that R will accept his (v, y) afterwards.
- D will generate 2η check vectors $(b_1, c_1), \ldots, (b_{2\eta}, c_{2\eta})$ and send them to R. He sends the corresponding values $y_1, \ldots, y_{2\eta}$ to I.
- I randomly chooses η indices $i_1, \ldots, i_\eta$ and sends them to R.
  - Let $\tilde{i}_1, \ldots, \tilde{i}_\eta$ be the other η indices.
- R sends $(b_{i_1}, c_{i_1}), \ldots, (b_{i_\eta}, c_{i_\eta})$ to I.
- R verifies that $c_{i_j} = v + b_{i_j} y_{i_j}$ for all j. If all checks out, then I thinks that R will accept.
- Later, I sends $(v, y_{\tilde{i}_1}, \ldots, y_{\tilde{i}_\eta})$ to R. R verifies all remaining check vectors. He accepts if at least one check vector is correctly verified.

Check vectors with malicious dealer

- If D is dishonest then the proof y sent to I might not match the check vector (b, c) sent to R.
- I, when receiving (v, y), wants to be sure that R will accept his (v, y) afterwards.
- D will generate 2η check vectors $(b_1, c_1), \ldots, (b_{2\eta}, c_{2\eta})$ and send them to R. He sends the corresponding values $y_1, \ldots, y_{2\eta}$ to I.
- I randomly chooses η indices $i_1, \ldots, i_\eta$ and sends them to R.
  - Let $\tilde{i}_1, \ldots, \tilde{i}_\eta$ be the other η indices.
- R sends $(b_{i_1}, c_{i_1}), \ldots, (b_{i_\eta}, c_{i_\eta})$ to I.
- R verifies that $c_{i_j} = v + b_{i_j} y_{i_j}$ for all j. If all checks out, then I thinks that R will accept.
- Later, I sends $(v, y_{\tilde{i}_1}, \ldots, y_{\tilde{i}_\eta})$ to R. R verifies all remaining check vectors. He accepts if at least one check vector is correctly verified.
- **Exercise.** What is the probability that R rejects, although I thought he would accept?
- **Exercise.** What is the probability that R will accept a value different from v?

Verified-at-the-end VSS

- In Verified-at-the-end VSS, a malicious dealer is caught during the recovery protocol.
- Also, the dealer cannot change his mind after the sharing protocol.
- The sharing protocol has two phases:
  - Sharing the secret.
  - Verifying the check vectors.

Sharing the secret

- Dealer randomly generates the polynomial $f(x) = v + \sum_{j=1}^{t-1} a_i x^i$ and sends the share $s_i = f(i)$ to each P_i.
- Dealer generates the check vectors $(\mathbf{b}_{ij}, \mathbf{c}_{ij})$ and the proofs $\mathbf{y}_{ij}$ for s_i. Sends the vector to P_j and proof to P_i.
  - Each of $\mathbf{b}_{ij}$, $\mathbf{c}_{ij}$, $\mathbf{y}_{ij}$ is actually a 2η-tuple of elements of $\mathbb{Z}_{p'}$.

Verifying the check vectors

- P_i wants to know whether P_j will accept his proof $\mathbf{y}_{ij}$.
- On the broadcast channel P_i asks P_j to publish η components of the check vector $(\mathbf{b}_{ij}, \mathbf{c}_{ij})$. Components are chosen by P_i.
- P_j does so (on broadcast channel).
- The dealer has two options:
  - Broadcast "I approve".
  - Broadcast a new $(\mathbf{b}_{ij}, \mathbf{c}_{ij})$ and send the corresponding new $\mathbf{y}_{ij}$ privately to P_i.
- Party P_i verifies the (received components of) the check vector.
  - If OK, move on to P_{j+1}.
  - If not OK, ask the dealer to broadcast s_i. Do not move on.
    - The value broadcast by dealer is taken as s_i by all parties.

Exercises

- Show that this part of the protocol does not expose data that is not known to dishonest parties (except for halves of check vectors).
- At this point, let a coalition be a set of parties $\mathbf{C} \subseteq \{P_1, \ldots, P_n\}$, such that for all $P, P' \in \mathbf{C}$, party P knows that P' will accept his share during recovery. Show that there is a coalition containing all honest parties.
  - A broadcast share is always accepted.

Recovery protocol

- D broadcasts the (coefficients of the) polynomial f.
- Each P_i sends to each P_j his share s_i and the proof $\mathbf{y}_{ij}$.
  - If the share s_i was broadcast then P_i does nothing.
- Each P_i verifies each received $(s_j, \mathbf{y}_{ji})$ with respect to the check vector $(\mathbf{b}_{ji}, \mathbf{c}_{ji})$ that he has.
- Each P_i verifies whether $f(j) = s_j$ for each share s_j that he accepted on the previous step.
- If this check succeeds for all accepted s_j, then P_i takes $f(0)$ as the secret v.
- If this check does not succeed for some accepted s_j then P_i broadcasts "dealer is malicious".
- A dealer whose maliciousness gets at least t votes is disqualified.

Exercises

- Show that all honest parties will arrive at the same value of the secret v.
- Show that an honest dealer is not disqualified.

Unconditionally secure VSS

- Here, during the dealing protocol, the dealer gives zero-knowledge proof that f has degree at most $\leq t-1$.
- In the beginning, D sends out the shares s_i as always.
  - No check vectors are necessary.
- Each P_i will use (n,t)-Verified-at-the-end VSS to share s_i. After that, each honest party P_i will have
  - His share s_i.
  - A polynomial f^i of degree at most $t-1$, such that $f^i(0) = s_i$.
  - The share β_i^j of s_j at point i. If P_j is honest then $\beta_i^j = f^j(i)$.
  - A check vector $(\mathbf{b}_{ki}^j, \mathbf{c}_{ki}^j)$ allowing P_i to verify that the share β_k^j is a correct share of s_j for party P_k.
  - A proof $\mathbf{y}_{ik}^j$ allowing P_i to prove to P_k that his share β_i^j is a correct share of s_j for party P_i.
  - Belief that all other parties accept the shares β_i^j that he is holding. (Everybody will accept β_i^j if it has been broadcast.)

The ZK proof

- Dealer picks a random polynomial f of degree $\leq t-1$.
- Dealer sends $s_i = f(i)$ to P_i.
- Each P_i will use (n,t)-Verified-at-the-end VSS to share s_i. After that, each honest party P_i will have f^i, β_i^j, $(\mathbf{b}_{ki}^j, \mathbf{c}_{ki}^j)$, $\mathbf{y}_{ik}^j$.
- Each P_i also shares $s_i = s_i + s_i$ using the polynomial $f^i = f^i + f^i$.
  - The check vectors $(\mathbf{b}_{ki}^j, \mathbf{c}_{ki}^j)$ and proofs $\mathbf{y}_{ik}^j$ are independently created and verified.
- One of the parties P_i (chosen in round-robin manner) asks the dealer to reveal either f or $f = f + f$.
- Dealer reveals f. Each P_i checks whether $f(i) = s_i$.
  - If unsatisfied, asks the dealer to broadcast s_i and s_i.
  - Dealer complies. Each P_j checks that $f(i) = s_i$.
- For each i, the parties run the recovery protocol of Verified-at-the-end VSS for s_i shared with f^i. Each P_j checks if $s_i = f(i)$. If not, disqualify P_i.

Exercises

- Show that no data unknown to the adversary is broadcast.
- Show that an honest party is not disqualified.
- Show that after $O(\eta)$ rounds, all values s_i that have been broadcast or that are held by still qualified players lay on the same polynomial of degree at most $t-1$.

Recovery of v

- The recovery protocols of Verified-at-the-end VSS are run for still hidden shares s_i.
- These shares are used to reconstruct f.

The VSS has the following properties:

- If the dealer is honest then he won't be disqualified.
- After the ZK proof (all rounds of which can be run in parallel), the secret value v has been uniquely determined for all honest parties.
  - It is also determined whether the recovery protocol will produce a v or not.
  - The dealer will not be disqualified during the recovery.

Summary

- The secret is shared with Shamir's scheme.
- Each share is shared with Shamir's scheme.
- Each share^2 created by P_i for P_j has check vectors for each P_k.
- P_j is sure that P_k will accept this check vector.
- A ZK-style proof is given that the shares lay on a polynomial of degree at most $\leq (t-1)$.
  - A random polynomial of degree $\leq (t-1)$ is generated and shared and shared^2 together with check vectors.
  - Either the random polynomial or (original+random) polynomial is opened.
  - The check vectors are used to catch malicious parties P_i.
  - Comparision of shares and opened polynomial is used to catch malicious D.
- During the recovery, D does not matter any more.

MPC with Rabin's and Ben-Or's VSS

- For each wire, the value it is carrying is distributed using the VSS.
- The inputs are shared using the VSS. The outputs are recovered using the VSS.
- Adding two wires ($v = v + v$):
  - $s_i = s_i + s_i$. $f^i = f^i + f^i$. $\beta_i^j = \beta_i^j + \beta_i^j$.
  - P_i sends to P_k the new check vector $(\mathbf{b}_{jk}^i, \mathbf{c}_{jk}^i)$ and to P_j the corresponding proof $\mathbf{y}_{jk}^i$. P_j verifies that P_k will accept this proof for β_j^i.
  - **Exercise.** Why not reuse the existing check vectors?
- Multiplying with a constant ($v = cv$):
  - $s_i = cs_i$. $f^i = cf^i$. $\beta_i^j = c\beta_i^j$.
  - $\mathbf{b}_{ki}^j = c \cdot \mathbf{b}_{ki}^j$. $\mathbf{c}_{ki}^j = c \cdot \mathbf{c}_{ki}^j$. $\mathbf{y}_{ik}^j = \mathbf{y}_{ik}^j$.
    - Recall that $\mathbf{c}_{ik}^j[z] = \beta_i^j + \mathbf{b}_{ik}^j[z] \cdot \mathbf{y}_{ik}^j[z]$.

Multiplication ($v = v \cdot v$)

- Verified-at-the-end sharings of s_i and s_i are extended to fully verified sharings.
  - All shares[2] β_i^j and β_i^j are shared using the verified-at-the-end sharing scheme, giving us shares[3] γ_k^{ji} and γ_k^{ji} and corresponding check vectors and proofs.
  - ZK-proof is given that all shares β_j^i lay on a polynomial of degree at most $t-1$.
    - Presumably, this polynomial is f^i.
  - Same for β and f.
- Each party P_i shares $s_i = s_i \cdot s_i$ using full VSS.
- Each party P_i proves in ZK that $s_i = s_i \cdot s_i$.
  - Next slides. . .
- v is computed as a suitable linear combination of $s_1, \ldots, s_n$.

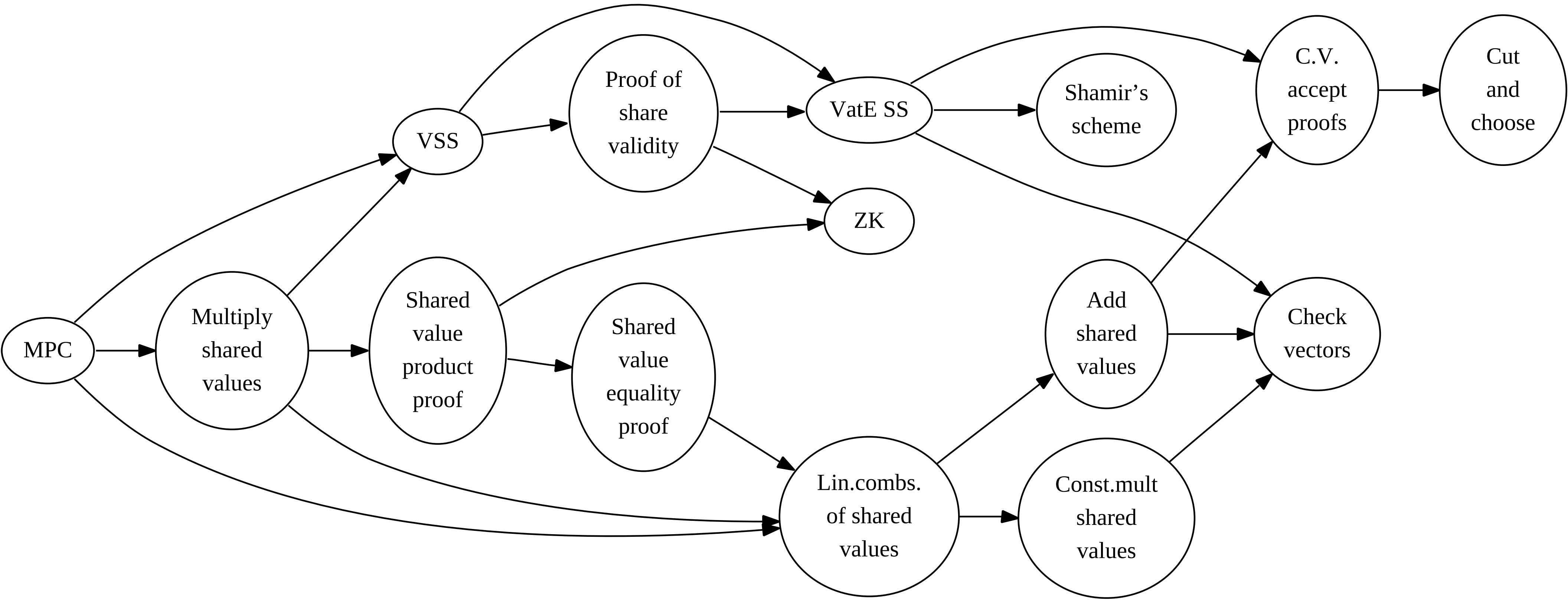
Proving that $v = v$

- The dealer has shared v and v.
- Use MPC to compute $v - v$.
- Recover the shared value. Check that it is 0.

Proving that $v = v \cdot v$

- Recall that we compute in a field $\mathbb{Z}_p$, where $n < p \leq 2n$ (except check vectors).
- The dealer has shared v, v and v.
- The dealer shares the entire multiplication table of $\mathbb{Z}_p$.
  - Let $\mathbf{T} = \{(x, y, z) \,|\, x, y \in \mathbb{Z}_p, z = xy\}$.
  - Let $(x_1, y_1, z_1), \ldots, (x_{p^2}, y_{p^2}, z_{p^2})$ be randomly permuted $\mathbf{T}$.
  - Dealer shares all x_i, y_i, z_i using full VSS.
- One of the P_i (chosen by round-robin) requests one of:
  - Open the entire table. Everybody checks that it was indeed the multiplication table of $\mathbb{Z}_p$.
  - Show the line (v, v, v). The dealer names $i \in \{1, \ldots, p^2\}$ and proves that $v = x_i$, $v = y_i$, $v = z_i$.

Components of Rabin's and Ben-Or's MPC



Homomorphic encryption systems

- Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an IND-CPA-secure public-key encryption system. Let the plaintext space R be a ring.
- $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is homomorphic, if there exist efficient algorithms
  - to compute $\mathcal{E}_k(a + b)$ from $\mathcal{E}_k(a)$ and $\mathcal{E}_k(b)$;
  - to compute $\mathcal{E}_k(ca)$ from $\mathcal{E}_k(a)$ and $c \in R$.

Paillier's cryptosystem

- Setup: generate large primes p and q.
- Public key: $N = pq$.
- Private key: $sk = \phi(N) \cdot (\phi(N)^{-1} \bmod N)$.
  - Note that $sk \equiv 1 \pmod{N}$.
- Encryption of $m \in \mathbb{Z}_N$ is $(1+N)^m \cdot r^N$ in $\mathbb{Z}_{N^2}$, where r is a random element of $\mathbb{Z}_N$.
  - The order of $1+N$ in N in $\mathbb{Z}_{N^2}$:
  - $(1+N)^N = 1 + \binom{N}{1}N + \binom{N}{2}N^2 + \cdots$. Every element in this sum, starting from the second, is divisible by N^2.
  - $(1+N)^p = 1 + pN + \binom{p}{2}N^2 + \cdots = 1 + pN$ in $\mathbb{Z}_{N^2}$. This does not equal 1.
  - Actually, $(1+N)^m = 1 + mN$ in $\mathbb{Z}_{N^2}$.
- To decrypt $c \in \mathbb{Z}_{N^2}$, first compute c^{sk} in $\mathbb{Z}_{N^2}$.

Decrypting

(computation in $\mathbb{Z}_{N^2}$)

$$((1+N)^m r^N)^{sk} = (1+mN)^{sk} \cdot ((r^N)^{\phi(N)})^{\phi(N)^{-1} \bmod N} = (1+mN)^{sk} = 1 + sk \cdot m \cdot N$$

- From this we find $sk \cdot m$ in $\mathbb{Z}_{N^2}$. Casting it into $\mathbb{Z}_N$ (divide by N and take the remainder) gives us m.

MPC from threshold homomorphic cryptosystem

- Assume that the keys have been distributed:
  - everybody knows pk;
  - each party P_i knows his secret key share sk_i.
  - At least t parties out of n must help to decrypt.
- The function f is represented by a circuit of addition, scalar multiplication, and multiplication gates.
- A value v on a wire is represented by $\mathcal{E}_{pk}(m)$.
  - All parties know $\mathcal{E}_{pk}(m)$.
  - Sharing of an input: encrypt it and broadcast the result.
  - Opening an output: at least t parties help to decrypt the value on output wire.
- Addition and scalar multiplication — every party performs the operation with the encrypted value(s) by itself.

Multiplying a and b

- Let $\mathcal{E}_{pk}(a)$ and $\mathcal{E}_{pk}(b)$ be known to everybody.
- Each party P_i chooses a random $d_i \in \mathbb{Z}_N$.
- P_i broadcasts $\mathcal{E}_{pk}(d_i)$ and $\mathcal{E}_{pk}(d_i b)$.
- Everybody computes $\mathcal{E}_{pk}(a + \sum_{i=1}^{n} d_i)$.
- This ciphertext is decrypted, everybody learns $a + \sum_{i=1}^{n} d_i$.
- Everybody computes $\mathcal{E}_{pk}((a + \sum_{i=1}^{n} d_i) \cdot b - \sum_{i=1}^{n} d_i b)$.

- This protocol can be made secure against malicious adversaries.

Threshold RSA

- n parties, at least t needed to decrypt.
- Primes p, q, public modulus $N = pq$, public exponent e, secret exponent $d = e^{-1} \bmod \phi(N)$.
- A dealer chooses all of those values.
  - Let e be a prime that is larger than n.
- The dealer shares d using Shamir's t-out-of-n secret sharing, working in $\mathbb{Z}_{\phi(N)}$. It sends the i-th share s_i to the party P_i.
  - For any set $\mathbf{C} \subseteq \{1, \ldots, n\}$, where $|\mathbf{C}| = t$, there exist coefficients $\tilde{r}_i^{\mathbf{C}}$, such that $d = \sum_{i \in \mathbf{C}} \tilde{r}_i^{\mathbf{C}} s_i$.
  - But finding such $\tilde{r}_i^{\mathbf{C}}$ requires the knowledge of $\phi(N)$.
  - There are public coefficients $r_i^{\mathbf{C}}$, such that $n! \cdot d = \sum_{i \in \mathbf{C}} r_i^{\mathbf{C}} s_i$.

Decryption

- Publicly decrypting $m^e = c \in \mathbb{Z}_N$: each party P_i publishes $m_i = c^{s_i} \bmod N$.
- Given a set of plaintext shares m_i, where $i \in \mathbf{C}$, compute c' by

$$c' = \prod_{i \in \mathbf{C}} m_i^{r_i^{\mathbf{C}}} \quad .$$

- $c' = m^{n!}$. As $n! \perp e$, there exist (public) coefficients $a, b \in \mathbb{Z}$, such that $ae + b(n!) = 1$.
- Compute $m = c^a + c'^b$.
- Threshold Paillier is doable in the same way.