General Secure Multiparty computation

We know how to do 2-Party Private Computation (using Oblivious Transfer)
↳ base case (K=2).

TTP

A — B

Any one node is corrupted (under adversary control)
↳ (Read only control)
↳ no issues [doesn't know computation]

Assume that we can make a TTP for K nodes, even if upto K-1 nodes are corrupted, adversary would not know the computation

1 2 - - - - K

TTPK

For K+1

1 2 - - - - K K+1

TTPK

TTP (2 Party Priv.)

Need to show that if atleast one node is not corrupted, computation is secure

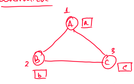
if node (K+1) is secure
⟶ TTP is secure (1 node of 2PPC is not corrupted)

if n ∈ [1, ---- K] is secure.
⟶ TTPK is secure (By induction)
⟶ TTP is secure (1 node of 2PPC is not corrupted)

if we have 3 nodes, if 2 are corrupted, we can create TTP using PKC (Oblivious transfer shown before)

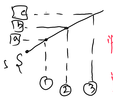
However if just 1 node is corrupted, we do not need PKC, can just use (Decentralize)
(In general, for t faults n ≥ 2t does not apply)

Decentralize



XOR is over K bits (even if 2 values among a,b,c are known we still do not know TTP value)

instead create a line (at random) through (1,A), (2,b), (3,C) [TTP value is y-intercept]
or more specifically, just consider a line with TTP value = s as intercept



if you know any 2 pts. we can find intercept (s)
if we know just one pt we cannot find s

Shamir's Secret Sharing.

n shares

up to t of them should not reveal anything about secret s (in TTP)

p(x) ⟶ random t-degree polynomial (with coefficients ∈ F > n)
p(0) = s
p(i) = s_i ∀ i ∈ [1, n]

Building Instruction Set (For addⁿ & multiplicⁿ)

Addⁿ
z ← x + y works

Assume underlying polynomial of x_i = p
i.e p(0) = x.
p(1) = x_1
⋮
p(i) = x_i

Assume underlying polynomial of y_i = q
q(0) = y
q(1) = y_1
⋮
q(i) = y_i

∴ for f = p+q. of degree t s.t
r(0) = x + y.
r(1) = x_1 + y_1
⋮
r(i) = x_i + y_i

∴ Directly Adding two secrets is fine.

Scalar Multiplicⁿ

z ← λx. works
underlying poly of x ⟶ p
∴ ∃ r = λp s.t
r(0) = λx
r(1) = λx_1
⋮
r(i) = λx_i

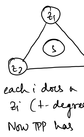
Multiplicⁿ
z_i ← x_i * y_i still works
underlying polynomial of x ⟶ p
" " " y ⟶ q

r = p*q

r(0) = p(0) * q(0) = x * y
r(1) = p(1) * q(1) = x_1 * y_1
⋮
r(i) = p(i) * q(i) = x_i * y_i

However this creates a 2t degree polynomial
⇓
need to convert it t-degree polynomial
not possible unless (n > 2t)

Protocol (no clue wtf i am writing)



λ_i = public.
s = xy = $\sum \lambda_i z_i$

each i does a secret sharing with TPP and shares z_i (t-degree secret sharing)
Now TPP has z_i ⟶ can calculate $\sum \lambda_i z_i$ to store (s)
↳ z_i = t degree
∴ s = t degree.