MIT 6.875

Foundations of Cryptography
Lecture 23

Security against Malicious (Active) Adversaries

Secure Two-Party Comp: New Def

(possibly randomized) $F(x, y; r) = (F_A(x, y; r), F_B(x, y; r))$

Input: $\boldsymbol{x}$ **Input:** $\boldsymbol{y}$

Alice Bob

There exists a PPT simulator SIM_A such that for any x and y:

$$\left(SIM_A\left(x, F_A(x, y)\right), F(x, y)\right) \cong \left(View_A(x, y), F(x, y)\right)$$

i.e. the joint distribution of the view and the output is correct

Counterexample

Randomized functionality $F(1^n, 1^n) = (r, \perp)$.

Protocol:

Alice picks a random r, outputs it **and sends it to Bob**.

Is this secure?

Secure acc. to old def, insecure acc. to new def.

Ergo, old def is insufficient.

Issues to Handle

1. Input (In)dependence: A malicious party could choose her input to depend on Bob's, something she cannot do in the ideal world.

Example (on the board): $F\big((a,b),x\big) = (\perp, ax+b)$

2. Randomness: A malicious party could choose her "random string" in the protocol the way she wants, something she cannot do in the ideal world.

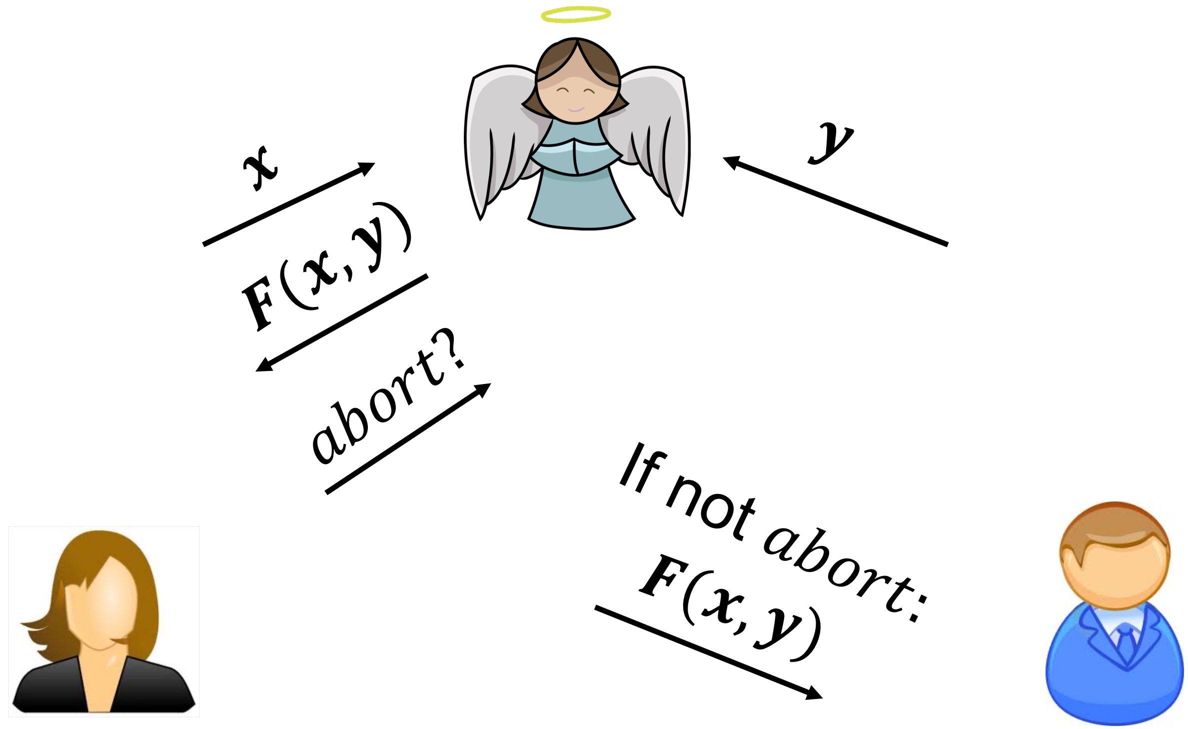
Example (on the board): our OT *protocol*

unavoidable

3. (Un)fairness: A malicious party could block the honest party from learning the output, while learning it herself.

4. Deviate from Protocol Instructions.

New (Less) Ideal Model



The “GMW Compiler”

Theorem **[Goldreich-Micali-Wigderson’87]**:
Assuming one-way functions exist, there is a general way to transform any semi-honest secure protocol computing a (possibly randomized) function F into a maliciously secure protocol for F.

Input Independence

1. Input (In)dependence: A malicious party could choose her input to depend on Bob's, something she cannot do in the ideal world.

Solution: Each party commits to their input in sequence, and provides a **zero-knowledge proof of knowledge** of the underlying input.

Solution: Coin-Tossing Protocol

2. Randomness: A malicious party could choose her "random string" in the protocol the way she wants, something she cannot do in the ideal world.

Def: Realize the functionality $F(1^n, 1^n) = (r, Com(r))$.

$$\xrightarrow{\quad Com(r_1) \quad}$$

$$\xleftarrow{\quad r_2 \quad}$$

Output $r = r_1 \oplus r_2$

Output $(Com(r_1), r_2)$

Zero Knowledge Proofs

4. Deviate from Other Protocol Instructions.

Solution: Each message of each party is a *deterministic* function of their input, their random coins and messages from party B.

When party A sends a message $m = m(x_A, r_A, \overline{msg_B})$, they also prove in zero-knowledge that they did so correctly. That is, they prove in ZK the following NP statement:

$$(m, \overline{msg_B}, XCom, RCom)\text{: } \exists\, x_A, r_A \text{ s.t.}$$
$$m = m(x_A, r_A, \overline{msg_B}) \ \wedge XCom = \mathrm{Com}(x_A) \wedge$$
$$RCom = \mathrm{Com}(r_A)$$

Optimizations

Optimization 1: Preprocessing OTs

Random OT tuple (or AND tuple, or Beaver tuple after D. Beaver): Alice has (α, γ_a) and Bob has (β, γ_b) which are random s.t. $\boldsymbol{\gamma_a \oplus \gamma_b = \alpha\beta}$.

Theorem: Given O(1) many *random* OT tuples, we can do OT with information-theoretic security, exchanging O(1) bits.

Optimization 2: OT Extension

Theorem
[Beaver'96, Ishai-Kushilevitz-Nissim-Pinkas'03]:

Given O(λ) many *random* OT tuples, we can generate n OT tuples exchanging O(n) bits --- as opposed to the trivial O($n\lambda$) bits --- and using only symmetric-key crypto.

Complexity of the 2-party solution

Number of OT protocol invocations $= 2 \, * \, \#AND$ gates

Can be made into O(#inputs · λ): Yao's garbled circuits

Number of rounds = AND-depth of the circuit

Can be made into O(1) rounds: Yao's garbled circuits

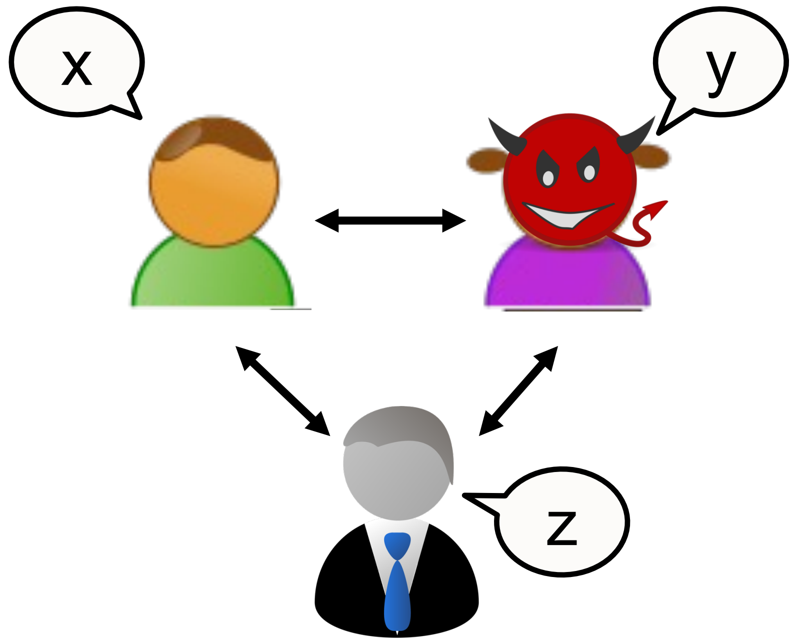
Communication in bits =

$$O(\#AND \cdot \lambda + \#outputs)$$

Can be made into O(λ #inputs) using FHE: but FHE is computationally more expensive concretely.

Secure Multi Party Computation w/ Information-theoretic security

Secure *Multi*-party Computation

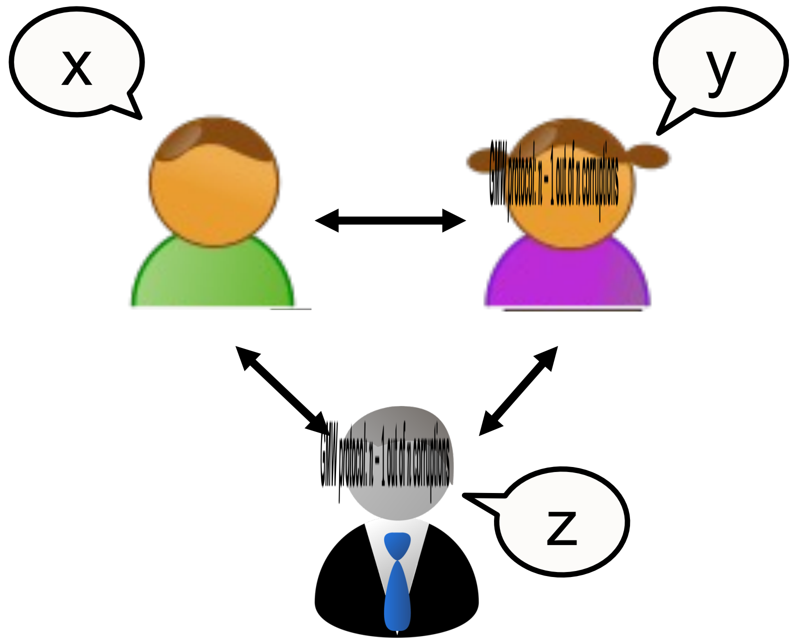


***TODAY*: HONEST MAJORITY**

Information-theoretically Secure Protocols for n parties with $< n/2$ corruptions

[BenOr-Goldwasser-Wigderson'88, Chaum-Crepeau-Damgard'88, BenOr-Rabin'89]

Secure *Multi*-party Computation

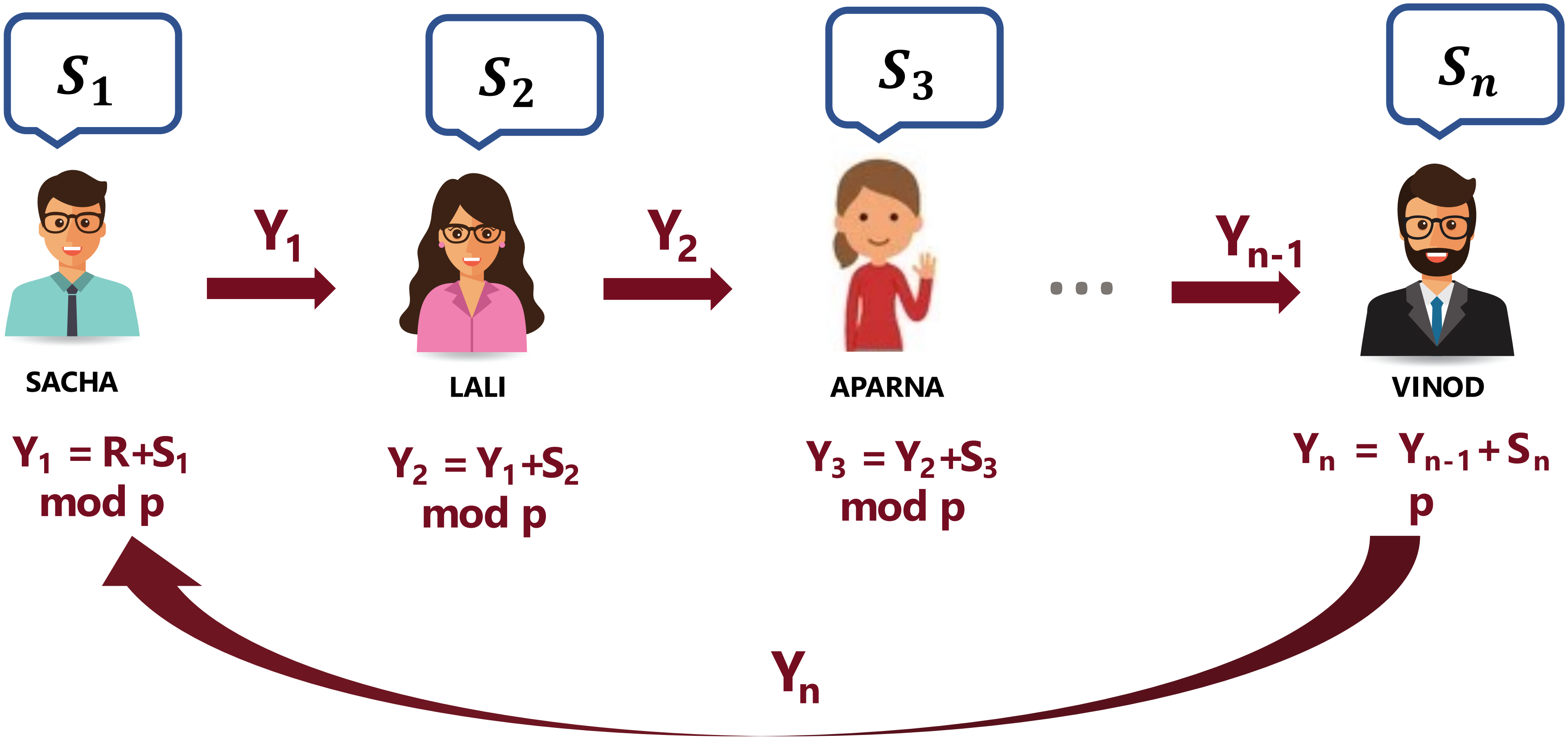


***GMW protocol*: $n-1$ out of n corruptions**

The Goldreich-Micali-Wigderson Protocol for n parties with $< n$ corruptions, using Oblivious Transfer
(which can only be *computationally* secure)

AN EXAMPLE

COMPUTING THE AVERAGE SALARY IN THIS ROOM



$$Y_n - R = \sum_{i=1}^{n} S_i \text{ mod p} = \sum_{i=1}^{n} S_i$$

(if p large enough)

Is this secure?

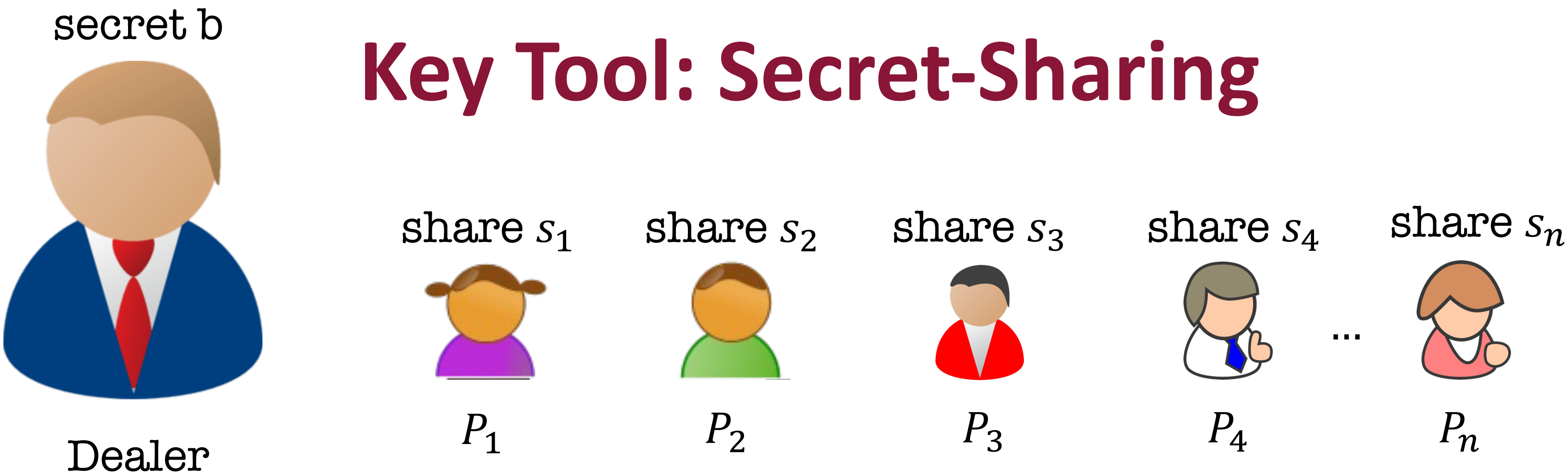
IT-Secure MPC with Honest Majority

***Theorem* [BenOr-Goldwasser-Wigderson'88, Chaum-Crepeau-Damgard'88]**:

Any n-party computation problem can be solved with information-theoretic security as long $< \frac{n}{2}$ parties collude.

***Key Tool:* Shamir's Secret Sharing**

Key Tool: Secret-Sharing



- ❑ Any **"authorized"** subset of players **can recover** b.
- ❑ No other subset of players **has any info** about b.

- ○ Threshold (or t-out-of-n) SS [Shamir'79, Blakley'79]:

  "authorized" subset = has size $\geq$ t.

secret b $\in Z_p$

n-out-of-n Secret Sharing

Dealer

P_1

P_2

P_3

P_4

...

P_n

share s_1: random

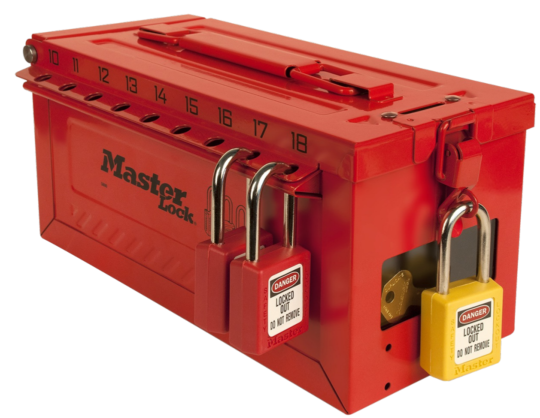
share s_2: random

share s_3: random

share s_4: random

...

share $s_n = b - (s_1 + s_2 + \cdots + s_{n-1})$ mod p



secret b $\in Z_p$

1-out-of-n Secret Sharing

Dealer

P_1 P_2 P_3 P_4 … P_n

share s_1 = b

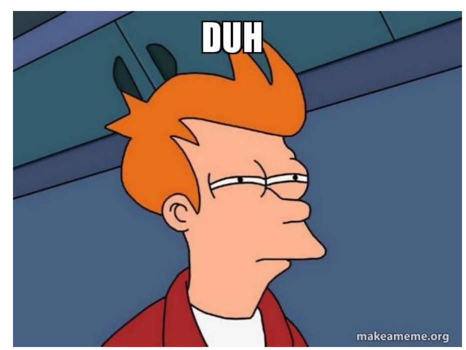
share s_2 = b

share s_3 = b

share s_4 = b

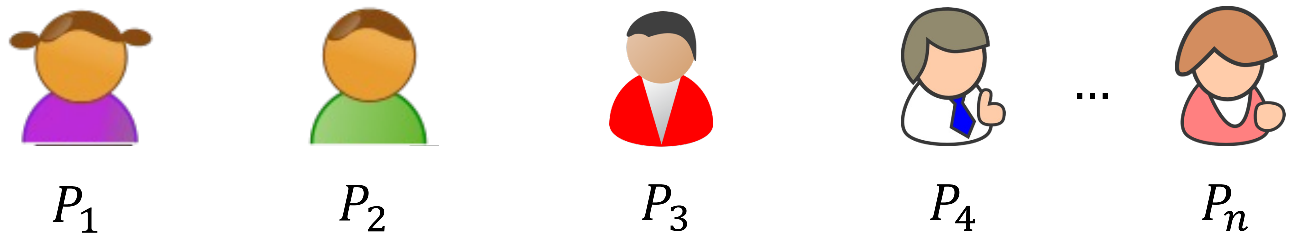
…

share $s_n = b$



secret b $\in Z_p$

2-out-of-n Secret Sharing?



Here is a solution.

Repeat for every two-person subset $\{P_i, P_j\}$: • Generate a 2-out-of-2 secret sharing (s_i, s_j) of b. • Give s_i to P_i and s_j to P_j

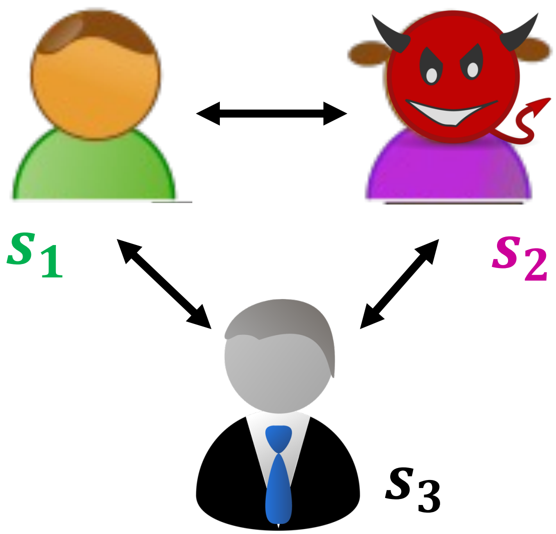
What is the size of shares each party gets?

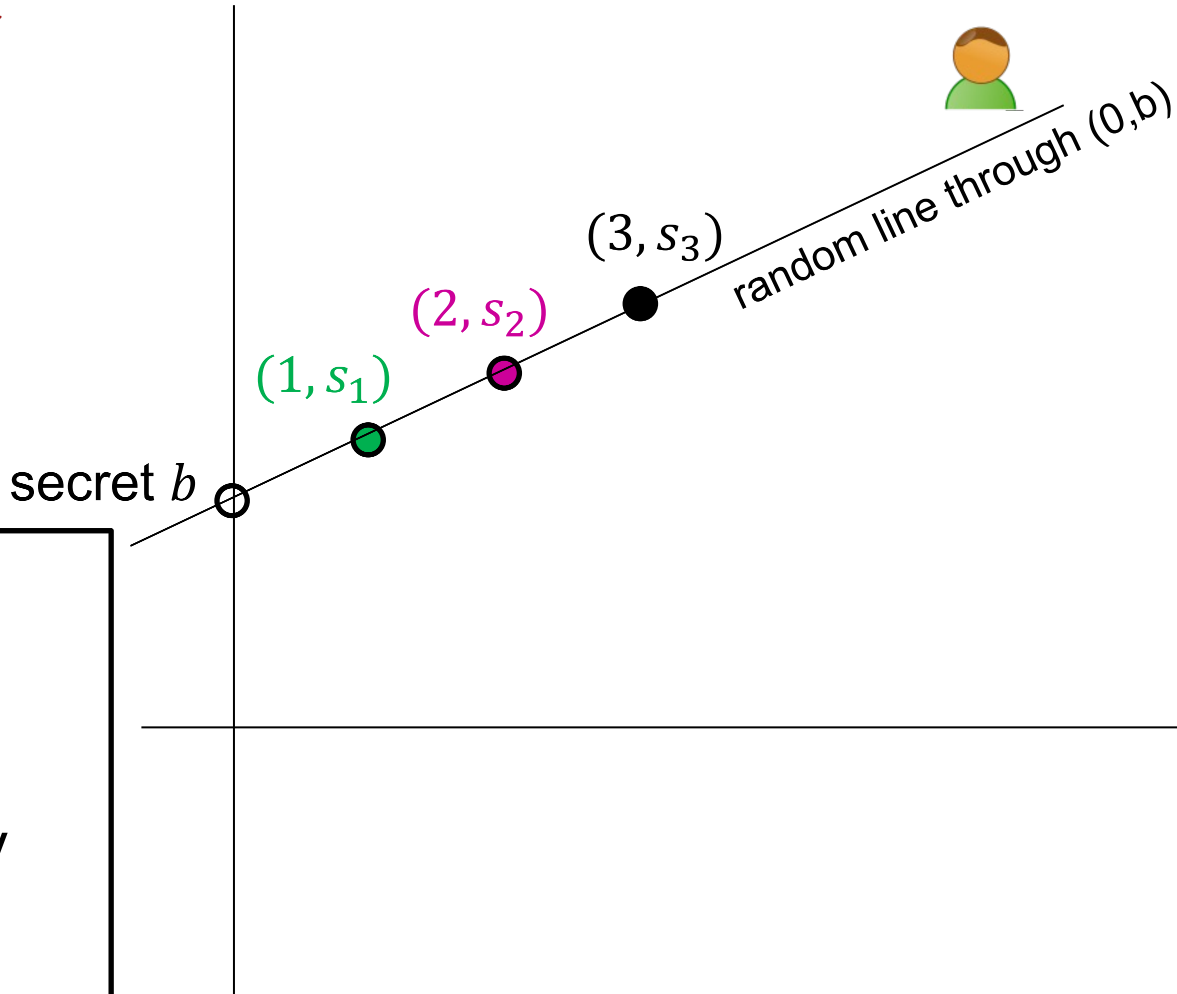
How does this scale to t-out-of-n?

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

Shamir's 2-out-of-n Secret Sharing





Each share s_i is truly random (independent of secret b)

Any two shares uniquely determine b.

Shamir's 2-out-of-n Secret Sharing

1. The dealer picks a uniformly random line **(mod p)** whose constant term is the secret b.

$$f(x) = ax + b \text{ where } a \text{ is uniformly random mod } p$$

2. Compute the shares:

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Correctness: can recover secret from any two shares.

Proof: Parties i and j, given shares $s_i = ai + b$ and $s_j = aj + b$ can solve for b $\left(= \frac{js_i - is_j}{j-i}\right)$.

Shamir's 2-out-of-n Secret Sharing

1. The dealer picks a uniformly random line **(mod p)** whose constant term is the secret b.

$f(x) = ax + b$ where a is uniformly random mod p

2. Compute the shares:

$$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Security: any single party has no information about the secret.

Proof: Party i's share $s_i = a * i + b$ is uniformly random, independent of b, as a is random and so is $a * i$.

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

1. The dealer picks a uniformly random degree-(t-1) polynomial **(mod p)** whose constant term is the secret b.

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

2. Compute the shares:

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Correctness: can recover secret from any t shares.

Security: the distribution of $any\ t-1$ shares is independent of the secret.

Note: need p to be larger than the number of parties n.

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Correctness: *via Vandermonde matrices.*

Let's look at shares of parties $P_1, P_2, \ldots, P_t$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \ldots \\ s_t \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^{t-1} \\ 1 & 3 & 3^2 & \ldots & 3^{t-1} \\ 1 & \ldots & \ldots & \ldots & \ldots \\ 1 & t & t^2 & \ldots & t^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \ldots \\ a_{t-1} \end{bmatrix} \pmod{p}$$

t-by-t Vandermonde matrix which is ***invertible***

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Correctness: Alternatively, *Lagrange interpolation* gives an explicit formula that recovers b.

$$b = f(0) = \sum_{i=1}^{t} f(i) \left(\prod_{1 \le j \le t, j \ne i} \frac{-x_j}{x_i - x_j} \right)$$

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \dots, s_i = f(i), \dots, s_n = f(n)$$

Security:

Let's look at shares of parties $P_1, P_2, \dots, P_{t-1}$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \dots \\ s_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^{t-1} \\ 1 & 3 & 3^2 & \dots & 3^{t-1} \\ 1 & \dots & \dots & \dots & \dots \\ 1 & t-1 & (t-1)^2 & \dots & (t-1)^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \dots \\ a_{t-1} \end{bmatrix} \pmod{p}$$

$(t-1)$*-by-*t *Vandermonde matrix*

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

Security: For every value of b there is a unique polynomial with constant term b and shares $s_1, s_2, \ldots, s_{t-1}$.

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \ldots \\ s_{t-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & 2 & 2^2 & \ldots & 2^{t-1} \\ 1 & 3 & 3^2 & \ldots & 3^{t-1} \\ 1 & \ldots & \ldots & \ldots & \ldots \\ 1 & t-1 & (t-1)^2 & \ldots & (t-1)^{t-1} \end{bmatrix} \begin{bmatrix} b \\ a_1 \\ a_2 \\ \ldots \\ a_{t-1} \end{bmatrix} \pmod{p}$$

$(t-1)$*-by-*t *Vandermonde matrix*

Shamir's t-out-of-n Secret Sharing

Key Idea: Polynomials are Amazing!

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + b$$

where a_i are uniformly random mod p

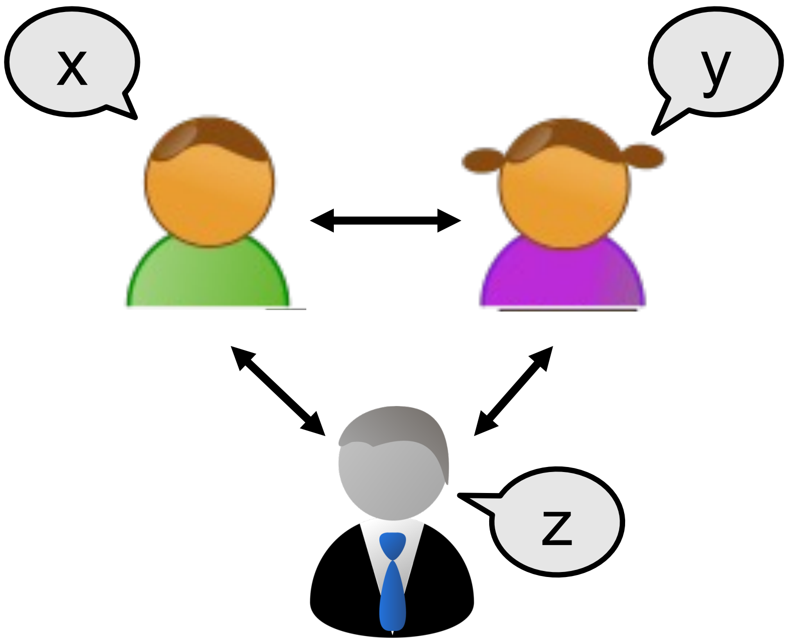
$$s_1 = f(1), s_2 = f(2), \ldots, s_i = f(i), \ldots, s_n = f(n)$$

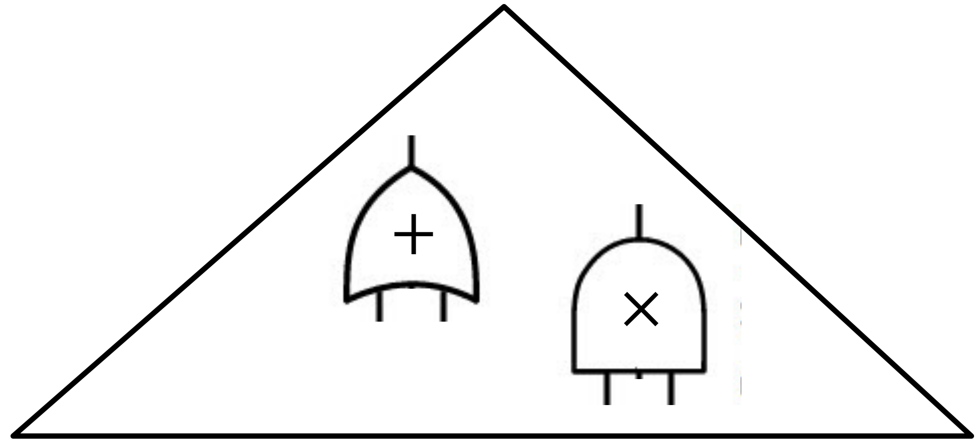
Security: For every value of b there is a unique polynomial with constant term b and shares $s_1, s_2, \ldots, s_{t-1}$.

Corollary: for every value of the secret b is equally likely given the shares $s_1, s_2, \ldots, s_{t-1}$. In other words, the secret b is perfectly hidden given $t - 1$ shares.

Secure Multiparty Computation

[BenOr-Goldwasser-Wigderson'88, Chaum-Crepeau-Damgard'88, BenOr-Rabin'89]





Arithmetic circuit for F

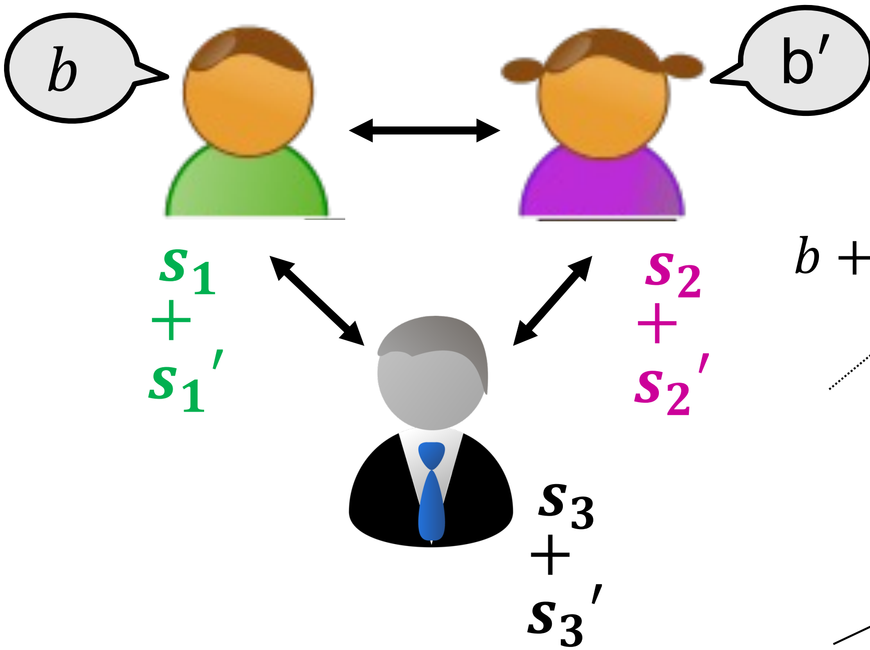
1. Each party secret-shares their input on a degree-t polynomial.

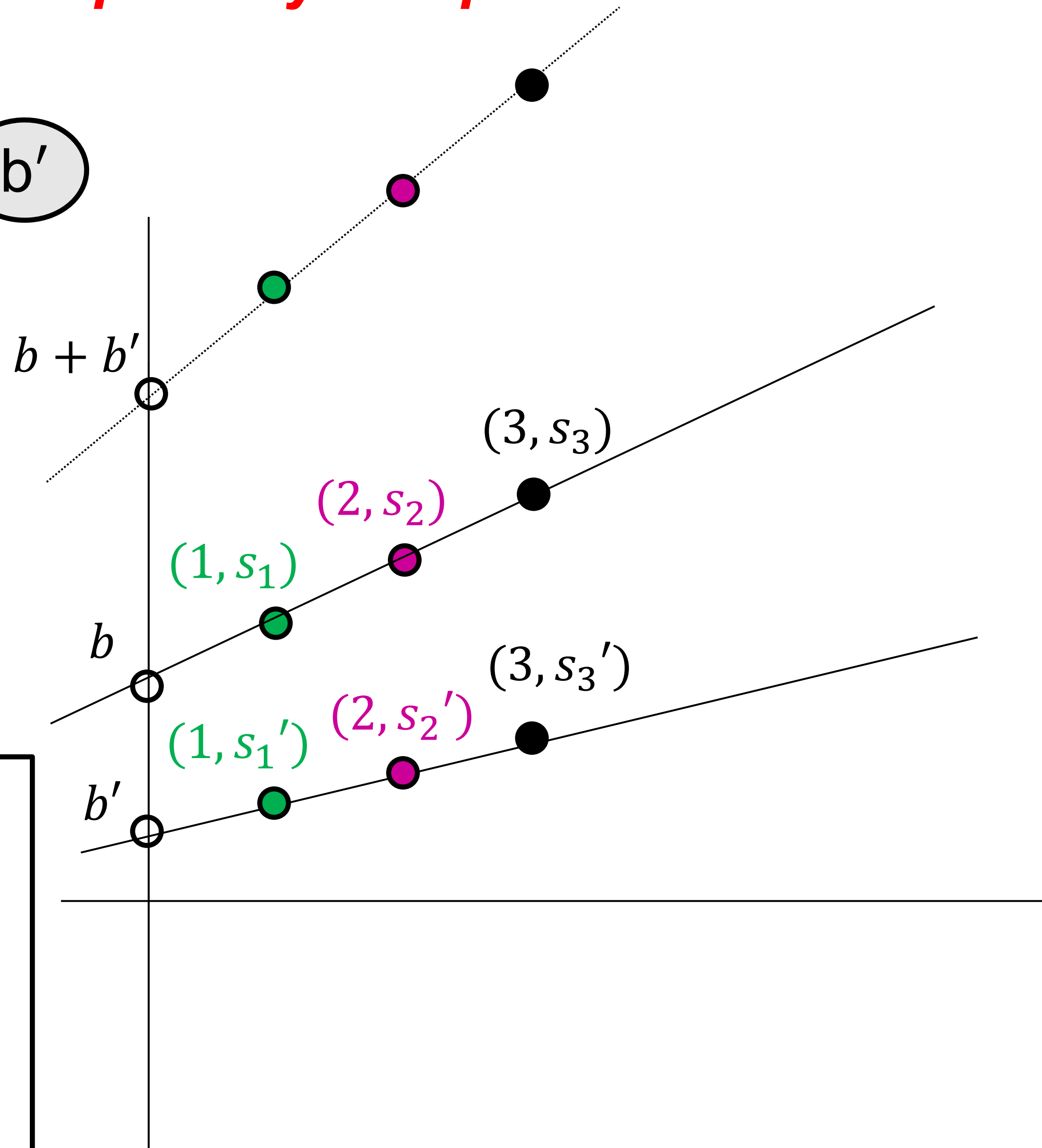
   // so, security against t corruptions

2. Proceed gate by gate, maintaining the invariant that the parties holds a secret sharing of every wire value.
3. Exchange the output shares & reconstruct the output.

Secure Multiparty Computation

Key Insight: Can homomorphically compute on shares!



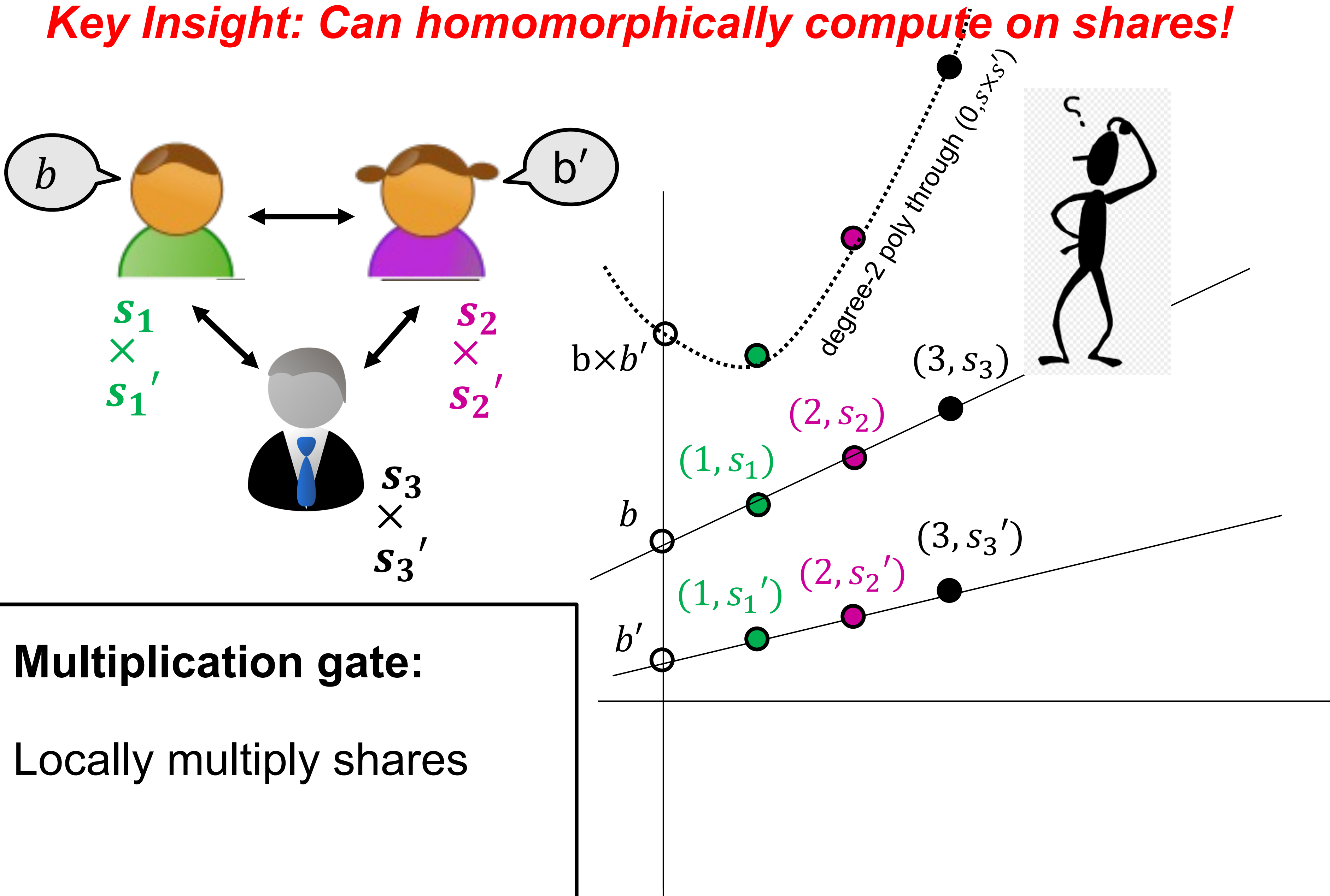


Addition gate:

Locally add shares

Secure Multiparty Computation

Key Insight: Can homomorphically compute on shares!



Multiplication gate:

Locally multiply shares

Multiplication

In general, after a single multiplication, the shares will live on a degree-2t polynomial.

Need $2t + 1$ shares to reconstruct.

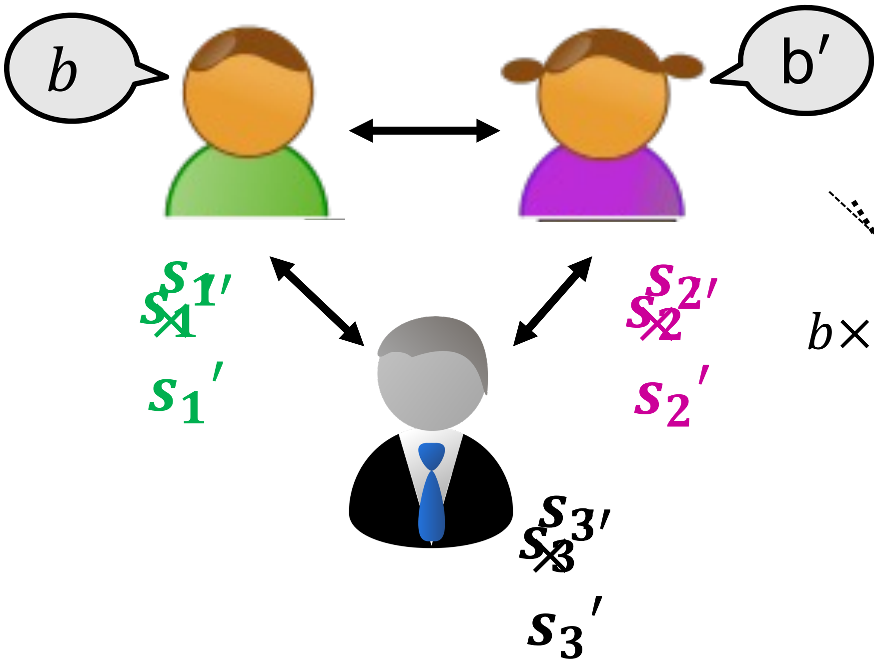
We know that $n > 2t$, so the n shares together have enough information to recover the product of the secrets!

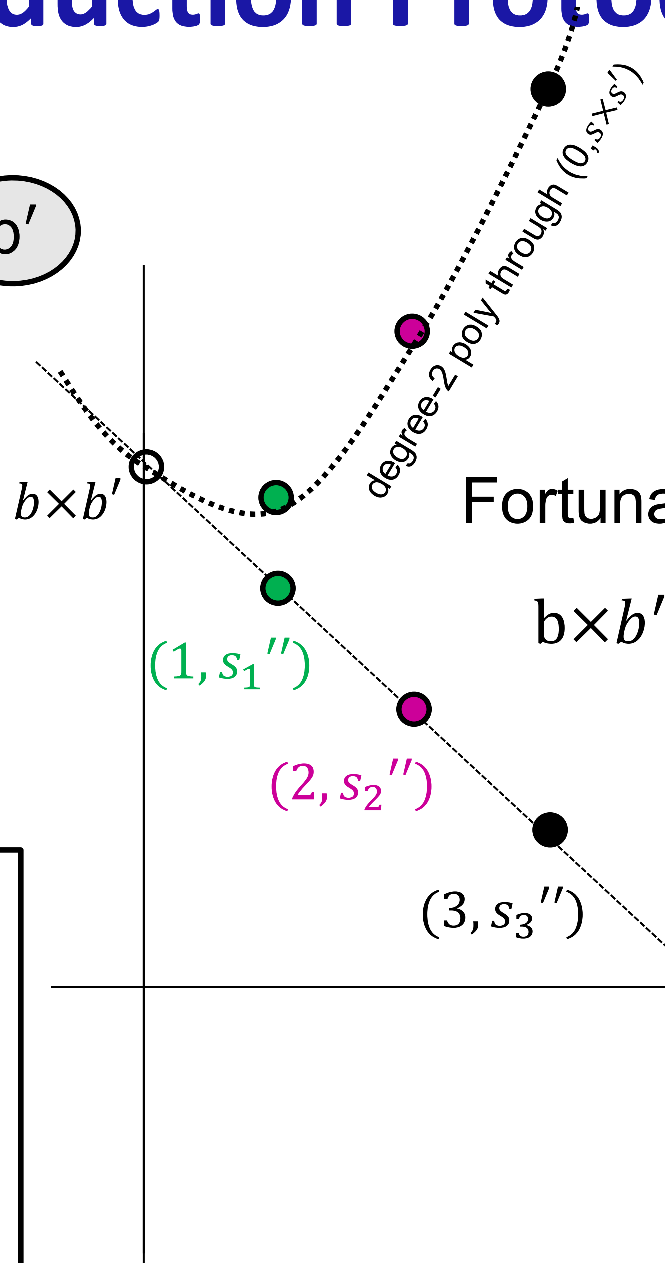
What's more, we also know that this recovery process is a linear function of the shares.

$$\mathrm{b} \times \mathrm{b}' = \sum \lambda_i \, s_i s_i{}'$$

for some publicly known coefficients λ_i.

Degree Reduction Protocol





Fortunately:

$$\mathrm{b} \times b' = \sum \lambda_i \, s_i s_i'$$

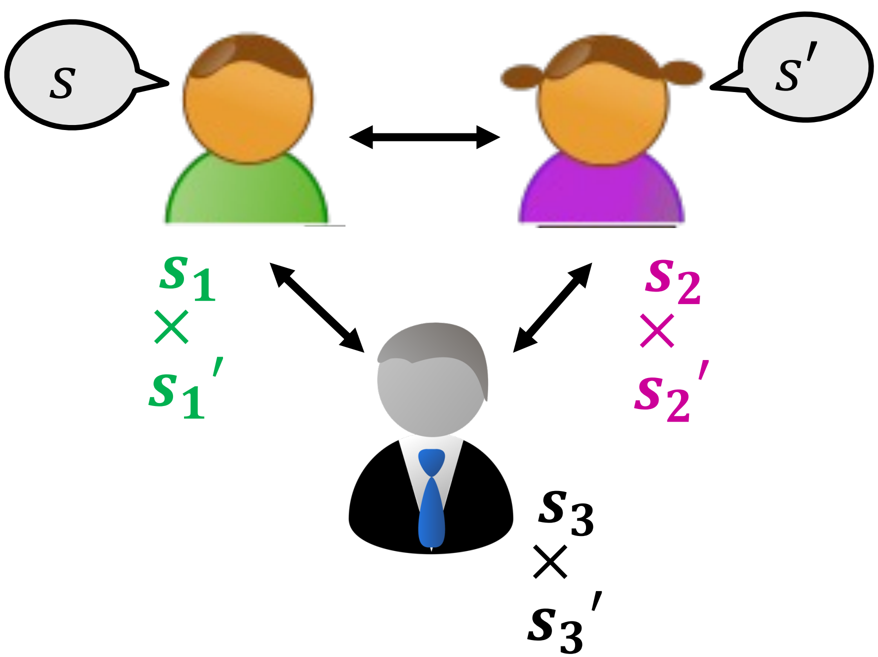
Multiplication gate:

Locally multiply shares & run a degree reduction protocol.

Degree Reduction Protocol

Convert shares on a degree-2t polynomial into shares on a degree-t polynomial

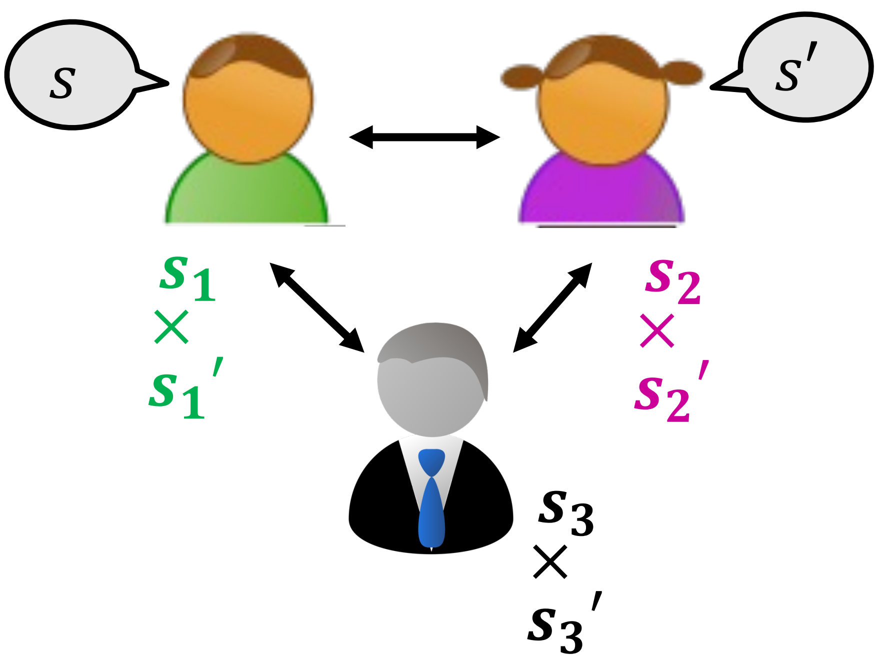
Idea: "homomorphically" compute the **linear function** $\sum \lambda_i * (\cdot)$ on the local product shares.



1. Each party t-out-of-n shares $s_i \times s_i'$ to all parties
2. Each party computes a linear combination of the shares it receives using coefficients λ_i.

This is the moral equivalent of bootstrapping in FHE!

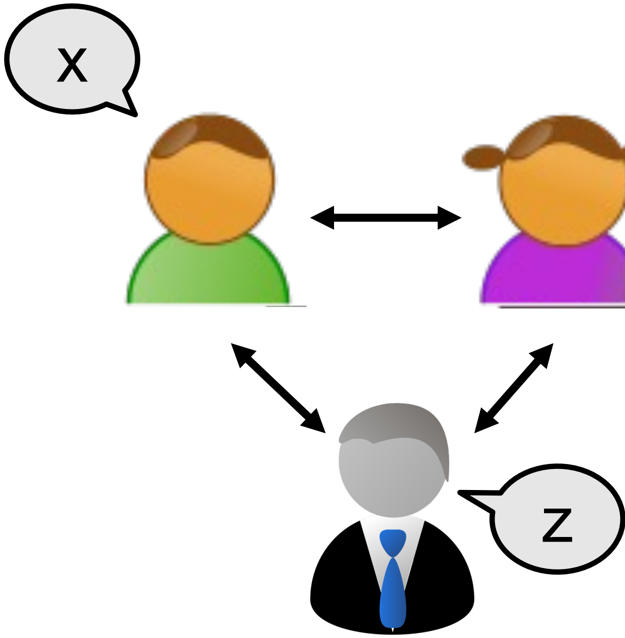
Idea: "homomorphically" compute the **linear function** $\sum \lambda_i * (\cdot)$ on the local product shares.

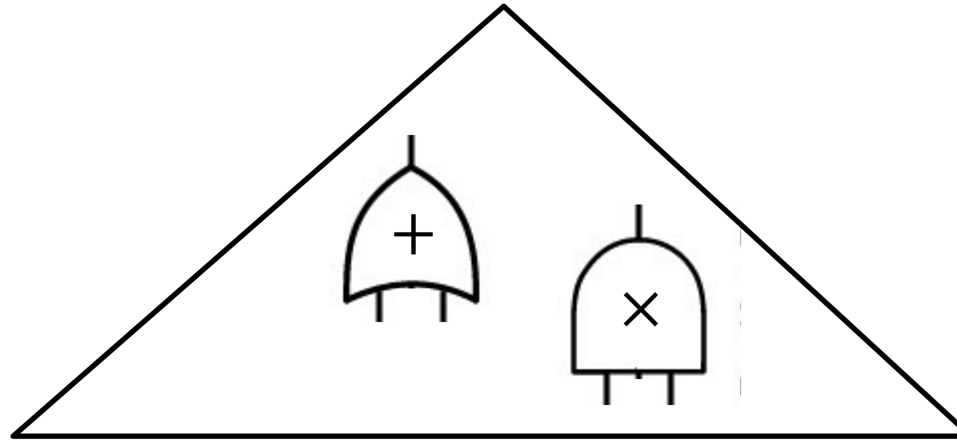


1. Each party t-out-of-n shares $s_i \times s_i'$ to all parties

2. Each party computes a linear combination of the shares it receives using coefficients λ_i.

Secure Multiparty Computation





Arithmetic circuit for F

1. Each party secret-shares their input.

2. Proceed gate by gate:
   ADD: locally add shares
   MULT: locally mult shares and do degree reduction.

3. Exchange the output shares & reconstruct the output.

Communication Complexity ∝ #AND gates

Security Intuition

1. Any subset of t parties do not get any information about other parties’ inputs from the input shares.

2. Security of the degree-reduction protocol: any subset of t parties sees completely random numbers

3. The output lives on a random polynomial of degree t whose constant term is the circuit output. The shares, therefore, reveal only the circuit output.

Threshold Decryption and Signing

Secret sharing is useful way beyond MPC.

An example: distributed storage of keys.

Another example, threshold decryption:

distributed storage of decryption key + non-interactive distributed (or threshold) decryption

Threshold El Gamal

Public key: g^x

Secret key: x

I am paranoid about losing x so I share it among n servers.

I secret-share x into n shares $x_1, \ldots, x_n$ s.t. $\sum_{i=1}^{n} x_i = x \pmod{q}$

Threshold Decryption:

Given a ciphertext $(g^r, g^{rx}m)$, the servers each compute a decryption share $(g^r)^{x_i}$.

Multiplying the decryption shares gives us $\prod (g^r)^{x_i} = g^{rx}$ which in turn gives us m after division.

Threshold Decryption and Signing

Secret sharing is useful way beyond MPC.

An example: distributed storage of keys.

Another example, threshold decryption:
distributed storage of decryption key + non-interactive distributed (or threshold) decryption

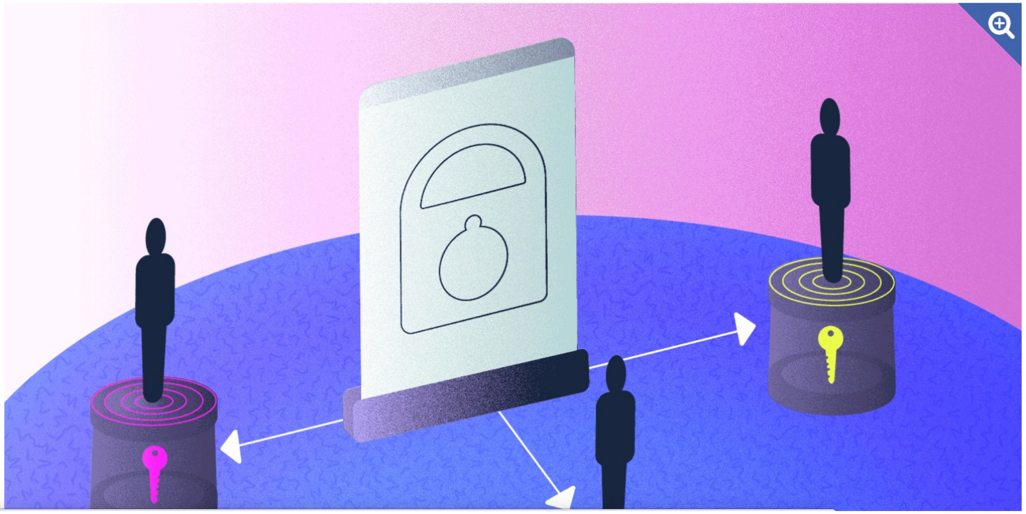
Yet another example, threshold signing.

NEWS

NIST Kick-Starts ‘Threshold Cryptography’ Development Effort

Establishing the emerging technique’s building blocks is a near-term focus.

July 07, 2020



Share

MEDIA CONTACT

Chad Boutin
charles.boutin@nist.gov
(301) 975-4261

ORGANIZATIONS

- Information Technology Laboratory
  - Computer Security Division
    - Cryptographic Technology Group
    - Security Test, Validation and Measurement Group