Secure Multi-Party Computations

An Introduction

Christian Zielinski

Last updated in 2015

Outline

Outline

Secure multi-party computations

- Consider n parties, with private inputs $x_1, \dots, x_n$
- They want to compute a function $f(x_1, \dots, x_n)$ in a secure way
- Security means here
  - Privacy: The respective inputs remain private
  - Correctness: The output is guaranteed to be correct
  - Fairness: Each party learns the result
- This should even hold when some parties try to cheat
- **The following presentation is primarily based on Ref. [1, 2]**

Questions at hand

- How to carry out computations without revealing the inputs?
- How to deal with cheating (corrupted) parties?
- How to define security formally?
- What is the upper limit of corrupted parties allowed?
- How does this bound depend on the assumption made about the attacker?

Motivation and applications

- Multi-party computations (**MPC**s) have a wide range of applications
- Auctions
  - Several parties are bidding for a product
  - Winning party and maximum bid should be determined, without revealing bids of other parties
- Electronic voting schemes
  - Each party votes for a candidate
  - Only the result is made public, the votes remain secret
- Yao's Millionaires' Problem, c.f. Ref. [3]
  - A group of millionaires wants to find out who is the richest
  - Nobody wants to reveal how wealthy they are

Outline

Adversaries

- To discuss secure MPCs, we have to define security
- Hence we have to make assumptions about cheating parties
- Typically one models them by considering an **adversary**
- This adversary can take over (**corrupt**) certain subsets of parties
- We assume one adversary, assuming the worst-case scenario of coordinated corrupted parties (**monolithic adversary**)
- We assume that at the beginning of the protocol honest (i.e. not corrupted) parties do not know which parties are corrupted

Passive and active security

We distinguish two cases of corruption

Definitions

Passive corruption:

- Adversary has *full information* of corrupted parties
- However, corrupted parties still follow the protocol

Active corruption:

- Adversary has *full control* over the corrupted parties
- Might deviate from the protocol to obtain sensitive data

Communication channels

Parties have to communicate and coordinate

Definitions

The **information-theoretic model**:

- All parties have pairwise secure channels
- Adversary has no access to messages sent between honest parties

The **cryptographic model**:

- The adversary has access to all messages sent
- Messages cannot be altered, i.e. the communication channel is authenticated

Sometimes we take a **broadcast** channel into account:

- All honest parties are assumed to receive the message (**consensus broadcast**)

$\mathcal{A}$-adversaries

- We define an **adversary structure** $\mathcal{A} \subset \mathcal{P}(P)$ as a family of subsets of the parties P
- An $\mathcal{A}$-**adversary** can only corrupt subsets of parties in $\mathcal{A}$
  - $\mathcal{A}$ is monotone
- Typically we consider **threshold adversaries**, i.e. $\mathcal{A}$ contains all subsets of up to some cardinality t
- An **adaptive** $\mathcal{A}$-adversary can corrupt a new party during execution, if the total set is in $\mathcal{A}$ (otherwise call it **static**)

Security in an ideal world

- How to define security in general?
- Here we introduce the concept of an **ideal world**
- A protocol is secure if an adversary does not learn more in the **real world** about the computations then in the ideal case
- More formal: Consider a function f, which should be securely evaluated in a MPC setting
- We introduce an ideal functional $\mathcal{F}^f_{\mathrm{SFE}}$, which is **incorruptible** and **leaks no private information**
- Then the MPC problem reduces to the parties securely sending their inputs to $\mathcal{F}^f_{\mathrm{SFE}}$ and receive the final result

Secure implementations

- In the real world $\mathcal{F}^f_{\mathrm{SFE}}$ is implemented by a protocol π^f_{SFE}
- We call π^f_{SFE} a **secure implementation** of $\mathcal{F}^f_{\mathrm{SFE}}$ if an adversary is unable to learn more about the computations than in the ideal world (without help of trusted parties)
- More formally assume an $\mathcal{A}$-adversary and let

$$\mathfrak{I} = \mathrm{IDEAL}_{\mathcal{A}}\left(\mathcal{F}^f_{\mathrm{SFE}}\right)$$

  denote what an adversary learns in the ideal world
- Similarly define

$$\mathfrak{R} = \mathrm{REAL}_{\mathcal{A}}\left(\pi^f_{\mathrm{SFE}}\right)$$

  for the execution of the protocol in the real world

Degrees of security

- With $\mathrm{SIM}(\mathfrak{I})$ we denote a **simulated protocol** using only the information of $\mathfrak{I}$, which we can compare with the real world protocol $\mathfrak{R}$
- If $\mathfrak{R}$ contains no more information than $\mathrm{SIM}(\mathfrak{I})$:
  - π^{f}_{SFE} is a **perfect secure implementation**
  - No unwanted information leaks
- If $\mathfrak{R}$ only contains additional statistical deviations from $\mathrm{SIM}(\mathfrak{I})$:
  - π^{f}_{SFE} is a **statistically secure implementation**
- If $\mathfrak{R}$ is only computationally indistinguishable from $\mathrm{SIM}(\mathfrak{I})$:
  - π^{f}_{SFE} is a **computationally secure implementation**
  - Adversary cannot distinguish due to computational bounds

Outline

Threshold adversaries

- We focus on threshold adversaries, i.e. the adversary can corrupt any set of parties up to cardinality t
- In the information-theoretic with adaptive adversaries we have the following results:

	Passive	Active w/ BC	Active w/o BC
Perfect	$n/2$	$n/3$	$n/3$
Statistical	$n/2$	$n/2$	$n/3$
Computational	n	$n/2$	$n/2$

Table: Maximal obtainable threshold t with n parties (taken from Ref. [1])

- Here we do not discuss general $\mathcal{A}$-adversaries, see Ref. [1]

Perfect security with passive adversary

- Assume n parties and a passive threshold adversary with threshold t
- We construct a perfectly secure protocol in the information-model theoretic for $t < n/2$
- We employ Shamir's $(t+1, n)$-scheme, calculating in a finite field $\mathbb{F}$
- Assume parties agreed to calculate $s' = \mathfrak{O}(s)$ with secret s
    - Secret s has been securely shared, so that party i has share s_i
    - Carry out operations $s_i' = \mathfrak{O}_i(s_i)$
    - Shares $\{s_i'\}$ allow to uniquely reconstruct s' by $t+1$ parties

Recap: Shamir's scheme

- Assume n parties and threshold $1 \leq t \leq n$
- Take a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq n+1$
- Let $s \in \mathbb{F}$ be the secret and define distinct elements $P_1, \ldots, P_n \in \mathbb{F} \setminus \{0\}$
- Sample a random polynomial p with $\deg p \leq t-1$ and $p(0) = s$
- Protocol:
  - Distribution phase: dealer shares $s_i = p(P_i)$ privately with party i
  - Reconstruction phase: $\geq t$ parties can reconstruct $p(x)$ (and hence $p(0) = s$)

Recap: Addition

- Assume P_i has share a_i and b_i
- Assume a and b have been shared with (random) polynomials p_a and p_b of degree $\leq t$
- We want to securely evaluate $c = a + b$
  - Each party adds $c_i = a_i + b_i$ locally
  - The $\{c_i\}$ uniquely determine the polynomial $p_c = p_a + p_b$
  - Polynomial p_c encodes the result as $p_c(0) = p_a(0) + p_b(0) = a + b = c$
- As $\deg p_c \leq t$ we find that $t + 1$ parties can reconstruct c
- In the special case of adding a (public) constant k party i just calculates $s_i' = s_i + k$

Recap: Multiplication

- The case of multiplying by a (public) constant k is similar
- To securely evaluate $s' = k \cdot s$, every party calculates $s'_i = s_i \cdot k$
- Shares $\{s'_i\}$ determine polynomial $p_{s'} = k \cdot p_s$, which encodes $p_{s'}(0) = k \cdot p_s(0) = k \cdot s = s'$
- What about the general case of $c = a \cdot b$ with a and b secretly shared?
- Every party can calculate $a_i \cdot b_i$
  - Uniquely determines the polynomial $p_c = p_a \cdot p_b$
  - Decodes the result as $p_c(0) = p_a(0) \cdot p_b(0) = a \cdot b = c$
  - **But is of degree $\mathbf{\deg p_c \leq 2t}$!**

Secure degree reduction (I)

- As $t < n/2$ we can at least uniquely define p_c
- Now **securely reduce the degree** of p_c, so that $\deg p_c \leq t$
- First observe by means of Lagrange interpolation

$$a \cdot b = \sum_{1 \leq i \leq n} \underbrace{\left[\frac{\prod_{1 \leq j \leq n}^{j \neq i} (-P_j)}{\prod_{1 \leq j \leq n}^{j \neq i} (P_i - P_j)} \right]}_{\equiv r_i} a_i \cdot b_i$$

- Hence we have a linear combination of the result c in terms of shares $\{a_i \cdot b_i\}$ at hand
- The **recombination vector** $r_1, \ldots, r_n$ can be calculated from public information

Secure degree reduction (II)

- In the next step each party acts as a dealer and **re-shares** their share $a_i \cdot b_i$ using a polynomial $\mathfrak{c}_i$ of $\deg \mathfrak{c}_i \leq t$
- This results in party i having shares u_{ji} $(j = 1, \ldots, n)$
- We can then consider the polynomial

$$\mathfrak{c} = \sum_{1 \leq i \leq n} r_i \cdot \mathfrak{c}_i$$

- Observe that $\deg \mathfrak{c} \leq t$ and

$$\mathfrak{c}(0) = \sum_{1 \leq i \leq n} r_i \cdot \mathfrak{c}_i(0) = \sum_{1 \leq i \leq n} r_i \cdot a_i b_i = a \cdot b = c$$

- Hence party i computes

$$c_i = \sum_{1 \leq \ell \leq n} r_\ell \cdot u_{\ell i} = \sum_{1 \leq \ell \leq n} r_\ell \cdot \mathfrak{c}_\ell(P_i) = \mathfrak{c}(P_i),$$

which is a $(t+1, n)$-SSS share c_i of $c = a \cdot b$

Privacy

- Party i only deals with their respective shares
- After reconstruction party i has share c_i of result $c = a + b$ or $c = a \cdot b$
- Shares belong to a $(t+1, n)$-SSS, but adversary can only corrupt up to t parties
- No information about other parties' input besides what is implied by their shares and the final result

General functions (I)

- Using addition and multiplication we can compute more general functions
- We represent the function as an arithmetic circuit:

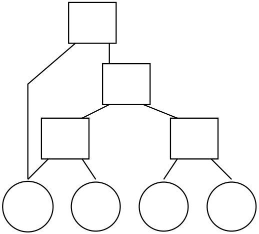

Figure: A simple arithmetic circuit

General functions (II)

- We can do this without loss of generality (if f is feasible)
- Well known in the special case of $\mathbb{F} = \{0, 1\}$, so called Boolean circuits
- Any computable function can be represented using only $\textsc{and}$ and $\textsc{not}$ gates
  - a $\textsc{and}$ b can be represented by $a \cdot b$
  - $\textsc{not}$ a can be represented by $1 - a$
  - The computer is a "proof by example"
- In the general case represent a function $f : \mathbb{F}^n \to \mathbb{F}$ by an arithmetic circuit consisting of addition and multiplication gates
- Calculations proceed gate by gate

The protocol

To compute a function $y = f(x_1, \ldots, x_n)$, we represent it as an arithmetic circuit

- Each party begins with private input $x_i \in \mathbb{F}$ and shares it using a $(t+1, n)$-SSS with all participants
- The calculation proceeds gate by gate, so that at each point all inputs and intermediate results are shared with a $(t+1, n)$-SSS
- From all remaining gates we randomly choose one for which all inputs are available
- At the end P_i broadcasts its share y_i of the final result $y = f(x_1, \ldots, x_n)$

Correctness and privacy

- Correctness follows from correctness of Shamir's scheme and algorithms for addition and multiplication
- Privacy follows from the facts that:
  - The adversary was assumed to only corrupt up to t parties
  - All values are shared with a $(t+1, n)$-scheme, so the adversary cannot interfere anything about the honest party's inputs
  - The corrupted parties only learn their own inputs and outputs
- *Everybody* learns the final result (fairness)

Tightness of bound (I)

- What if $t \geq n/2$?
- Then there is no protocol with **perfect privacy** and **perfect correctness**
  - Assuming correctness, a infinite powerful passive advisor can violate privacy!
- An example:
  - Consider two parties P_i with input bit b_i $(i = 1, 2)$
  - They want to securely compute $r = b_1 \wedge b_2$
  - Both have additional randomness $r_i \in \{0, 1\}^{\star}$

Tightness of bound (II)

- Both are exchanging messages m_{ij}, $j = 1, \ldots, N$.
  Define the **transcript**

$$\mathscr{T} = (m_{11}, m_{21}, m_{12}, m_{22}, \ldots, m_{1N}, m_{2N}, r)$$

- Let $\mathscr{T}(b_1, b_2)$ be the set of transcripts for given b_1 and b_2
- Can then show that

$$\mathscr{T}(0,0) \cap \mathscr{T}(0,1) = \emptyset$$

- Hence if $b_1 = 0$ party P_1 can check if $\mathscr{T} \in \mathscr{T}(0,0)$ or $\mathscr{T} \in \mathscr{T}(0,1)$ and deduce b_2
  - Might be unfeasible in the real world

Active adversaries

- We now want to deal with an active adversary
  - We assume that $t < n/2$ for this part
- In the presence of an active adversary a broadcast (BC) channel cannot be taken for granted
  - Corrupted party might send different things to different parties
  - However, in the case discussed here there are effective protocols to emulate a BC
- Corrupted parties can now:
  - Deviate from the protocol
  - Give wrong inputs
  - Might even refuse to respond

Verifiable secret sharing schemes

- We need a **Verifiable Secret Sharing (VSS)** scheme
    - A VSS is a SSS, that allows the parties to verify that they have consistent shares
- We implement the active secure protocol by emulating the previous protocol and:
    - We make all **parties committed to their respective shares**
    - We ensure that all shares are computed correctly

Modeling security

- In the ideal world all a corrupted party can do is specify an alternative input x_i' for $\mathcal{F}_{\mathrm{SFE}}^{f}$
- We require that all deviations of the protocol can be modeled by choosing alternative inputs
- What if a corrupted party refuses to give any input?
    - The protocol can potentially deadlock
    - Possible solution: other parties simulate this party with input $x_i = 0$

Commitments

- How can a party P_i commit to a value $a \in \mathbb{F}$?
- To model this we introduce another ideal functional $\mathcal{F}_{\mathrm{COM}}$
  - In the real world this will be implemented collectively by the other parties
- Can then commit to a and access a via $\mathcal{F}_{\mathrm{COM}}$ using a interface with given commands

Interface of $\mathcal{F}_{\mathrm{COM}}$

- We now define an interface for $\mathcal{F}_{\mathrm{COM}}$ consisting of commands
- For execution *all honest parties have to agree on the command* send to $\mathcal{F}_{\mathrm{COM}}$ as the implementation will require them to actively participate
- Basic commands for committing and revealing of values
    - Values committed to not known by other parties than the committer
- Also implementing manipulation commands
    - Add and multiply committed shares
    - Allows us to eventually to emulate $\mathcal{F}_{\mathrm{SFE}}^{f}$ by $\mathcal{F}_{\mathrm{COM}}$

Basic commands of $\mathcal{F}_{\mathrm{COM}}$

- `commit` of P_i to $a \in \mathbb{F}$, denoted by $P_i : [a]_i \Leftarrow a$
  - After successful execution $\mathcal{F}_{\mathrm{COM}}$ stores (P_i, a)
- `public commit` of all parties to $a \in \mathbb{F}$
  - By $[a]_i \Leftarrow a$ we denote the use of the `public commit` command to force P_i to commit to a
  - $\mathcal{F}_{\mathrm{COM}}$ stores a for all P_i
- `open` of $a \in \mathbb{F}$ to all parties assuming some $[a]_i$ is stored, denoted by $a \Leftarrow [a]_i$
  - All parties learn a
- `designated open` of $a \in \mathbb{F}$ to party P_j assuming some $[a]_i$ is stored, denoted by $P_j : a \Leftarrow [a]_i$
  - Party P_j learns a

Manipulation commands of $\mathcal{F}_{\mathrm{COM}}$

- **`add`** of two values $a, b \in \mathbb{F}$, assuming some $[a]_i$ and $[b]_i$ is stored
  - Denoted by $[a+b]_i \leftarrow [a]_i + [b]_i$
- **`multiplication by a constant`** of $a \in \mathbb{F}$ with an $\alpha \in \mathbb{F}$, assuming some $[a]_i$ is stored
  - Denoted by $[\alpha a]_i \leftarrow \alpha [a]_i$
- **`transfer`** of $a \in \mathbb{F}$ to all parties assuming some $[a]_i$ is stored
  - P_j learns a and commits to it, denoted by $[a]_j \leftarrow [a]_i$
- **`multiplication`** of two values $a, b \in \mathbb{F}$, assuming some $[a]_i$ and $[b]_i$ is stored
  - Denoted by $[a \cdot b]_i \leftarrow [a]_i \cdot [b]_i$

Implementation

- The **`transfer`** and **`multiplication`** commands are high level commands
  - Can be implemented using the other commands
- As an example we show how to implement the **`multiplication`** command
- For brevity we omit here the implementation of the **`transfer`** command
  - Details can be found in [1]

The **multiplication** command

We implement $[a \cdot b]_i \leftarrow [a]_i \cdot [b]_i$

1. $P_i : [c]_i \Leftarrow a \cdot b$ (P_i knows a and b)

If P_i is honest, c will be correct. But P_i might cheat. Hence every P_k carries out the following consistency check:

1. P_i chooses $\alpha \in \mathbb{F}$ uniform at random
2. $P_i : [\alpha]_i \Leftarrow \alpha$ (**commit** to α)
3. $P_i : [\gamma]_i \Leftarrow \alpha b$ (**commit** to αb)
4. P_k broadcasts a **challenge** $e \in \mathbb{F}$ uniform at random
5. $[A]_i \leftarrow e\,[a]_i + [\alpha]_i$; $A \Leftarrow [A]_i$ (**open** A)
6. $[D]_i \leftarrow A\,[b]_i - e\,[c]_i - [\gamma]_i$; $D \Leftarrow [D]_i$ (**open** D)
7. The proof is accepted if $D = 0$

The **multiplication** command

- If $[c]_i = [a]_i \cdot [b]_i$ then $D = 0$.
- If P_i cheated and committed to a $c = a \cdot b + \Delta$, then $D \neq 0$ with probability $|\mathbb{F}|^{-1}$
    - As $\mathbb{F}$ is finite, $D = 0$ can still happen coincidentally
    - There are $n - t > n/2$ honest parties, so probability that all proofs of honest parties are accepted in the case of cheating is $\leq |\mathbb{F}|^{-n/2}$
    - By repeating the proof several times, probability can be further reduced
- We now present the actual protocol using $\mathcal{F}_{\mathrm{COM}}$

The active secure protocol (I)

Input sharing

- Party P_i holds input x_i and shares it using Shamir's scheme
- Ensure correct shares and that parties are committed to their shares

Protocol

1. $P_i : [x_i]_i \Leftarrow x_i$ (`commit` to input)
2. P_i chooses a polynomial $\mathscr{P}_i(z) = x_i + \sum_{j=1}^{t} \alpha_j z^j$ uniform at random
3. $P_i : [\alpha_j]_i \Leftarrow \alpha_j,\ \forall j$ (`commit` to coefficients)
4. $P_i : [\mathscr{P}_i(P_\ell)]_i \leftarrow x_i + \sum_{j=1}^{t} [\alpha_j]_i P_\ell^j,\ \ell = 1, \ldots, n$ (evaluating the shares for all parties)
5. $[\mathscr{P}_i(P_\ell)]_\ell \leftarrow [\mathscr{P}_i(P_\ell)]_i,\ \ell = 1, \ldots, n$ (`transfer` of all shares to the respective parties)

The active secure protocol (II)

Arithmetic operations

- Function f was assumed to be represented by an arithmetic circuit

Addition

1. $[c]_i \leftarrow [a]_i + [b]_i$

Multiplication

1. $[d_i]_i \leftarrow [a_i]_i \cdot [b_i]_i$ (local multiplication)
2. P_i shares $[d_i]_i$ (like in input sharing part), hence P_ℓ is committed to $[d_{i\ell}]_\ell$
3. $[c_\ell]_\ell \leftarrow \sum_{i=1}^{n} r_i \, [d_{i\ell}]_\ell$ (recombination with recombination vector $r_1, \ldots, r_n$)

The active secure protocol (III)

Reconstruction

- Party P_i committed to share y_i

If P_j is supposed to learn y:

1. $P_j : y_i \Leftarrow [y_i]_i,\ i = 1, \ldots, n$

Note:

- If share y_i is stored in $\mathcal{F}_{\mathrm{COM}}$, it is consistent
- If P_i cheats, it might be not recorded and the opening fails
- As there are $n - t > t$ honest parties, still can reconstruct y

Security of the protocol

- The protocol ensures correct and consistent shares at every point
- However, a corrupted party might refuse to carry out a given command
- If this happens with party P_j:
  - Input phase: other parties take input $x_j = 0$ and 0-polynomial for P_j
  - Addition cannot fail
  - Multiplication: if P_j has been disqualified, open its input and restart the calculation and openly simulate this party
  - Reconstruction was already discussed

Implementation of $\mathcal{F}_{\mathrm{COM}}$ (I)

- Import question: how to implement the ideal functional $\mathcal{F}_{\mathrm{COM}}$ by a protocol π_{COM}?
- We will emulate it by all (honest) parties
- Assume information-theoretic scenario with $t < n/3$
  - In the cryptographic scenario can relax to $t < n/2$
- We just give an outline of the realization

Implementation of $\mathcal{F}_{\mathrm{COM}}$ (II)

- Idea: can implement `commit` of party P_j to $a \in \mathbb{F}$ by using a SSS
  - Then we easy to implement `open`, `add` and `multiplication by a constant` (and hence `transfer` and `multiplication`)
  - If P_j is honest, adversary will not learn a
  - But: If P_j is corrupted, might give **inconsistent shares**
- Hence we have to **force consistent shares**

Corrupted shares

- Note that if even $< n/3$ shares are corrupted, the secret is still uniquely defined
- Consider secret $s \in \mathbb{F}$ shares with polynomial p with $\deg p \leq t$
- The shares are
$$\mathbf{s} = (p(P_1), \dots, p(P_n))$$
- Consider an error $\mathbf{e} \in \mathbb{F}^n$ with Hamming-weight $w_{\mathrm{H}}(\mathbf{e}) \leq t$
- Then both $\mathbf{s}$ and $\tilde{\mathbf{s}} = \mathbf{s} + \mathbf{e}$ uniquely define the same s

Forcing consistent shares (I)

- In principle can check for consistent shares, if dealer broadcasts the polynomial
  - But this reveals the secret

Instead use algorithm:

1. P_j shares secret $s \in \mathbb{F}$ with shares $\{s_i\}$
2. P_j picks a $r \in \mathbb{F}$ uniform at random and shares $\{r_i\}$
3. A challenge $e \in \mathbb{F}$ is chosen and $\forall i$ party P_i computes

$$a_i = e \cdot s_i + r_i$$

4. Then make consistency check of $\{a_i\}$ for value $a = e \cdot s + r$

Forcing consistent shares (II)

- Value a is randomly distributed, revealing is unproblematic
- If shares $\{s_i\}$ and $\{r_i\}$ are consistent, so are the $\{a_i\}$
- Otherwise probability that the $\{a_i\}$ are consistent by coincident is $|\mathbb{F}|^{-1}$
- What if the $\{a_i\}$ are consistent, but a corrupted party broadcasts a value $\tilde{a}_i \neq a_i$?
  - We employ **dispute control**

Dispute control (I)

- With each party we associate a public **dispute set** $D_i \subseteq \{P_1, \ldots, P_n\}$
  - At the beginning $D_i = \emptyset$
- If some party P_j broadcasts an inconsistent share $\tilde{a}_j$:
  - P_j is added to D_i (P_i is dealer)
  - If dealer P_i is honest $P_j \in D_i$ means that P_j is corrupted
  - Test is repeated with $a_j = 0$ (**corrected sharing**)

Dispute control (II)

- If in one of the repetitions:
  - P_i broadcasts a polynomial with $p(P_j) \neq 0$ for $P_j \in D_i$
  - $|D_i| > t$ (at least one honest party is in dispute with P_i)
- Then: remaining parties accuse dealer P_i of being corrupt
  - All messages from P_i will be ignored from now on
- Employing dispute control allows us to ensure consistent shares and to implement $\mathcal{F}_{\mathrm{COM}}$ (which emulates $\mathcal{F}_{\mathrm{SFE}}^{f}$)

Conclusions

- We showed how to implement MPCs using **Shamir's Scheme**
- For a **passive** threshold adversary we have to require $t < n/2$
- For an **active** threshold adversary in the information-theoretic scenario we need $t < n/3$
  - Protect against active attacks with **VSS**s and **dispute control**
- **For more information refer to Ref. [1, 2]**

Bibliography

Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen.
Multiparty computation, an introduction.
Contemporary cryptology, pages 41–87, 2009.

Martin Hirt and Ueli Maurer.
Complete characterization of adversaries tolerable in secure multi-party computation.
In *Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing*, pages 25–34. ACM, 1997.

Andrew Chi-Chih Yao.
Protocols for secure computations.
In *FOCS*, volume 82, pages 160–164, 1982.