# Lecture 11: Multi-Party Computation

Chris Brzuska

November 16, 2022

Secure Multi-Party Computation (MPC) allows several parties to perform a computation such that the parties learn nothing except for the output of the computation. These lecture notes contain the following:

- an example of secret-sharing of a password using xor (warm-up)
- Security of secret-sharing: The simulation paradigm
- Threshold secret-sharing
- an example of securely computing an AND function (2nd warm-up)
- a description of how to securely evaluate an arbitrary function
- Shamir's secret-sharing scheme

## 1 Secret-Sharing

Let us start our exposition with a little story.

### 1.1 Alice shares her password

Alice went on a research visit to Aarhus University to collaborate on a research project on practically efficient secure multi-party computation. Upon arrival, in shock, she realizes that she has worked a lot from home over the past month, but completely forgot to push her changes to the Git from her personal computer. Now, she is in Aarhus without all her results from the last month. She is sad and frustrated. She was so looking forward to her research visit! So, she calls her flatmate Bianca. "Bianca, can you go to my computer and push the changes I made to the Git?"–"Sure, what's your password?"–"It's... ...uhm... ...is this a good idea? Wait, I got an idea how to give you the password securely. I'll use multiple channels and if at least one of them is secure from an attacker, then the password is secure. So, on this voice channel, let me send you the bit sequence 100101 100110 001010. I'll send you some other bit sequences over other channels, and you xor the results, okay?" – "Alright, that's a cool idea!" So, Alice sends to Bianca over encrypted chat the sequence 001100 101011 001101 and via an anonymous call platform transmits the sequence 010010 100000 101110, and Bianca reconstructs the password as follows

$$\begin{array}{rl} & 100101\ 100110\ 001010 \\ \oplus & 001100\ 101011\ 001101 \\ \oplus & \underline{010010\ 100000\ 101110} \\ = & 111011\ 101101\ 101001, \end{array}$$

which Bianca then transforms into ascii code, uses the result as a password[1] and pushes Alice's changes to the Git.

### 1.2 Secret Sharing Syntax

The technique which Alice and Bianca use is known as *secret sharing*. Here, an encoding algorithm splits a secret $s$ into multiple shares $sh_1, ..., sh_t$ such that a decoding algorithm can put $sh_1, ..., sh_t$ together to recover the secret $s$.

[1] The example is short so that the example is easy to calculate, but of course, in real-life, we would want to use much longer passwords.

**Definition 1.1** (Syntax Secret Sharing)**.** A *secret sharing scheme* $\mathsf{secsha}$ consists of two PPT algorithms $(\mathsf{encode}, \mathsf{decode})$ with an associated domain $D_\lambda$ for each security parameter $\lambda$ such that the following correctness condition holds:

$$\forall \lambda \in \mathbb{N} \forall s \in D_\lambda \, \forall t \in \mathbb{N} : \Pr_{sh_1,...,sh_t \leftarrow\$ \mathsf{encode}(s,t,1^\lambda)}[s = \mathsf{decode}(sh_1, ..., sh_t)] = 1 - \mathsf{negl}(\lambda),$$

where $\mathsf{negl}(\lambda)$ is a negligible function in $\lambda$.

## 1.3 The simulation paradigm

So, how do we define the *security* of a secret sharing scheme? We want to state that an adversary who only sees $t-1$ of the shares but not all shares does not gain any information about the secret $s$, except for the length of the secret. Thus, what we'll demand is that there is an efficient algorithm which is only given the length of $s$ and which can simulate the $t-1$ shares to the adversary in a way that is indistinguishable from what the adversary sees in the real world. In the literature, this algorithm is called a *simulator*, and using this definitional style is convenient: For defining the security of a primitive, we don't need to say *how* the simulator works, we just need to say that it exists.

**Definition 1.2** (IND-CSA)**.** A *secret sharing scheme* $\mathsf{secsha}$ is *indistinguishable under chosen secret attacks* if there exists a PPT algorithm $\mathcal{S}$ such that the games $\texttt{Gind-csa}^0_{\mathsf{secsha}}$ and $\texttt{Gind-csa}^1_{\mathcal{S}}$ are indistinguishable, i.e., for all PPT adversaries $\mathcal{A}$, the advantage

$$\mathbf{Adv}^{\texttt{Gind-csa}^0_{\mathsf{secsha}},\texttt{Gind-csa}^1_{\mathcal{S}}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gind-csa}^0_{\mathsf{secsha}}\right] - \Pr\left[1 = \mathcal{A} \rightarrow \texttt{Gind-csa}^1_{\mathcal{S}}\right]\right|$$

is negligible, where the games $\texttt{Gind-csa}^0_{\mathsf{secsha}}$ and $\texttt{Gind-csa}^1_{\mathcal{S}}$ are defined as

| $\texttt{Gind-csa}^0_{\mathsf{secsha}}$ | |
|---|---|
| Parameters | |
| $\lambda$: | security parameter |
| $\mathsf{secsha}$: | secret sharing scheme |
| **Package State** | |
| $d$: | call-once bit |

| $\mathsf{SECSHA}(s,t,i)$ |
|---|
| **assert** $d \neq 1$ |
| $d \leftarrow 1$ |
| **assert** $i \in \{1,..,t\}$ |
| $sh_1,..,sh_t \leftarrow\$ \mathsf{secsha}(s,t,1^\lambda)$ |
| **return** $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t$ |

| $\texttt{Gind-csa}^1_{\mathcal{S}}$ | |
|---|---|
| Parameters | |
| $\lambda$: | security parameter |
| $\mathcal{S}$: | simulator |
| **Package State** | |
| $d$: | call-once bit |

| $\mathsf{SECSHA}(s,t,i)$ |
|---|
| **assert** $d \neq 1$ |
| $d \leftarrow 1$ |
| **assert** $i \in \{1,..,t\}$ |
| $\ell \leftarrow \lvert s \rvert$ |
| $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t \leftarrow\$ \mathcal{S}(t,\ell,i,1^\lambda)$ |
| **return** $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t$ |

How is this similar/different from our definition of IND-CPA security for encryption? Also there, we wanted to capture that the adversary learns nothing except for the length of the transmitted message and captured it by defining an ideal world where the adversary sees an encryption of zeroes of the same length. In a way, "encrypting zeroes" can be seen as a special simulation strategy. We could also define symmetric encryption by demanding that there exists a simulator which simulates ciphertexts, given only the length of the message ($\rightarrow$ Exercise Sheet 10). The simulation paradigm is a popular paradigm for defining security of secure multi-party computation, and we will use it throughout this lecture. For now, let us return to our concrete example.

## 1.4 XOR Secret-Sharing

| $\mathsf{secsha}_{\mathrm{xor}}.\mathsf{encode}(s,t,1^\lambda)$ | $\mathsf{secsha}_{\mathrm{xor}}.\mathsf{decode}(sh_1,..,sh_t)$ | $\mathcal{S}_{\mathrm{xor}}(t,\ell,i,1^\lambda)$ |
|---|---|---|
| **for** $j=1..t-1$ : | | **for** $j=1..i-1$ : |
| $sh_j \leftarrow\!\!\$\ \{0,1\}^{\lvert s\rvert}$ | | $sh_j \leftarrow\!\!\$\ \{0,1\}^{\ell}$ |
| $sh_t \leftarrow s \bigoplus_{j=1}^{t-1} sh_j$ | $s \leftarrow \bigoplus_{j=1}^{t} sh_j$ | **for** $j=i+1..t$ : |
| | | $sh_j \leftarrow\!\!\$\ \{0,1\}^{\ell}$ |
| **return** $(sh_1,..,sh_t)$ | **return** $s$ | **return** $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t$ |

Figure 1: XOR Secret Sharing

Now, we can show the following theorem via inlining:

**Theorem 1.** For all PPT adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}_{\mathcal{A}}^{\mathtt{Gind\text{-}csa}^0_{\mathsf{secsha}_{\mathrm{xor}}},\mathtt{Gind\text{-}csa}^1_{\mathcal{S}_{\mathrm{xor}}}}(\lambda) :=$

$$\left|\Pr\left[1=\mathcal{A}\rightarrow \mathtt{Gind\text{-}csa}^0_{\mathsf{secsha}_{\mathrm{xor}}}\right]-\Pr\left[1=\mathcal{A}\rightarrow \mathtt{Gind\text{-}csa}^1_{\mathcal{S}_{\mathrm{xor}}}\right]\right|$$

is 0.

*Proof.* We can prove this theorem by inlining the code of $\mathsf{secsha}_{\mathrm{xor}}$ into $\mathtt{Gind\text{-}csa}^0_{\mathsf{secsha}_{\mathrm{xor}}}$ and the code of $\mathcal{S}_{\mathrm{xor}}$ into $\mathtt{Gind\text{-}csa}^1_{\mathcal{S}_{\mathrm{xor}}}$. We can then observe that $\mathtt{Gind\text{-}csa}^0_{\mathsf{secsha}_{\mathrm{xor}}}$ and $\mathtt{Gind\text{-}csa}^1_{\mathcal{S}_{\mathrm{xor}}}$ return exactly the same distributions to the adversary. For $i=t$, this is relatively easy to see: Since $sh_t$ is not returned in the real game, we do not need to assign it and thus, we just have $t-1$ shares which are sampled independently and uniformly at random and returned to the adversary in both games. In the following inlined game for $i=t$, observe that the cyan line can be removed:

| $\mathtt{Gind\text{-}csa}^0_{\mathsf{secsha}_{\mathrm{xor}}}$ | $\mathtt{Gind\text{-}csa}^1_{\mathcal{S}_{\mathrm{xor}}}$ |
|---|---|
| Parameters | Parameters |
| $\lambda$: security parameter | $\lambda$: security parameter |
| $\mathsf{secsha}_{\mathrm{xor}}$: secret sharing scheme | $\mathcal{S}_{\mathrm{xor}}$: simulator |
| Package State | Package State |
| $d$: call-once bit | $d$: call-once bit |
| $\mathsf{SECSHA}(s,t,i=t)$ | $\mathsf{SECSHA}(s,t,i=t)$ |
| **assert** $d\neq 1$ | **assert** $d\neq 1$ |
| $d\leftarrow 1$ | $d\leftarrow 1$ |
| **assert** $i\in\{1,..,t\}$ | **assert** $i\in\{1,..,t\}$ |
| | $\ell\leftarrow\lvert s\rvert$ |
| **for** $j=1..t-1$ : | **for** $j=1..t-1$ : |
| $sh_j \leftarrow\!\!\$\ \{0,1\}^{\lvert s\rvert}$ | $sh_j \leftarrow\!\!\$\ \{0,1\}^{\ell}$ |
| $sh_t \leftarrow s \bigoplus_{j=1}^{t-1} sh_j$ | |
| **return** $sh_1,..,sh_{t-1}$ | **return** $sh_1,..,sh_{t-1}$ |

For $i \neq t$, we need to observe that in the real secret-sharing scheme, we can actually swap the roles of any $sh_i$ with the role of the share $sh_t$ and obtain the same distribution. Then, the same argument applies, i.e., in the description below, the assignment to $sh_i$ (marked in cyan) can be removed.

$\underline{\texttt{Gind-csa}^0_{\mathsf{secsha}_{\mathrm{xor}}}}$

Parameters

| | |
|---|---|
| $\lambda$: | security parameter |
| $\mathsf{secsha}_{\mathrm{xor}}$: | secret sharing scheme |

Package State

| | |
|---|---|
| $d$: | call-once bit |

SECSHA$(s,t,i)$

**assert** $d \neq 1$
$d \leftarrow 1$
**assert** $i \in \{1,..,t\}$

**for** $j = 1..i-1 \wedge i+1..t$ :
  $sh_j \leftarrow\!\$\ \{0,1\}^{|s|}$
$sh_i \leftarrow s \bigoplus_{j=1}^{t-1} sh_j$

**return** $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t$

$\underline{\texttt{Gind-csa}^1_{\mathcal{S}_{\mathrm{xor}}}}$

Parameters

| | |
|---|---|
| $\lambda$: | security parameter |
| $\mathcal{S}_{\mathrm{xor}}$: | simulator |

Package State

| | |
|---|---|
| $d$: | call-once bit |

SECSHA$(s,t,i)$

**assert** $d \neq 1$
$d \leftarrow 1$
**assert** $i \in \{1,..,t\}$
$\ell \leftarrow |s|$
**for** $j = 1..i-1 \wedge i+1..t$ :
  $sh_j \leftarrow\!\$\ \{0,1\}^{\ell}$

**return** $sh_1,..,sh_{i-1},sh_{i+1},..,sh_t$

□

## 1.5 Additive Homomorphism

We say that a secret-sharing scheme is *additively homomorphic* if summing the shares of a secret-sharing scheme corresponds to summing up the secret[2], i.e., when running the following experiment

Experiment$(s^0,s^1,t)$

$sh^0_1,..,sh^0_t \leftarrow\!\$\ \mathsf{secsha}(s^0,t,1^\lambda)$
$sh^1_1,..,sh^1_t \leftarrow\!\$\ \mathsf{secsha}(s^1,t,1^\lambda)$
**for** $j = 1..t$
  $sh^{\oplus}_i \leftarrow sh^0_i \oplus sh^1_i$
$s^{\oplus} \leftarrow \mathsf{decode}(sh^{\oplus}_1,..,sh^{\oplus}_t)$
**return** $s^{\oplus}$

then $s^{\oplus}$ will be equal to $s^0 \oplus s^1$. This is indeed the case for $\mathsf{secsha}_{\mathrm{xor}}$. In the next section, we will use this feature for secure multi-party computation.

[2]We here express properties while using xor as addition, but this notion is meaningful more generally as well for other notions of addition as we will see in Section 5.

## 2 Computing the AND securely

Penguin A, B and C want to go to a restaurant together, but each of them has a different budget, and they don't want to reveal their budget since this could make their friends uncomfortable. We will now see how secret-sharing can help them make a decision that suits everyone without revealing their different budgets. The penguins decide restaurant by restaurant. Let's say they start with the very fancy *Poisson Rouge*. Each penguin has an opinion on the restaurant, 0 means no and 1 means yes.

$$A : 0$$
$$B : 0$$
$$C : 1$$

The penguins want to compute the AND function of their bits in such a way that they only learn the result 0 (since $0 = 0 \wedge 0 \wedge 1$) but nothing else. That is,

(0) None of the participants with a 1 should be able to distinguish whether the other penguins held the values 00, 01 or 10.

(1) None of the participants with a 0 should be able to distinguish whether the other penguins held the values 00, 01, 10 or 11.

We can also express this property using the simulation paradigm: There exists simulator $\mathcal{S}$ which can emulate a party's view, given only that party's input and the output of the protocol. How can we construct such a protocol. Let us consider the following protocol:
If a penguin's bit is 0, it samples a two random strings $r_0 \leftarrow\$ \{0,1\}^\lambda$ and $r_1 \leftarrow\$ \{0,1\}^\lambda$.
If a penguin's bit is 1, it samples a single random string $r \leftarrow\$ \{0,1\}^\lambda$ and then assigns $r_0 \leftarrow r$ and $r_1 \leftarrow r$.
Now, each penguin A, B, C holds two shares, let us denote them by $r_0^A$, $r_1^A$, $r_0^B$, $r_1^B$, $r_0^C$, and $r_1^C$. Now, penguin A passes $r_0^A$ to penguin B. Penguin B xors $r_0^B$ on top and passes the result to penguin C. Penguin C xors $r_0^C$ on top and passes the result to penguin A. Now, penguin A xors $r_1^A$ on top and passes the result to penguin B. Penguin B xors $r_1^B$ on top and passes the result to penguin C. Finally, penguin C xors $r_1^C$ on top. If the resulting string is the all-zeroes string, then penguin C tells A and B that this is the restaurant of their common choice. Else, penguin C tells the other penguins that they need to keep looking.
This protocol achieves correctness, because if all penguins have a 1, then

$$r_0^A \oplus r_0^B \oplus r_0^C \oplus r_1^A \oplus r_1^B \oplus r_1^C$$

is equal to

$$r_0^A \oplus r_0^B \oplus r_0^C \oplus r_0^A \oplus r_0^B \oplus r_0^C$$

which is the all-zeroes string, since all cancel out.
In turn, if at least one penguin has a 0, then

$$r_0^A \oplus r_0^B \oplus r_0^C \oplus r_1^A \oplus r_1^B \oplus r_1^C$$

is equal to a uniformly random string from $\{0,1\}^\lambda$ which is the very unlikely to be the all-zeroes string.
Now, what about the security of this protocol? We will see that we can simulate the view of penguin A and penguin B, but that we cannot perfectly emulate the view of penguin C. Penguin A creates its own random strings and then sees another uniformly random string. Penguin B sees two uniformly random strings and creates its own uniformly random string. But penguin C sees the xors of the shares of penguin A and penguin B and can thus derive the result of their computation, i.e., penguin C can distinguish between the case that penguin A and B both hold a 1 and the other cases. In case that penguin C holds a 1 himself, he can already knows this information, so in that case, this is fine. But if penguin $C$ holds a 0, then the result of the computation does not reveal this information.

# 3 Secure evaluation of an arbitrary function

In this section, we describe how two parties can securely evaluate a function on their inputs such that both parties learn

- the output of the function, but
- nothing else.

Especially, the parties do not learn the input of the other party. Security in this protocol holds as long as the parties follow the protocol. This security notion is known as *honest-but-curious*.

## 3.1 Oblivious transfer

1-out-of-4-oblivious transfer (OT) is a protocol which allows party A(lice) to transfer one out of 4 values to party B(ianca) such that

- Bianca chooses which of the 4 values she wants to learn.
- Bianca learns nothing about the 3 other values.
- Alice does not learn which of the 4 values Bianca learned.

For an example of an OT protocol, see the exercise sheet. Note that the exercise sheet treats a 1-out-of-2-OT protocol. For completeness, Appendix A describes how to build a 1-out-of-4-OT protocol out of a 1-out-of-2-OT protocol.

## 3.2 Securely evaluating a function

For two parties A(lice) and B(ianca) to evaluate a function $f$ on their input bitrings $a$ and $b$, they first describe the function $f$ as a Boolean circuit made out of $\oplus$ (xor) and $\wedge$ (and) gates. The protocol then proceeds gate-by-gate from top to bottom. We now describe each of the steps which Alice and Bianca perform.

**Input.** In order to secret-share a bit $a_i$ of her own input, Alice samples two random values $x$ and $y$ such that $a_i = x \oplus y$. This is analogous to the xor-based secret-sharing we discussed before. Then, Alice sends one of the two shares over to Bianca and stores the other share for herself. Bianca also secret-shares here input bits in this way.

**$\oplus$ gate** For an $\oplus$ gate, if Alice has a share $x_{\text{left}}$ and $x_{\text{right}}$ for the input to the $\oplus$ gate, then she computes a share $z_{\text{Alice}} = x_{\text{left}} \oplus x_{\text{right}}$ for the output of the $\oplus$ gate. Analogously, Bianca has a share $y_{\text{left}}$ and $y_{\text{right}}$ for the input to the $\oplus$ gate and computes a share $z_{\text{Bianca}} = y_{\text{left}} \oplus y_{\text{right}}$ for the output of the $\oplus$ gate. Note that if

$$a = x_{\text{left}} \oplus x_{\text{right}} b = y_{\text{left}} \oplus y_{\text{right}}$$

then

$$a \oplus b = z_{\text{left}} \oplus z_{\text{right}}$$

as required by definition of $z_{\text{left}}$ and $z_{\text{right}}$.

**$\wedge$ gate** For a $\wedge$ gate, we are now going to use 1-out-of-4-OT. Namely, Alice has a share $x_{\text{left}}$ and $x_{\text{right}}$ for the input to the $\wedge$ gate, and Bianca has a share $y_{\text{left}}$ and $y_{\text{right}}$ for the input to the $\wedge$ gate. Alice now samples a random bit $\sigma$ and computes the values

$$\begin{aligned} s_{00} &\leftarrow ((x_{\text{left}} \oplus 0) \wedge (x_{\text{right}} \oplus 0)) \oplus \sigma \\ s_{10} &\leftarrow ((x_{\text{left}} \oplus 1) \wedge (x_{\text{right}} \oplus 0)) \oplus \sigma \\ s_{01} &\leftarrow ((x_{\text{left}} \oplus 0) \wedge (x_{\text{right}} \oplus 1)) \oplus \sigma \\ s_{11} &\leftarrow ((x_{\text{left}} \oplus 1) \wedge (x_{\text{right}} \oplus 1)) \oplus \sigma. \end{aligned}$$

and then, Alice and Bianca perform a 1-out-of-4-OT with $s_{00}$, $s_{10}$, $s_{01}$ and $s_{11}$ as Alice's 4 inputs and $b_0 = y_{\text{left}}$ and $b_1 = y_{\text{right}}$ as Bianca's inputs. In the end, the value which Bianca learned and $\sigma$ are a secret-sharing of the output of the $\wedge$ gate.

**Output gate** Finally, in the end of the computation, Alice and Bianca obtain the output of the computation simply by xoring their final shares.

# 4 Literature

This was a very brief introduction into secure multi-party computation. If you are curious for how to perform arbitrary multi-party computation, you can have a look into this tutorial by Yehuda Lindell `https://eprint.iacr.org/2016/046.pdf` or follow the talks of the Bar-Ilan Winter School on Secure Multi-Party Computation `http://cyber.biu.ac.il/event/the-5th-biu-winter-school/`.

# 5 Shamir's Secret Sharing Scheme

In this section, we cover Shamir's Secret Sharing scheme which has interesting and advanced features. We start by discussing these features first and then turn to Shamir's Secret Sharing scheme.

## 5.1 Threshold Secret-Sharing

Imagine that Alice sends mail pigeons to Bianca instead of using electronic mails, and let us assume that some mail pigeons might decide to fly south instead of flying to Bianca[3]. In this case, Biance might not be able to recover the secret. Thus, Alice might consider to use a *threshold* secret-sharing instead which allows Bianca to recover the secret already from less shares.
Syntactically, a threshold secret-sharing scheme gives a number $n$ of desired shares *and* a threshold $t$ to the encoding algorithm, and the encoding algorithm generates $n$ shares such that $t$ of these $n$ shares suffice to already recover the secret. Moreover, any $t-1$ shares yield no information.

**Definition 5.1** (Syntax Threshold Secret Sharing)**.** A *threshold secret sharing scheme* $\mathsf{thsecsha}$ consists of two PPT algorithms $(\mathsf{encode}, \mathsf{decode})$ with an associated domain $D_\lambda$ for each security parameter $\lambda$ such that the following correctness condition holds:

$$\begin{aligned} &\forall \lambda \in \mathbb{N} \forall s \in D_\lambda \, \forall n \geq t \in \mathbb{N}, \text{ for all } \{i_1, .., i_t\} \subseteq \{1, .., n\} : \\ &\qquad \Pr_{sh_1,...,sh_n \leftarrow \$\mathsf{encode}(s,n,t,1^\lambda)}[s = \mathsf{decode}(sh_{i_1}, ..., sh_{i_t})] = 1 - \mathsf{negl}(\lambda), \end{aligned}$$

where $\mathsf{negl}(\lambda)$ is a negligible function in $\lambda$.

[3]This story has not been verified by an ornithologist :-)

The security of a threshold secret sharing scheme is defined by enhancing the previously discussed IND-CSA notion into an IND-CSTA notions which additionally allows the adversary to choose a threshold and decide which shares it would like to learn.

**Definition 5.2** (IND-CSTA)**.** A *threshold secret sharing scheme* $\mathsf{thsecsha}$ is *indistinguishable under chosen secret and thresholds attacks* if there exists a PPT algorithm $\mathcal{S}$ such that the games $\texttt{Gind-csta}^0_{\mathsf{thsecsha}}$ and $\texttt{Gind-csta}^1_{\mathcal{S}}$ are indistinguishable, i.e., for all PPT adversaries $\mathcal{A}$, the advantage

$$\mathbf{Adv}^{\texttt{Gind-csta}^0_{\mathsf{thsecsha}},\texttt{Gind-csta}^1_{\mathcal{S}}}_{\mathcal{A}}(\lambda) := \big|\Pr\big[1 = \mathcal{A} \to \texttt{Gind-csta}^0_{\mathsf{thsecsha}}\big] - \Pr\big[1 = \mathcal{A} \to \texttt{Gind-tcsa}^1_{\mathcal{S}}\big]\big|$$

is negligible, where the games $\texttt{Gind-csta}^0_{\mathsf{thsecsha}}$ and $\texttt{Gind-csta}^1_{\mathcal{S}}$ are defined as

| $\underline{\texttt{Gind-csta}^0_{\mathsf{thsecsha}}}$ | $\underline{\texttt{Gind-csta}^1_{\mathcal{S}}}$ |
|---|---|
| Parameters | Parameters |
| $\lambda$: security parameter | $\lambda$: security parameter |
| $\mathsf{secsha}$: secret sharing scheme | $\mathcal{S}$: simulator |
| Package State | Package State |
| $d$: call-once bit | $d$: call-once bit |

| $\underline{\mathsf{SECSHA}(s, n, t, \{i_1, .., i_{t-1}\})}$ | $\underline{\mathsf{SECSHA}(s, n, t, \{i_1, .., i_{t-1}\})}$ |
|---|---|
| **assert** $d \neq 1$ | **assert** $d \neq 1$ |
| $d \leftarrow 1$ | $d \leftarrow 1$ |
| **assert** $\lvert\{i_1, .., i_{t-1}\}\rvert = t - 1$ | **assert** $\lvert\{i_1, .., i_{t-1}\}\rvert = t - 1$ |
| **assert** $\{i_1, .., i_{t-1}\} \subseteq \{1, .., n\}$ | **assert** $\{i_1, .., i_{t-1}\} \subseteq \{1, .., n\}$ |
| | $\ell \leftarrow \lvert s\rvert$ |
| $sh_1, .., sh_n \leftarrow\!\!\$\ \mathsf{thsecsha}(s, n, t, 1^\lambda)$ | $sh_{i_1}, .., sh_{i_{t-1}} \leftarrow\!\!\$\ \mathcal{S}(n, t, \ell, \{i_1, .., i_{t-1}\}, 1^\lambda)$ |
| **return** $sh_{i_1}, .., sh_{i_{t-1}}$ | **return** $sh_{i_1}, .., sh_{i_{t-1}}$ |

## 5.2 Shamir's Secret Sharing Scheme

| $\underline{\mathsf{secsha}^q_{\mathrm{Shamir}}.\mathsf{encode}(s, n, t, 1^\lambda)}$ | $\underline{\mathsf{secsha}^q_{\mathrm{Shamir}}.\mathsf{decode}(sh_{i_1}, .., sh_{i_t})}$ | $\underline{\mathcal{S}^q_{\mathrm{Shamir}}(n, t, \ell, \{i_1, .., i_{t-1}\}, 1^\lambda)}$ |
|---|---|---|
| $a_0 \leftarrow s$ | Interpolate to find a polynomial | **for** $j = 1..t-1$ : |
| **for** $j = 1..t-1$ : | that passses through | $x_j \leftarrow\!\!\$\ \{0, .., q-1\}$ |
| $a_j \leftarrow\!\!\$\ \{0, .., q-1\}$ | $sh_{i_1}, .., sh_{i_t}$ | $y_j \leftarrow\!\!\$\ \{0, .., q-1\}$ |
| **for** $j = 1..n$ : | $s \leftarrow a_0$ | $sh_{i_j} \leftarrow (x_j, y_j)$ |
| $x_j \leftarrow\!\!\$\ \{0, .., q-1\}$ | **return** $s$ | **return** $sh_{i_1}, .., sh_{i_{t-1}}$ |
| $y_j \leftarrow \sum_{m=0}^{t-1} a_m (x_j)^m \bmod q$ | | |
| $sh_j \leftarrow (x_j, y_j)$ | | |
| **return** $(sh_1, .., sh_t)$ | | |

Figure 2: Shamir's secret-sharing scheme, parametrized by a prime number $q$. Secrets are allowed to be any number in $\{0, .., q-1\}$

# A 1-out-of-2 OT implies 1-out-of-4 OT

Alice holds four inputs $a_{00}$, $a_{10}$, $a_{01}$ and $a_{11}$. Bianca holds two bits $b_0$ and $b_1$. The goal of 1-out-of-4-OT is that Bianca learns $b_{b_0||b_1}$, where $||$ denotes concatenation. No one should learn anything else.
To build a 1-out-of-4 OT from a 1-out-of-2 OT, Alice now chooses four different keys $k_0^0$, $k_1^0$, $k_0^1$, and $k_1^1$ and then performs two 1-out-of-2-OT-protocols with Bianca. In the first 1-out-of-2 OT protocol, Bianca learns $k_{b_0}^0$, and in the second 1-out-of-2 OT protocol, Bianca leanrs $k_{b_1}^1$. Alice then generates four ciphertexts using double encryption.

$$
\begin{aligned}
c_{00} &\leftarrow\!\!\$ \ (\mathsf{enc}(k_0^0, \mathsf{enc}(k_0^1, (a_{00})))) \\
c_{10} &\leftarrow\!\!\$ \ (\mathsf{enc}(k_1^0, \mathsf{enc}(k_0^1, (a_{10})))) \\
c_{01} &\leftarrow\!\!\$ \ (\mathsf{enc}(k_0^0, \mathsf{enc}(k_1^1, (a_{01})))) \\
c_{11} &\leftarrow\!\!\$ \ (\mathsf{enc}(k_1^0, \mathsf{enc}(k_1^1, (a_{11}))))
\end{aligned}
$$

and sends $c_{00}, c_{10}, c_{01}$ and $c_{11}$ over to Bianca. Bianca can only decrypt $c_{b_0||b_1}$, since for all of the other four ciphertexts, at least one of the keys is missing. This method is very related to *garbling schemes*, which we did not treat in this course. If you are curious, you can ask Kirthi for more information about garbling.