# Privacy Enhancing Technologies FS2026
# Exercise Sheet 6 (version 1)

Florian Tramèr

**Problem 1: Conceptual Questions.** For each of the following statements, say whether it is TRUE or FALSE. Write at most one sentence to justify your answer.

(a) In a world where $\texttt{P} = \texttt{NP}$,

i. additive secret sharing schemes exist.

ii. Two parties $P_1$ and $P_2$ can securely compute the function $f(x_1, x_2) = x_1 + x_2$ with semi-honest security.

(b) For every function $f : \mathcal{X} \times \mathcal{X} \to \{0,1\}$, there exists a secure two-party computation protocol for $f$, such that if the two parties' inputs are equal, the protocol reveals to each party the input of the other party.

(c) Let $p$ be a prime number. A set of $d+1$ points $(x_0, y_0), \ldots, (x_d, y_d) \in \mathbb{Z}_p^2$ determines a unique polynomial $f \in \mathbb{Z}_p[X]$ of degree $d$ such that $f(x_i) = y_i$ for every $i = 0, 1, \ldots, d$.

**Problem 2: Maliciously secure MPC.** We assume an authenticated broadcast channel which reliably delivers messages to all parties (i.e., when party $i$ sends message $m$, all parties receive $m$ and know that party $i$ sent it to everyone). We further assume that no party aborts the protocol by not sending a message, and that Beaver triplets have been set up in advance.

(a) Consider the MPC protocol from the lecture applied to some circuit $\mathcal{C} : \{0,1\}^n \to \{0,1\}$. Show that a malicious party can cause all benign parties to receive an arbitrary answer $\hat{b} \in \{0,1\}$.

(b) As a first step towards a maliciously-secure protocol, we need a *fair* communication channel. Such a channel has the following interface:

- $\texttt{Push}(x^{(i)})$: the $i$-th party pushes their message.
- $\texttt{Pull} \to x^{(i)}$ or $\perp$: if all $n$ parties have pushed their message and party $i$ calls `Pull`, all other parties receive party $i$'s message. If one or more parties have not pushed yet, the call has no effect.

Informally, this communication channel ensures that no party can choose their message based on the value of other messages. Show how to implement `Push` and `Pull` using tools seen in class, and argue that your construction is secure.

(c) Show that if all parties communicate over a fair channel, a malicious party can still cause all benign parties to receive an incorrect output $\hat{b} \neq \mathcal{C}(x^{(1)}, \ldots, x^{(n)})$.

(d) To prevent such tampering, as described in (c), modern MPC protocols *authenticate* the secret shares and check that opened values are correct. Specifically, for each secret-shared value $x$, there is an associated "tag"

$$t = a \cdot x \ ,$$

where $a$ is a global secret that is additively secret-shared among all parties. The value $x$ and the tag $t$ are always secret shared among all parties.

Given shares of $[a]$ and of values $[x], [y]$ with associated tags $[t], [u]$, show how the parties can compute authentication tags for $[x + y]$ and $[x \cdot y]$.

(e) Suppose all parties open the shares of some value $[x]$. Give a protocol that allows the parties to check that $x$ has a correct tag. Your protocol should require all parties to send a single message to all other parties. If all parties behave honestly, none should learn the global secret $a$. Prove that your scheme reveals nothing about $a$ to honest parties, and that a cheating party (who reveals some incorrect share $\hat{x}_i = x_i + \Delta$) must know $a$ to avoid being caught.

**Problem 3: Beaver squares.**

(a) Suppose that parties $P_1, \ldots, P_n$ have additive shares of some value $[x]$ and want to compute shares of $[x^2]$. Show how to do this using a trusted dealer who generates Beaver multiplication triples.

(b) Now assume that the trusted dealer provides shares of a random *square*, i.e., $[a], [a^2]$ for some random $a \leftarrow^{\$} \mathbb{Z}_p$. Show how the parties can securely compute shares of $[x^2]$ from the shares of $[x]$.

How does your solution compare in terms of communication complexity (between parties, without counting the dealer's communication) to the solution in (a)?

(c) Let $f(X)$ be a polynomial of degree $d$ over $\mathbb{Z}_p$, with public coefficients $\beta_0, \ldots, \beta_d$. The parties want to securely compute shares of $[f(x)]$ from the shares of $[x]$.

Show how to do this using a trusted dealer who sends each party $d$ values.

**Hint:** Recall the binomial expansion: $(a+b)^k = \sum_{i=0}^{k} \binom{k}{i} a^i b^{k-i}$.

**Problem 4: Threshold signatures.**

(a) Let $g$ be a polynomial of degree $d$ over $\mathbb{Z}_p$. Let $I$ be a set of $d+1$ distinct elements of $\mathbb{Z}_p$. Show that $g$ can be written as

$$g(x) = \sum_{i \in I} g(i) \underbrace{\prod_{i' \in I \setminus \{i\}} \frac{x - i'}{i - i'}}_{\lambda_i(x)}$$

The polynomials $\lambda_i(x)$ are called *Lagrange interpolation coefficients*.

(b) Let $\mathbb{G}$ be a group of prime order $p$. Let $I \subseteq \mathbb{Z}_p$ be a set of size $t$, and let $i^* \in \mathbb{Z}_p$. Given $I$, $i^*$, as well as a collection of group elements of the form $\{h^{\omega(i)}\}_{i \in I}$, where $h \in \mathbb{G}$ and $\omega \in \mathbb{Z}_p[X]$ is a polynomial of degree at most $t-1$, we can efficiently compute the group element $h^{\omega(i^*)}$.

(c) Consider the BLS signature scheme defined below.

**Definition 1** (BLS signature scheme)**.** Let $\mathbb{G}$ be a group of prime order $p$ with generator $g$. Let $H$ be a hash function that maps messages to group elements in $\mathbb{G}$. Let $\mathcal{O}_{DDH}$ be a DDH oracle that, given $g^a, g^b, g^c$, outputs 1 if $c = ab$ and 0 otherwise.[a]

- $\texttt{KeyGen}() \to (\texttt{sk}, \texttt{pk})$: choose a random $\alpha \xleftarrow{\$} \mathbb{Z}_p$, compute $u \leftarrow g^\alpha$, and output $\texttt{pk} := u \in \mathbb{G}$ and $\texttt{sk} := \alpha$.
- $\texttt{Sign}(\texttt{sk}, m) \to \sigma$: compute $\sigma \leftarrow H(m)^\alpha \in \mathbb{G}$, where $\texttt{sk} = \alpha$.
- $\texttt{Verify}(\texttt{pk}, m, \sigma) \to \{0, 1\}$: output $\mathcal{O}_{DDH}(\texttt{pk}, H(m), \sigma)$.

[a] We can implement this oracle by choosing a special group $\mathbb{G}$ that admits an efficient *pairing* operation (we won't go into the details of pairings in this course).

Given a BLS signature scheme, define a *threshold* version of the scheme, which allows a set of $t$-out-of-$n$ signers to sign a message $m$, but no less. The scheme should consist of the following algorithms:

- $\texttt{KeyGen}(N, t) \to (\texttt{pk}, \texttt{pk}_1, \ldots, \texttt{pk}_N, \texttt{sk}_1, \ldots, \texttt{sk}_N)$: Generates a public key $\texttt{pk}$, N public key shares $\texttt{pk}_1, \ldots, \texttt{pk}_N$, and N signing key shares $\texttt{sk}_1, \ldots, \texttt{sk}_N$.
- $\texttt{Sign}(m, \texttt{sk}_i) \to \sigma_i$: Computes a signature share $\sigma_i$ for message $m$ using signing key share $\texttt{sk}_i$.
- $\texttt{Combine}(m, I, \{(\texttt{pk}_i, \sigma_i)\}_{i \in I}) \to \sigma$: Combines the signature shares $\sigma_i$ for message $m$ from the set of $t$ signers $I \subseteq [N]$ into a final signature $\sigma$.
- $\texttt{Verify}(m, \texttt{pk}, \sigma) \to \{0, 1\}$: Verifies the signature $\sigma$ for message $m$ using the public key $\texttt{pk}$.

The public key $\texttt{pk}$ and the final signature $\sigma$ in your scheme should each consist of a single group element, regardless of the values of $t$ and $N$.

Show that your scheme is correct, and informally argue that it is secure.

(d) Show that the combine algorithm can detect if a malicious signer gives a bad signature share $\sigma_i$.