Constructing protocols that are secure under concurrent general composition in the timing model.

Recap: zk protocols: sequential \[ \text{consider only} \] parallel \[ \text{fixed roles.} \] concurrent

Commitment protocols: non-malleability - specific type of composition.

⇒ So far we talked about specific tasks & specific type of composition.

Today: General tasks & general composition.

General task: A set of parties \( P_1, \ldots, P_n \).

Each part receives a private input, and they wish to jointly compute some function \( F \) of their inputs in a secure way, where some of the parties are malicious.

Defining security (for general tasks):

Security in the stand-alone model: (generalizes the security)

Protocol \( P \) securely realizes \( F \) if

\[ \forall \text{PPT } A \text{ (that corrupts any subset } I \text{ of parties)} \forall \text{PPT } S \text{ (that also corrupts the parties in } I \text{) s.t.} \]

\[ \{ \text{Real}_{AI}(x, z, 2) \} \leq \{ \text{Ideal}_{SI}(x, z, 2) \} \text{ for all } \]

\[ x \in \{0, 1\}^* \]
where:

$\text{Real}_{\mathcal{A},\mathcal{I}}(n,\Xi,\mathcal{I})$: is the sequence of outputs of all parties,
where parties $P_i$ not in $\mathcal{I}$ interact according to $\mathcal{A}$ on input $x_i$, and parties in $\mathcal{I}$ interact according to $A(\Xi,\mathcal{I})$.

$\text{Ideal}_{\mathcal{E},\Xi}(n,\Xi,\mathcal{I})$: is the sequence of outputs of all parties after communicating only with a trusted party computing $F$. Parties $P_i$ not in $\mathcal{I}$ send to the trusted party their input $x_i$ and output the value $y_i$ they receive from the trusted party. Parties $P_i$ in $\mathcal{I}$ send to the trusted party the value $x_{i}'$ given to it by $S(\Xi,\mathcal{I})$, and outputs the value given to it by $S(I,\Xi,\mathcal{I},y_i)$, where $y_i$ is the value that the trusted party sends to party $P_i$.

The adversary that we consider is static, malicious, and control the delivery of all msgs (though cannot modify or duplicate msgs).

Notice: Any functionality $F$ can be securely realized trivially by a protocol $\mathcal{P}$ which instructs all parties to simply output 1 (without interacting).

Non-triviality: A protocol $\mathcal{P}$ is said to be non-trivial if all parties generate outputs (not 1) when the adversary does not corrupt any party and delivers all msgs.
Notice the similarity to ZK:
- \( \forall V^* \exists S \text{ st } (R,V) \subseteq S \)
- \( \forall A \exists S \text{ s.t. } \text{Real} \neq \text{Ideal} \)

Fundamental Thm [GMW87]: Assuming the existence of enhanced trapdoor permutations, any multiparty functionality \( F \) can be securely realized by a non-trivial protocol.

So far: Security for any multiparty task in the stand-alone setting

- **Sequential composition:**
  Every secure protocol remains secure when composed sequentially (similar to ZK)

- **Parallel composition:**
  \( \forall F \exists \) protocol \( P \) for securely realizing \( F \), s.t. \( P \) remains secure under parallel composition [GL03]

- **Concurrent self-composition:**
  considers the case where a protocol \( P \) for securely computing some function \( F \) is run concurrently many times.

  Negative results: "Most" interesting functionalities (such as ZK) cannot be securely realized under concurrent self-composition [CKLO3, LO3]

  (unless we assume honest majority or trusted setup)

  (assumptions such as CRS model)
Goal: Concurrent general composition

considers the case where a protocol $F$ is run securely, realizing some function $F$, in a network of arbitrary protocols running concurrently. We model this arbitrary network as a "calling protocol" $\mathcal{C}$. $\mathcal{C}$ is a protocol that contains ideal calls to $F$. $F^*$ is the composed protocol obtained by replacing the ideal calls to $F$ with executions of $F$.

Intuitively, security is defined by requiring that $\forall \mathcal{C}$ (that contains ideal calls to $F$), an adversary $A$ interacting with $F^*$ can do no more harm than in $\mathcal{C}^F$ $\Rightarrow F$ behaves just like an ideal call to $F$, even when running concurrently with any other protocol $\mathcal{C}$.

Definition (Security under concurrent general composition): $F$ securely realizes $F$ under concurrent general composition if $\forall \mathcal{C}$ (with ideal calls to $F$) and $\forall PPT A$ (interacting with $\mathcal{C}^F$) that corrupts parties in $\mathcal{C}$, $PPT A$ (interacting with $\mathcal{C}^F$) that corrupts parties in $\mathcal{C}$, s.t.

$$\{\text{Real}_{F,A,I}^\mathcal{C}(n,x,z)\}_{x \in \{0,1\}^n} \approx \{\text{Hybrid}_{F,A,I}^\mathcal{C}(n,x,z)\}_{x \in \{0,1\}^n}$$

where:

$\text{Real}_{F,A,I}^\mathcal{C}(n,x,z)$: is the sequence of outputs of all parties, where parties $P_j$ not in $I$ interact according to $\mathcal{C}^F$.
on input $x$, and parties in $I$ interact according to $A(x, z, n)$, and output whatever is instructed by $A$.

$\text{Hybrid}_{F}^{n}(x, y, z)$ is the sequence of outputs of all parties, where parties $P_y$ not in $I$ interact according to $E^F$ on input $x$, and parties in $I$ interact according to $S(n, x, y, z)$.

Remarks:
- This definition is equivalent to the UC security definition [C01].
- Note that if $F$ is only allowed to send ideal calls to $F$, then we get concurrent self-composition.

[CK03]: There exist large classes of functions (including ZK) that cannot be securely realized under concurrent general composition (with non-trivial protocols).

[LO3]: The above impossibility results also hold for the case of self-composition.

What to do?

Weaken notions of security

Idea: provide the hybrid adversary S with more power than real adversary $A$

Every $F$ can be securely in the concurrent setting, assuming:

- honest majority [C01]
- CRS model [CLOS02]
- PK model [BCCN04]

Network Assumptions

- bounded concurrency [PO4]
- running time
Goal: Achieve concurrent general composition in the timing model

Timing model: Every party has a local clock.

- This model has been previously used in cryptography to achieve concurrent ZK [CNS98,GO2]

Timing model assumptions:

- E-clock drift: The parties' local clocks proceed at "approximately" the same rate. Specifically, when one local clock advances $t$ time units every $c$ time units, where $\frac{1}{2}c \leq t' \leq c (1 + \epsilon)$

- A-latency: There is a known upper bound $A$ on the latency of the network (we assume for simplicity that local computation is instantaneous).

Note: The second assumption is far more problematic than the first.

The security of our protocol relies only on the first assumption.

The latency assumption is only used to ensure non-triviality of our protocol.
Moreover, our protocols have the property that if they fail to converge only because of high network latency, then they can be restarted without losing safety.

Note: We assume given to the adversary in its control model: We work the protocol in the absence of clock. The clock is used by the adversary.
Main Theorem: Assume enhanced trapdoor permutations exist. Then \( \exists \alpha > 0 \) s.t. \( \forall f \in \mathcal{F} \), \( \forall \mathcal{F} \) that securely realizes \( f \) under concurrent general composition in the timing model with \( \alpha \), assuming all network msgs are delayed (by \( d = d(\mathcal{F}, \alpha) \) time units).

\[ \text{Let } \forall \mathcal{F} \in \mathcal{F} \text{ s.t. } \text{Real }_{\mathcal{F}, \alpha} \subseteq \text{Hybrid }_{\mathcal{F}, \alpha} \]

However, \( \mathcal{F} \) is non-trivial under \( (\alpha, \mathcal{A}) \)-timing assumptions.

Main Idea:

[CLOS02]: Every \( f \) can be computed securely under concurrent general composition (by a non-trivial protocol) in the CRS-model.

I.e., \( \forall \mathcal{F} \in \mathcal{F} \text{ s.t. } \text{Real }_{\mathcal{F}, \alpha} \subseteq \text{Hybrid }_{\mathcal{F}, \alpha} \)

Our attempt: Securely compute \( \text{Real }_{\mathcal{F}, \alpha} \) under concurrent general composition in the \( (\alpha, \mathcal{A}) \)-timing model.

Unfortunately, we show that this cannot be achieved.

(Will not explain in class).

Instead, we did succeed in proving the following claim.

Claim: \( \exists \) protocol \( \mathcal{F} \) for securely realizing \( \text{Real }_{\mathcal{F}, \alpha} \) in the \( (\alpha, \mathcal{A}) \)-timing model (assuming \( \mathcal{F} < V \) ) under:

- concurrent self-composition
- concurrent general composition where all msg are delayed.
Protocol $\mathcal{P}$ for securely realizing $\mathcal{F}$:

We're going to use two special instructions:
- **timeout ($s_{(E,A)}$)**: If more than $s_{(E,A)}$ time units passed (and some msg was not yet received), then output timeout.
- **delay ($d_{(E,A)}$)**: Wait $d_{(E,A)}$ time units before sending the next msg.

**Overview:** Assume 2-parties for simplicity.

**Basic Idea:**

1. Each party commits to a random value, and gives a zk proof of knowledge (zkPOK) of the committed value.
2. Each party reveals its committed value (without decommitting), and gives a zkP that the revealed value is indeed the one committed to.

*The idea is that the simulator and force the output to be any value (and in particular Reqs), by extracting all the committed values, and then simulating one of the honest parties revealing a value that would lead to Reqs and simulating its zkP.*

**Crucial point:** The zkP's and the zkPOK's must be "independent" of each other.

In the stand-alone setting, this independence is achieved.
by running the proofs sequentially.
(Technically, this allows rewinding)

* In the concurrent setting it is impossible to enforce any scheduling that will ensure independence.

* In the concurrent synchronous setting one can use (a variant of) Chor-Rabin scheduling $8m$.

(Originally used for the stand-alone synchronous case, to reduce the number of rounds, and thus avoiding sequential executions.

This scheduling assumes each party $P_j$ has a unique identifier $ID_j$ with exactly $m$ zeros and $m$ ones. Consider $8m$ time slots, where party $P_j$ executes the protocol in the $j$th time slot iff $(ID_j)_j = 1$.

Thus $P_k, P_j$ at time slot $k$ where $P_k$ is executing and $P_j$ is not, and vice versa.

* We show: The Chor-Rabin scheduling can be adapted to the timing model (by inserting timeout and delay instructions), resulting with the following property:

Pairwise disjointness: for two sessions $j$ at least one execution in the first session that does not overlap with any of the executions of the second session, and vice versa.
We are not going to elaborate further on the construction of a pairwise disjoint scheduling. It was shown that such a scheduling exists in the (e,a)-timing model with $a \leq \sqrt{3}$. In order to allow restarting the protocol without harming the security (in case the protocol failed to conclude due to high network latency), we need to limit the scheduling to be early in the protocol (before Bob is revealed). We achieve this by using the specific ZK of Feige-Shamir [9]:

$$P(u) \rightarrow e^2 \rightarrow V$$

Pairwise disjoint schedule will be applied only to this part

$$\text{ZKPOK} \leftarrow [f^-(u) v f^-(u)]$$

Soundness follows from the witness hiding of a ZKPOK with 2 independent witnesses.]

Our protocol consists of 3 phases:

Phases 1: Each party sends $\chi_i, \chi_i'$ and gives a ZKPOK: $\text{ZKPOK}(f^-(u_i) v f^-(u_i'))$ according to a pairwise disjoint scheduling.
Phase 2: Each party commits to a random value $r_i$: $c_i = \text{com}(r_i, s_i)$, and gives to every party $P_j$ a $\text{WIPROK}(c_i, f^{-1}(u_i^j) \lor f^{-1}(v_i^j))$.

Phase 3: Each party sends $r_i$ and gives to every party $P_j$ a $\text{WIPROK}(s_i, f^{-1}(u_i^j) \lor f^{-1}(v_i^j))$.

- Notice that the pairwise disjoint scheduling is before the $r_i$'s are revealed!

Alert: Many details are missing. This is not exactly the protocol. In particular we need strong WIPROKs (which means an extractor can extract with overwhelming probability from any prover who succeeds with non-negligible probability) rather than regular WIPROKs.