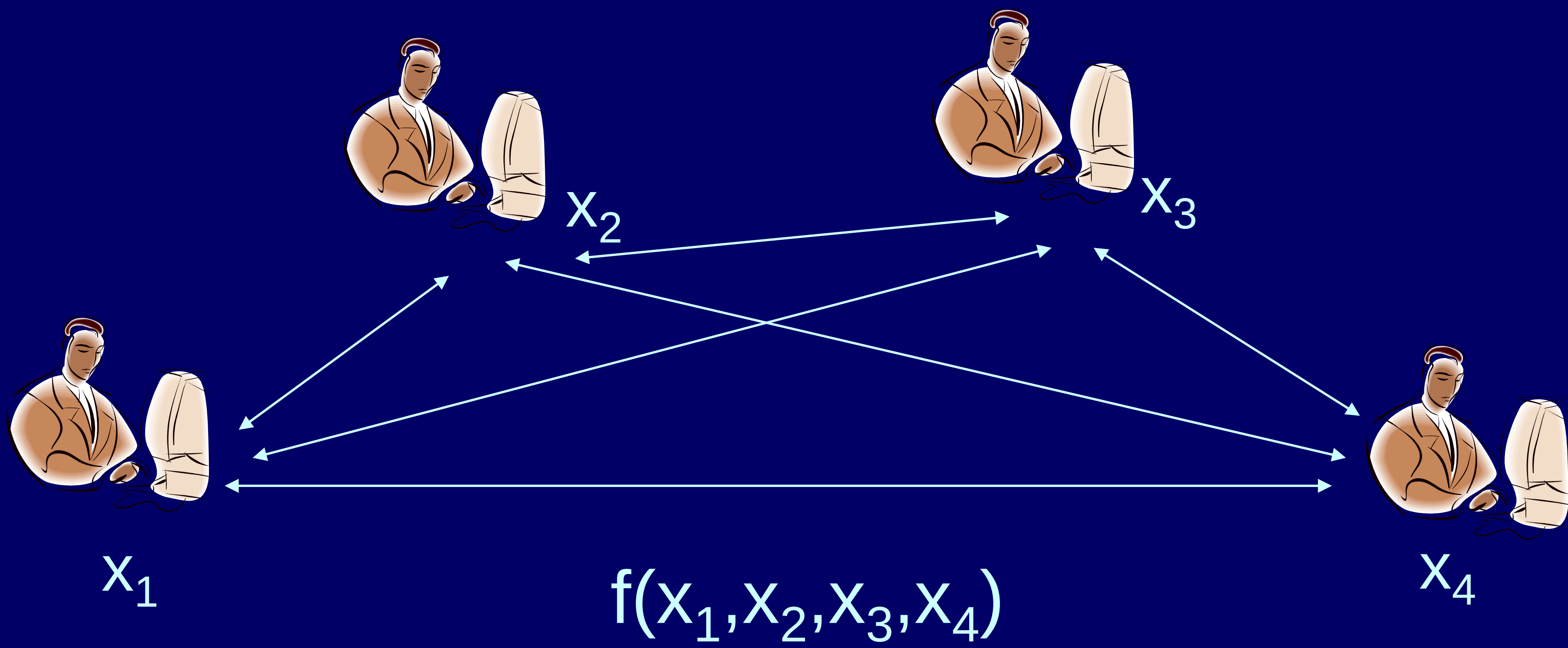
New Constructions for Universally Composable Computation Using Tamper Proof Hardware

Vipul Goyal UCLA

(Joint work with Nishanth Chandran and Amit Sahai)

Secure Multi-Party Computation [Yao86, GMW87]



No information other than $f(x_1,x_2,x_3,x_4)$

Secure Multi-Party Computation Contd..

- Initially considered only in the isolated setting. General positive results by [Yao86, GMW87]
- Canetti [Canetti01] introduced the Universal Composability framework to study protocols in complex environments like the internet

Feasibility of UC Computation

- UC computation known to be impossible for a large class of functionalities [CF01, CKL03]
- The above far-reaching impossibility results hold only in the *plain model* (no trust assumptions, no setup: the vanilla model)

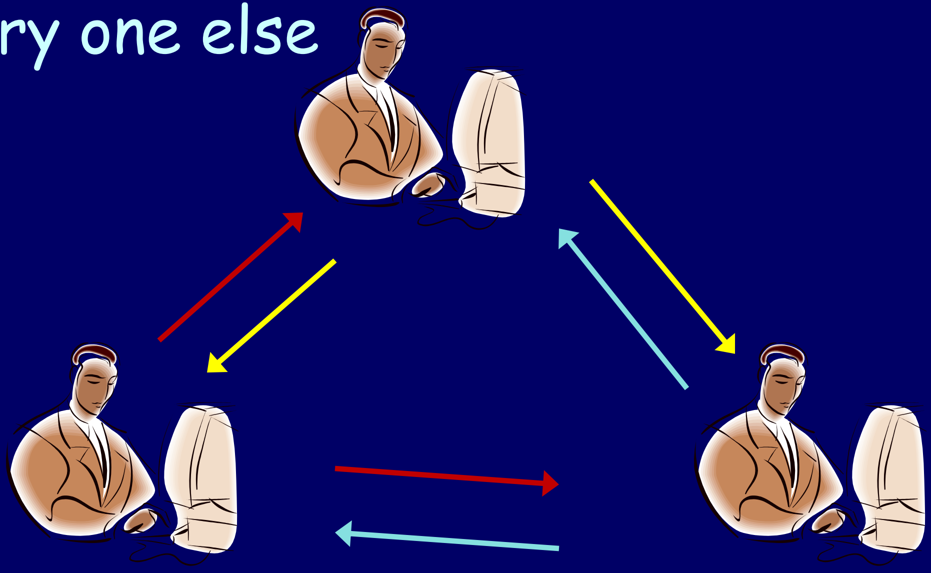
Augmenting the Model

- Feasibility of UC can be regained assuming:
  - A majority of the parties are honest [BGW89, Can01]
  - There exists a trusted "common reference string" (CRS) available to all parties [CLOS02]
  - Other setup assumptions like public key registration

- Katz [Katz07] proposed a "physical assumption" sufficient for UC computation. Does not require a party to place any trust in others.

UC Computation using Tamper Proof Hardware [Katz 07]

- Token Exchange [One time Process]: Every party sends **tamper proof hardware** (TPH) tokens to every one else



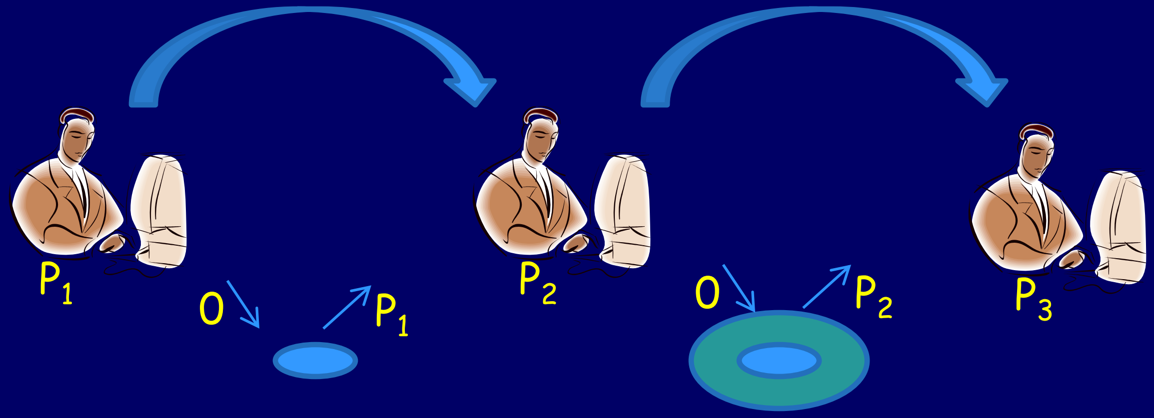
- Tokens cannot communicate back with its creator
- During the protocol execution, interaction with tokens received required

UC Computation using Tamper Proof Hardware Contd ..

- Katz modeled tokens as ITM (which run a multi-round protocol)
  - Thus tokens have to reliably keep state (even when e.g. the power supply is cut off)
- General feasibility results based on DDH provided
- Security proofs based on rewinding the token received from malicious parties
  - Assumption: malicious sender "knows" the code of the tokens which he distributed

Knowing the code

- Undesirable: doesn't capture real life attacks where an adversary passes a token received from an honest party to another honest party
- A naïve fix:



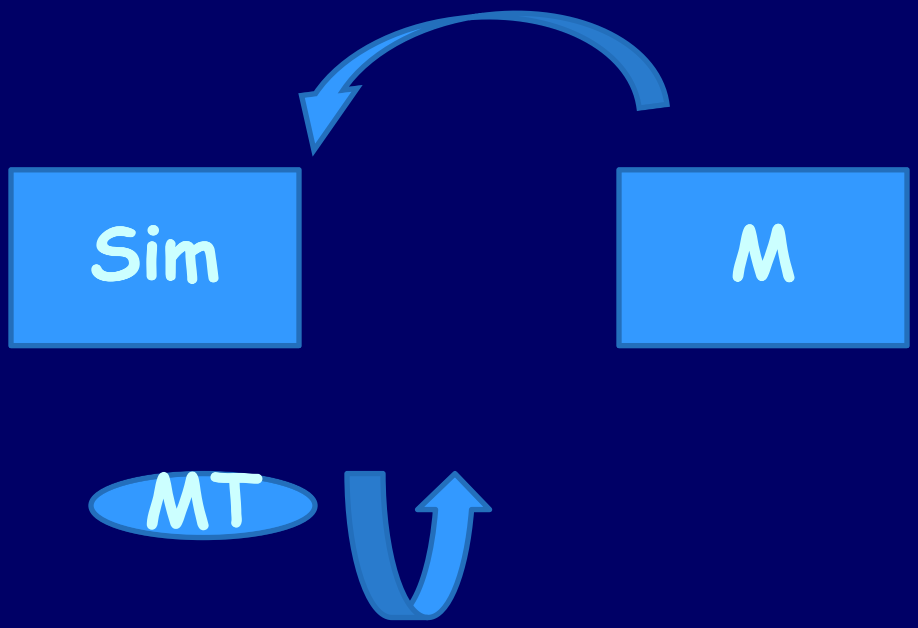
- Additionally, more sophisticated attacks can be imagined where tokens of one type in one protocol used to create tokens of another type in other protocols

Our Contributions

- New constructions: Improvements in several different directions, substantially different techniques
  - Knowing the code: Our security proofs are not based on rewinding the malicious tokens
  - Resettable Tokens: Interaction with the tokens modeled as simple request/reply protocol. Hence tokens not only resettable but completely stateless
  - Our UC commitment protocol is based on one way permutations

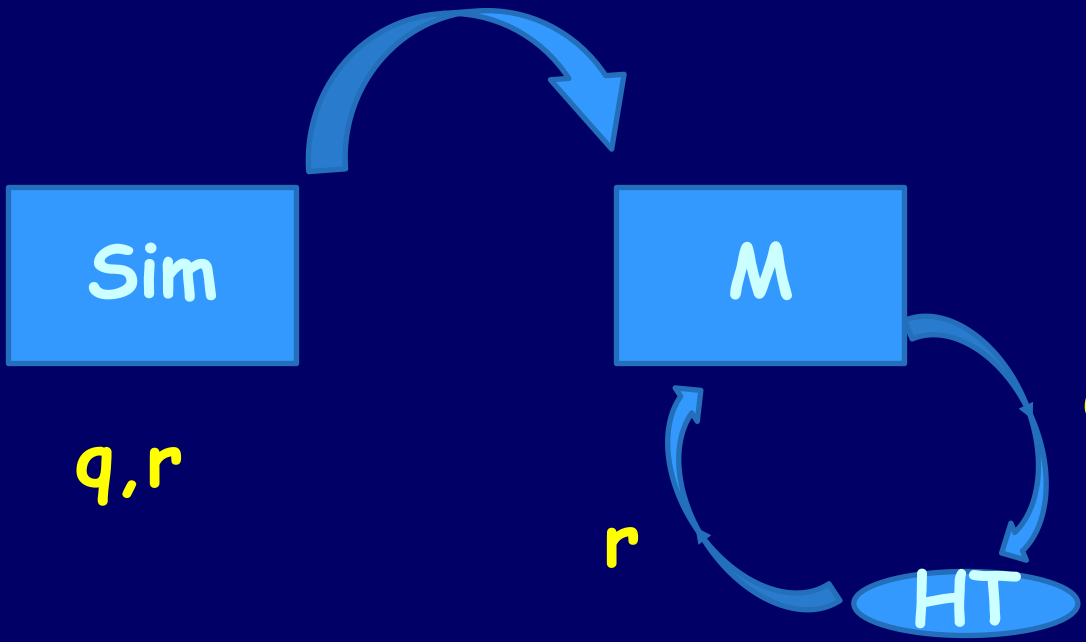
Our Construction: Key Idea

- Source of extra power of simulator in [Katz 07]: Rewinding malicious tokens



Our Construction: Key Idea

- Our idea: Sim given access to queries made by a malicious party to an honest token

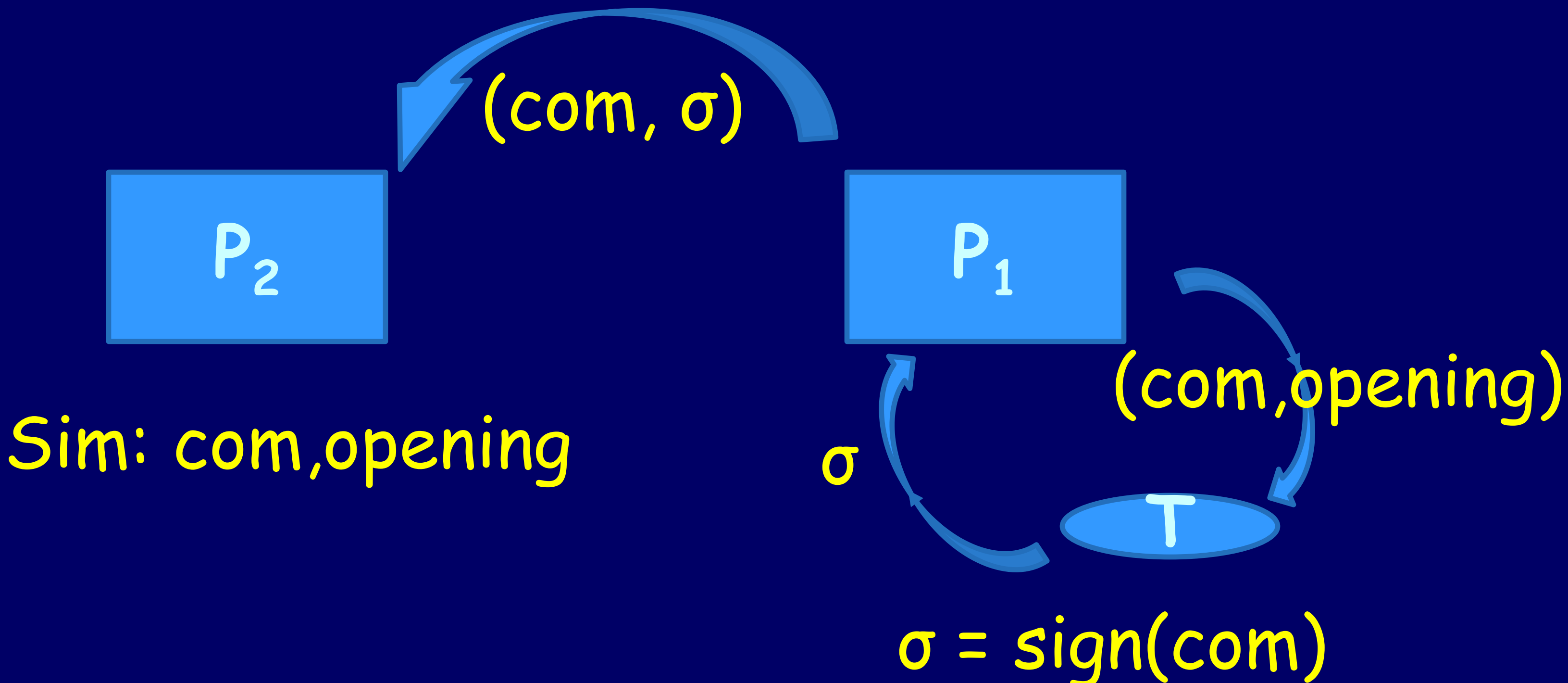


- Similar to how proofs are done in the Random Oracle Model

Our Construction: Exploiting this power

To commit to P_2, P_1 has to:

- Feed the commitment and opening to the tokens sent by P_2
- Obtain a signature on it



Our Construction: Main Issues

- Selective Abort: Token, for example, gives signature if commitment is to *0* but aborts otherwise
  - Solution: First get signatures on both: commitment to 0 as well as to 1. Then use the appropriate one.
- P_1 can't send σ in clear: Information about opening leaked potentially
  - Send com(σ) instead + prove its validity
- Proving the validity is tricky: information about σ should not be leaked. We use *concurrent zero knowledge* for this purpose

Our Construction: Most Difficult Aspects

- Ensure that an adversary can't commit to σ + prove its validity without querying the token to obtain σ
- Extract σ in such a case and show signature forgery
- Take this analysis "outside the UC framework" in the form of a soundness lemma

UC -Com(a): High Level

Generate n coms to 0 + n coms to 1 + signatures

→ commitments: com(Σ = signatures)

Generate and send shares of Σ

$\Sigma_0^i \oplus \Sigma_1^i = \Sigma$

→ com(Σ_0^1),com(Σ_1^1), … com(Σ_0^k),com(Σ_1^k)

← q_1 .. q_k

P$_1$ P$_2$

→ Open Relevant Commits

→ Concurrent ZK: All are valid shares of Σ + Σ represents valid signatures on commitments

Analysis

- Extraction straightline: Sim just looks at the queries made by the committer

- Extraction Abort Lemma: To complete UC-Com protocol, P_1 has to query the token and get a signature
  - Proven "outside the UC framework"
  - We rewind the Env to extract this signature
  - *Challenge + opening shares* mechanism enables the extraction of the forged signature

Other Independent Works [DNW08, MS08]

- Among other things, give constructions based on general assumptions. However, do not solve the main problems addressed in this work
- Both works are in the rewinding based simulator paradigm as [Katz07]
  - Thus, the assumption that sender knows the code of its tokens is required
- Tokens are required to execute a multi-round protocol
  - Resettable/stateless tokens not sufficient

Open Questions

- Obtain properties achieved in [DNW08, MS08] with a non-rewinding simulator
- Obtain simpler and efficient constructions

Thank You