How To Simulate It

A Tutorial on the Simulation Proof Technique

Kurt Pan

Fudan University

June 8, 2021

Outline

Outline


拟真训练
training simulations,

Section 1

Semantic Security as Simulation

Semantic Security as Simulation

Definition (Semantically Secure)

A private-key encryption scheme (G, E, D) is *semantically secure* (in the private-key model) if for every non-uniform probabilistic-polynomial time algorithm $\mathcal{A}$ there exists a non-uniform probabilistic-polynomial time algorithm $\mathcal{A}'$ such that for every probability ensemble $\{X_n\}_{n\in\mathbb{N}}$ with $|X_n| \leq \text{poly}(n)$, every pair of polynomially-bounded functions $f, h : \{0,1\}^* \to \{0,1\}^*$, every positive polynomial $p(\cdot)$ and all sufficiently large n :

$$\Pr_{k\leftarrow G(1^n)}\left[\mathcal{A}\left(1^n, E_k\left(X_n\right), 1^{|X_n|}, h\left(1^n, X_n\right)\right) = f(1^n, X_n)\right]$$
$$< \Pr\left[\mathcal{A}'\left(1^n, 1^{|X_n|}, h\left(1^n, X_n\right)\right) = f(1^n, X_n)\right] + \frac{1}{p(n)}$$

(The probability in the above terms is taken over X_n as well as over the internal coin tosses of the algorithms G, E and $\mathcal{A}$ or $\mathcal{A}'$.)

Semantic Security as Simulation

Simulation(ideal/real world comparison)

Simulation is a way of comparing what happens in the "real world" to what happens in an "ideal world" where the primitive in question is secure by definition.

Simulator $\mathcal{A}'$: Upon input $1^n, 1^{|X_n|}, h = h(1^n, X_n)$, algorithm $\mathcal{A}'$ works as follows

1. $\mathcal{A}'$ runs the key generation algorithm $G(1^n)$ in order to receive k .
2. $\mathcal{A}'$ computes $c = E_k\left(0^{|X_n|}\right)$ as an encryption of "garbage" .
3. $\mathcal{A}'$ runs $\mathcal{A}\left(1^n, c, 1^{|X_n|}, h\right)$ and outputs whatever $\mathcal{A}$ outputs.

Simulation-based proofs

The simulator somehow simulates an execution for the adversary while handing it "garbage" that looks indistinguishable. Then, the proof proceeds by showing that the simulation is "good", or else the given assumption can be broken.

Section 2

Simulation for Semi-Honest Adversaries

Defining Security for Semi-Honest Adversaries

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time *functionality* and let π be a two-party protocol for computing f.
- The *view* of the i th party ($i \in \{1, 2\}$) during an execution of π on (x, y) and security parameter n is denoted by $\text{view}_i^{\pi}(x, y, n)$ and equals $\left(w, r^i; m_1^i, \ldots, m_t^i\right)$, where $w \in \{x, y\}$ (its input depending on the value of i), r^i equals the contents of the i th party's internal random tape, and m_j^i represents the j th message that it received.
- The *output* of the i th party during an execution of π on (x, y) and security parameter n is denoted by $\text{output}_i^{\pi}(x, y, n)$ and can be computed from its own view of the execution. We denote the joint output of both parties by $\text{output}^{\pi}(x, y, n) = (\text{output}_1^{\pi}(x, y, n), \text{output}_2^{\pi}(x, y, n))$.

Defining Security for Semi-Honest Adversaries

Definition (π securely computes f in the presence of static semi-honest adversaries)

if there exist probabilistic polynomial-time algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\{(\mathcal{S}_1(1^n, x, f_1(x,y)), f(x,y))\}_{x,y,n} \overset{c}{\equiv} \{(\text{view}_1^\pi(x,y,n), \text{output}^\pi(x,y,n))\}_{x,y,n}$$
$$\{(\mathcal{S}_2(1^n, y, f_2(x,y)), f(x,y))\}_{x,y,n} \overset{c}{\equiv} \{(\text{view}_2^\pi(x,y,n), \text{output}^\pi(x,y,n))\}_{x,y,n}$$

where $x, y \in \{0,1\}^*$ such that $|x| = |y|$, and $n \in \mathbb{N}$.

Defining Security for Semi-Honest Adversaries

Definition (π securely computes f in the presence of static semi-honest adversaries(f is deterministic))

- **Correctness**, meaning that the output of the parties is correct, i.e. there exists a negligible function μ such that for every $x, y \in \{0,1\}^*$ and every n

$$\Pr\left[\text{output}^{\pi}(x, y, n) \neq f(x, y)\right] \leq \mu(n),$$

- **Privacy**, meaning that the view of each party can be (separately) simulated,i.e. there exist probabilistic-polynomial time S_1 and S_2 such that

$$\left\{\mathcal{S}_1\left(1^n, x, f_1(x, y)\right)\right\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \overset{c}{\equiv} \left\{\text{view}_1^{\pi}(x, y, n)\right\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}}$$
$$\left\{\mathcal{S}_2\left(1^n, y, f_2(x, y)\right)\right\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}} \overset{c}{\equiv} \left\{\text{view}_2^{\pi}(x, y, n)\right\}_{x,y\in\{0,1\}^*;n\in\mathbb{N}}$$

Oblivious Transfer for Semi-Honest Adversaries

Definition (Bit Oblivious Transfer Functionality)

$$f((b_0, b_1), \sigma) = (\lambda, b_\sigma), \text{ where } b_0, b_1, \sigma \in \{0, 1\}$$

- P_1 has a pair of input bits (b_0, b_1) and P_2 has a choice bit σ.
- P_1 receives no output (denoted by the empty string λ), and in particular learns nothing about σ.
- P_2 receives the bit of its choice b_σ and learns nothing about the other bit $b_{1-\sigma}$.

Definition (Enhanced Trapdoor Permutations)

A collection of *trapdoor permutations* is a collection of functions $\{f_\alpha\}_\alpha$ accompanied by four probabilistic-polynomial time algorithms I, S, F, F^{-1} such that:

- $I(1^n)$ selects a random n -bit index α of a permutation f_α along with a corresponding trapdoor τ. Denote by $I_1(1^n)$ the α -part of the output.
- $S(\alpha)$ samples an (almost uniform) element in the domain (equivalently, the range) of f_α. We denote by $S(\alpha; r)$ the output of $S(\alpha)$ with random tape r; for simplicity we assume that $r \in \{0,1\}^n$
- $F(\alpha, x) = f_\alpha(x)$, for α in the range of I_1 and x in the range of $S(\alpha)$.
- $F^{-1}(\tau, y) = f_\alpha^{-1}(y)$ for y in the range of f_α and (α, τ) in the range of I.

for every non-uniform probabilistic polynomial time adversary $\mathcal{A}$ there exists a negligible function μ such that for every n,

$$\Pr\left[\mathcal{A}(1^n, \alpha, r) = f_\alpha^{-1}(S(\alpha; r))\right] \leq \mu(n)$$

where $\alpha \leftarrow I_1(1^n)$ and $r \in_R \{0,1\}^n$ is random.

Definition (Hard-core Predicate)

A hard-core predicate B of a family of enhanced trapdoor permutations $\left(I, S, F, F^{-1}\right)$ if for every non-uniform probabilistic-polynomial time adversary $\mathcal{A}$ there exists a negligible function μ such that for every n,

$$\Pr\left[\mathcal{A}\left(1^n, \alpha, r\right) = B\left(\alpha, f_\alpha^{-1}(S(\alpha; r))\right)\right] \leq \frac{1}{2} + \mu(n)$$

Oblivious Transfer for Semi-Honest Adversaries

Protocol

1. P_1 runs $I(1^n)$ to obtain a permutation-trapdoor pair (α, τ). P_1 sends α to P_2.
2. P_2 runs $S(\alpha)$ twice; denote the first value obtained by x_σ and the second by $y_{1-\sigma}$. Then, P_2 computes $y_\sigma = F(\alpha, x_\sigma) = f_\alpha(x_\sigma)$, and sends y_0, y_1 to P_1.
3. P_1 uses the trapdoor τ and computes $x_0 = F^{-1}(\alpha, y_0) = f_\alpha^{-1}(y_0)$ and $x_1 = F^{-1}(\alpha, y_1) = f_\alpha^{-1}(y_1)$. Then, it computes $\beta_0 = B(\alpha, x_0) \oplus b_0$ and $\beta_1 = B(\alpha, x_1) \oplus b_1$, where B is a hard-core predicate of f. Finally, P_1 sends (β_0, β_1) to P_2.
4. P_2 computes $b_\sigma = B(\alpha, x_\sigma) \oplus \beta_\sigma$ and outputs the result.

Oblivious Transfer for Semi-Honest Adversaries

Theorem

Assume that $\left(I, S, F, F^{-1}\right)$ *constitutes a family of enhanced trapdoor permutations with a hard-core predicate* B*. Then, Protocol above securely computes the functionality* $f\left(\left(b_0, b_1\right), \sigma\right) = \left(\lambda, b_\sigma\right)$ *in the presence of static semi-honest adversaries.*

Proof.

P_1 is corrupted

$\mathcal{S}_1$ is given (b_0, b_1) and 1^n :

1. $\mathcal{S}_1$ chooses a uniformly distributed random tape r for P_1
2. $\mathcal{S}_1$ computes $(\alpha, \tau) \leftarrow I(1^n; r)$, using the r from above.
3. $\mathcal{S}_1$ runs $S(\alpha)$ twice with independent randomness to sample values y_0, y_1.
4. $\mathcal{S}_1$ outputs $\left(\left(b_0, b_1\right), r; \left(y_0, y_1\right)\right)$; the pair $\left(y_0, y_1\right)$ simulates the incoming message from P_2 to P_1 in the protocol.

Claim

$$\{(F(\alpha, x_0), y_1)\} \overset{s}{\equiv} \{(y_0, y_1)\} \overset{s}{\equiv} \{(y_0, F(\alpha, x_1))\}$$

$$\{\mathcal{S}_1(1^n, (b_0, b_1))\} \overset{s}{\equiv} \{\text{view}_1^{\pi}((b_0, b_1), \sigma)\}$$

P_2 is corrupted

Proof.

Simulator $\mathcal{S}_2$ receives for input 1^n plus (σ, b_σ):

1. $\mathcal{S}_2$ chooses a uniform random tape for P_2. Since P_2 's randomness is for running $S(\alpha)$ twice, we denote the random tape by r_0, r_1.
2. $\mathcal{S}_2$ runs $I(1^n)$ and obtains (α, τ).
3. $\mathcal{S}_2$ computes $x_\sigma = S(\alpha; r_\sigma)$ and $y_{1-\sigma} = S(\alpha; r_{1-\sigma})$, and sets $x_{1-\sigma} = F^{-1}(\tau, y_{1-\sigma})$.
4. $\mathcal{S}_2$ sets $\beta_\sigma = B(\alpha, x_\sigma) \oplus b_\sigma$, where b_σ is P_2 's output received by $\mathcal{S}_2$.
5. $\mathcal{S}_2$ sets $\beta_{1-\sigma} = B(\alpha, x_{1-\sigma})$.
6. $\mathcal{S}_2$ outputs $(\sigma, r_0, r_1; \alpha, (\beta_0, \beta_1))$.

□

Claim

When $b_{1-\sigma} = 0$, for every $\sigma, b_\sigma \in \{0,1\}$ and every n:

$$\{\mathcal{S}_2(1^n, \sigma, b_\sigma)\} \equiv \{\text{view}_2^\pi((b_0, b_1), \sigma)\}$$

Claim

When $b_{1-\sigma} = 1$

$$\{(\sigma, r_0, r_1; \alpha, (B(\alpha, x_\sigma) \oplus b_\sigma, B(\alpha, x_{1-\sigma})))\} \stackrel{c}{=}$$

$$\{(\sigma, r_0, r_1; \alpha, (B(\alpha, x_\sigma) \oplus b_\sigma, B(\alpha, x_{1-\sigma}) \oplus 1))\}$$

Section 3

Simulating the View of Malicious Adversaries – Zero Knowledge

Defining Zero Knowledge

Definition (IP)

An *interactive proof system* for a language L involves a prover P and a verifier V, where upon common input x, the prover P attempts to convince V that $x \in L$. The prover is often given some private auxiliary-input that "helps" it to prove the statement in question to V.

1. **Completeness**: when honest P and V interact on common input $x \in L$, then V is convinced of the correctness of the statement that $x \in L$ (except with at most negligible probability).
2. **Soundness**: when V interacts with any (cheating) prover P^* on common input $x \notin L$, then V will be convinced with at most negligible probability. (Thus V cannot be tricked into accepting a false statement.)

Defining Zero Knowledge

Definition (bb-cZK)

Let (P, V) be an interactive proof system for an $\mathcal{NP}$ -language L, and let R_L be the associated $\mathcal{NP}$ -relation. We say that (P, V) is *black-box computational zero knowledge* if there exists a probabilistic-polynomial time oracle machine $\mathcal{S}$ such that for every non-uniform probabilistic polynomial time algorithm V^* it holds that:

$$\{\text{output}_{V^*}\left(P(x,w), V^*(x,z)\right)\}_{(x,w)\in R_L, z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathcal{S}^{V^*(x,z,r,\cdot)}(x)\right\}_{x\in L, z\in\{0,1\}^*}$$

where r is uniformly distributed, and where $V^*(x, z, r, \cdot)$ denotes the next-message function of the interactive machine V^* when the common input x, auxiliary input z and random-tape r are fixed (i.e., the next message function of V^* receives a message history $\vec{m}$ and outputs $V^*(x, z, r, \vec{m})$).

Commitment Schemes

Definition

We denote by Com a *non-interactive perfectly binding commitment scheme*. Let $c = \mathrm{Com}_n(x; r)$ denote a commitment to x using random string r and with security parameter n. We will typically omit the explicit reference to n and will write $c = \mathrm{Com}(x; r)$.
Let $\mathrm{Com}(x)$ denote a commitment to x using uniform randomness.
Let $\mathrm{decom}(c)$ denote the decommitment value of c; to be specific, if $c = \mathrm{Com}(x; r)$ then $\mathrm{decom}(c) = (x, r)$.

- **Computational Hiding** commitments to different strings are computationally indistinguishable. For bit commitments, $\mathcal{C}_0 \stackrel{c}{\equiv} \mathcal{C}_1$ where $\mathcal{C}_b = \{\mathrm{Com}(b; U_n)\}_{n \in \mathbb{N}}$ is the ensemble of commitments to bit b.
- **Perfect Binding** the sets of all commitments to different values are disjoint; that is, for all $x_1 \neq x_2$ it holds that $C_{x_1} \cap C_{x_2} = \emptyset$ where $C_{x_1} = \{c \mid \exists r : c = \mathrm{Com}(x_1; r)\}$ and $C_{x_2} = \{c \mid \exists r : c = \mathrm{Com}(x_2; r)\}$.

Non-Constant Round Zero Knowledge

Zero-Knowledge Proof for 3-Coloring

- *Common input*: a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$
- *Auxiliary input for the prover*: a coloring of the graph $\psi : V \to \{1, 2, 3\}$ such that for every $(v_i, v_j) \in E$ it holds that $\psi(v_i) \neq \psi(v_j)$
- *The proof system*: Repeat the following $n \cdot |E|$ times (using independent randomness each time):
  1. The prover selects a random permutation π over $\{1, 2, 3\}$, defines $\phi(v) = \pi(\psi(v))$ for all $v \in V$, and computes $c_i = \mathrm{Com}(\phi(v_i))$ for all i. The prover sends the verifier the commitments $(c_1, \ldots, c_n)$
  2. The verifier chooses a random edge $e \in_R E$ and sends e to the prover.
  3. Let $e = (v_i, v_j)$ be the edge received by the prover. The prover sends $\mathrm{decom}(c_i), \mathrm{decom}(c_j)$ to the verifier.
  4. Let $\phi(v_i)$ and $\phi(v_j)$ denote the respective decommitment values from c_i and c_j. The verifier checks that the decommitments are valid, that $\phi(v_i), \phi(v_j) \in \{1, 2, 3\}$, and that $\phi(v_i) \neq \phi(v_j)$. If not, it halts and outputs 0 .

  If the checks pass in all iterations, then the verifier outputs 1.

- **Soundness** By repeating the proof $n \cdot |E|$ times (where n is the number of nodes in the graph), the prover can get away with cheating with probability at most $\left(1 - \frac{1}{|E|}\right)^{n \cdot |E|} < e^{-n}$ which is negligible.
- **Zero Knowledge** In each execution a new random coloring of the edges is committed to by the prover, and the verifier only sees the colors of a single edge. Thus, the verifier simply sees two (different) random colors for the endpoints of the edges each time. This clearly reveals nothing about the coloring of the graph. We must prove this intuition by constructing a simulator.

Rewinding

- game save points
- virtual machine snapshot

Theorem

Let Com *be a perfectly-binding commitment scheme with security for non-uniform adversaries. Then, the 3-coloring protocol is bb-cZK.*

Proof.

Simulator $\mathcal{S}$ is given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ and oracle access to some probabilistic-polynomial time $V^*(x, z, r, \cdot)$, and works as follows:

1. $\mathcal{S}$ initializes the message history transcript $\vec{m}$ to be the empty string λ.
2. Repeat $n \cdot |E|$ times:
   a. $\mathcal{S}$ sets $j = 1$.
   b. $\mathcal{S}$ chooses a random edge $(v_k, v_\ell) \in_R E$ and chooses two random different colors for v_k and v_ℓ. Formally, $\mathcal{S}$ chooses $\phi(v_k) \in_R \{1, 2, 3\}$ and $\phi(v_\ell) \in_R \{1, 2, 3\} \setminus \{\phi(v_k)\}$. For all other $v_i \in V \setminus \{v_k, v_\ell\}$, $\mathcal{S}$ sets $\phi(v_i) = 0$.
   c. For every $i = 1, \ldots, n$, $\mathcal{S}$ computes $c_i = \mathrm{Com}(\phi(v_i))$.

Proof.

2. d. $\mathcal{S}$ "sends" the vector $(c_1, \ldots, c_n)$ to V^*. Formally, $\mathcal{S}$ queries $\vec{m}$ concatenated with this vector to its oracle (indeed $\mathcal{S}$ does not interact with V^* and so cannot actually "send" it any message). Let $e \in E$ be the reply back from the oracle.
   - d. If $e = (v_k, v_\ell)$, then $\mathcal{S}$ completes this iteration by concatenating the commitments $(c_1, \ldots, c_n)$ and $(\text{decom}(c_k), \text{decom}(c_\ell))$ to $\vec{m}$. Formally, $\mathcal{S}$ updates the history string $\vec{m} \leftarrow (\vec{m}, (c_1, \ldots, c_n), (\text{decom}(c_k), \text{decom}(c_\ell)))$
   - d. If $e \neq (v_k, v_\ell)$ then $\mathcal{S}$ sets $j \leftarrow j + 1$. If $j = n \cdot |E|$, then $\mathcal{S}$ outputs a fail symbol $\perp$. Else (when $j \neq n \cdot |E|$), $\mathcal{S}$ returns to Step 2b (i.e., $\mathcal{S}$ tries again for this i). This return to Step 2b is the *rewinding* of V^* by the simulator.
3. $\mathcal{S}$ outputs whatever V^* outputs on the final transcript $\vec{m}$.

□

- $\mathcal{S}$ **runs in polynomial-time:** each repetition runs for at most $n \cdot |E|$ iterations, and there are $n \cdot |E|$ repetitions.
- To prove that $\mathcal{S}$ **generates a transcript that is indistinguishable from a real transcript**, i.e.

$$\{ \text{output}_{V^*} (P(G, \psi), V^*(G, z))\} \stackrel{c}{\equiv} \left\{ \mathcal{S}^{V^*(G,z,r,\cdot)}(G) \right\}$$

  we need to prove a reduction to the security of the commitment scheme.
- Alternative simulator $\mathcal{S}'$ who is given a valid coloring ψ as auxiliary input.
  $\mathcal{S}'$ works in exactly the same way as $\mathcal{S}$ (choosing e at random, rewinding, and so on) except that in every iteration it chooses a random permutation π over $\{1, 2, 3\}$, sets $\phi(v) = \pi(\psi(v))$, and computes $c_i = \text{Com}(\phi(v_i))$ for all i, exactly like the real prover.

Claim 1

$$\{\text{output}_{V^*}\left(P(G,\psi), V^*(G,z)\right)\} \equiv \left\{\mathcal{S}'^{V^*(G,z,r,)}(G,\psi) \mid \mathcal{S}'^{V^*(G,z,r,\cdot)}(G,\psi) \neq \perp\right\}$$

Claim 2

$$\left\{\mathcal{S}'^{V^*(G,z,r,\cdot)}(G,\psi) \mid \mathcal{S}'^{V^*(G,z,r,\cdot)}(G,\psi) \neq \perp\right\} \stackrel{c}{\equiv} \left\{\mathcal{S}'^{V^*(G,z,r,\cdot)}(G,\psi)\right\}$$

Claim 3

$$\left\{\mathcal{S}'^{V^*(G,z,r,\cdot)}(G,\psi)\right\} \stackrel{c}{\equiv} \left\{\mathcal{S}^{V^*(G,z,r,\cdot)}(G)\right\}$$

Proof.

- Assume by contradiction, that there exists a probabilistic-polynomial time verifier V^*, a probabilisticpolynomial time distinguisher D, and a polynomial $p(\cdot)$ such that for an infinite sequence (G, ψ, z) where $(G, \psi) \in R$ and $z \in \{0,1\}^*$,

$$|\Pr\left[D\left(G, \psi, z, \mathcal{S}'^{V^*(G,z,r,\cdot)}(G, \psi)\right) = 1\right]$$
$$-\Pr\left[D\left(G, \psi, z, \mathcal{S}^{V^*(G,z,r,\cdot)}(G)\right) = 1\right]| \geq \frac{1}{p(n)}$$

- We construct a adversary $\mathcal{A}$ for the commitment experiment LR-commit. $\mathcal{A}$ receives (G, ψ, z) on its advice tape.

1. $\mathcal{A}$ initializes V^* with input graph G, auxiliary input z and a uniform random tape r
2. $\mathcal{A}$ runs the instructions of $\mathcal{S}'$ with input (G, ψ) and oracle $V^*(x, z, r; \cdot)$, with some changes. For every iteration of the simulation:
   a. For the randomly chosen edge $e = (v_k, v_\ell)$, adversary $\mathcal{A}$ generates commitments $c_k = \mathrm{Com}(\phi(v_k))$ and $c_\ell = \mathrm{Com}(\phi(v_\ell))$ by itself.
   b. For all other i (i.e., for all $i \in \{1, \ldots, n\} \backslash \{k, \ell\}$), adversary $\mathcal{A}$ queries its LR-oracle with the pair $(0, \phi(i))$. Denote by c_i the commitment received back.

   $\mathcal{A}$ simulates $\mathcal{S}'$ querying V^* with the commitments $(c_1, \ldots, c_n)$ as a result of the above. Observe that $\mathcal{A}$ can decommit to v_k, v_ℓ as needed by $\mathcal{S}'$, since it computed the commitments itself.
3. When $\mathcal{S}'$ concludes, then $\mathcal{A}$ invokes D on the output generated by $\mathcal{S}'$, and outputs whatever D outputs.

Proof.

$$\Pr\left[\text{LR-commit}_{\text{Com},\mathcal{A}}(1^n) = 1 \mid b = 1\right] = \Pr\left[D\left(G, z, \mathcal{S}'^{V^*(G,z,r,\cdot)}(G, \psi)\right) = 1\right]$$

$$\Pr\left[\text{LR-commit}_{\text{Com},\mathcal{A}}(1^n) = 1 \mid b = 0\right] = \Pr\left[D\left(G, z, \mathcal{S}^{V^*(G,z,r,\cdot)}(G)\right) = 0\right]$$

$$\begin{aligned}
&\Pr\left[\text{LR-commit}_{\text{Com},\mathcal{A}}(1^n) = 1\right] \\
&= \tfrac{1}{2}\Pr\left[\text{LR-commit}_{\text{Com},\mathcal{A}}(1^n) = 1 \mid b = 1\right] \\
&+ \tfrac{1}{2}\Pr\left[\text{LR-commit}_{\text{Com},\mathcal{A}}(1^n) = 1 \mid b = 0\right] \\
&= \tfrac{1}{2}\Pr\left[D\left(G, z, \mathcal{S}'^{V^*(G,z,r,\cdot)}(G, \psi)\right) = 1\right] + \tfrac{1}{2}\Pr\left[D\left(G, z, \mathcal{S}^{V^*(G,z,r,)}(G)\right) = 0\right] \\
&= \tfrac{1}{2}\Pr\left[D\left(G, z, \mathcal{S}'^{V^*(G,z,r,\cdot)}(G, \psi)\right) = 1\right] \\
&+ \tfrac{1}{2}\left(1 - \Pr\left[D\left(G, z, \mathcal{S}^{V^*(G,z,r,\cdot)}(G)\right) = 1\right]\right) \\
&= \tfrac{1}{2} + \\
&\tfrac{1}{2}\left(\Pr\left[D\left(G, z, \mathcal{S}'^{V^*(G,z,r,\cdot)}(G, \psi)\right) = 1\right] - \Pr\left[D\left(G, z, \mathcal{S}^{V^*(G,z,r,)}(G)\right) = 1\right]\right) \\
&\geq \tfrac{1}{2} + \tfrac{1}{2p(n)}.
\end{aligned}$$

Discussion on the proof technique

- Two differences between $\mathcal{S}$ and a real prover: (a) the flow of $\mathcal{S}$ that involves choosing e and rewinding, and (b) the commitments that are incorrectly generated.
- This technique is often used in simulation-based proofs, and in some cases there are **series of simulators** that bridge the differences between the real execution and the simulation. This is similar to **sequences of hybrids in game-based proofs**, with the only difference being that the sequence here is from the simulation to the real execution (or vice versa).

Constant-Round Zero-Knowledge

为什么简单的重置策略会失败. 我们希望图同构证明系统的可靠性错误非常小，而不是上面的1/2。上述协议可以通过顺序执行多次（验证者接受当且仅当所有的完整执行都接受）来降低这一可靠性错误（同时保持零知识性），但这会大幅增加交互的次数（轮复杂性，我们通常称发送一次消息为一轮）。另一种方式就是并行执行多次协议来降低可靠性错误，这样可以保持交互轮数（3轮）不变。但它导致的一个问题就是上述简单的重置策略无法使得我们能够在多项式时间内完成模拟：假设进行n次并行，这时V的挑战问题变成了n个 （可有一个n长的比特串表示），模拟器S同时猜中这n个问题的概率此时就变成 $1/2^n$ ，这会导致S的猜中为止的期望循环次数为 2^n ，进而需要指数时间来完成模拟。3-轮零知识的存在性目前仍是一个巨大的公开问题。最近Canetti和Chen等人在Fiat-Shamir启发式（将公开掷币的交互协议转化为非交互协议的一种方法）方面的突破性工作[CCH+, STOC 19]中，给出了这一问题的强有力负面证据。

The Goldreich-Kahan Proof System

1. The prover sends the first message of a (two-round) perfectly-hiding commitment scheme, denoted $\mathrm{Com}_{\mathrm{h}}$.
2. The verifier chooses $N \stackrel{\text{def}}{=} n \cdot |E|$ random edges $e_1, \ldots, e_N \in_R E$. Let $q = (e_1, \ldots, e_N)$ be the query string; the verifier commits to q using the perfectly-hiding commitment $\mathrm{Com}_{\mathrm{h}}$
3. The prover prepares the first message in N parallel executions of the basic three-round proof system (i.e., commitments to N independent random colorings of the graph), and sends commitments to all using the perfectly-binding commitment scheme Com.
4. The verifier decommits to the string q.
5. If the verifier's decommitment is invalid, then the prover aborts. Otherwise, the prover sends the appropriate decommitments in every execution. Specifically, if e_i is the edge in the i th execution, then the prover decommits to the nodes of that edge in the i th set of commitments to colorings.
6. The verifier outputs 1 if and only if all checks pass.

Theorem

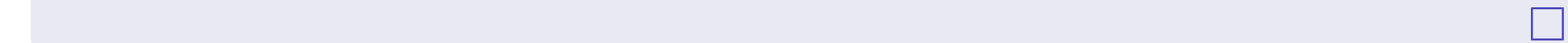
Let $\mathrm{Com}_{\mathrm{h}}$ *and* Com *be perfectly-hiding and perfectly-binding commitment schemes, respectively. Then, GK Protocol above is bb-cZK with an expected polynomial-time simulator.*

Proof.

1. $\mathcal{S}$ hands V^* the first message of $\mathrm{Com}_{\mathrm{h}}$.
2. $\mathcal{S}$ receives from V^* its perfectly-hiding commitment c to some query string $q = (e_1, \ldots, e_N)$, where $N = n \cdot |E|$.
3. $\mathcal{S}$ generates N vectors of n commitments to 0 , hands them to V^*, and receives back its reply.
4. If V^* aborts by not replying with a valid decommitment to c (and the decommitment is to a vector of N edges), then $\mathcal{S}$ aborts and outputs whatever V^* outputs. Otherwise, let $q = (e_1, \ldots, e_N)$ be the decommitted value. $\mathcal{S}$ proceeds to the next step.
5. Rewinding phase $-\mathcal{S}$ repeatedly rewinds V^* back to the point where it receives the prover commitments, until it decommits to q from above:

Proof.

5.
   1. $\mathcal{S}$ generates N vectors of commitments $\vec{c}_1, \ldots, \vec{c}_N$, as follows. Let $e_i = (v_j, v_k)$ in q. Then, the j th and k th commitments in $\vec{c}_i$ are to random distinct colors in $\{1, 2, 3\}$ and all other commitments are to 0. Simulator $\mathcal{S}$ hands all vectors to V^*, and receives back its reply.
   2. If V^* does not generate a valid decommitment, then $\mathcal{S}$ returns to the previous step (using fresh randomness).
   3. If V^* generates a valid decommitment to some $q' \neq q$, then $\mathcal{S}$ outputs $\mathrm{ambiguous}$ and halts.
   4. Otherwise, V^* exits the loop and proceeds to the next step.
6. $\mathcal{S}$ completes the proof by handing V^* decommitments to the appropriate nodes in all of $\vec{c}_1, \ldots, \vec{c}_N$, and outputs whatever V^* outputs.



Simulator may not run in expected polynomial-time

- The probability that the verifier decommits correctly when receiving the first prover *commitments to pure garbage* is not necessarily the same as the probability that it decommits correctly when receiving the *simulator-generated commitments*.
- If the verifier was all powerful, it could break open the commitments and purposefully make the simulation fail by decommitting when it receives pure garbage (or fully valid commitments) and not decommitting when it receives commitments that can be opened only to its query string.
- we can only argue that this doesn't happen by a reduction to the commitments, and this also means that there may be a negligible difference. Thus, we actually have that the expected running time of the simulator is

$$\text{poly}(n) \cdot \left(1 - \epsilon(n) + \epsilon(n) \cdot \frac{1}{\epsilon(n) - \mu(n)}\right)$$

let ϵ denote the probability that V^* does not abort


我依然在做梦
I'm still dreaming.


FUDAN UNIVERSITY
1905

$\mu(n) = 2^{-n/2} - 2^{-n}$ and $\epsilon(n) = \mu(n) + 2^{-n} = 2^{-n/2}$. Then,

$$\frac{\epsilon(n)}{\epsilon(n) - \mu(n)} = \frac{2^{-n/2}}{2^{-n/2} - \left(2^{-n/2} - 2^{-n}\right)} = \frac{2^{-n/2}}{2^{-n}} = 2^{n/2}$$

- simulation does not run in expected polynomial-time.
- This is solved by ensuring that the simulator $\mathcal{S}$ never runs "too long".
- $\mathcal{S}$ runs the rewinding phase in Step 5 of the simulation up to n times. Each time, $\mathcal{S}$ limits the number of rewinding attempts in the rewinding phase to $n/\tilde{\epsilon}$ iterations.

$\mathcal{S}$ first estimates the value of $\epsilon(n)$ which is the probability that V^* does not abort given garbage commitments. This is done by repeating Step 3 of the simulation (sending fresh random commitments to all zeroes) until $m = O(n)$ successful decommits occurs (for a polynomial $m(n)$; to be exact $m = 12n$ suffices), where a successful decommit is where V^* decommits to q, the string it first decommitted to.
We remark that as in the original strategy, if V^* correctly decommits to a different $q' \neq q$ then $\mathcal{S}$ outputs ambiguous. Then, an estimate $\tilde{\epsilon}$ of ϵ is taken to be m/T, where T is the overall number of attempts until m successful decommits occurred. This suffices to ensure that the probability that $\tilde{\epsilon}$ is not within a constant factor of $\epsilon(n)$ is at most 2^{-n}.

Claim

Simulator $\mathcal{S}$ runs in expected-time that is polynomial in n

$$\text{poly}(n) \cdot \epsilon(n) \cdot \left(\frac{12n}{\epsilon(n)} + n \cdot \frac{n}{\tilde{\epsilon}} \right) = \text{poly}(n) \cdot \frac{\epsilon(n)}{\tilde{\epsilon}} = \text{poly}(n)$$

Claim

The probability that $\mathcal{S}$ outputs fail is negligible in n

Claim

The probability that $\mathcal{S}$ outputs $\mathrm{ambiguous}$ is negligible in n

Claim

The output distribution generated by $\mathcal{S}$ is computationally indistinguishable from the output of V^* in a real proof with an honest prover.

Honest-Verifier Zero Knowledge

- A proof system is honest-verifier zero knowledge if the zero-knowledge property holds for semi-honest verifiers.
- Σ-protocol / Schnorr Signature
- Consider the basic 3-coloring protocol described run $n \cdot |E|$ times in parallel. Specifically, the prover generates $n \cdot |E|$ sets of commitments to random colorings and sends them to the verifier. The verifier chooses $q = (e_1, \ldots, e_N)$ at random and sends q to the prover. Finally, the prover decommits as in the protocol.

Theorem

If Com *is a perfectly-binding commitment scheme, then the parallel 3-coloring protocol is honest-verifier zero knowledge.*

Proof.

$\mathcal{S}$:

1. Let $N = n \cdot |E|$. Then, for $i = 1, \ldots, N$, $\mathcal{S}$ chooses a random edge $e_i \in E$, and sets $q = (e_1, \ldots, e_N)$. Let r_q be the random coin tosses that define q.
2. For every $i = 1, \ldots, N$:
   a. Let $e_i = (v_j, v_k)$.
   b. $\mathcal{S}$ chooses random $\phi(v_j) \in_R \{1, 2, 3\}$ and $\phi(v_k) \in_R \{1, 2, 3\} \setminus \{\phi(v_j)\}$. For all other $v_\ell \in V \setminus \{v_j, v_k\}$, $\mathcal{S}$ sets $\phi(v_\ell) = 0$
   c. $\mathcal{S}$ sets the commitment vector $\vec{c}_i = \left(c_i^1, \ldots, c_i^n\right) = (\mathrm{Com}(\phi(v_1)), \ldots, \mathrm{Com}(\phi(v_n)))$.
   d. $\mathcal{S}$ sets the decommitment vector $\vec{d}_i = \left(\mathrm{decom}\left(c_i^j\right), \mathrm{decom}\left(c_i^k\right)\right)$

□

Section 4

Defining Security for Malicious Adversaries

Execution in the ideal model

Definition (Ideal Execution)

Denote the participating parties by P_1 and P_2 and let $i \in \{1, 2\}$ denote the index of the corrupted party, controlled by an adversary $\mathcal{A}$.
An ideal execution for a function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ proceeds as follows:

- **Inputs**: Let x denote the input of party P_1, and let y denote the input of party P_2. The adversary $\mathcal{A}$ also has an auxiliary input denoted by z. All parties are initialized with the same value 1^n on their security parameter tape (including the trusted party).
- **Send inputs to trusted party**: The honest party P_j sends its prescribed input to the trusted party.
  The corrupted party P_i controlled by $\mathcal{A}$ may either abort (by replacing the input with a special abort_i message), send its prescribed input, or send some other input of the same length to the trusted party.
  Denote the pair of inputs sent to the trusted party by (x', y') .

Definition

- **Early abort option**: If the trusted party receives an input of the form abort_i for some $i \in \{1, 2\}$, it sends abort_i to the honest party P_j and the ideal execution terminates. Otherwise, the execution proceeds to the next step.
- **Trusted party sends output to adversary**: At this point the trusted party computes $f_1(x', y')$ and $f_2(x', y')$ and sends $f_i(x', y')$ to party P_i .
- **Adversary instructs trusted party to continue or halt**: $\mathcal{A}$ sends either $\mathrm{continue}$ or abort_i to the trusted party. If it sends $\mathrm{continue}$, the trusted party sends $f_j(x', y')$ to the honest party P_j. If $\mathcal{A}$ sends abort_i, the trusted party sends abort_i to party P_j.
- **Outputs**: The honest party always outputs the output value it obtained from the trusted party. The corrupted party outputs nothing. The adversary $\mathcal{A}$ outputs any arbitrary function of the prescribed input of the corrupted party, the auxiliary input z, and the value $f_i(x', y')$ obtained from the trusted party.

Execution in the ideal model

Definition

The ideal execution of f on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted by $\text{IDEAL}_{f,\mathcal{A}(z),i}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

- In the case of no honest majority (and in particular in the two-party case), it is in general impossible to achieve **guaranteed output delivery** and **fairness**.
- This "weakness" is therefore incorporated into the ideal model by allowing the adversary in an ideal execution to abort the execution or obtain output without the honest party obtaining its output.

Execution in the real model

Definition

let π be a two-party protocol for computing f, meaning that when P_1 and P_2 are both honest, then the parties output $f_1(x, y)$ and $f_2(x, y)$, respectively, after an execution of π with respective inputs x and y.
The real execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted by $\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the real execution of π

- there exists no trusted third party
- the adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy.
- the honest party follows the instructions of π.
- a simple network setting where the protocol proceeds in rounds, where in each round one party sends a message to the other party

Defining Security for Malicious Adversaries

Definition

Let f be a two-party functionality and let π be a two-party protocol that computes f. Protocol π is said to **securely compute f with abort in the presence of static malicious adversaries** if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $i \in \{1, 2\}$

$$\left\{\text{IDEAL}_{f,\mathcal{S}(z),i}(x, y, n)\right\}_{x,y,z,n} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),i}(x, y, n)\right\}_{x,y,z,n}$$

where $x, y \in \{0, 1\}^*$ under the constraint that $|x| = |y|$, $z \in \{0, 1\}^*$ and $n \in \mathbb{N}$.

- a secure protocol (in the real model) **emulates** the ideal model (in which a trusted party exists).
- adversaries in the ideal model (simulator) are able to **simulate** executions of the real-model protocol.

Discussion

- Definition implies **privacy** (meaning that nothing but the output is learned), **corrrectness** (meaning that the output is correctly computed) and more.
- Since the IDEAL and REAL distributions must be indistinguishable, this in particular implies that the output of the adversary in the IDEAL and REAL executions is indistinguishable. Thus, whatever the adversary learns in a real execution can be learned in the ideal model. In the ideal model, the adversary cannot learn anything about the honest party's input beyond what is revealed in the output.
- Regarding correctness, if the adversary can cause the honest party's output to diverge from a correct value in a real execution, then this will result in a non-negligible difference between the distribution over the honest party's output in the real and ideal executions.

Modular Sequential Composition

The hybrid model

- Parties both interact with each other (as in the real model) and use trusted help (as in the ideal model).
- - **standard messages** : regular messages of π that are sent amongst the parties
  - **ideal messages**: the messages that are sent between parties and the trusted party.
- the parties run a protocol π that contains "ideal calls" to a trusted party computing some functionalities $f_1, \ldots, f_{p(n)}$. These ideal calls are just instructions to send an input to the trusted party. Upon receiving the output back from the trusted party, the protocol π continues.
- The protocol π is such that f_i is called before f_{i+1} for every i

Sequential composition – malicious adversaries

Let $f_1, \ldots, f_{p(n)}$ be probabilistic polynomialtime functionalities and let π be a two-party hybrid-model protocol that uses ideal calls to a trusted party computing $f_1, \ldots, f_{p(n)}$.
The $f_1, \ldots, f_{p(n)}$ -**hybrid execution** of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter n, denoted $\mathrm{HYBRID}_{\pi,\mathcal{A}(z),i}^{f_1,\ldots,f_{p(n)}}(x, y, n)$ is defined as the output of the honest party and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing $f_1, \ldots, f_{p(n)}$

Sequential composition – malicious adversaries

Definition (Real protocol $\pi^{\rho_1,\ldots,\rho_{p(n)}}$)

All standard messages of π are unchanged.
When a party is instructed to send an ideal message α to the trusted party to compute f_j, it begins a real execution of ρ_j with input α instead. When this execution of ρ_j concludes with output y, the party continues with π as if y were the output received from the trusted party for f_j (i.e., as if it were running in the hybrid model).

Theorem

Let $p(n)$ be a polynomial, let $f_1, \ldots, f_{p(n)}$ be two-party probabilistic polynomial-time functionalities and let $\rho_1, \ldots, \rho_{p(n)}$ be protocols such that each ρ_i securely computes f_i in the presence of malicious adversaries. Let g be a two-party functionality and let π be a protocol that securely computes g in the $f_1, \ldots, f_{p(n)}$ -hybrid model in the presence of malicious adversaries. Then, $\pi^{\rho_1,\ldots,\rho_p(n)}$ securely computes g in the presence of malicious adversaries.

Section 5

Determining Output – Coin Tossing

Subsection 1

Coin Tossing a Single Bit

Coin Tossing a Single Bit

- The protocol by Blum for **tossing a single coin** securely : The protocol securely computes the functionality $f_{\mathrm{ct}}(\lambda, \lambda) = (U_1, U_1)$ where U_1 is a random variable that is uniformly distributed over $\{0, 1\}$.
- It is possible for one party to see the output and then abort before the other receives it (e.g., in the case that it is not a favorable outcome for that party). Because it is impossible for two parties to toss a coin fairly so that neither party can cause a premature abort or bias the outcome
- We defined a real model where protocols proceed in rounds and in each round one message is sent from one party to the other.

Blum's Coin Tossing of a Single Bit

1. P_1 chooses a random $b_1 \in \{0,1\}$ and a random $r \in \{0,1\}^n$ and sends $c = \mathrm{Com}(b_1; r)$ to P_2.
2. Upon receiving c, party P_2 chooses a random $b_2 \in \{0,1\}$ and sends b_2 to P_1.
3. Upon receiving b_2, party P_1 sends (b_1, r) to P_2 and outputs $b = b_1 \oplus b_2$. (If P_2 does not reply, or replies with an invalid value, then P_2 sets $b_2 = 0$.)
4. Upon receiving (b_1, r) from P_1, party P_2 checks that $c = \mathrm{Com}(b_1; r)$. If yes, it outputs $b = b_1 \oplus b_2$; else it outputs $\perp$.

Theorem

Assume that Com is a perfectly-binding commitment scheme. Then, Protocol above securely computes the bit coin-tossing functionality defined by $f_{\mathrm{ct}}(\lambda, \lambda) = (U_1, U_1)$.

[9]I would like to add a personal anecdote here. The first proof of security that I read that followed the ideal/real simulation paradigm with security for malicious adversaries was this proof by Oded Goldreich (it appeared in a very early draft on Secure Multiparty Computation that can be found at `www.wisdom.weizmann.ac.il/~oded/pp.html`). I remember reading it multiple times until I understood why all the complications were necessary. Thus, for me, this proof brings back fond memories of my first steps in secure computation.

I would like to add a personal anecdote here. The first proof of security that I read that followed the ideal/real simulation paradigm with security for malicious adversaries was this proof by Oded Goldreich (it appeared in a very early draft on Secure Multiparty Computation that can be found at www.wisdom.weizmann.ac.il/ oded/pp.html). I remember reading it multiple times until I understood why all the complications were necessary. Thus, for me, this proof brings back fond memories of my first steps in secure computation.
– by Lindell

The simulator here is the ideal-model adversary. It **externally interacts** with the trusted party computing the functionality, and **internally interacts** with the real-model adversary as part of the simulation.
(Internal interaction is not real, and is just the simulator internally feeding messages to $\mathcal{A}$ that it runs as a subroutine)

> "撒谎对自己有利的时候,为什么要说实话?" 罗素转向哲学,是希望在之前只有怀疑的地方找到确定性,而维特根斯坦,则是上述那种问题激起的强制倾向把他**拽进**了哲学。可以说,是哲学找的他,而非他找的哲学。在他的体验里,那个问题的两难是讨厌的侵扰和费解的谜,强加于他、俘虏了他,令他不能好好过日常生活,除非哪天能用一个满意的解答将其驱除。 --《维特根斯坦传：天才之为责任》

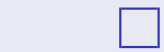
The simulator needs to send the trusted party the corrupted party's input and receive back its output. In this specific case of coin tossing, the parties have no input, and so the adversary just receives the output from the trusted party (formally, the parties send an empty string λ as input so that the trusted party knows to compute the functionality).
The challenge of the simulator is to make the output of the execution that it simulates equal the output that it received from the trusted party.

P_2 is corrupted

Proof.

Simulator $\mathcal{S}$:

1. $\mathcal{S}$ sends λ externally to the trusted party computing f_{ct} and receives back a bit b.
2. $\mathcal{S}$ initializes a counter $i = 1$.
3. $\mathcal{S}$ invokes $\mathcal{A}$, chooses a random $b_1 \in_R \{0,1\}$ and $r \in_R \{0,1\}^n$ and internally hands $\mathcal{A}$ the value $c = \mathrm{Com}(b_1; r)$ as if it was sent by P_2.
4. If $\mathcal{A}$ replies with $b_2 = b \oplus b_1$, then $\mathcal{S}$ internally hands $\mathcal{A}$ the pair (b_1, r) and outputs whatever $\mathcal{A}$ outputs. (As in the protocol, if $\mathcal{A}$ does not reply or replies with an invalid value, then this is interpreted as $b_2 = 0$.)
5. If $\mathcal{A}$ replies with $b_2 \neq b \oplus b_1$ and $i < n$, then $\mathcal{S}$ sets $i = i + 1$ and returns back to Step 3 .
6. If $i = n$, then $\mathcal{S}$ outputs fail.



P_2 is corrupted

Claim 1

$\mathcal{S}$ outputs fail with negligible probability

Claim 2

Conditioned on $\mathcal{S}$ does not output fail, the output distributions IDEAL and REAL are statistically close.

- **Real**: In a real execution, b_1 and r are uniformly distributed.
- **Ideal**: In an ideal execution, a random b is chosen, and then random b_1 and r are chosen under the constraint that $b_1 \oplus \mathcal{A}(\mathrm{Com}(b_1; r)) = b$.

P_1 is corrupted

Proof.

Simulator $\mathcal{S}$:

1. $\mathcal{S}$ sends λ externally to the trusted party computing f_{ct} and receives back a bit b.
2. $\mathcal{S}$ invokes $\mathcal{A}$ and internally receives the message c that $\mathcal{A}$ sends to P_1.
3. $\mathcal{S}$ internally hands $\mathcal{A}$ the bit $b_2 = 0$ as if coming from P_2, and receives back its reply. Then, $\mathcal{S}$ internally hands $\mathcal{A}$ the bit $b_2 = 1$ as if coming from P_2, and receives back its reply.
   a. If $\mathcal{A}$ replies with a valid decommitment (b_1, r) such that $\mathrm{Com}(b_1; r) = c$ in both iterations, then $\mathcal{S}$ externally sends $\mathrm{continue}$ to the trusted party. In addition, $\mathcal{S}$ defines $b_2 = b_1 \oplus b$, internally hands $\mathcal{A}$ the bit b_2, and outputs whatever $\mathcal{A}$ outputs.
   b. If $\mathcal{A}$ does not reply with a valid decommitment in either iteration, then $\mathcal{S}$ externally sends abort_1 to the trusted party. Then, $\mathcal{S}$ internally hands $\mathcal{A}$ a random bit b_2 and outputs whatever $\mathcal{A}$ outputs.

P_1 is corrupted

Proof.

- If $\mathcal{A}$ replies with a valid decommitment (b_1, r) such that $\mathrm{Com}(b_1; r) = c$ only when given b_2 where $b_1 \oplus b_2 = b$, then $\mathcal{S}$ externally sends $\mathrm{continue}$ to the trusted party. Then, $\mathcal{S}$ internally hands $\mathcal{A}$ the bit $b_2 = b_1 \oplus b$ and outputs whatever $\mathcal{A}$ outputs.
- If $\mathcal{A}$ replies with a valid decommitment (b_1, r) such that $\mathrm{Com}(b_1; r) = c$ only when given b_2 where $b_1 \oplus b_2 \neq b$, then $\mathcal{S}$ externally sends abort_1 to the trusted party. Then, $\mathcal{S}$ internally hands $\mathcal{A}$ the bit $b_2 = b_1 \oplus b \oplus 1$ and outputs whatever $\mathcal{A}$ outputs.

Claim

In each case, the joint distributions over $\mathcal{A}$'s output and the honest party's output are identical in the real and ideal executions.

Discussion

- In the malicious setting, many additional issues needed to be dealt with:
  1. The adversary can send any message and so the simulator must "interact" with it.
  2. The adversary may abort in some cases and this must be carefully simulated so that the distribution is not skewed when aborts can happen.
  3. The adversary may abort after it receives the output and before the honest party receives the output. This must be correlated with the abort and $\mathrm{continue}$ instructions sent to the trusted party, in order to ensure that the honest party aborts with the same probability in the real and ideal executions, and that this behavior matches the view of the adversary.
- The need to consider the joint distribution over the outputs, and to simulate for the output received from the trusted party, adds considerable complexity.

Securely Tossing Many Coins and the Hybrid Model

- To toss $\ell = \text{poly}(n)$ many coins in a constant number of rounds: $f_{\text{ct}}^{\ell}(\lambda, \lambda) = \left(U_{\ell(n)}, U_{\ell(n)}\right)$.
- Assume that we are given a constant-round protocol that securely computes the **zero-knowledge proof of knowledge functionality** for any $\mathcal{NP}$ -relation. This functionality is parameterized by a relation $R \in \mathcal{NP}$ and is defined by $f_{zk}^{R}((x, w), x) = (\lambda, R(x, w))$.
- **Hybrid functionalities**:
  - Let $L_1 = \{c \mid \exists(x, r) : c = \text{Com}(x; r)\}$ be the language of all valid commitments, and let R_1 be its associated $\mathcal{NP}$ -relation (for statement c the witness is x, r such that $c = \text{Com}(x; r)$).
  - Let $L_2 = \{(c, x) \mid \exists r : c = \text{Com}(x; r)\}$ be the language of all pairs of commitments and committed values, and let R_2 be its associated $\mathcal{NP}$ -relation (for statement (c, x) the witness is r such that $c = \text{Com}(x; r)$).
  - The parties have access to a trusted party that computes the zero-knowledge proof of knowledge functionalities $f_{zk}^{R_1}$ and $f_{zk}^{R_2}$ associated with relations R_1 and R_2, respectively.

Multiple Coin Tossing

1. P_1 chooses a random $\rho_1 \in \{0,1\}^{\ell(n)}$ and a random $r \in \{0,1\}^{\text{poly}(n)}$ of length sufficient to commit to $\ell(n)$ bits, and sends $c = \text{Com}(\rho_1; r)$ to P_2.
2. P_1 sends $(c, (\rho_1, r))$ to $f_{\text{zk}}^{R_1}$.
3. Upon receiving c, party P_2 sends c to $f_{zk}^{R_1}$ and receives back a bit b. If $b = 0$ then P_2 outputs $\perp$ and halts. Otherwise, it proceeds.
4. P_2 chooses a random $\rho_2 \in \{0,1\}^{\ell(n)}$ and sends ρ_2 to P_1.
5. Upon receiving ρ_2, party P_1 sends ρ_1 to P_2 and sends $((c, \rho_1), r)$ to $f_{\text{zk}}^{R_2}$. (If P_2 does not reply, or replies with an invalid value, then P_1 sets $\rho_2 = 0^{\ell(n)}$)
6. Upon receiving ρ_1, party P_2 sends (c, ρ_1) to $f_{\text{zk}}^{R_2}$ and receives back a bit b. If $b = 0$ then P_2 outputs $\perp$ and halts. Otherwise, it outputs $\rho = \rho_1 \oplus \rho_2$.
7. P_1 outputs $\rho = \rho_1 \oplus \rho_2$.

Proving in the hybrid model

$\mathcal{S}$ has many types of interactions :

1. External interaction with the trusted party: this is real interaction where $\mathcal{S}$ sends and receives messages externally.
2. Internal simulated interaction with the real adversary $\mathcal{A}$: this is simulated interaction and involves internally invoking $\mathcal{A}$ as a subroutine on incoming messages. This interaction is of two sub-types:
   a. Internal simulation of real messages between $\mathcal{A}$ and the honest party.
   b. **Internal simulation of ideal messages** between $\mathcal{A}$ and the trusted party computing the functionality used as a subprotocol in the hybrid model.The simulator directly receives the input that the adversary sends and can write any output that it likes.

Theorem

Assume that Com *is a perfectly-binding commitment scheme and let* ℓ *be a polynomial. Then, Protocol above securely computes the functionality* $f_{\mathrm{ct}}^{\ell}(\lambda, \lambda) = \left(U_{\ell(n)}, U_{\ell(n)}\right)$ *in the* $\left(f_{\mathrm{zk}}^{R_1}, f_{\mathrm{zk}}^{R_2}\right)$ *-hybrid model.*

P_1 corrupted

Proof.

Simulator $\mathcal{S}$:

1. $\mathcal{S}$ invokes $\mathcal{A}$, and receives the message c that $\mathcal{A}$ sends to P_2, and the message $(c', (\rho_1, r))$ that $\mathcal{A}$ sends to $f_{\mathrm{zk}}^{R_1}$.
2. If $c' \neq c$ or $c \neq \mathrm{Com}(\rho_1; r)$, then $\mathcal{S}$ sends abort_1 to the trusted party computing f_{ct}^{ℓ}, simulates P_2 aborting, and outputs whatever $\mathcal{A}$ outputs. Otherwise, it proceeds to the next step.
3. $\mathcal{S}$ sends 1^n to the external trusted party computing f_{ct}^{ℓ} and receives back a string $\rho \in \{0,1\}^{\ell(n)}$.
4. $\mathcal{S}$ sets $\rho_2 = \rho \oplus \rho_1$ (where ρ is as received from f_{ct}^{ℓ} and ρ_1 is as received from $\mathcal{A}$ as part of its message to $f_{zk}^{R_1}$), and internally hands ρ_2 to $\mathcal{A}$.

Proof.

5. $\mathcal{S}$ receives the message ρ_1' that $\mathcal{A}$ sends to P_2, and the message $((c'', \rho_1''), r'')$ that $\mathcal{A}$ sends to $f_{\mathrm{zk}}^{R_2}$. If $c'' \neq c$ or $\rho_1' \neq \rho_1''$ or $c \neq \mathrm{Com}(\rho_1''; r'')$ then $\mathcal{S}$ sends abort_1 to the trusted party computing f_{ct}^{ℓ}, simulates P_2 aborting, and outputs whatever $\mathcal{A}$ outputs. Otherwise, $\mathcal{S}$ externally sends $\mathrm{continue}$ to the trusted party, and outputs whatever $\mathcal{A}$ outputs.

□

Claim

Joint distribution over $\mathcal{A}$'s view and P_2 's output is identical in the real and ideal executions.

P_2 corrupted

Proof.

Simulator $\mathcal{S}$:

1. $\mathcal{S}$ sends 1^n to the external trusted party computing f_{ct}^{ℓ} and receives back a string $\rho \in \{0,1\}^{\ell(n)}$. $\mathcal{S}$ externally sends continue to the trusted party (P_1 always receives output).
2. $\mathcal{S}$ chooses a random $r \in \{0,1\}^{\text{poly}\,(n)}$ of sufficient length to commit to $\ell(n)$ bits, and computes $c = \text{Com}\left(0^{\ell(n)}; r\right)$
3. $\mathcal{S}$ internally invokes $\mathcal{A}$ and hands it c.
4. $\mathcal{S}$ receives back some ρ_2 from $\mathcal{A}$.
5. $\mathcal{S}$ sets $\rho_1 = \rho_2 \oplus \rho$ and internally hands $\mathcal{A}$ the message ρ_1 .
6. $\mathcal{S}$ receives some pair (c', ρ_1') from $\mathcal{A}$ as it sends to $f_{\text{zk}}^{R_2}$ (as the "verifier"). If $(c', \rho_1') \neq (c, \rho_1)$ then $\mathcal{S}$ internally simulates $f_{zk}^{R_2}$ sending 0 to $\mathcal{A}$. Otherwise, $\mathcal{S}$ internally simulates $f_{zk}^{R_2}$ sending 1 to $\mathcal{A}$.
7. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

P_2 corrupted

Proof.

Let $\mathcal{S}'$ work in the same way as $\mathcal{S}$ except that instead of receiving ρ externally from the trusted party, $\mathcal{S}'$ chooses ρ by itself (uniformly at random) after receiving ρ_2 from $\mathcal{A}$. In addition, the output of the honest party is set to be ρ. $\mathcal{S}'$ outputs the pair (ρ, output ($\mathcal{A}$)), where output ($\mathcal{A}$) is the output of $\mathcal{A}$ after the simulation. □

Claim 1

$$\left\{\mathcal{S}'\left(1^n\right)\right\}_{n\in\mathbb{N}} \equiv \left\{\mathsf{IDEAL}_{f_{\mathrm{ct}}^{\ell},\mathcal{S}}\left(1^n,1^n,n\right)\right\}_{n\in\mathbb{N}}$$

Claim 2

$$\left\{\mathcal{S}'\left(1^n\right)\right\}_{n\in\mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{REAL}_{\pi,\mathcal{A}}\left(1^n,1^n,n\right)\right\}_{n\in\mathbb{N}}$$

Discussion

- No rewinding is necessary.
- Nnot necessary to justify that the simulation is polynomial time,
- Not necessary to justify that the output distribution is not skewed by the rewinding procedure.

Section 6

Extracting Inputs – Oblivious Transfer

RAND procedure

Definition

Let $\mathbb{G}$ be a multiplicative group of prime order q. Define the probabilistic procedure

$$RAND(g, x, y, z) = (u, v) = \left(g^s \cdot y^t, x^s \cdot z^t\right)$$

where $s, t \in_R \mathbb{Z}_q$ are uniformly random.

Claim

Let g be a generator of $\mathbb{G}$ and let $x, y, z \in \mathbb{G}$. If (g, x, y, z) do not form a Diffie-Hellman tuple (i.e., there does not exist $a \in \mathbb{Z}_q$ such that $y = g^a$ and $z = x^a$), then $RAND(g, x, y, z)$ is uniformly distributed in $\mathbb{G}^2$.

Oblivious Transfer

- Oblivious transfer functionality : $f_{\text{ot}}\left(\left(x_0, x_1\right), \sigma\right) = \left(\lambda, x_\sigma\right)$
- Simulator must **extract** the input from the adversary, send it to the trusted party and receive back the output. The view generated by the simulator must then correspond to this input and output.
- Let $L = \{(\mathbb{G}, q, g_0, x, y, z) \mid \exists a \in \mathbb{Z}_q : y = (g_0)^a \wedge z = x^a\}$ be the language of all Diffie-Hellman tuples , and let R_L be its associated $\mathcal{NP}$ -relation. The parties have access to a trusted party that computes the zero-knowledge proof of knowledge functionality $f_{\text{zk}}^{R_L}$ associated with relation R_L.

Oblivious Transfer

1. Party P_2 chooses random values $y, \alpha \in_R \mathbb{Z}_q$ and computes $g_1 = (g_0)^y$, $h_0 = (g_0)^\alpha$ and $h_1 = (g_1)^{\alpha+1}$ and sends (g_1, h_0, h_1) to party P_1.
2. P_2 sends statement $\left(\mathbb{G}, q, g_0, g_1, h_0, \frac{h_1}{g_1}\right)$ and witness α to $f_{\text{zk}}^{R_L}$.
3. P_1 sends statement $\left(\mathbb{G}, q, g_0, g_1, h_0, \frac{h_1}{g_1}\right)$ to $f_{\text{zk}}^{R_L}$ and receives back a bit. If the bit equals 0 , then it halts and outputs $\perp$. Otherwise, it proceeds to the next step.
4. P_2 chooses a random value $r \in_R \mathbb{Z}_q$, computes $g = (g_\sigma)^r$ and $h = (h_\sigma)^r$, and sends (g, h) to P_1
5. P_1 computes $(u_0, v_0) = RAND(g_0, g, h_0, h)$, and $(u_1, v_1) = RAND(g_1, g, h_1, h)$. P_1 sends P_2 the values (u_0, w_0) where $w_0 = v_0 \cdot x_0$, and (u_1, w_1) where $w_1 = v_1 \cdot x_1$.
6. P_2 computes $x_\sigma = w_\sigma / (u_\sigma)^r$.
7. P_1 outputs λ and P_2 outputs x_σ

Theorem

Assume that the Decisional Diffie-Hellman problem is hard in the auxiliary-input group G. Then, Protocol above securely computes f_{ot} in the presence of malicious adversaries.

P_1 is corrupted

Proof.

1. $\mathcal{S}$ internally invokes $\mathcal{A}$ controlling P_1
2. $\mathcal{S}$ chooses $y, \alpha \in_R \mathbb{Z}_q$ and computes $g_1 = (g_0)^y$, $h_0 = (g_0)^\alpha$ and $h_1 = (g_1)^\alpha$. (Note that $h_1 = (g_1)^\alpha$ and not $(g_1)^{\alpha+1}$)
3. $\mathcal{S}$ internally hands (g_1, h_0, h_1) to $\mathcal{A}$.
4. When $\mathcal{A}$ sends a message intended for $f_{zk}^{R_L}$. If the message equals $\left(\mathbb{G}, q, g_0, g_1, h_0, \frac{h_1}{g_1}\right)$ then $\mathcal{S}$ internally hands $\mathcal{A}$ the bit 1 as if it came from $f_{zk}^{R_L}$. If the message equals anything else, then $\mathcal{S}$ simulates $\mathcal{A}$ receiving 0 from $f_{\mathrm{zk}}^{R_L}$.
5. $\mathcal{S}$ chooses a random value $r \in_R \mathbb{Z}_q$, computes $g = (g_0)^r$ and $h = (h_0)^r$, and internally sends (g, h) to $\mathcal{A}$. (This is exactly like an honest P_2 with input $\sigma = 0$.)

Proof.

6. When $\mathcal{A}$ sends messages $(u_0, w_0), (u_1, w_1)$ then simulator $\mathcal{S}$ computes $x_0 = w_0 / (u_0)^r$ and $x_1 = w_1 / (u_1)^{r \cdot y^{-1} \bmod q}$.
7. $\mathcal{S}$ sends (x_0, x_1) to the trusted party computing f_{ot} .
8. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs, and halts.

□

DDH assumption

P_2 is corrupted

Proof.

1. $\mathcal{S}$ internally invokes $\mathcal{A}$ controlling P_2.
2. $\mathcal{S}$ internally obtains (g_1, h_0, h_1) from $\mathcal{A}$, as it intends to send to P_1.
3. $\mathcal{S}$ internally obtains an input tuple and α from $\mathcal{A}$, as it intends to send to $f_{\text{zk}}^{R_L}$.
4. $\mathcal{S}$ checks that the input tuple equals $\left(\mathbb{G}, q, g_0, g_1, h_0, \frac{h_1}{g_1}\right)$, that $h_0 = (g_0)^\alpha$ and $\frac{h_1}{g_1} = (g_1)^\alpha$.
5. $\mathcal{S}$ internally obtains a pair (g, h) from P_2. If $h = g^\alpha$ then $\mathcal{S}$ sets $\sigma = 0$. Otherwise, it sets $\sigma = 1$.
6. $\mathcal{S}$ externally sends σ to the trusted party computing f_{ot} and receives back x_σ.
7. $\mathcal{S}$ computes $(u_\sigma, v_\sigma) = RAND(g_\sigma, g, h_\sigma, h)$ and $w_\sigma = v_\sigma \cdot x_\sigma$. In addition, $\mathcal{S}$ sets $(u_{1-\sigma}, w_{1-\sigma})$ to be independent uniformly distributed in $\mathbb{G}^2$.

Proof.

8. $\mathcal{S}$ internally hands $(u_0, w_0), (u_1, w_1)$ to $\mathcal{A}$.
9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs and halts.

□

Section 7

The Common Reference String Model – Oblivious Transfer

CRS model

- plain model with no trusted setup
- a trusted setup is used to obtain additional properties
- CRS can be used to achieve non-interactive zero knowledge , which is impossible in the plain model.
- CRS is used to achieve security under composition
- In the CRS model, in the real model the parties are provided the same string generated by M, whereas in the ideal model the simulator chooses the string. Since the real and ideal models must be indistinguishable, this means that the CRS chosen by the simulator must be indistinguishable from the CRS chosen by M.
- The motivation behind this definition is that if an adversary can attack the protocol in the real model, then it can also attack the protocol in the ideal model with the simulator.
- The simulator must have additional power beyond that of a legitimate party. In the CRS model, it is possible to construct a simulator that does not rewind the adversary, since its additional power is in choosing the CRS itself.

CRS model

Two ways to define security in the CRS model:

- to include the CRS in the output distributions.
- to define an ideal CRS functionality $f_{\text{crs}}(1^n, 1^n) = (M(1^n), M(1^n))$. Then, one constructs a protocol and proves its security in the f_{crs} -hybrid model.

Hybrid functionality f_{crs} : A group $\mathbb{G}$ of order q(of length n) with generator g_0 is sampled, along with three random elements $g_1, h_0, h_1 \in_R \mathbb{G}$ of the group. f_{crs} sends $(\mathbb{G}, q, g_0, g_1, h_0, h_1)$ to P_1 and P_2.

OT

1. P_2 chooses a random value $r \in_R \mathbb{Z}_q$, computes $g = (g_\sigma)^r$ and $h = (h_\sigma)^r$, and sends (g, h) to P_1
2. P_1 computes $(u_0, v_0) = RAND(g_0, g, h_0, h)$, and $(u_1, v_1) = RAND(g_1, g, h_1, h)$. P_1 sends P_2 the values (u_0, w_0) where $w_0 = v_0 \cdot x_0$, and (u_1, w_1) where $w_1 = v_1 \cdot x_1$
3. P_2 computes $x_\sigma = w_\sigma / (u_\sigma)^r$.
4. P_1 outputs λ and P_2 outputs x_σ.

Section 8

Advanced Topics

Subsection 1

Composition and Universal Composability

Composition and Universal Composability

- stand-alone model implies security under sequential composition
- security under concurrent composition
- adding an **environment machine** which is essentially an interactive distinguisher. The environment writes the inputs to the parties' input tapes and reads their outputs. In addition, it externally interacts with the adversary throughout the execution. The environment's "goal" is to distinguish between a real protocol execution and an ideal execution.

Proofs in the Random Oracle Model

Proofs in the Random Oracle Model

- Random oracle model is used to gain higher efficiency or other properties otherwise unobtainable.
- Whether or not the distinguisher obtains access to the random oracle, and if yes, how.
  - If the distinguisher does not have any access: very weak definition, sequential composition will not be guaranteed.
  - If provide with the same randomly chosen oracle as the parties and the (real and ideal) adversary: obtain a *non-programmable* random oracle which may not be strong enough.
  - Provide the random oracle, but in the ideal world to allow the simulator to still control the oracle: a somewhat strange formulation, but something of this type seems necessary in some cases.

Adaptive Security

Adaptive Security

- **static adversaries** where the subset of corrupted parties is fixed before the protocol execution begins.
- **adaptive adversary** can choose which parties to corrupt throughout the protocol, based on the messages viewed
- **no erasures model**: parties cannot securely erase data
- **erasures model**: parties can securely erase data

Thanks!