Key idea: Basic scheme from PKE but instead of evaluating f using the garbled circuit, we instead evaluate the FHE decryption function, which has complexity smaller than f

Security: ABE security: labels not associated with $ct_{f(x)}$ hidden by semantic security

Garbling security: Can simulate garbled circuit + labels given only FHE.Decrypt$(\cdot,\cdot)$ and $f(x)$ ← removes dependence on FHE secret key

FHE security: Replace encryption of x_0 with x_1

Still only secure in the single-key setting (since garbled circuit is not reusable)

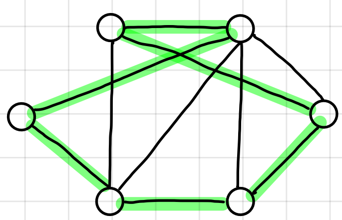
Recap: functional encryption (FE) provides fine-grained access to encrypted data

↳ general mechanism for achieving confidentiality with computation

Next: NIZK from lattices – integrity for computations

Useful building block: Blum's protocol for graph Hamiltonicity [previously: may have encountered protocol for 3-coloring]

Hamiltonian path problem: given a graph $G=(V,E)$, decide whether there is a cycle that visits every node exactly once



Deciding whether a graph contains a Hamitonian cycle is NP-complete

We will build a Σ-protocol for graph Hamiltonicity (3-round public-coin ZK protocol):

prover

1. Sample random permutation
   $\pi \xleftarrow{R} \text{Perm}[V]$
2. Commit to edges in the permuted graph
   $\forall i,j \in [n]$: if $(i,j) \in E$, $c_{\pi(i),\pi(j)} \leftarrow \text{Commit}(1)$
   else, $c_{\pi(i),\pi(j)} \leftarrow \text{Commit}(0)$

technically, commitments will depend on a set of public parameters / common reference string

verifier

$\xrightarrow{\{c_{ij}\}_{i,j\in[n]}}$

$b \xleftarrow{R} \{0,1\}$

$\xleftarrow{b}$

if $b=0$: reveal π and open all c_{ij}

if $b=1$: open only the edges corresponding to the Hamitonian cycle

$\longrightarrow$

Completeness: Follows by inspection.

Soundness: Suppose G does not have a Hamiltonian cycle.

Suppose that the commitments are statistically binding

1. Suppose prover commits to a graph G' that is not isomorphic to G.
   Then, if $b=0$, prover cannot succeed.

2. Suppose prover commits to a graph G' where $G' = \pi(G)$ for some permutation π.
Then, if $b = 1$, prover <u>cannot</u> succeed.

In both cases, prover succeeds with probability at most $1/2$.

<u>HVZK</u>: Simulator operates as follows:

1. Sample challenge bit $b \xleftarrow{R} \{0,1\}$
2. If $b = 0$, sample a random permutation π and construct commitments to $\pi(G)$. Output commitments along with openings.
   If $b = 1$, sample a random cycle graph, commit to 1 for edges in cycle graph and 0 elsewhere. Output all commitments and openings for edges in cycle graph.

Correctness of simulation:

- $b = 0$ case is perfectly simulated
- $b = 1$ case is computationally indistinguishable from real transcript (since commitments are hiding)
- challenge bit sampled as in real protocol

To amplify soundness, we can repeat the protocol λ times in parallel.
Resulting protocol has soundness $2^{-\lambda}$.
↳ For every choice of prover message, there is now a <u>single</u> bad challenge string $c \in \{0,1\}^{\lambda}$ that allows prover to succeed

will be a useful property later on

<u>Designated-verifier NIZK</u>: secret key is needed to check proofs (i.e., <u>single</u> verifier)

- $\mathrm{Setup}(1^{\lambda}) \to (pk, sk)$
- $\mathrm{Prove}(pk, x, w) \to \pi$
- $\mathrm{Verify}(sk, x, \pi) \to 0/1$

<u>Completeness</u>: If $R(x,w) = 1$, then $(pk, sk) \leftarrow \mathrm{Setup}(1^{\lambda})$, $\pi \leftarrow \mathrm{Prove}(pk, x, w)$,
$$\Pr[\mathrm{Verify}(sk, x, \pi) = 1] = 1$$

<u>Soundness</u>:

<u>adversary</u>		<u>challenger</u>
		$(pk, sk) \leftarrow \mathrm{Setup}(1^{\lambda})$
	$\xleftarrow{\quad pk \quad}$	
	$\xrightarrow{\quad x, \pi \quad}$	

↳ adversary wins if $x \notin \mathcal{L}$ and $\mathrm{Verify}(sk, x, \pi) = 1$

<u>Important</u>: adversary does <u>not</u> have oracle access to Verify (<u>non-reusable</u> soundness)

<u>Zero-Knowledge</u>: for all efficient adversaries, there exists an efficient simulator S such that for all $(x, w) \in R$:

$$\left\{ \begin{array}{l} (pk, sk) \leftarrow \mathrm{Setup}(1^{\lambda}) \\ \pi \leftarrow \mathrm{Prove}(pk, x, w) \\ \text{output } (pk, sk, x, \pi) \end{array} \right\} \overset{c}{\approx} \left\{ \begin{array}{l} (\tilde{pk}, \tilde{sk}, \tilde{\pi}) \leftarrow S(1^{\lambda}, x) \\ \text{output } (\tilde{pk}, \tilde{sk}, x, \tilde{\pi}) \end{array} \right\}$$

Formulated as <u>one-time</u> security notion

general theme: <u>removing</u> interaction via setup

<u>One-time DV-NIZK from PKE</u>: Move challenge in Σ-protocol to the public key [similar trick used to get single-key FE]

in the case of FE, we took Yao's protocol and moved the OT into the public key

<u>prover</u>		<u>verifier</u>
	$\xrightarrow{\quad \text{commitments} \quad}$	
	$\xleftarrow{\quad b_1, \dots, b_{\lambda} \xleftarrow{R} \{0,1\} \quad}$	
	$\xrightarrow{\quad \text{responses} \quad}$	

move the challenge to the public key

Setup(1^λ): for $i \in [\lambda]$ and $b \in \{0,1\}$, $(pk_i^{(b)}, sk_i^{(b)}) \xleftarrow{R} \text{PKE.Setup}(1^\lambda)$

sample challenge $b_1, \dots, b_\lambda \xleftarrow{R} \{0,1\}$

output $pk = \{pk_i^{(b)}\}_{i \in [\lambda], b \in \{0,1\}}$

$sk = (sk_1^{(b_1)}, \dots, sk_\lambda^{(b_\lambda)}, b_1, \dots, b_\lambda)$

Prove(pk, x, w): construct first message $\sigma_1, \dots, \sigma_\lambda$ of λ copies of the Σ-protocol

for each σ_i, compute response $z_i^{(0)}$ and $z_i^{(1)}$ to be the responses associated with σ_i and challenge bit 0/1

compute $ct_i^{(b)} \leftarrow \text{PKE.Encrypt}(pk_i^{(b)}, z_i^{(b)})$

output proof $\pi = (\sigma_1, \dots, \sigma_\lambda, ct_1^{(0)}, ct_1^{(1)}, \dots, ct_\lambda^{(0)}, ct_\lambda^{(1)})$

Verify(sk, x, π): decrypt responses $z_i^{(b_i)} \leftarrow \text{PKE.Decrypt}(sk_i^{(b_i)}, ct_i^{(b_i)})$

check that $(\sigma_i, b_i, z_i^{(b_i)})$ is valid for each $i \in [\lambda]$

Completeness: Follows by correctness of PKE + completeness of Σ-protocol

Soundness: Follows by soundness of Σ-protocol

[The public key pk perfectly hides the challenge bits $b_1, \dots, b_\lambda$ and $b_1, \dots, b_\lambda \xleftarrow{R} \{0,1\}$]

Zero-Knowledge: Simulator takes input x and operates as follows:

1. Use HVZK simulator to sample transcripts (σ_i, b_i, z_i) for each $i \in [\lambda]$
2. Sample $(pk_i^{(b)}, sk_i^{(b)}) \leftarrow \text{PKE.Setup}(1^\lambda)$ for $i \in [\lambda]$ and $b \in \{0,1\}$
3. For each $i \in [\lambda]$, $ct_i^{(b_i)} \leftarrow \text{PKE.Encrypt}(sk_i^{(b_i)}, z_i)$

   $ct_i^{(1-b_i)} \leftarrow \text{PKE.Encrypt}(sk_i^{(1-b_i)}, 0^{|z_i|})$
4. Output $pk = \{pk_i^{(b)}\}_{i \in [\lambda], b \in \{0,1\}}$

   $sk = (sk_1^{(b_1)}, \dots, sk_\lambda^{(b_\lambda)})$

   $\pi = (\sigma_1, \dots, \sigma_\lambda, ct_1^{(0)}, ct_1^{(1)}, \dots, ct_\lambda^{(0)}, ct_\lambda^{(1)})$

To show that this transcript is computationally indistinguishable from real transcript, we can use a hybrid argument:

- Hyb_0: real game
- Hyb_1: replace $ct_i^{(1-b_i)}$ with encryption of all-zeroes string
- Hyb_2: replace (σ_i, b_i, z_i) with simulated transcripts

PKE security

HVZK security of Σ-protocol

Construction is not reusable. Suppose adversary has oracle access to Verify$(sk, \cdot, \cdot)$ — models reusability of the verification key sk

Problem: Malicious prover can learn the challenge (same challenge used for verification)

Underlying Σ-protocol trivially broken if prover knows the challenge (e.g., can use HVZK simulator to obtain transcript)

How do we recover $b_1, \dots, b_\lambda$ from oracle access to Verify$(sk, \cdot, \cdot)$?

prover | verifier

1. Start with a valid proof of a true statement x:

$\sigma_1, \dots, \sigma_\lambda$

honest proof: ciphertexts encrypt correct response

$$\begin{bmatrix} ct_1^{(0)} & ct_2^{(0)} & \cdots & ct_\lambda^{(0)} \\ ct_1^{(1)} & ct_2^{(1)} & \cdots & ct_\lambda^{(1)} \end{bmatrix}$$

$\xrightarrow{\quad x, \pi \quad}$

Verifier decrypts $ct_i^{(b_i)}$ and checks response

Observe: Verifier cannot decrypt $ct_i^{(1-b_i)}$ and its output does not depend on $ct_i^{(1-b_i)}$

2. Replace $ct_1^{(0)}$ with encryption of $\perp$

$\xrightarrow{\quad x, \pi' \quad}$

If verifier still accepts, then it did not decrypt $ct_1^{(0)}$ so $b_1 \neq 0 \Rightarrow b_1 = 1$

If verifier rejects, then it decrypts $ct_1^{(0)}$ so $b_1 = 0$

↳ Whether verifier rejects or not leaks b_1!

First construction: from circular-secure FHE (not quite LWE, but close)