Privacy Enhancing Technologies

1 Commitment Schemes

Def. Commitment scheme algorithm $\texttt{Commit} : \mathcal{M} \times \mathcal{R} \to \mathcal{C}$

- **(Statistically) Hiding**: c leaks nothing about m
- **(Computationally) Binding**: No PTT adversary can produce $c_1 = c_2$ for $m_1 \neq m_2$

Random oracle model Replace hash function by *completely random function* $H : \mathcal{X} \to \mathcal{Y}$.

Thm. $\texttt{Commit}(m, r) \coloneqq H(m, r)$ is stat. hiding and comp. binding in ROM ($|\mathcal{K}|$ and $|\mathcal{C}|$ have size $2^{\text{poly}(\lambda)}$)

Def. Pedersen Comm. Let $\mathbb{G}$ be group of prime order q, let $g, h \in \mathbb{G}$ generators s.t. $g^x = h$ and x **unknown**.

$\texttt{Commit}(m, r) \coloneqq g^m h^r = g^{m+r \cdot x}$

Properties

- Linearly homomorphic ($\texttt{Commit}(m_0, r_0) \cdot \texttt{Commit}(m_1, r_1) = \texttt{Commit}(m_0 + m_1, r_0 + r_1)$)
- Statistically hiding (for any c and m there is a unique r)
- Computationally binding (same advantage as in DLog)

How to **break DLog**: find to representations of same element: $g^{m_0} h^{r_0} = c = g^{m_1} h^{r_1}, m_0 \neq m_1$
$\implies m_0 + x \cdot r_0 = m_1 + x \cdot r_1 \implies x = \frac{m_1 - m_0}{r_0 - r_1} \bmod q$

Def. DDH Problem Consider the distributions
$D_0 = \{(g^\alpha, g^\beta, g^{\alpha\beta}) : \alpha, \beta \leftarrow^{\$} \mathbb{Z}_q\}$
$D_1 = \{(g^\alpha, g^\beta, g^\gamma) : \alpha, \beta, \gamma \leftarrow^{\$} \mathbb{Z}_q\}$
$\implies$ $\text{AdvDDH}(\mathcal{A}, \mathbb{G}) = |\Pr[W_0] - \Pr[W_1]|$, where W_0, W_1 are events that $\mathcal{A}$ outputs 1 given a sample from D_0 or D_1.

DDH Assumption $\text{AdvDDH}(\mathcal{A}, \mathbb{G}) = \text{negl}(\lambda)$ for all PTT $\mathcal{A}$

Thm. If DDH Assumption holds and $H : \mathbb{Z}_q \times \mathcal{X} \to \mathbb{G}$ is a random function, then **PRF** $F(k, x) = H(x)^k$ is secure.

Vector commitment Produces commitment c of size λ and opening proof π of size $= (\log n \cdot \lambda)$: $H(m_1), ..., H(m_n)$ are leaves of binary tree. Each node with children v_l, v_r has value $H(v_l, v_r)$. Commitment c is root hash.

To open at position i, walk from leaf i to root, revealing values of the sibling nodes along the path.

2 Zero Knowledge Proofs

NP Language NP is class of languages $\mathcal{L}$ s.t. there exists efficient, deterministic algo M s.t.
$x \in \mathcal{L} \iff \exists w \in \{0,1\}^{\text{poly}(|x|)}$ s.t. $M(x, w) = 1$

Def. Interactive Proof unbounded prover, efficient verifier

- **Complete**: $\forall x \in \mathcal{L} : \Pr[\langle P, V \rangle(x) = 1] \geq \frac{2}{3}$
- **Sound**: $\forall x \notin \mathcal{L}, \forall$ provers $P^* : \Pr[\langle P^*, V \rangle(x) = 1] \leq \frac{1}{3}$

Def. ZK Proof Interactive proof system $\langle P, V \rangle$ that satisfies completeness and soundness, and $\forall$ `(malicious)` V^*:
$\exists$ simulator $\texttt{Sim}_{V^*}$ s.t. $\forall x \in \mathcal{L} : \texttt{View}[\langle P, V^* \rangle] \approx^c \texttt{Sim}_{V^*}(x)$

Ex. ZK proof for Ham-Cycle Assuming comm. scheme w/ perfect binding, computational hiding: valid ZK proof w/ perfect completeness and soundness error $\leq \frac{1}{2}$

1. P(G, Ham-Cycle): pick random permutation, insert edges into permuted adjacency matrix and commit to each value
2. V(G): sample random challence $c \leftarrow^{\$} \{0, 1\}$
3. P: if $c = 0$, open commitments. Else, open all commitments where edges are in hamiltonian cycle
4. V: if $c = 0$, check if graphs are isomorphic. Else, check that a cycle was opened

Sim(G):

1. Guess $\hat{c} \leftarrow^{\$} \{0, 1\}$. If $\hat{c} = 0$, commit to random permutation of G. Else, commit to complete graph
2. Send commitments and receive c
3. If $c \neq \hat{c}$ abort and restart
4. Else, open requested commitments and output the view

Parallel vs. in sequence In sequence: run the protocol λ times one-after-the-other. Parallel: run λ instances with λ provers interacting with their verifiers. Tricky to build an efficient simulator for parallel, since if verifier is malicious, it can sample the challenges in a way unknown to the verifier and easily detect a simulated view.

Def. Proof of Knowledge Proof system $\langle P, V \rangle$ for NP lang $\mathcal{L}$ is proof of knowledge w/ knowledge error ϵ if efficient extractor $\texttt{Ext}$ exists s.t. $\forall x, P^*$:
$\Pr[M(x, w) = 1 : w \leftarrow \texttt{Ext}^{P^*}(x)] \geq \Pr[\langle P, V \rangle(x) = 1] - \epsilon$
The extractor can rewind the prover.

Def. Sigma protocol Three-round protocol $\langle P, V \rangle$ of the form *commitment* $u \to$ *challenge* $c \to$ *response* z with:

- perfect completeness: accepts all valid statements
- special soundness: there exists $\texttt{Ext}$, s.t. given 2 accepting transcripts $(u, c, z), (u, c', z')$ outputs w s.t. $M(x, w) = 1$
- honest verif. ZK: $\texttt{Sim}$ outputs computationally indistinguishable transcript from a honest verifier's view

Thm. Sigma prot. gives interact. proof w/ sound. err. $\leq \frac{1}{|C|}$

Non-interactive NIZK Simulator can program the RO for verifier's RO queries, as long as responses are indist. from random

Fiat-Shamir transform Turns Sigma-protocol into NIZK: compute challenge as hash of prior messages of prover ($c = H(x, u)$)
For NIZK, we require **negl. soundness and completeness**!

3 SNARGs

Lem. Equality Test To check if polynomials f and g are equal, check equality at a random point. This has perfect completeness and soundness error $\frac{d}{|\mathbb{F}|}$.

Fundamental theorem of algebra Any non-zero polynomial over field $\mathbb{F}$ of degree d has at most d roots over $\mathbb{F}$.

Lem. Schwartz-Zippel Let $f(X_1, ..., X_n)$ be multivariate polynomial of total degree d. Let Ω a finite subset of $\mathbb{F}$ and $r_1, ..., r_n \in \Omega$ uniformly random. Then $\Pr[f(r_1, ..., r_n) = 0] \leq \frac{d}{|\Omega|}$

Thm. Let $\Omega \subseteq \mathbb{F}$, $|\Omega| = k \leq d$. Let $Z_\Omega(X) = \prod_{a \in \Omega}(X - a)$.
Then: f is zero on $\Omega \iff f(X)$ divisible by $Z_\Omega(X)$

Thm. PCP Let L be an NP language. There exist two efficient algorithms $(\mathcal{P}, \mathcal{V})$ defined as follows:

- Prover $\mathcal{P}$ is deterministic algo that takes as input a statement $x \in \{0,1\}^n$, a witness $w \in \{0,1\}^h$ and outputs a bitstring $\pi \in \{0,1\}^m$, where $h, m = poly(n)$. We refer to π as the proof string.
- Verifier algo $\mathcal{V}^\pi$ is randomized and takes as input a statement $x \in \{0,1\}^n$ and has oracle access to proof string $\pi \in \{0,1\}^m$. Verifier reads $O(1)$ bits of π (chosen non-adaptively).
- **Complete**: $\forall x \in \mathcal{L}$ if w witness for x:
  $\Pr[\mathcal{V}^\pi(x) = 1 : \pi \leftarrow \mathcal{P}(x, w)] = 1$
- **Sound**: if $x \notin \mathcal{L}$, then for all $\pi \in \{0,1\}^m$:
  $\Pr[\mathcal{V}^\pi(x) = 1] \leq \frac{1}{2}$

Prove that $f(z) = y$ at some point z using ZeroTest: we can write $f(X) - y = (X - z) \cdot q(X)$, i.e. $q(X) = \frac{f(X) - y}{X - z}$

PLONK IOP Represent an arithmetic circuit as polynomials, using $w \in \mathbb{F}$ a root of unity (i.e. $w^{3|C|+|I|} = 1$). When applying Fiat-Shamir, this becomes a SNARG for NP.

- $f_{trace}(w^{-j}) =$ input j
- $f_{trace}(w^{3l}) =$ left input to gate l
- $f_{trace}(w^{3l+1}) =$ right input to gate l
- $f_{trace}(w^{3l+2}) =$ output of gate l

To verify, then check the following:

- Output of last gate is 0 ($f_{trace}(w^{3|C|-1}) = 0$)
- Public inputs: $f_{trace} - f_{in} = 0$ (ZeroTest)
- each gate is evaluated correctly (ZeroTest, using selector polynomial): $f_{sel}(x) \cdot (f_{trace}(x) + f_{trace}(wx)) + (1 - f_{sel}(x)) \cdot (f_{trace}(x) \times f_{trace}(wx)) = f_{trace}(w^2 x)$
- wiring is implemented correctly (outputs = inputs)

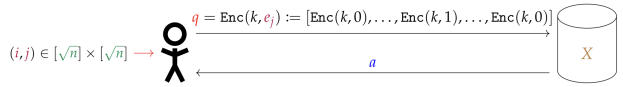
4 Private Information Retrieval (PIR)

2-server PIR View the DB as a matrix X of $\sqrt{n} \times \sqrt{n}$ bits. Then, using $O(n)$ communication, load the j-th row and read the i-th bit to retrieve X_{ij}.

- $\texttt{Query}((i,j))$: $q_0 \leftarrow^{\$} \mathbb{Z}_2^{\sqrt{n}}$, $q_1 = q_0 \oplus e_j$
- $\texttt{Answer}(X, q_b)$: $a_b \leftarrow X \cdot q_b \in \mathbb{Z}_2^{\sqrt{n}}$
- $\texttt{Reconstruct}(a_0, a_1)$: $x_i \leftarrow (a_0 \oplus a_1)_i$

Linearly homomorphic encryption
$\texttt{Enc}(k, m_1) + \texttt{Enc}(k, m_2) = \texttt{Enc}(k, m_1 + m_2)$

Single-server PIR Using LHE and load balancing:



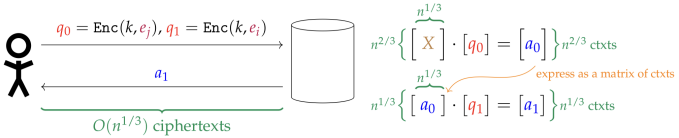
$(\texttt{Dec}(k,a))_i = \texttt{Dec}(k, \texttt{Enc}(k, X_j)_i) = X_{ij}$

$a = X \cdot q = X \cdot \texttt{Enc}(k, e_j) = \texttt{Enc}(k, X \cdot e_j) = \texttt{Enc}(k, X_j)$

requires only adding ciphertexts

The total communication is $|q| + |a| = \sqrt{n} + \sqrt{n}$ ciphertexts, or $O(\lambda\sqrt{n})$ bits.

Recursive PIR View X as $n^{\frac{2}{3}} \times n^{\frac{1}{3}}$ matrix and recurse:



Watch out for ciphertext expansion!

5 Oblivious RAM

Def. Privately read and write to private RAM

- **Correctness**: ORAM answers correctly if RAM is honest
- **Security**: For any two poly-sized seq. of equal length: $\{\texttt{Access}(op_1), ..., \texttt{Acc}(op_k)\} \approx^c \{\texttt{Access}(op'_1), ..., \texttt{Acc}(op'_k)\}$ i.e. only the number of operations is leaked

A $\sqrt{n}$ Construction Any $k \leq n$ ops that access distinct locations are indistinguishable. The client keeps stash of $\sqrt{n}$ words in internal state.

`while (true)`: The RAM is shuffled according to some random permutation π (oblivious sorting, data-independent). If some value is modified in the stash, update the RAM. Process $\sqrt{n}$ ops:

- if address in stash, read random address and discard
- else, read address and store in stash
- if op is write, update in stash

Amortized cost per access: $O(\sqrt{n/k})$ where k is size of stash and $k = \sqrt{n}$ for balanced costs. Shuffle in $O(n \log^2 n)$ time, then process $\sqrt{n}$ ops with 1 RAM access each.

Lower bound $O(\log n)$ accesses per op, there are schemes that achieve this bound. For $O(\text{polylog} n)$ use tree-based ORAM.

6 Multi-party computation (MPC)

Def. n Parties $P_1, ..., P_n$ with inputs $x_1, ..., x_n$ want to compute $y \leftarrow f(x_1, ..., x_n)$.

Adv should learn nothing more about honest parties' inputs than if they were interacting with trusted third party. An adversary learns 1) inputs of corrupted parties and 2) output y of func f.

Def. Semi-honest security Adversary follows protocol spec, but can do additional poly-time local computation. Tries to learn honest parties' inputs. The view of the corrputed parties C should be indistinguishable from simulation.

$$\texttt{Sim}(C, \underbrace{\{x_i : i \in C\}}_{\text{the inputs of corrupted parties}}, \underbrace{y = f(x_1, \ldots, x_n)}_{\text{the output of the computation}}) \approx \underbrace{\{\texttt{View}_i : i \in C\}}_{\text{the views of all corrupted parties in a real protocol execution}}$$

Def. Malicious security Adversary may arbitrarily deviate from protocol to learn about honest parties' inputs or fool them into accepting incorrect output.

7 Examples

New DDH triples Build new DDH triple from (X, Y, Z): sample $r \leftarrow^{\$} \mathbb{Z}_q$, set $X' = X \cdot g^r$, $Z' = Z \cdot Y^r$

PRF security Using PRF $F(k,x) = H(x)^k$, PRF adversary has access to RO for H. DDH adversary simulates a PRF challenger and receives (X,Y,Z). When getting RO query m_i, sample random $r_i \leftarrow^{\$} \mathbb{Z}_q$, set $H(m_i) = X \cdot g^{r_i}$ and store (m_i, r_i).

When getting a PRF query for m_j, return $Z \cdot Y^{r_j}$. Simulation is perfect, since RO responses are random. If $b = 0$, Z is random and so is $Z' = Z \cdot Y^{r_j}$.

If $b = 1$, $Z' = (X')^\beta = H(m_j)^\beta = F(\beta, m_j)$.

SNARGs Prove that $fib(100) = x$. Prover puts the following polynomials in the box: $f_{fib}(X)$ and $q(X) = \frac{f_{fib}(X+2) - f_{fib}(X) - f_{fib}(X+1)}{\prod_{i=0}^{98}(X-i)}$. The verifier queries $f_{fib}(0)$, $f_{fib}(1)$, $f_{fib}(100)$ and $f_{fib}(r)$, $f_{fib}(r+1)$, $f_{fib}(r+2)$, $q(r)$ and checks $f_{fib}(r+2) - f_{fib}(r) - f_{fib}(r+1) = q(r) \cdot \prod_{i=0}^{98}(r-i)$

This has perfect completeness and soundness error $\frac{2d}{|\mathbb{F}|}$.

PIR from FHE Given secret-key FHE scheme that supports enc, NOT, AND, OR: build 1-server PIR for DB of n bits.

Write $i \in [n]$ as binary string $i = i_1 i_2 ... i_{\log n}$ and send to server, along with enc. of 0 and 1.

For each index $j = j_1 j_2 ... j_{\log n}$ the server replaces j_i by corresponding ciphertext and homomorph. evaluates $C_{EQ}(i,j)$ over encrypted bits. Then, multiply this with ciphertext of bit X_j and sum up all ciphertexts (with OR).

PIR from DPF A DPF creates keyed functions $\texttt{Eval}(k_0, \cdot)$, $\texttt{Eval}(k_1, \cdot)$ s.t. $\texttt{Eval}(k_0, \cdot) + \texttt{Eval}(k_1, \cdot) = P_{x,y}(\cdot)$.

To retrieve i-th bit, generate keys $(k_0, k_1) \leftarrow \texttt{Gen}(i, 1)$.

$\left(\sum_j \texttt{Eval}(k_0, j) \cdot X_j\right) \oplus \left(\sum_j \texttt{Eval}(k_1, j) \cdot X_j\right) = \sum_j P_{i,1}(j) \cdot X_j$

For *PIR with keywords*, generate keys $(k_0, k_1) \leftarrow \texttt{Gen}(w', 1)$.