Probabilistic Polynomial Time

PPT: Probabilistic Polynomial Time

A language L is BPP (bounded from half probabilistic polynomial time) if $\exists M \in PPT$ s.t.

$\forall x \in LPr[M(x) = 1] > \frac{2}{3}$

$\forall x \notin LPr[M(x) = 0] < \frac{1}{3}$

We can run the machine M many times and output the majority of results.

Chernoff Bound

Let $x_1 ... x_n$ be independent random variables with identical probability distribution $0 \leq P \leq 1$.

Let $Y = \sum x$ then $Pr[|Y - E(Y)| > \delta * n] < 2 * e^{-\frac{\delta^2 * n}{2}}$.

Chebyshev Bound

Let $x_1 ... x_n$ be pariwise-independent random variables with identical probability distribution $0 \leq P \leq 1$.

Let $Y = \sum x$ then $Pr[|Y - E(Y)| > \delta * n] < \frac{1}{4 * \delta^2 * n}$

Neglible Function

A function f is neglible if:

$\forall c, \exists N_c$ s.t. $\forall n \geq N_c : f(n) < \frac{1}{n^c}$

Dot Product

$< a \cdot b >= (\sum x_i * p_i) \mod 2$ (bitmask a using mask b then reduction XOR)

One-Way Function

Informally

- Easy (polynomial-time) to compute
- Hard to invert

Formally

1) Challenger generages random x with length k, computes $f(x)$, and sends to Adversary

2) Adversary attempts to compute in polynomial-time x and makes guess y
3) Adversary wins if $f(x) = f(y)$

As k increases, probability of A winning is a negligible function

$\forall c, \forall A \in PPT, \exists N_c s.t. \forall |x| > N_c : Pr[A(f(x)) \in f^{-1}(f(x))] < \frac{1}{|x|^c}$

Hard-Core Bits

Suppose f is a one-way function. Suppose $y = f(x)$ and pick p.

If f is a one-way function, then calculating $< x \cdot p >$ is easy but finding $< y \cdot p > = < x \cdot p >$ is hard. Formally:

f is one-way $\rightarrow Pr[A(f(x), p) = < x \cdot p >] < \frac{1}{2} + \epsilon$

Therefore dot-product is a hard-core bit for any one way functions

Proof of Hard-Core Bits

By contrapositive, if an inverter exists for a hard-core bit, then it can invert the one-way function. Formally:

$Pr[A(f(x), p) = < x \cdot p >] > \frac{1}{2} + \epsilon \rightarrow A$ inverts f

Define $A(f(x), p) = < x \cdot p > = A$ predicts $< x \cdot p >$

Trivial Case

Suppose that adversary A predicts $< x \cdot p >$ with certainty. We can construct many $p = (0...1...0)$ which extracts a certain bit in x. We only need to do this for each bit in x.

Special Case

Suppose that adversary A predicts $< x \cdot p >$ with probability more than $\frac{3}{4} + \epsilon$. Pick p as a random string, then flip the ith bit to generate p'. If A is correct both times, then $< x \cdot p > \oplus < x \cdot p' > = x_i$ which occurs with probabilty slightly mroe than 50%. Therefore we can run this many times and take the majority.

General Case

Suppose that adversary A predicts $< x \cdot p >$ with probability more than $\frac{1}{2} + \epsilon$. Define Good $= \{x | Pr[A \text{ predicts } < x \cdot p >] > \frac{1}{2} + \frac{\epsilon}{2}\}$. We claim that at least $\frac{\epsilon}{2}$ x's are in Good.

Therefore:

$Pr[A \text{ predicts } < x \cdot p >] \leq$
$Pr[A \text{ predicts } < x \cdot p > | x \in Good] * Pr[x \in Good] +$
$Pr[A \text{ predicts } < x \cdot p > | x \notin Good] * Pr[x \notin Good]$

We note that by definition:

$Pr[A \text{ predicts} < x \cdot p > | x \in Good] = 1$
$Pr[x \in Good] = \frac{\epsilon}{2}$, $\setminus$ $Pr[A \text{ predicts} < x \cdot p > | x \notin Good]$
$Pr[A \text{ predicts} < x \cdot p > | x \notin Good] = \frac{1}{2}$
$Pr[x \in Good] = \frac{\epsilon}{2}$, $Pr[A \text{ predicts} < x \cdot p > | x \notin Good] = 1$

Therefore $Pr[A \text{ predicts} < x \cdot p >] \leq \frac{1}{2} + \frac{\epsilon}{2}$. This is a contradiction with our original assumption that A predicts $< x \cdot p >$ with probability more than $\frac{1}{2}$.

We can then create A to invert f by picking $p_1 ... p_{c*\log(n)}$ and guessing $b_1 =< p_1 \cdot x > ... b_{c*\log(n)} =< p_{c*\log(n)} \cdot x >$. The probability that all b's are guessed correctly is $\frac{1}{n^c}$. Compute $P =$ xor of all possible combinations of p's and $B =$ xor of all possible combinations of b's. Note that P and B must be correct with probability $\frac{1}{n^c}$ and are pairwise independent. For some fresh $< x \cdot p >$ we can calculate ith bit of x: $x^i = \text{majority}(A(p_k^i, x^i) \oplus b_k)$. We can check which p result in correct guesses, and the probability of getting x wrong decreases as a Chernoff bound.

Commitment Protocols

We want to commit some information without revealing the information, and we want a commitment to be binding without being able to change what was committed. We want to guarantee binding and hiding.

Binding: Sender cannot change their commitment after it is sent.

Hiding: Reciever learns nothing about the value of the commitment until it is uncommitted.

Committment Protocol using One-Way Permutation

We can construct a commitment protocol with security paramter k using a one-way function f that is a permutation (1-1 and onto). To commit a bit b, pick random x, p and send $f(x), p, < x \cdot p > \oplus b$. To decommit, sender can send x.

Binding: Perfect

Hiding: Computational

Pseudorandom Generator

Define $PRG(s) \rightarrow y$ for some small seed $|x| = k$ outputs a large random y which is indistinguishable from random.

y is random if there does not exist $M \in PPT$ such that it can predict that a y was generated by real random or PRG with probabilty greater than negligible.

Samplable Distributuions

A distribution D is samplable if $\exists S \in PPT, \forall x, Pr[x] = Pr(S \to x)$.

Indistinguishable

Two samplable distributions X, Y are polynomial time indistinguisable if $\forall c \forall J \in PPT, \exists N_c$ s.t. $\forall n > N_c$:

$|(Pr_X[J(x \in X)] = 1) - (Pr_Y[J(y \in Y) = 1)]| < \frac{1}{n^c}$

The difference in probabilities between whether J predicts 1 over the distributions X and Y are within marginal difference.

Next Bit Test

A PRG passes the Next Bit Test if a $J \in PPT$ which recieves a stream of PRG bits can guess the next bit with probability less than $\frac{1}{2} + \epsilon$.

Proving Indistinguishable $\to$ Passes Next Bit Test

If a J can distinguish a series of samples of random distributions X, Y with probability ϵ, then it can distinguish a single sample with probability $\frac{\epsilon}{m}$.

Assume that there exists some J' such that

$|(Pr_X[J'(x_1...x_n \in X)] = 1) - (Pr_Y[J'(y_1...y_n \in Y) = 1)]| > \epsilon(n)$

Consider "hybrids" P_j where the first j samples come from Y and the remaining samples come from X

$P_0 = Pr_{X,Y}(T'(x_1, x_2, ...x_n))$
$P_1 = Pr_{X,Y}(T'(y_1, x_2, ...x_n))$
$P_2 = Pr_{X,Y}(T'(y_1, y_2, ...x_n))$
...
$P_k = Pr_{X,Y}(T'(x_1, x_2, ...x_n))$

Therefore $P_0 - P_k > \epsilon(n)$ and $P_0 + (-P_1 + P_1 - P_2 + P_2 ... - P_{k-1} + P_{k-1} - P_k) > \epsilon(n)$
$(P_0 - P_1) + (P_1 - P_2) + ... + (P_{k-1} - P_k) > \epsilon(n)$

Because $P_0 - P_k > \epsilon(n)$ and there are k terms, then there must be some $P_{j-1} - P_j > \frac{\epsilon(n)}{k}$.

We can guess the value of j for which this is true. Then for each bit starting at k, we can guess a random bit and ask the judge whether it thinks it is more likely to be random or PRG. We can then use this to predict the next bit of the PRG. Therefore Indisguinshibility $\to$ Next Bit Test

Proving Passes Next Bit Test $\rightarrow$ Indistinguishable

Extending seed by 1 bit: Given a 1-way permutation f, then $PRG(x,p) \rightarrow f(x), p, < x \cdot p >$, we have extended $2n \rightarrow 2n+1$

Extending seed by n bits: We define the next bit test, where a PRG sends next random bits at a time to J and J predicts with probability $< \frac{1}{2} + \epsilon$.

We show that a PRG secure against the Next Bit Test $\rightarrow$ Indistinguishability. Assume f is a one-way permutation, then

$PRG(x,p) = (f(x) \rightarrow x_1, < x \cdot p >) \rightarrow (f(x_1) \rightarrow x_2, < x_1 \cdot p >) \rightarrow ... \rightarrow (f(x_{n-1}) \rightarrow x_n, p)$

and output $(p, < x_n \cdot p >, < x_{n-1} \cdot p >, ..., < x_1 \cdot p >)$. If J is able to predict the next bit, which is a hard-core bit of f, then it must be able to invert f.

Using PRG as a Commitment Protocol

Given a $PRG : \{0,1\}^n \rightarrow \{0,1\}^{3n}$ that is indistinguishable and bit b that committer wants to commit to reciever. The reciever sends a random value Y where $|Y| = 3n$. The commiter picks random seed s and computes $PRG(s) = X$, if $b = 0$ then committer sends X, otherwise committer sends $X \oplus Y$. To open, send s.

Hiding: If the reciever can distinguish X and $X \oplus Y$, then reciever can distinguish pseudorandom from random.

Binding: Suppose there are $s_1, s_2 \rightarrow PRG \rightarrow x_1, x_2$ such that $x_1 \oplus x_2 = Y$. If the committer wants to open a 0, committer sends s_1, otherwise send s_2. For some Y that the reciever picks, there are 2^{2n} possible combinations of s_1, s_2 and 2^{3n} possible Y. Therefore the probability that the committer cna find s_1, s_2 for some Y is $\frac{2^2 n}{2^3 n} = \frac{1}{2^n}$.

Pseudorandom Functions

Idea: We want a PRG with random access.

Let a $PRG(s = \{0,1\}^n) \rightarrow \{0,1\}^{2^n}$, we define a $PRF(s, ADDR) \rightarrow \{0,1\}$ which samples a single bit among a polynomial number of addresses from the PRG in polynomial time.

Security: Assume $J \in PPT$. Given a stream of bits, J is unable to distinguish if the bits come from a PRF or a random string.

Constructing PRF from PRG

We construct $PRF(s, ADDRR)$. Given a $PRG(s \in \{0,1\}^n) \rightarrow x_0 \in \{0,1\}^n | x_1 \in \{0,1\}^n$, then we construct a tree with root node s and leaves x_0, x_1. We can continue to build a tree by inputing x_0 and x_1 as seeds to the PRG

and generating $x_{00}, x_{01}, x_{10}, x_{11}$. We define the PRF as the traversal down the binary tree using the $ADDR$ and output the leaf node it reaches.

Proving PRF Security

If $\exists$ distinguisher for PRF, then $\exists$ extended distinguisher for PRG.

Extended PRG Indistinguishability: Given $s_1...s_n \rightarrow PRG(s_1)...PRG(s_n)$, a $J \in PPT$ cannot distinguish the PRG outputs from random. By hybrid argument, we can show that if such a distinguisher exists, then we can construct a distinguisher for a single s.

Suppose there exists a $J \in PPT$ that can distinguish between a PRF tree and a random binary tree with probability $> \epsilon(n)$. Construct the hybrids where the top j levels are from random and the bottom $n-j$ levels are from the PRF. By hybrid argument, there are two trees such that J can distinguish with probability $> \frac{\epsilon(n)}{k}$. We can select a level in which one hybrid is from PRF but from random in the other. Since J can distinguish between the two trees, then we can use this to judge whether this level is from the PRG or from random. Therefore J contradicts the extended PRG indistinguisability.

Public Key Encryption

Idea: generate a public key which can be publicly avaliable, and a private key that is secret

For some random r, $Gen(1^k, r) \rightarrow (pk, sk)$

Signature Scheme

For some document:

$Sign(sk, D) \rightarrow \sigma_{pk,D}$

$Verify(pk, D, \sigma_{pk,D}) \rightarrow True/False$

Correctness: Verify outputs True for all D signed with sk corresponding with pk.

Unforgeability: A challenger recieves $m_1...m_k$ from adversary and sends signatures of $m_1...m_k$ to adversary. The adversary cannot find $m' \neq m_1...m_k$ such that $Verify(pk, m') \rightarrow True$.

Creating a Signature Scheme - Lamport

Assume we have a 1-way function f. We can generate secret keys x_0, x_1 and calcualte public keys $f(x_0) = y_0, f(x_1) = y_1$. To sign a 0, we show x_0, and to

sign a 1 we show x_1. To verify, we show that the signed x corresponds with the $y = f(x)$ that was signed.

Proof of unforgeability: Assume an adversary can forge a signature with non-neligible probability. Suppose we generate some z and repalce y_0 with z. With probability $\frac{1}{2}$, the adversary will forge z by finding some x_z such that $f(x_z) = z$. However since we did not know this x_z, then the adversary has just inverted f, thus a contradiction.

Exting to n bits: we can generate multiple x_0 and x_1 for each bit position we want to sign.

However, we want to accomplish this wihtout arbitrarily large keys.

Using Collision Resistant Hash Functions

We can use a collision resistant hash function to reduce the size of the document, then sign the hash of the document. If a valid signature exists for a different document, then there must have been a collision in the hash function.

Collision resistant Hash Functions

A hash function h is collison resistant if an adversary cannot find x_1, x_2 such that $h(x_1) = h(x_2)$.

U1WHF: A hash function h is universal and 1-way if an adversary cannot find x_2 given x_1 such that $h(x_1) = h(x_2)$

Constructing A 1-way Universal hash Function

Let $h(a, b, x) =$ delete first bit $(ax + b)$

Let f be a 1-way permutation $f(\{0,1\}^n) \to \{0,1\}^n$.

We can create a universal 1-way hash function as U1WHF$(x) = h(f(x))$

Discrete Log

Given $h = g^x \mod p$, it is believed to be hard to find x given h.

Eliptic Curves

Given curve of form $ax^3 + bx + c = 0$, we iteratively sample two points and find a thrid that is colinear, then reflect that over x axis. Keep iterating using this protocol many times

Public Key Encryption

Diffe-Hellman

Alice and Bob agrees on $g, p \in Z^p$

Alice picks random x, Bob picks random y

Key Exchange: Alice sends $g^x \mod p$, Bob sends $g^y \mod p$. Both can compute $g^{xy} \mod p$ as a shared secret.

Alice's secret key $sk = x$ and can publish the public key $pk = (h = g^x \mod p, p, g)$

Encrypt message m and random r, we can send $Enc(m, r) = (g^r \mod p, h^r * m \mod p) = c$

Decrypt message m using r, we can send $Dec(u, v) = (\frac{v}{u^x} \mod p)$

Security

Indistinguishability:

A Challenger generates pk, sk and sends pk to Adversary. Adversary chooses two different messages m_1, m_2 and sends to the Challenger to encrypt using sk. Adversary wins if they can distinguish which message m_1 or m_2 was encrypted with probability more than $\frac{1}{2} + \epsilon$.

DDH Indistinguishable

The distributions $(g, g^a, g^b, g^c...)$ and $(g, g^a, g^b, g^{ab}, ...)$ are indistinguishable from each other.

Semantic

Suppose a series of messages come from a samplable distribution D. Challenger draws a message m from D.

Adversary must predict any $f(m)$ given only D or given D and $Enc(m)$ with negligibly similar probability.

Lunchtime Attacks (CCA-1)

The challenger generates sk, pk and sends pk to Adversary. Adversary sends $c_1...c_n$ ciphertexts **nonadaptively or adaptively** to the Challenger to decrypt. Adversary sends $m_0 \neq m_1$ and Challenger encrypts m_0, m_1. Given the chosen ciphertexts and known decryptions, the Adversary is not able to distinguish between the encryptions of m_0, m_1.

CCA-2

The challenger generates sk, pk and sends pk to Adversary. The adversary is allowed to generate decryptions of any number of ciphertexts $c_1 ... c_n$ adaptively. Challenger encrypts m_0, m_1 as before and the adversary is able to ask for decryptions adaptively as long as they are not for m_0 or m_1. The Adversary wins if they are able to distinguish between the two messages with probability $> \frac{1}{2} + \epsilon$.

Constructing Non-Adaptive CCA-1 Encryption (Cramer - Shoup)

CS-Light: Pick $sk = x, y, a, b$, pk=$g_1, g_2, h = g_1^x * g_2^y, c = g_1^a * g_2^b$. - $Enc(m, r) = (g_1^r, g_2^r, h^r * m, c^r)$ - $Dec(u, v, e, w) = $ - Check if not $w = u^a * v^b$ then output fail - output $\frac{e}{u^x * v^y}$

Proof of Non-Adaptive CCA-1 Security

Assume towards contradiction that $\exists A \in PPT$ that can distinguish two encrypted messages m_0, m_1 with $Pr > \frac{1}{2} + \epsilon$. We show that the probability A can distinguish ciphertexts made with DDH tuples $\{g_1, g_2, g_3, g_4, ...\} = \{g_1, g_1^a, g_1^b, g_1^{ab}, ...\}$ versus from random $\{g_1, g_2, g_3, g_4, ...\} \in R$ has a gap in probability.

Challenger computes (x, y, a, b), g_1, g_2, h, c as before. Challenger decrypts the non-adaptive $c_1, ..., c_n$ ciphertexts honestly. When the Adversary presents m_0, m_1, we instead send $(g_3, g_4, g_3^x * g_4^y * m, g_3^a * g_4^b)$.

This is a valid decryption because:

$ u = g_3 = g_1^b, \ v = g_4 = g_1^{ab}, \ e = g_1^{bx} g_1^{aby} \ m, \ w = g_1^{ba} * g_1^{abb} $.

Since $w = g_1^{aa} * g_1^{abb} = u^a * v^b$ and $\frac{e}{u^x * v^y} = \frac{g_1^{bx} * g_1^{aby} * m}{g_1^{bx} * g_1^{aby}} = m$. if we consider $b = r$ then we see that $g_3 = g_1^r$ and $g_4 = g_2^r$. Therefore A's advantage in distinguishing the ciphertexts is the same as in distinguishing DDH tuples.

However, if $\{g_1, g_2, g_3, g_4, ...\} \in R$. We run the protocol as before. When the Adversary sends (u, v, e, w):

Case 1: $u = g_1^r, v = g_2^r \rightarrow log_{g_1}(u) = log_{g_2}(v) = r$

Define $\alpha = log_{g_1}(g_2)$ then Adversary learns

$m = \frac{e}{u^x * v^y} \rightarrow log_{g_1}(m) = log_{g_1}(e) - x * log_{g_1}(u) - y * log_{g_1}(v)$
$= log_{g_1}(e) - x * log_{g_1} - y * \alpha * log_{g_2}(v)$
$= log_{g_1}(e) - r * (x + \alpha * y)$

Then the Adversary learns the realtion $x + \alpha * y$. This does not leak any new information about the private key because we could have computed $log_{g_1} h = x + \alpha * y$. Therefore the Adversary does not gain any additional information and distinguishes the two ciphertexts with $Pr = \frac{1}{2}$.

Case 2: $log_{g_1}(u) = r \neq log_{g_2}(v) = r'$

Then the adversary's chances of finding w that for which the decruption will succeed is negligible. Recall:

$c = g_1^a * g_2^b \rightarrow log_{g_1}(c) = a + \alpha * b$
$w = u^a * v^b \rightarrow log_{g_1}(w) = a * r + \alpha * b * r'$

Note both equations are linearly independent. When the decryption fails, then Adversary can only eliminate one (a, b) pair for each fail. Therefore because the Adversary is limited to a polynomial number of ciphertext, then the adversary cannot find a, b

Construction CCA-2 Secure Encryption

CCA-2 CS-Full: Pick $sk = x, y, a, b, a', b'$, pk=$g_1, g_2, h = g_1^x * g_2^y, c = g_1^a * g_2^b, d = g_1^{a'} * g_2^{b'}$. Also publish a collision resistant hash function f.

- Define $\beta = f(g_1^r, g_2^r, h^r * m)$, $Enc(m, r) = (g_1^r, g_2^r, h^r * m, (c * d^\beta)^r)$
- \$Dec(u, v, e, w) = \$
  - Check if not $w = u^{a+\beta*a'} * v^{b+\beta*b'}$ then output fail
  - else output $\frac{e}{u^x * v^y}$

Proof of CCA-2 Security

Assume towards contradiction that $\exists A \in PPT$ that can distinguish two encrypted messages m_0, m_1 with $Pr > \frac{1}{2} + \epsilon$. We show that the probability A can distinguish ciphertexts made with DDH tuples $\{g_1, g_2, g_3, g_4, ...\} = \{g_1, g_1^a, g_1^b, g_1^{ab}, ...\}$ versus from random $\{g_1, g_2, g_3, g_4, ...\} \in R$ has a gap in probability.

Challenger computes (x, y, a, b, a', b'), (g_1, g_2, h, c, d) as before. Challenger decrypts the non-adaptive $c_1, ..., c_n$ ciphertexts honestly. When the Adversary presents m_0, m_1, we instead send $(g_3, g_4, g_3^x * g_4^y * m, g_3^{a+\beta*a'} * g_4^{b+\beta*b'})$.

Case 1 $u = g_1^r, v = g_2^r \rightarrow log_{g_1}(u) = log_{g_2}(v) = r$

If the tuples come from DDH, then the Adversary's advantage is the same as before and the Adversary does not learn any additional information.

Case 2: $log_{g_1}(u) = r \neq log_{g_2}(v) = r'$

The adversary learns that :

$c = g_1^a * g_2^b \rightarrow log_{g_1}(c) = a + e * b$

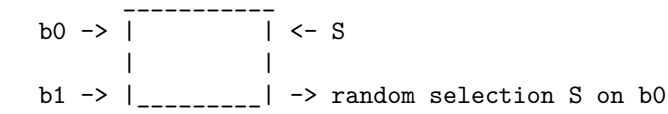
$log_{g_1}(d) = a' + e * \beta$

$log_{g_1}(w) = r * (a + \beta * a') + r'(b + \beta * b')$

During the second round of decryptions when Adversary sends (u, v, e, w). If $u = u*, v = v*, \alpha = \alpha*$ but $w \neq w*$ then it will fail with large probability. Otherwise, if the hashes collide $f(w*) = f(w)$, then you have found a collision in a collision resistant hash function. Therefore the Adversary learns nothing about the message.

Secure Multiparty Computation

Given 1-out-of-2 Oblivious Transfer. If A sends b_0, b_1 then B secretely recieves random selection S on b_0 or b_1 without knowing either input.



We can compute AND by feeding in $b_0 = x \cdot 0, b_1 = x \cdot 1$ and $S = y$, the output is therefore $x \cdot y$ which is AND. However, since B knows y then they can know the value of x if $y = 1$.

Additive Secret Sharing

Suppose A and B want to compute a function on their secret input x and y. For each bit of x, A picks $x_0 \oplus x_1 = x$, keeps x_0 and sends x_1 to B. B picks $y_0 \oplus y_1 = y$, keeys y_1 and sends y_0 to A. A has x_0, y_0 and B has x_1, y_1.

To compute $x \oplus y$, compute $(x_0 \oplus y_0) \oplus (x_1 \oplus y_1)$. Therefore, A and B can both compute XOR on their shares, then send their results to each other and reconstruct the result.

To compute $x \cdot y$, note that $x \cdot y = (x_0 \oplus x_1) \cdot (y_0 \oplus y_1) = x_0 \cdot y_0 \oplus x_0 \cdot y_1 \oplus x_1 \cdot y_0 \oplus x_1 \cdot y_1$. A can compute $x_0 \cdot y_0$ and B can compute $x_1 \cdot y_1$. To exchange the missing intermediate terms, A picks random r and using 1-2-OT sends $r \oplus x_0 \cdot y_1$ to B without learning y_1. B can use the same mechanism and random s to send $s \oplus x_1 \cdot y_0$ to A.

Therefore, $x \cdot y = (x_0 \cdot y_0 \oplus r \oplus s \oplus x_1 \cdot y_0) \oplus (x_1 \cdot y_1 \oplus r \oplus s \oplus x_0 \cdot y_1)$. Since A and B have each secret for their part then can send the result of each part and compute the final result.

Protectionn Against Collusion

GMW: We can extend the 1-2-OT protocol where each party generates $r_1...r_k$ for each of the other k parties. To compute, each party may need to engage in 1-2-OT protocol to determine intermediate results individually with each other party. This is resilient to $n-1$ collusion, but does not allow any parties to drop out before the protocol is complete.

BGW: We can use a Lagrange sum to build polynomials $f(x) = y_1 * L_1(x) + ... + y_n * L_n(x)$. For k parties, each party with secret input s can generate a random Lagrange polynomial $f(x) = s + ax + bx^2 + cx^3 + ...$ with degree k and send $f(n)$ to party n for $n > 0$ (including themselves). Therefore $f(0) = s$ but no other party knows each other's $f(0)$.

To compute $\alpha + \beta$ given secret shares $f(x) = \alpha + ...$, $g(x) = \beta + ...$ A computes $f(1) + g(1)$ and B computes $f(2) + g(2)$. Both parties can release their shares and they can recompute the Lagrange sum from the two points.

To compute $\alpha * \beta$ given secret shares $f(x) = \alpha + ...$, $g(x) = \beta + ...$ A computes $y_1 = f(1) * g(2)$ and B computes $y_2 = f(2) * g(2)$. Therefore A can pick a new polynomial Q and compute $y_1 * L_1(0) + Q(x)$ and sends to each party for $x = 1...k$. B picks R and computes $y_2 * L_2(0) + Q(x)$. Then all parties sum their shares. Since multiplication doubles the degree of polynomials, then if we start with degree $\frac{n}{k}$ then after multiplication the polynomial has degree $\frac{2n}{k}$ and therefore we only need that many parties to finish the computation.

A majority of colluding parties can reconstruct the polynomial from their shares. A majority of colluding parties can reconstruct the polynomial from their shares.

Yao's Garbled Circuits

Suppose two parties want to compute $f(x = x_0|x_1) = y$. We assign one party is the Garbler and one is the Evaluator.

$Garble(f, r) \to \hat{f}$

$Garble(x, r) \to \hat{x}$

Such that $\hat{f}(\hat{x}) = y$. The evaluator can then garble $y \to \hat{y}$ using ob oblivious transfer.

We say that it is secure if $\forall x_1, x_2 s.t. f(x_1) = f(x_2)$ then $\hat{f}(\hat{x_1}) \approx \hat{f}(\hat{x_2})$ are computationally indistinguishable. There exists a simulator that garbles a different circuit such that the truth table distributions are indistinguishable.

For AND gate:

Consider private key encryption $E_k(m) = c$. Garbler can generate random keys k_0, k_1, labels b_0, b_1, and c_0, c_1. Garbler can encrypt

$E_{k_0}(E_{b_0}(c_0)), E_{k_0}(E_{b_1}(c_0)), E_{k_1}(E_{b_0}(c_0)), E_{k_1}(E_{b_1}(c_1))$. Evaluator can evaluate inputs to the AND gate as k_0 or k_1 on one wire and b_0 or b_1 on the other. Evaluator will get the key pair to decrypt one of the encryptions on c_0 or c_1.

For OR gate:

Similar idea, create encryption of results and store in a truth table. Encrypt inputs and evaluator is able to decrypt one value of the truth table.

Proof of Security

Assume that an adversary can distinguish one circuit from another on the same input. Therefore there is at least one gate where the adversary is able to distinguish the two circuits. Since the output of the gate is a decryption of a key in the truth table of the gate. Therefore if the adversary is able to distinguis the circuits, then adversary must be able to distinguish between encryptions which therefore breaks the encryption.

Optimizations

Question: how many random bits required to compute XOR/AND

XOR: 1 random bit

- Suppose parties $p_1...p_n$ want to compute $\oplus(b_1...b_n)$. p_1 can compute $b_1 \oplus r \to m$ and send to p_2. p_2 computes $m \oplus b_2 \to m$ etc. So to compute XOR, you only need 1 bit.

AND: ~5 random bits, proof that it needs more than 2 random bit

- If parties can compute f 1-privately using constant random bits, then f has a linear size circuit.

1-way Trapdoor Permutation Family

Given one-way permutation f, we build a trapdoor f^{-1} which can invert the function that can invert f in polynomial time.

Constructing Oblicious Transfer Protocol

Rabin-OT: 1/2 chance to recieve bit otherwise get nothing.

Sender generates $3n$ bits and sends using Rabin-OT. By Chernoff bound, Reciever has a marginal chance of recieving less than n bits and the probability of recieving more than $2n$ bits is also marginal. Reciever selects s_0, s_1 mutually disjoint sets of size n bits. Sender sends $b_0 \oplus s_0$ and $b_1 \oplus s_1$.

PIR

Idea: want to retrieve data from a database without server(s) knowing which position was retrieved.

CGKS Two-Server

Two servers which promise not to collude with database size n.

Database: matrix $DB_1 : \sqrt{n} \times \sqrt{n}$, $DB_2 : \sqrt{n} \times \sqrt{n}$.

Suppose a user wants item i, j from the database. User picks random $r : \sqrt{n}$ and sends to DB_1. DB_1 XORs columns where $r_i = 1$ to get y_1. User generates r' by flipping bit i in r and sends to DB_2 which generates y_2 User can construct column i by computing $y_1 \oplus y_2$.

Reducing communication to $\sqrt[3]{n}$

By adding additional servers, we can reduce the communication cost of the protocol. The user wants to retrieve i, j. User picks r_1, r_2 and DB_1 database sends the XOR of cells with $x_1 = 1, x_2 = 1$. User sends to $DB_2 : r'_1, r_2$, $DB_3 : r_1, r'_2$, $DB_4 : r'_1, r'_2$ by flipping the row, column, and both. The XOR of all 4 values is the value of i, j.

We can extend this principle to 3 dimensions and 8 servers. The user wants to retrieve point i, j, k and sends random x, y, z to DB_1. The user then flips the i, j, k bits combinationally for each other database. XOR of all these values will be the value.

We can collapse the databases by only using DB_1 and $DB - 8$. DB_1 recieves x, y, z and DB_8 recieves x^i, y^j, z^k. Note that DB_1 can simulate $DB_2...DB_4$ by toggling x, y, z for every possible i, j, k. DB_8 can simulate $DB_5..DB_7$ by untoggling x^i, y^j, z^k for every possible i, j, k. Therefore the communication overhead is $\sqrt[3]{n}$ using 2 servers.

Locally Decodable Codes

In a general PIR scheme, user asks q_1 to DB_1 and q_2 to DB_2 and recieves a_1, a_2

Suppose we generate the answers for possible queries for q_1, q_2. Even if an adversary corrupts a constant portion of the database, there is still a large constant probability that a specific point is recoverable by reading two positions. We dont have to generate the entire code to recover one position.

Single Database Implementation

Given a database $x_1...x_n$ and assume additively homorphic encryption such that $E(x) \oplus E(y) = E(x + y)$. The user generates query vector $q = 0000...1....000$

and encrypts each bit. The database adds an encryption of the database to the encryption of the query and returns to the user.

Offline & Online System

Offline: User sends $\sqrt{n}$ set of queries with length $\sqrt{n}$ to DB_1. Server sends XOR of each set.

Online : User picks a set of queries s_i that contains i to retrieve. Send $s_i - i$ to DB_2 abd returns the XOR.

User can recover i by subtraction both results.

Oblivious RAM

Idea: We want to create a system that hides the patterns of read and write accesses.

User generates a random permutation π using a PRF of the data entry locations. When the user wants to do a read, they compute the permutation and get the lcoation. This works as long as the user only reads a location at most once.

Extending More than 1 Read

User stores a local cache and stores read values to that location. On a subsequent read, the user can return the local cache value. When the local cache runs out, we can shuffle the remote database with the local cache and pick a new permuation.

Use a sorting network:

- Using a compatator gate: $c(a, b) -> max(a, b), min(a, b)$
- The sorting network sorts any set of input integers

We can obliviously sort the database by using the sorting network to sort against random indices.

Secure Indirect Addressing

Indirect addressing: We can use a hash function to take an address and output a randomized address. we can then use a hash table to store the values. Each bin in the hash table will have on average $O(logn)$ values.

To implement secure indirect addressing, we can use a PRF to randomize addresses. We can again construct a hash table.

We can also move the user's local cache to the remote database since it needs to be scanned fully anyways. This allows the user to have constant memory.

Heirarchical ORAM - Poly Log Overhead

We can reduce the $\sqrt{n}$ cache by having a heiarchy of caches starting from the topmost, smallest buffer down to the final hashtable. Each level of cache has its own PRF. For each read, we compute the PRF for each level and search that level for the item. If it is found, then we continue down searching random values, and cache it into the topmost buffer. When each level fills, we shuffle it with the lower level.

Cuckoo Hash Table

Instead of making a hash table with many buckets, we can use two hash functions h_1, h_2. If there are collisions with h_1 we can evict the existing element to a new location using h_2. We can use the cuckoo hash tables to replace the heiarchical levels in Heierarchical ORAM. However, this leaks information because for small levels, certain keys are incompatible becuaseh they would create a cycle of evictions. Therefore, we need to use a stash to put those items when that happens.

Panorama / Optorama

We can shuffle together two adjacent levels of the Heirarchical ORAM using $\log(n)$ size sorting network.