CSCI 5440: Cryptography
The Chinese University of Hong Kong, Fall 2020

Lecture 11
17 and 23 November 2020

The motivating application for fully homomorphic encryption was secure outsourcing of computation: A protocol by which a weak Alice gets a powerful Bob to carry out a computation for her without revealing information about her inputs. In short homomorphic encryption enables privacy of outsourced computation. This lecture is about integrity of outsourced computation: How can Bob convince Alice that the answer he calculated is the correct one?

1 The Lund-Fortnow-Karloff-Nisan (LFKN) protocol

Suppose Alice wants to count the *number* of valid 3-colorings of some n-vertex graph G. In general this is a brute force-type problem: Alice has no better recourse than trying out all 3^n colorings and tracking how many of them are valid. When n is in the fourties this type of work may be too intensive for Alice but quite feasible for Bob's powerful server. Alice, however, wants to be convinced that Bob did the work and didn't just give her some random answer.

Counting colorings may sound esoteric, but it turns out to be closely related to certain calculations in thermodynamics. It is in fact representative of a large class of computations. We'll say more about this shortly.

Remember that a 3-coloring is valid if the endpoints of all edges have different colors. The first step is to describe the test for a valid coloring by an arithmetic circuit. Let's represent the three colors by the integers -1, 0, and 1: Variable x_u takes value -1 if vertex u is colored red, 0 if green, 1 if blue. The condition "u and v have different colors" then says that $x_u - x_v$ takes a nonzero value. The polynomial

$$p(x_u, x_v) = 1 - \frac{((x_u - x_v)^2 - 4)((x_u - x_v)^2 - 1)}{4} = \begin{cases} 1, & \text{if } x_u = x_v, \\ 0, & \text{if not} \end{cases}$$

indicates if this condition holds. The product polynomial

$$P(x_1, \dots, x_n) = \prod_{\text{edges } \{u,v\}} p(x_u, x_v)$$

indicates the validity of the coloring represented by $x = (x_1, \dots, x_n)$: It evaluates to 1 when the coloring is valid and to 0 when it is not. For a graph with m edges P is a polynomial of degree $4m$.

The number of valid 3-colorings of the graph equals the value of the sum

$$S = \sum_{x_1,\dots,x_n \in \{-1,0,1\}} P(x_1, \dots, x_n). \tag{1}$$

The LFKN protocol is designed to certify the values of sums of this form. First the colors $\{-1, 0, 1\}$ are embedded in a finite field $\mathbb{F}_q$ for a sufficiently large prime number q. The main component of the LFKN protocol is this subprotocol for eliminating the first variable:

1. Prover sends the univariate polynomial $r(x_1) = \sum_{x_2,\dots,x_n \in \{-1,0,1\}} P(x_1, \dots, x_n)$ described by its $4m + 1$ coefficients in $\mathbb{F}_q$.

2. Verifier checks that $r(-1)+r(0)+r(1)=S$, then chooses a random $a_1 \sim \mathbb{F}_q$ and asks for a proof that
$$r(a_1) = \sum_{x_2,\dots,x_n \in \{-1,0,1\}} P(a_1,\dots,x_n). \tag{2}$$

This is repeated n times until all variables are eliminated. At this point the verifier the summation has vanished and Verifier needs to be convinced that $P(a_1,\dots,a_n)$ is the value claimed by Prover in the last round. This amounts to evaluating $P(a_1,\dots,a_n)$, which Verifier can do on his own. Verifier accepts if this last evaluation checks out.

If Prover's claim (1) was correct then Verifier accepts with probability 1. We will now prove that if claim (1) does not hold, then Verifier is likely to reject after interacting with a cheating prover. The proof is surprisingly short. The main tool is this lemma which analyzes the soundness of the variable elimination protocol.

Lemma 1. *If claim* (1) *is violated modulo q, then for any polynomial r^* sent by the prover, either Verifier rejects or claim* (2) *is violated for $r^*(a_1)$ except with probability at most $4m/q$.*

Proof. We prove the contrapositive. Suppose claim (2) is correct with probability more than $4m/q$. This means
$$r(x_1) = \sum_{x_2,\dots,x_n \in \{-1,0,1\}} P(x_1,\dots,x_n)$$
for at least $4m+1$ values of x_1. Both sides are univariate polynomials of degree $4m$, and they agree on $4m+1$ values. Since a polynomial of degree d is fully determined by any $d+1$ of its values, the two polynomials must be the same polynomial. So $r(-1)+r(0)+r(1)=S$ implies that
$$\sum_{x_1 \in \{-1,0,1\}} \sum_{x_2,\dots,x_n \in \{-1,0,1\}} P(x_1,\dots,x_n) = S$$
and claim (1) must be true. □

Theorem 2. *If claim* (1) *is violated modulo q then the verifier in the LFKN protocol rejects any cheating prover with probability at least $4mn/q$.*

Proof. By Lemma 1 and a union bound, the probability that Verifier asks for a proof of a correct claim at any point in the protocol is at most $4mn/q$. With the remaining probability, Verifier's final evaluation rejects. □

If q is a prime larger than 3^n then the number of colorings does not change modulo q, so the validity of claim (1) modulo q and over the integers is the same.

Nisan's protocol is very efficient from the verifier's perspective. The verifier needs to evaluate n degree-$4m$ polynomials three times and evaluate P once, which can be done in $O(mn)$ arithmetic operations modulo q. The prover, on the other hand, has to calculate several sums like (1), which requires $O(m3^n)$ arithmetic operations. This is not much larger than the time it takes to count the number of valid 3-colorings by brute force.

The LFKN protocol can more generally be used to certify counts of discrete configurations of any type, as long as the validity of a configuration can be efficiently checked. It was generalized by Shamir to certify the outcome of any long computation that uses a bounded amount of memory. Once it is "compiled" to a sufficiently simple model, the outcome of a computation that takes m bits of memory and runs in time t can be verified in time proportional to $m \cdot \log t$ (but the prover's running time may be as large as exponential in m). The remarkable history of these protocols, which were invented in the early days of email, is described in Babai's entertaining article *Email and the unexpected power of interaction.*

2 Modeling general computations

The verifier's complexity in the LFKN protocol grows linearly with the amount of memory required by the computation so the protocol is unsuitable for computations that require a lot of memory.

In a general delegation scenario Prover needs to convince Verifier that some program was correctly executed on Verifier's input. As usual we would like to represent the execution of this program by a circuit. So far in the course we were unconcerned about how this conversion of programs into circuits works, but to obtain efficient delegation protocols it is important that the resulting circuit has a special structure that we now describe.

Recall that a circuit is nothing more than a directed acyclic graph whose internal nodes are labeled by gates. Let's assume for simplicity that we only NAND gates with two inputs and one output are allowed so that the circuit is completely specified by the graph structure. When the computation represented by the circuit is time-consuming the circuit graph is proportionately large.

However, for circuits that represent the execution of relatively short computer programs the *adjacency predicate* of the graph can be computed very efficiently. If a circuit has and size s then its underlying graph has s nodes.[1] A node in the graph can therefore be indexed by a string of $\lceil \log s \rceil$ bits. The structure of the graph is specified by the adjacency predicate $A(u, v)$ that tells if there is an edge from node u to node v. For example, for the circuit in Figure 1 (a), $A(1, 5)$ and $A(2, 4)$ are true while $A(1, 4)$ is false.

The adjacency predicate itself can be represented by a circuit A that takes $2\lceil \log s \rceil$ bits of input and produces one bit of output. Computational complexity theory tells us that A is a very efficient circuit: Its size $O(\log s)$, i.e. it is linear in the length of its inputs x and y.

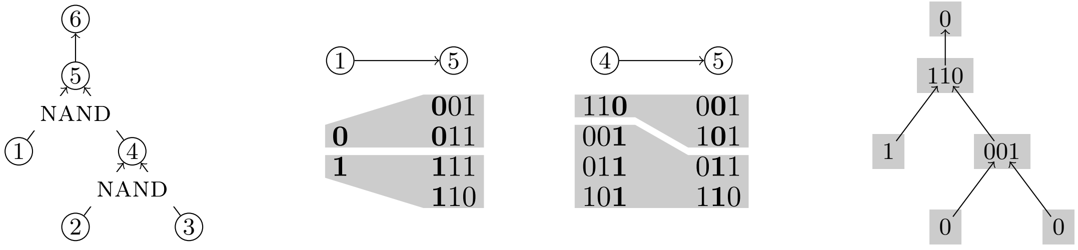

Figure 1: A (a) NAND circuit graph, (b) example coloring constraints for adjacent nodes, and (c) a coloring induced by circuit evaluation.

The validity of a circuit evaluation can be represented as a type of graph coloring problem. The colors of the nodes represent assignments to their incident wires. Adjacent vertices must assign colors that are consistent with respect to their shared wire.

Here is a specific implementation. The possible colors of the input and output nodes are the values 0 and 1 that the circuit inputs and outputs may take. The colors of the internal nodes represent the possible assignments to their input and output wires. There are four possible colors: 001, 010, 100, and 110. The intended meaning of color abc is that if the left input wire has value a and the right input wire has value b then the output wire has value c. If two vertices share a wire then the value among this wire must be consistent.

[1] We subsume the input and output nodes in the size in this discussion.

In Figure 1, vertices 1 and 5 could be colored $C(1) = 1$ and $C(5) = 111$ because their shared wire could take value 1, but they cannot be colored $C(1) = 0$ and $C(5) = 110$ because there is no consistent assignment to their shared wire. Similarly, vertices 4 and 5 might take colors $C(4) = 110$ and $C(5) = 101$ (their shared wire takes value zero), but not $C(4) = 011$ and $C(5) = 001$.

Certifying the correctness of a computation amounts to proving the existence of a color assignment to the internal nodes of the underlying circuit that satisfies the all the coloring constraints. For example, the assignment $C(4) = 001$ and $C(5) = 110$ in Figure 1 (c) is a proof that the circuit evaluates to $C(6) = 0$ on input $C(1) = 1$, $C(2) = 0$, $C(3) = 0$.

In summary, verifying large computations can be reduced to proving that some large graph whose adjacency predicate can be computed very efficiently can be properly "colored." The coloring constraints we saw are more general than those for graph 3-colorability but it turns out that this is not an important distinction.

With some additional manipulation it is possible to arrive at the this formulation: Prover and Verifier are given a 2^n-vertex graph specified by its adjacency predicate A of size $O(n)$. In addition, the honest Prover knows a valid 3-coloring of the graph. The objective is for Prover to convince Verifier that the graph is 3-colorable. The catch is that the size of Verifier is much smaller than the size of the graph: The intention is for the verifier to run in time polynomial in n even though the graph and its coloring have size exponential in n.

3 Certifying general computations

The statement "C is a valid 3-coloring of the graph" can be represented by the equation

$$\sum_{u,v \in \{0,1\}^n} \big((C(u) - C(v))^2 - 1\big)\big((C(u) - C(v))^2 - 4\big) \cdot A(u, v) = 0, \tag{3}$$

where u and v range over all 2^n vertices of the graph.[2] When u and v are not adjacent, $A(u, v)$ is zero. When u and v are adjacent, the other term vanishes exactly when the colors of u and v are different, assuming colors are represented by the integers -1, 0, 1. So the summation vanishes exactly when C is a valid 3-coloring.

This algebraic representation suggests that the LFKN protocol may be applicable in this (more general) setting as well. The complexity of the LFKN protocol is governed by the degree of the polynomial described by the expression under the sum, which in turn is determined by the degree of the functions A and C. The adjacency predicate A has size linear in n, so viewed as an arithmetic circuit it computes a polynomial of degree $O(n)$.[3]

Regarding C it is not a priori clear what its degree might be. In general C should be able to represent any candidate coloring of the graph's vertices, that is any function from the vertices $\{0, 1\}^n$ to the three colors $\{-1, 0, 1\}$. It turns out that such a function can always be described by a *multilinear polynomial* whose degree is at most n.

Instead of describing the general procedure let's work out an example in the case $n = 2$. Suppose the intended coloring is $C(00) = 0$, $C(01) = 1$, $C(10) = -1$, $C(11) = 1$. We intend to specify this coloring by a polynomial of the form

$$C(xy) = a + bx + cy + dxy.$$

[2] The modulus q needs to be sufficiently large, say $q > 4 \cdot 2^n$ suffices.

[3] The relevant parameter is in fact the depth. In general, a Boolean circuit of depth d can have degree as large as 2^d, but it turns out that A can be implemented in depth $\log n + O(1)$.

The color assignment then specifies the system of equations

$$\begin{aligned} 0 &= C(00) = a \\ 1 &= C(01) = a + c \\ -1 &= C(10) = a + b \\ 1 &= C(11) = a + b + c + d. \end{aligned}$$

By successive elimination we find that $a = 0$, $b = -1$, $c = 1$, and $d = 1$, so $C(xy) = -x + y + xy$.

In general, given the coloring C, the unique multilinear polynomial that represents it can be computed by solving the corresponding system of equations. This can be done in time $O(n2^n)$ for an 2^n-vertex graph.

In conclusion, both the coloring C and the adjacency circuit A can be represented by polynomials of degree linear in n (i.e. logarithmic in the graph size), so it looks like certifying claim (3) should be amenable to the LFKN protocol.

There is however a significant complication here: Verifier does not know the coloring C. It might very well be the case that C cannot even be represented by a small circuit that Verifier can evaluate on his own. Kilian, using a protocol of Babai, Fortnow, and Lund, suggested outsourcing the evaluation of C to Prover along these lines:

1. Prover sends a succinct (Merkle tree) commitment of the "database" C to Verifier.
2. Verifier and Prover run the LFKN protocol for claim (3).
3. In the last step of the LFKN protocol, Verifier needs to validate a claim of the form
$$\big((C(a) - C(b))^2 - 1\big)\big((C(a) - C(b))^2 - 4\big) \cdot A(a, b) = c$$
for some random values a and b in $\mathbb{F}_q$. To do this, Prover reveals the values $C(a)$ and $C(b)$ and the certificates and Verifier completes the validation.

There are a couple of ways in which a prover can cheat in this protocol. First, there is nothing to prevent the prover from claiming that the color $C(u)$ of a vertex is, say, 5 or 77. The equivalence of Claim (3) and the existence of a 3-coloring assumed that the coloring C only takes the intended values -1, 0, and 1. So Verifier needs a mechanism to ensure that the committed values are of this type.

Second, the analysis of the LFKN protocol works under the assumption that C is a low-degree polynomial (and could fail otherwise). There is nothing to prevent the prover from committing to a high-degree polynomial.

Certifying that C takes only -1, 0, 1 values on $\{0, 1\}^n$ amounts to the sumcheck

$$\sum_{u \in \{0,1\}^n} r^u (C(u) - 1) C(u) (C(u) + 1) = 0 \tag{4}$$

for a random r in $\mathbb{F}_q$. Here r^u means r raised to the number whose binary expansion is u. If C represents a 3-coloring on $\{0, 1\}^n$ then the expression vanishes regardless of the value of r. If it doesn't then it is a non-zero polynomnial in r of degree at most $2^n - 1$, so the chance that a random r is a root is at most $(2^n - 1)/q$, which is small when q is bigger than 3^n.

To carry out the sumcheck (4) via the LFKN protocol the expression must be a low-degree polynomial in u. Although r^u looks like an exponential, as any other boolean function it can be represented by a multilinear polynomial. Here is the concrete representation:

$$r^u = r^{u_1} \cdot r^{2u_2} \cdots r^{2^{n-1}u_n} = (1 - u_1 + u_1 r) \cdot (1 - u_2 + u_2 r^2) \cdots (1 - u_n + u_n r^{2^{n-1}}).$$

Therefore expression 4 is a degree-$4n$ polynomial in u, assuming C itself is a degree-n polynomial.

It remains for the verifier to be convinced that the C that was committed to in step 1 is indeed a degree-n polynomial. In general this is impossible because Prover can modify the value of C at a random point, in which case C is no longer low-degree but Verifier is unlikely to detect the difference. As Verifier only queries C at random points, it is sufficient that C looks like a degree-n polynomial on most inputs.

The following test of Rubinfeld and Sudan achieves this very purpose: Ask to see the values of C on $n + 2$ inputs that lie along a random line, i.e. ask for the values $C(a), C(a + b), C(a + 2b), \ldots, C(a + (n + 1)b)$ for random a and b. If C had degree n, these values should satisfy the Lagrange interpolation formula. It turns out that if C differs from all degree-n polynomials on at least, say, 1% of inputs in $\mathbb{F}_q^n$ then Lagrange interpolation has a good chance of failing (over the choice of a and b). By repeating the test sufficiently many times Verifier can be confident that the commitment of the prover is, for all intents and purposes, a degree-n polynomial.

We now have all the ingredients to describe the protocol for certifying 3-colorability of a graph specified by its adjacency predicate:

The Babai-Fortnow-Lund / Kilian protocol.

1. Prover sends a succinct (Merkle tree) commitment of C to Verifier. Whenever Verifier asks to see a value $C(u)$, Prover certifies its consistency with the commitment.
2. Verifier and Prover run the degree-n test on C.
3. Verifier chooses a random $r \sim \mathbb{F}_q$ and runs the LFKN protocol for claim (4) with Prover.
4. Verifier and Prover run the LFKN protocol for claim (3).

The commitment in Step 1 ensures that Verifier's queries to C are indistinguishable from oracle access to C. Step 2 ensures that, assuming the Verifier makes random queries to C, those are indistinguishable from random queries to a degree-n polynomial. Step 3 ensures that C takes only the allowed color values on $\{0, 1\}^n$, and finally step 4 ensures that C is a valid 3-coloring of the graph described by A.