Pruebas de conocimiento nulo

Definicion

Una prueba de conocimiento nulo es una tecnica criptografica que permite probar la validez de una hecho/declaracion (*statement*) sin revelar el *statement* como tal. Existen dos partes en la prueba.

- El "*prover*" el cual es el que trata de probar el hecho.
- El "*verifier*" el cual es el responsable de validar el hecho.

Cabe destacar que las 'pruebas' no son pruebas en el contexto matematico (probar un *statement* mas alla de cualquier duda), si no que son pruebas probabilisticas. Es decir, se 'prueba' un *statement* con un alto grado de certeza (generalmente la probabilidad de error es extremedamente baja).

Solo se podra generar una prueba de un *statement* solo cuando se tenga posesion de cierta informacion secreta asociada al mismo (siendo esta la informacion que no se comparte). El *verifier* no podra probar el *statement* a otras partes, incluso despues de haberse convencido de que el *statement* es verdadero.

En general, para realizar una prueba de conocimiento nulo se necesita un protocolo en donde las dos partes interactuan; el *verifier* propone *desafios* aleatorios al *prover* quien debe contestar correctamente. La naturaleza aleatoria de estos desafios, una vez resueltos correctamente por el *prover*, hacen que el *verifier* acepte la 'prueba' de que el *statement* es verdadero (se convence probabilistamente, con una probabilidad de falso positivo muy baja). Esta interaccion entre las partes hace que el *verifier* no pueda hacerse pasar por el *prover*.

Una prueba de conocimiento nulo debe sastifacer los siguientes criterios:

- **Completitud**: Si el *input* es valido, la prueba debe siempre retorar *verdadero* (No debe existir la posibilidad de falsos negativos)
- **Robustez**: Si el *input* es invalido, es teoricamente imposible engañar al protocolo de prueba de conocimiento nulo a retornar *verdadero*. (Un *prover* mentiroso no puede hacer creer a un *verifier* algo invalido, a excepcion de un margen de probabilidad pequeño)
- **Conocimiento nulo**: El *verifier* aprende nada acerca del *statement* excepto su validez. (Prevee que el *verifier* derive el input original a partir de la prueba).

Pruebas interactivas

Una prueba interactiva esta compuesta de tres elementos:

- **Testigo**: El *prover* quiere probar conocimiento de cierta informacion secreta. Esta informacion secreta es *testigo* a la prueba y se asume que el *prover* la conoce y puede responder ciertas preguntas a partir de esta.
- **Desafio**: El *verifier* elige al azar una pregunta (que puede ser respondida) y le pide al *prover* que las responda
- **Respuesta**: El *prover* acepta la respuesta y envia la respuesat al *verifier*. Esta respuesta da cierto grado de confianza de que el *prover* realmente tiene conocimiento del *testigo*. Las repetidas realizaciones de este protocolo aumenta el grado de confianza hasta que el *verifier* este lo suficientemente convencido (con una cantidad finita, sensible de *desafios*)

Pruebas no interactivas

El problema de las pruebas interactivas es que el *prover* debe convencer a todos los *verifiers* realizando el mismo protocolo repetidas veces. Para evitar esto, las pruebas no interactivas se generan una sola vez; luego, cualquiera que quiera verificar que el *prover* tiene la informacion secreta lo puede hacer. Para poder "recrear" la prueba se necesita una *shared key* la cual fue usada al momento de crear la prueba, asi como el algoritmo de verificacion.

Pruebas interactivas de oraculo (*IOP*)

Un **oraculo* es una entidad abstracta que representa una maquina con conocimiento perfecto de la informacion. Ante una consulta, responde instantanea y/o correctamente a la misma, de manera deterministica.

En una prueba interactiva de conocimiento nulo, el *prover* actua como si fuera un oraculo (tiene conocimiento correcto del *statement* a probar), por lo que cualquier prueba interactiva de conocimiento nulo es a su vez una prueba interactiva de oraculo.

Esquema de compromiso de polinomio (*PCS*)

Un esquema de compromiso de polinomio permite a un *committer* a comprometerse a un polinomio a traves de una *palabra* que puede usarse para verificar evaluaciones en el mismo. Este compromiso "indirecto" permite al *committer* a no revelar el polinomio en si. La verificacion del comprimiso al polinomio debe ser eficiente. Una vez un *committer* se compromete con un polinomio, este no puede desligarse del mismo. Un compromiso debe corresponder a un polinomio especifico.

Existen multiples diferentes esquemas, los cuales se diferencias por, entre otros:

- **Setup con o sin confianza**: En los esquemas de setup con confianza se requiren un grupo de participantes en los cuales se confie para que generen ciertos parametros del esquema. Esta generacion dejan 'residuos toxicos' que pueden crear vulnerabilidades. Tambien, los participantes que realizan el setup puede generarlo de manera tal de crear vulnerabilidades. En cambio, en los setup sin confianza, este problema no existe. Por lo general, los esquema de setup sin confianza resultan ser menos eficientes, pues se basan en propiedades criptograficas mas complejas.
- **Escalabilidad**: Los diferentes esquemas pueden escalar de manera diferente frente al polinomio que se busca el compromiso: constante, lineal u otros. Esto hace que la huella en almacenamiento sea diferente entre los diferenes esquemas.
- **Eficiencia**: Algunos esquemas pueden comprobarse mas rapidamente que otros para el mismo polinomio.
- **Fuerza criptografica**: Que tan resistente es a ataques que busquen conocer la informacion oculta.

Cabe destacar que estas propiedades suelen ser exclusivas una otra a la otra. Es decir, si, por ejemplo, se quiere mas eficiencia, puede ser que se obtenga un esquema mas 'debil' criptograficamente.

(ZK-)SNARKs

(Zero-Knowledge) Succinct Non-Interactive Argument of Knowledge o Argumento Sucinto No Interactiva de Conocimiento (Nulo), donde:

- **Sucinta** se refiere a que la prueba de conocimiento nulo es mas pequeña que el *testigo* y puede ser verificada eficientemente.
- **No Interactiva** se refiere a que el unico mensaje que se intercambia entre el *prover* y el *verifier* es la prueba.
- **Conocimiento Nulo** se refiere a que se prueba que se posee cierto conocimiento, pero que el *verifier* no sabe cual el es este.
- **Argumento** se refiere a que existe una (pequeña) probabilidad de que se convenza al *verifier* con un argumento no correcto.

Refiere a la construccion de una prueba que, de manera probabilistica, prueba que tiene cierta informacion, sin necesariamente revelarla o dar ningun dato de donde esta se pueda derivar.

1. ***Gen*** es el setup del algoritmo, generando una *string* ***CRS*** (*Common Reference String*) usada en el proceso de prueba y una key de verificacion ***VRS***. Este *setup* es generalmente hecho por una parte confiable dado que genera **residuo toxico** en forma de la ***simulation trapdoor***. Existen maneras de distribuir el *setup* de forma que se necesiten muchas partes y con que una sola descarte su **residuo toxico** ya no se puede reconstruir la ***simulation trapdoor***

2. ***Prove*** es el proceso que toma como input ***CRS***, el *statement* ***U*** y un *testigo* ***W*** y da como resultado la prueba ***Pi***.

3. ***Verifiy*** es el proceso que toma como input ***VRS***, el *statement* ***U*** y la prueba ***Pi*** y retorna si es aceptada o no.

Aplicaciones

Las aplicaciones de pruebas de conocimiento nulo, y como tal zk-SNARKS son multiples; permiten aumentar la privacidad de una operacion que utiliza datos sensibles. Obviamente esta privacidad viene a un costo, en este caso, rapidez al momento de comprobar la operacion. Algunos ejemplos de sus aplicaciones son:

- **Blockchains y cripto-monedas**: En el caso de las cripto-monedas, permiten realizar transaccion sin revelar quienes son las partes involucradas o la transaccion. ZCash es un ejemplo (y unos de los primeros) de una cripto-moneda que utiliza zk-SNARKs como base de su privacidad.
- **Escalabilidad de la blockchain**: Permiten crear blockchains dependientes de una blockchain original (como puede ser Ethereum) donde se realizan transacciones costosas. Las transacciones realizadas fuera de la blockchain original se juntan, y se realizan un "*commit*" en la blockchain original. De esta forma, el costo de realizar transacciones se disminuye, se disminuye la "carga" en la blockchain original, y ademas da mas flexibilidad a la blockchain original. (Esto puede ser logrado a traves de *sidechains* o *layer 2* blockcains, donde la primera ofrece mas flexibilidad a cambio de ser mas complejas dado que no estan mas disociadas de la blockchain original)
- **Blockchains de tamaño constante**: Permite crear una blockchain de tamaño constante. Esto permite que los nodos y la red en general tenga una huella mucho mas pequeña (nodos menos poderosos, menos consumo energetico, etc.). En vez de agregar un bloque cada vez, extendiendo asi la red, se guarda la prueba de que la transaccion es correcta. Coda
- **Privacidad en aplicaciones / auditorias** zk-SNARKs pueden ser usado para que en aplicaciones donde se use informacion sensible, como datos personales o financieros, sean auditables sin tener que revelar esta informacion.