Explaining SNARKs Part I: Homomorphic Hidings

Ariel Gabizon | February 28, 2017 | Updated: October 1, 2018

Constructions of zk-SNARKs involve a careful combination of several ingredients; fully understanding how these ingredients all work together can take a while.

If I had to choose **one ingredient** whose role is most prominent, it would be what I will call here *Homomorphic Hiding* (HH) [1]. In this post we explain what an HH is, and then give an example of why it is useful and how it is constructed.

An HH $E(x)$ of a number x is a function satisfying the following properties:

- For most x's, given $E(x)$ it is hard to find x.
- Different inputs lead to different outputs – so if $x \neq y$, then $E(x) \neq E(y)$.
- If someone knows $E(x)$ and $E(y)$, they can generate the HH of *arithmetic expressions in x and y*. For example, they can compute $E(x+y)$ from $E(x)$ and $E(y)$.

Here's a toy example of why HH is useful for Zero-Knowledge proofs: Suppose Alice wants to prove to Bob she knows numbers x, y such that $x + y = 7$ (Of course, it's not too exciting knowing such x, y, but this is a good example to explain the concept with).

1. Alice sends $E(x)$ and $E(y)$ to Bob.
2. Bob computes $E(x+y)$ from these values (which he is able to do since E is an HH).
3. Bob also computes $E(7)$, and now checks whether $E(x+y) = E(7)$. He accepts Alice's proof only if equality holds.

As different inputs are mapped by E to different hidings, Bob indeed accepts the proof only if Alice sent hidings of x, y such that $x + y = 7$. On the other hand, Bob does not learn x and y, as he just has access to their hidings [2].

Now let's see an example of how such hidings are constructed. We actually cannot construct them for regular integers with regular addition, but need to look at *finite groups*:

Let n be some integer. When we write $A \bmod n$ for an integer A, we mean taking the remainder of A after dividing by n. For example, $9 \bmod 7 = 2$ and $13 \bmod 12 = 1$. We can use the $\bmod n$ operation to define an addition operation on the numbers $\{0, \ldots, n-1\}$: We do regular addition but then apply $(\bmod n)$ on the result to get back a number in the range $\{0, \ldots, n-1\}$. We sometimes write $(\bmod n)$ on the right to clarify we are using this type of addition. For example, $2 + 3 = 1 \pmod 4$. We call the set of elements $\{0, \ldots, n-1\}$ together with this addition operation the group $\mathbb{Z}_n$.

For a prime p, we can use the $\bmod p$ operation to also define a *multiplication* operation over the numbers $\{1, \ldots, p-1\}$ in a way that the multiplication result is also always in the set $\{1, \ldots, p-1\}$ – by performing regular multiplication of integers, and then taking the result $\bmod p$. [3] For example, $2 \cdot 4 = 1 \pmod 7$.

This set of elements together with this operation is referred to as the group $\mathbb{Z}_p^*$. $\mathbb{Z}_p^*$ has the following useful properties:

1. It is a *cyclic* group; which means that there is some element g in $\mathbb{Z}_p^*$ called a *generator* such that all elements of $\mathbb{Z}_p^*$ can be written as g^a for some a in $\{0, .., p-2\}$, where we define $g^0 = 1$.

2. The *discrete logarithm problem* is believed to be hard in $\mathbb{Z}_p^*$. This means that, when p is large, given an element h in $\mathbb{Z}_p^*$ it is difficult to find the integer a in $0, \ldots, p-2$ such that $g^a = h \pmod p$.

3. As "exponents add up when elements are multiplied", we have for a, b in $0, \ldots, p-2$ $g^a \cdot g^b = g^{a+b \pmod{p-1}}$.

Using these properties, we now construct an HH that "supports addition" – meaning that $E(x+y)$ is computable from $E(x)$ and $E(y)$. We assume the input x of E is from $\mathbb{Z}_{p-1}$, so it is in the range $\{0, \ldots, p-2\}$. We define $E(x) = g^x$ for each such x, and claim that E is an HH: The first property implies that different x's in $\mathbb{Z}_{p-1}$ are mapped to different outputs. The second property implies that given $E(x) = g^x$ it is hard to find x. Finally, using the third property, given $E(x)$ and $E(y)$ we can compute $E(x+y)$ as $E(x+y) = g^{x+y \bmod p-1} = g^x \cdot g^y = E(x) \cdot E(y)$.

---

[1] Homomorphic hiding is not a term formally used in cryptography, and is introduced here for didactic purposes. It is similar to but weaker than the well-known notion of a computationally hiding commitment. The difference is that an HH is a deterministic function of the input, whereas a commitment uses additional randomness. As a consequence, HHs essentially "hide most x's" whereas commitments "hide every x".

---

[2] Bob does learn *some* information about x and y. For example, he can choose a random x', and check whether x=x' by computing E(x'). For this reason the above protocol is not really a Zero-Knowledge protocol, and is only used here for explanatory purposes. In fact, as we shall see in later posts, HH is ultimately used in snarks to conceal verifier challenges rather than prover secrets.

---

[3] When p is not a prime it is problematic to define multiplication this way. One issue is that the multiplication result can be zero even when both arguments are not zero. For example when p=4, we can get 2*2=0 (mod 4).

Part II >>

Technical
#cryptography, explainers, zksnarks

< History of Hash Function Attacks | New Release: 1.0.7 >

Explaining SNARKs Part II: Blind Evaluation of Polynomials

Ariel Gabizon | March 13, 2017 | Updated: October 1, 2018

[<< Part I]

In this post, we recall the notion of a polynomial, and explain the notion of "blind evaluation" of a polynomial, and how it is implemented using Homomorphic Hiding (HH). (See Part I for an explanation of HH.) In future posts, we will see that blind evaluation is a central tool in SNARK constructions.

We denote by $\mathbb{F}_p$ the field of size p; that is, the elements of $\mathbb{F}_p$ are $\{0,\ldots,p-1\}$ and addition and multiplication are done $\mod p$ as explained in Part I.

Polynomials and linear combinations

Recall that a polynomial P of degree d over $\mathbb{F}_p$ is an expression of the form

$$P(X) = a_0 + a_1 \cdot X + a_2 \cdot X^2 + \ldots + a_d \cdot X^d$$

, for some $a_0,\ldots,a_d \in \mathbb{F}_p$.

We can *evaluate* P at a point $s \in \mathbb{F}_p$ by substituting s for X, and computing the resultant sum

$$P(s) = a_0 + a_1 \cdot s + a_2 \cdot s^2 + \ldots + a_d \cdot s^d$$

For someone that knows P, the value $P(s)$ is a *linear combination* of the values $1, s, \ldots, s^d$ – where linear combination just means "weighted sum", in the case of $P(s)$ the "weights" are $a_0,\ldots,a_d$.

In the last post, we saw the HH E defined by $E(x) = g^x$ where g was a generator of a group with a hard discrete log problem. We mentioned that this HH "supports addition" in the sense that $E(x+y)$ can be computed from $E(x)$ and $E(y)$. We note here that it also "supports linear combinations"; meaning that, given $a, b, E(x), E(y)$, we can compute $E(ax+by)$. This is simply because

$$E(ax+by) = g^{ax+by} = g^{ax} \cdot g^{by} = (g^x)^a \cdot (g^y)^b = E(x)^a \cdot E(y)^b.$$

Blind evaluation of a polynomial

Suppose Alice has a polynomial P of degree d, and Bob has a point $s \in \mathbb{F}_p$ that he chose randomly. Bob wishes to learn $E(P(s))$, i.e., the HH of the evaluation of P at s. Two simple ways to do this are:

- Alice sends P to Bob, and he computes $E(P(s))$ by himself.
- Bob sends s to Alice; she computes $E(P(s))$ and sends it to Bob.

However, in the *blind evaluation problem* we want Bob to learn $E(P(s))$ without learning P – which precludes the first option; and, most importantly, we don't want Alice to learn s, which rules out the second [1].

Using HH, we can perform blind evaluation as follows.

1. Bob sends to Alice the hidings $E(1), E(s), \ldots, E(s^d)$.
2. Alice computes $E(P(s))$ from the elements sent in the first step, and sends $E(P(s))$ to Bob. (Alice can do this since E supports linear combinations, and $P(s)$ is a linear combination of $1, s, \ldots, s^d$.)

Note that, as only hidings were sent, neither Alice learned s [2], nor Bob learned P.

Why is this useful?

Subsequent posts will go into more detail as to how blind evaluation is used in SNARKs. The rough intuition is that the verifier has a “correct” polynomial in mind, and wishes to check the prover knows it. Making the prover blindly evaluate their polynomial at a random point not known to them, ensures the prover will give the wrong answer with high probability if their polynomial is not the correct one. This, in turn, relies on the Schwartz-Zippel Lemma stating that “different polynomials are different at most points”.

---

[1] The main reason we don’t want to send P to Bob, is simply that it is large – (d+1) elements, where, for example, d~2000000 in the current Zcash protocol; this ultimately has to do with the “Succinct” part of SNARKs. It is true that the sequence of hidings Bob is sending to Alice above is just as long, but it will turn out this sequence can be “hard-coded” in the parameters of the system, whereas Alice’s message will be different for each SNARK proof.

---

[2] Actually, the hiding property only guarantees s not being recoverable from $E(s)$, but here we want to claim it is also not recoverable from the sequence $E(s), \ldots, E(s^d)$ that potentially contains more information about s. This follows from the d-power Diffie-Hellman assumption, which is needed in several SNARK security proofs.

Part III >>

Technical
#cryptography, explainers, zksnarks

< BLS12-381: New zk-SNARK Elliptic Curve Construction | Announcing the Zcash Foundation >

Explaining SNARKs Part III: The Knowledge of Coefficient Test and Assumption

Ariel Gabizon | March 28, 2017 | Updated: October 1, 2018

[<< Part II]

In Part II, we saw how Alice can blindly evaluate the hiding $E(P(s))$ of her polynomial P of degree d, at a point s belonging to Bob. We called this "blind" evaluation, because Alice did not learn s in the process.

However, there was something missing in that protocol – the fact that Alice is *able* to compute $E(P(s))$ does not guarantee she will indeed send $E(P(s))$ to Bob, rather than some completely unrelated value.

Thus, we need a way to "force" Alice to follow the protocol correctly. We will explain in part IV precisely how we achieve this. In this post, we focus on explaining the basic tool needed for that – which we call here the *Knowledge of Coefficient (KC) Test*.

As before, we denote by g a generator of a group G of order $|G| = p$ where the discrete log is hard. It will be convenient from this post onwards to write our group additively rather than multiplicatively. That is, for $\alpha \in \mathbb{F}_p$, $\alpha \cdot g$ denotes the result of summing α copies of g.

The KC Test

For $\alpha \in \mathbb{F}_p^*$ [1], let us call a pair of elements (a, b) in G an α-pair if $a, b \neq 0$ and $b = \alpha \cdot a$.

The KC Test proceeds as follows.

1. Bob chooses random $\alpha \in \mathbb{F}_p^*$ and $a \in G$. He computes $b = \alpha \cdot a$.
2. He sends to Alice the "challenge" pair (a, b). Note that (a, b) is an α-pair.
3. Alice must now respond with a *different* pair (a', b') that is also an α-pair.
4. Bob accepts Alice's response only if (a', b') is indeed an α-pair. (As he knows α he can check if $b' = \alpha \cdot a'$.)

Now, let's think how Alice could successfully respond to the challenge. Let's assume for a second that she *knew* α. In that case, she could simply choose any a' in G, and compute $b' = \alpha \cdot a'$; and return (a', b') as her new α-pair.

However, as the only information about α she has is $\alpha \cdot a$ and G has a hard discrete log problem, we expect that Alice cannot find α.

So how can she successfully respond to the challenge without knowing α?

Here's the natural way to do it: Alice simply chooses some $\gamma \in \mathbb{F}_p^*$, and responds with $(a', b') = (\gamma \cdot a, \gamma \cdot b)$.

In this case, we have:

$$b' = \gamma \cdot b = \gamma\alpha \cdot a = \alpha(\gamma \cdot a) = \alpha \cdot a',$$

so indeed (a', b') is an α-pair as required.

Note that if Alice responds using this strategy, she *knows the ratio* between a and a'. That is, she knows the coefficient γ such that $a' = \gamma \cdot a$.

The Knowledge of Coefficient Assumption [2] (KCA) states that *this is always the case*, namely:

KCA: If Alice returns a valid response (a', b') *to Bob's challenge* (a, b) *with non-negligible probability over Bob's choices of* a, α*, then she knows* γ *such that* $a' = \gamma \cdot a$.

The KC Test and Assumption will be important tools in Part IV.

What does "Alice knows" mean exactly

You may wonder how we can phrase the KCA in precise mathematical terms; specifically, how do we formalize the notion that "Alice knows γ" in a mathematical definition?

This is done roughly as follows: We say that, in addition to Alice, we have another party which we call *Alice's Extractor*. Alice's Extractor has access to Alice's inner state.

We then formulate the KCA as saying that: whenever Alice successfully responds with an α-pair (a', b'), Alice's Extractor outputs γ such that $a' = \gamma \cdot a$. [3]

---

[1] $\mathbb{F}_p^*$ denotes the non-zero elements of $\mathbb{F}_p$. It is the same as $\mathbb{Z}_p^*$ described in Part I.

---

[2] This is typically called the Knowledge of Exponent Assumption in the literature, as traditionally it was used for groups written multiplicatively.

---

[3] The fully formal definition needs to give the Extractor "a little slack" and states instead that the probability that Alice responds successfully but the Extractor does not output such γ is negligible.

Part IV >>

Technical
#cryptography, explainers, zksnarks

< New Release: 1.0.8 | BIP199 for Hashed Timelocked Contracts >

Explaining SNARKs Part IV: How to make Blind Evaluation of Polynomials Verifiable

Ariel Gabizon | April 11, 2017 | Updated: October 1, 2018

[<< Part III]

In this part, we build on Part II and III to develop a protocol for verifiable blind evaluation of polynomials, which we will define shortly. In Part V we'll start to see how such a protocol can be used for constructing SNARKs, so bear with me a little bit longer for the connection to SNARKs :).

Suppose, as in Part II, that Alice has a polynomial P of degree d and Bob has a point $s \in \mathbb{F}_p$ that he chose randomly. We want to construct a protocol that allows Bob to learn $E(P(s))$, i.e. the hiding of P evaluated at s, with two additional properties:

1. **Blindness**: Alice will *not* learn s (and Bob will not learn P).
2. **Verifiability**: The probability that Alice sends a value not of the form $E(P(s))$ for P of degree d that is known to her, but Bob still accepts – is negligible.

This is what we call *verifiable* blind evaluation of a polynomial. The protocol in Part II gave us the first item but not the second. To get verifiability we need an extended version of the Knowledge of Coefficient Assumption (KCA) that was presented in Part III.

The verifiability and blindness properties are useful together because they make Alice decide what polynomial P she will use *without seeing* s. [1] This, in a sense, commits Alice to an "answer polynomial" without seeing the "challenge point" s. This intuition will become more clear in the next parts of the series.

An Extended KCA

The KCA as we defined it in Part III essentially said something like this: If Bob gives Alice some α-pair $(a, b = \alpha \cdot a)$ and then Alice generates another α-pair (a', b'), then she knows c such that $a' = c \cdot a$. In other words, Alice knows the relation between a' and a.

Now, suppose that instead of one, Bob sends Alice several α-pairs $(a_1, b_1), \ldots, (a_d, b_d)$ (for the same α); and that again, after receiving these pairs, Alice is challenged to generate some other α-pair (a', b'). Recall that the main point is that Alice must do so although *she does not know* α.

As we saw in Part III, a natural way for Alice to generate such an α-pair, would be to take one of the pairs (a_i, b_i) she received from Bob, and multiply both elements by some $c \in \mathbb{F}_p^*$; if (a_i, b_i) was an α-pair, then $(c \cdot a_i, c \cdot b_i)$ will be one too. But can Alice generate α-pairs in more ways now that she's received *multiple* α-pairs? Perhaps using several of the received α-pairs *simultaneously* to get a new one?

The answer is yes: For example, Alice can choose *two* values $c_1, c_2 \in \mathbb{F}_p$ and compute the pair $(a', b') = (c_1 \cdot a_1 + c_2 \cdot a_2, c_1 \cdot b_1 + c_2 \cdot b_2)$. An easy computation shows that, as long as a' is non-zero, this is also an α-pair:

$b' = c_1 \cdot b_1 + c_2 \cdot b_2 = c_1\alpha \cdot a_1 + c_2\alpha \cdot a_2 = \alpha(c_1 \cdot a_1 + c_2 \cdot a_2) = \alpha \cdot a'.$

More generally, Alice can take any *linear combination* of the given d pairs – that is choose any $c_1, \ldots, c_d \in \mathbb{F}_p$ and define $(a', b') = (\sum_{i=1}^{d} c_i a_i, \sum_{i=1}^{d} c_i b_i)$.

Note that, if Alice uses this strategy to generate her α-pair she will know some linear relation between a' and $a_1, \ldots, a_d$; that is, she knows $c_1, \ldots, c_d$ such that $a' = \sum_{i=1}^{d} c_i \cdot a_i$.

The extended KCA states, essentially, that this is the only way Alice can generate an α-pair; that is, whenever she succeeds, she will know such a linear relation between a' and $a_1, \ldots, a_d$. More formally, suppose that G is a group of size p with generator g written additively as in Part III. The *d-power Knowledge of Coefficient Assumption* (d-KCA) [2] in G is as follows:

d-KCA: *Suppose Bob chooses random $\alpha \in \mathbb{F}_p^*$ and $s \in \mathbb{F}_p$, and sends to Alice the* α-pairs $(g, \alpha \cdot g), (s \cdot g, \alpha s \cdot g), \ldots, (s^d \cdot g, \alpha s^d \cdot g)$. *Suppose that Alice then outputs another* α-pair (a', b'). *Then, except with negligible probability, Alice knows $c_0, \ldots, c_d \in \mathbb{F}_p$ such that $\sum_{i=0}^{d} c_i s^i \cdot g = a'$.*

Note that in the d-KCA Bob does not send an arbitrary set of α-pairs, but one with a certain "polynomial structure". This will be useful in the protocol below.

The Verifiable Blind Evaluation Protocol

Assume that our HH is the mapping $E(x) = x \cdot g$ for a generator g of G as above.

For simplicity, we present the protocol for this particular E:

1. Bob chooses a random $\alpha \in \mathbb{F}_p^*$, and sends to Alice the hidings $g, s \cdot g, \ldots, s^d \cdot g$ (of $1, s, \ldots, s^d$) and also the hidings $\alpha \cdot g, \alpha s \cdot g, \ldots, \alpha s^d \cdot g$ (of $\alpha, \alpha s, \ldots, \alpha s^d$).
2. Alice computes $a = P(s) \cdot g$ and $b = \alpha P(s) \cdot g$ using the elements sent in the first step, and sends both to Bob.
3. Bob checks that $b = \alpha \cdot a$, and accepts if and only if this equality holds.

First, note that given the coefficients of P, $P(s) \cdot g$ is a linear combination of $g, s \cdot g, \ldots, s^d \cdot g$; and $\alpha P(s) \cdot g$ is a linear combination of $\alpha \cdot g, \alpha s \cdot g, \ldots, \alpha s^d \cdot g$. Thus, similarly to the protocol of Part II, Alice can indeed compute these values from Bob's messages for a polynomial P that she knows.

Second, by the d-KCA if Alice sends a, b such that $b = \alpha \cdot a$ then almost surely she knows $c_0, \ldots, c_d \in \mathbb{F}_p$ such that $a = \sum_{i=0}^{d} c_i s^i \cdot g$. In that case, $a = P(s) \cdot g$ for the polynomial $P(X) = \sum_{i=0}^{d} c_i \cdot X^i$ known to Alice. In other words, the probability that Bob accepts in Step 3 while at the same time Alice does not know such a P is negligible.

To summarize, using the d-KCA we've developed a protocol for verifiable blind evaluation of polynomials. In the next posts, we will see how this building block comes to play in SNARK constructions.

---

[1] In the fully formal proof, things are somewhat more subtle, as Alice *does* see some information about s before deciding on her P – for example, the hidings of $s, \ldots, s^d$.

---

[2] The d-KCA was introduced in a paper of Jens Groth.

Part V >>

Technical
#cryptography, explainers, zksnarks

< Bellman: zk-SNARKs in Rust | Security Announcement 2017-04-12 >

Explaining SNARKs Part V: From Computations to Polynomials

Ariel Gabizon | April 25, 2017 | Updated: October 1, 2018

[<< Part IV]

In the three previous parts, we developed a certain machinery for dealing with polynomials. In this part, we show how to translate statements we would like to prove and verify to the language of polynomials. The idea of using polynomials in this way goes back to the groundbreaking work of Lund, Fortnow, Karloff and Nisan.

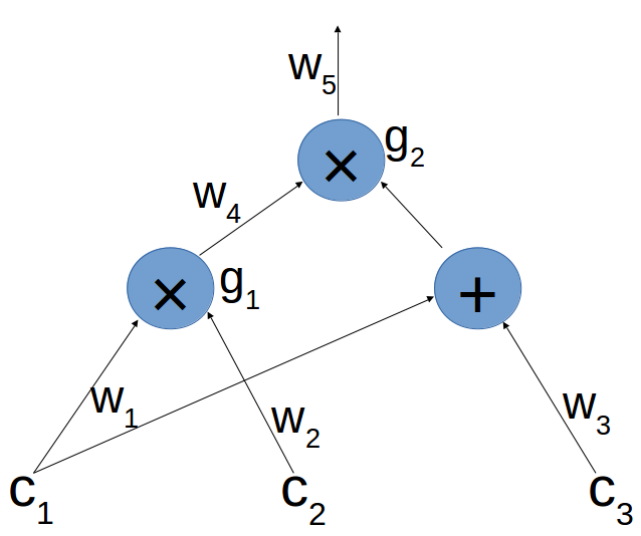
In 2013, another breakthrough work of Gennaro, Gentry, Parno and Raykova defined an extremely useful translation of computations into polynomials called a *Quadratic Arithmetic Program* (QAP). QAPs have become the basis for modern zk-SNARK constructions, in particular those used by Zcash.

In this post we explain the translation into QAPs by example. Even when focusing on a small example rather than the general definition, it is unavoidable that it is a lot to digest at first, so be prepared for a certain mental effort :).

Suppose Alice wants to prove to Bob she knows $c_1, c_2, c_3 \in \mathbb{F}_p$ such that $(c_1 \cdot c_2) \cdot (c_1 + c_3) = 7$. The first step is to present the expression computed from c_1, c_2, c_3 as an *arithmetic circuit*.

Arithmetic circuits

An arithmetic circuit consists of gates computing arithmetic operations like addition and multiplication, with wires connecting the gates. In our case, the circuit looks like this:



The bottom wires are the input wires, and the top wire is the output wire giving the result of the circuit computation on the inputs. As can be seen in the picture, we label the wires and gates of the circuit in a very particular way, that is needed for the next step of translating the circuit into a QAP.

- When the same outgoing wire goes into more than one gate, we still think of it as one wire – like w_1 in the example.
- We assume multiplication gates have exactly two input wires, which we call the left wire and right wire.
- We don't label the wires going from an addition to multiplication gate, nor the addition gate; we think of the inputs of the addition gate as going directly into the multiplication gate. So in the example we think of w_1 and w_3 as both

being right inputs of g_2.

A *legal assignment* for the circuit, is an assignment of values to the labeled wires where the output value of each multiplication gate is indeed the product of the corresponding inputs. So for our circuit, a legal assignment is of the form: $(c_1, \ldots, c_5)$ where $c_4 = c_1 \cdot c_2$ and $c_5 = c_4 \cdot (c_1 + c_3)$. In this terminology, what Alice wants to prove is that she knows a legal assignment $(c_1, \ldots, c_5)$ such that $c_5 = 7$. The next step is to translate this statement into one about polynomials using QAPs.

Reduction to a QAP

We associate each multiplication gate with a field element: g_1 will be associated with $1 \in \mathbb{F}_p$ and g_2 with $2 \in \mathbb{F}_p$. We call the points $\{1, 2\}$ our *target points*. Now we need to define a set of "left wire polynomials" $L_1, \ldots, L_5$, "right wire polynomials" $R_1, \ldots, R_5$ and "output wire polynomials" $O_1, \ldots, O_5$.

The idea for the definition is that the polynomials will usually be zero on the target points, except the ones involved in the target point's corresponding multiplication gate.

Concretely, as w_1, w_2, w_4 are, respectively, the left, right and output wire of g_1; we define $L_1 = R_2 = O_4 = 2 - X$, as the polynomial $2 - X$ is one on the point 1 corresponding to g_1 and zero on the point 2 corresponding to g_2.

Note that w_1 and w_3 are *both* right inputs of g_2. Therefore, we define similarly $L_4 = R_1 = R_3 = O_5 = X - 1$ – as $X - 1$ is one on the target point 2 corresponding to g_2 and zero on the other target point.

We set the rest of the polynomials to be the zero polynomial.

Given fixed values $(c_1, \ldots, c_5)$ we use them as coefficients to define a left, right, and output "sum" polynomials. That is, we define

$$L := \sum_{i=1}^{5} c_i \cdot L_i, R := \sum_{i=1}^{5} c_i \cdot R_i, O := \sum_{i=1}^{5} c_i \cdot O_i,$$

and then we define the polynomia $P := L \cdot R - O$.

Now, after all these definitions, the central point is this: $(c_1, \ldots, c_5)$ *is a legal assignment to the circuit if and only if* P *vanishes on all the target points*.

Let's examine this using our example. Suppose we defined L, R, O, P as above given some $c_1, \ldots, c_5$. Let's evaluate all these polynomials at the target point 1:

Out of all the L_i's only L_1 is non-zero on 1. So we have $L(1) = c_1 \cdot L_1(1) = c_1$. Similarly, we get $R(1) = c_2$ and $O(1) = c_4$.

Therefore, $P(1) = c_1 \cdot c_2 - c_4$. A similar calculation shows $P(2) = c_4 \cdot (c_1 + c_3) - c_5$.

In other words, P vanishes on the target points if and only if $(c_1, \ldots, c_5)$ is a legal assignment.

Now, we use the following algebraic fact: For a polynomial P and a point $a \in \mathbb{F}_p$, we have $P(a) = 0$ if and only if the polynomial $X - a$ *divides* P, i.e. $P = (X - a) \cdot H$ for some polynomial H.

Defining the *target polynomial* $T(X) := (X - 1) \cdot (X - 2)$, we thus have that T divides P if and only if $(c_1, \ldots, c_5)$ is a legal assignment.

Following the above discussion, we define a QAP as follows:

A Quadratic Arithmetic Program Q *of degree* d *and size* m *consists of polynomials* $L_1, \ldots, L_m$, $R_1, \ldots, R_m$, $O_1, \ldots, O_m$ *and a target polynomial* T *of degree* d.

An assignment $(c_1, \ldots, c_m)$ **satisfies** Q *if, defining* $L := \sum_{i=1}^{m} c_i \cdot L_i, R := \sum_{i=1}^{m} c_i \cdot R_i, O := \sum_{i=1}^{m} c_i \cdot O_i$ *and* $P := L \cdot R - O$, *we have that* T *divides* P.

In this terminology, Alice wants to prove she knows an assignment $(c_1, \ldots, c_5)$ satisfying the QAP described above with $c_5 = 7$.

To summarize, we have seen how a statement such as "I know c_1, c_2, c_3 such that $(c_1 \cdot c_2) \cdot (c_1 + c_3) = 7$" can be translated into an equivalent statement about polynomials using QAPs. In the next part, we will see an efficient protocol for proving knowledge of a satisfying assignment to a QAP.

>> Part VI

---

[1] In this post we tried to give the most concise example of a reduction to QAP; we also recommend Vitalik Buterin's excellent post for more details on the transformation from a program to a QAP.

Technical
#cryptography, explainers, zksnarks

< Payment Contexts & Reusing Shielded Addresses | Internet Money >

Explaining SNARKs Part VI: The Pinocchio Protocol

Ariel Gabizon | May 10, 2017

[<< Part V]

In part V we saw how a statement Alice would like to prove to Bob can be converted into an equivalent form in the "language of polynomials" called a Quadratic Arithmetic Program (QAP).

In this part, we show how Alice can send a very short proof to Bob showing she has a satisfying assignment to a QAP. We will use the Pinocchio Protocol of Parno, Howell, Gentry and Raykova. But first let us recall the definition of a QAP we gave last time:

A Quadratic Arithmetic Program Q of degree d and size m consists of polynomials $L_1, \ldots, L_m, R_1, \ldots, R_m, O_1, \ldots, O_m$ and a target polynomial T of degree d.

An assignment $(c_1, \ldots, c_m)$ **satisfies** *Q if, defining $L := \sum_{i=1}^{m} c_i \cdot L_i, R := \sum_{i=1}^{m} c_i \cdot R_i, O := \sum_{i=1}^{m} c_i \cdot O_i$ and $P := L \cdot R - O$, we have that T divides P.*

As we saw in Part V, Alice will typically want to prove she has a satisfying assignment possessing some additional constraints, e.g. $c_m = 7$; but we ignore this here for simplicity, and show how to just prove knowledge of *some* satisfying assignment.

If Alice has a satisfying assignment it means that, defining L, R, O, P as above, there exists a polynomial H such that $P = H \cdot T$. In particular, for any $s \in \mathbb{F}_p$ we have $P(s) = H(s) \cdot T(s)$.

Suppose now that Alice *doesn't* have a satisfying assignment, but she still constructs L, R, O, P as above from some unsatisfying assignment $(c_1, \ldots, c_m)$. Then we are guaranteed that T does not divide P. This means that for any polynomial H of degree at most d, P and $H \cdot T$ will be different polynomials. Note that P and $H \cdot T$ here are both of degree at most $2d$.

Now we can use the famous Schwartz-Zippel Lemma that tells us that two different polynomials of degree at most $2d$ can agree on at most $2d$ points $s \in \mathbb{F}_p$. Thus, if p is much larger than $2d$ the probability that $P(s) = H(s) \cdot T(s)$ for a randomly chosen $s \in \mathbb{F}_p$ is very small.

This suggests the following protocol sketch to test whether Alice has a satisfying assignment.

1. Alice chooses polynomials L, R, O, H of degree at most d.
2. Bob chooses a random point $s \in \mathbb{F}_p$, and computes $E(T(s))$.
3. Alice sends Bob the hidings of all these polynomials evaluated at s, i.e. $E(L(s)), E(R(s)), E(O(s)), E(H(s))$.
4. Bob checks if the desired equation holds at s. That is, he checks whether $E(L(s) \cdot R(s) - O(s)) = E(T(s) \cdot H(s))$.

Again, the point is that if Alice does not have a satisfying assignment, she will end up using polynomials where the equation does not hold identically, and thus does not hold at most choices of s. Therefore, Bob will reject with high probability over his choice of s in such a case.

The question is whether we have the tools to implement this sketch. The most crucial point is that Alice must choose the polynomials she will use, *without* knowing s. But this is exactly the problem we solved in the verifiable blind evaluation protocol, that was developed in Parts II-IV.

Given that we have that, there are four main points that need to be addressed to turn this sketch into a zk-SNARK. We deal with two of them here, and the other two in the next part.

Making sure Alice chooses her polynomials according to an assignment

Here is an important point: If Alice doesn't have a satisfying assignment, it doesn't mean she can't find *any* polynomials L, R, O, H of degree at most d with $L \cdot R - O = T \cdot H$, it just means she can't find such polynomials where L, R and O were "produced from an assignment"; namely, that $L := \sum_{i=1}^{m} c_i \cdot L_i, R := \sum_{i=1}^{m} c_i \cdot R_i, O := \sum_{i=1}^{m} c_i \cdot O_i$ for *the same* $(c_1, \ldots, c_m)$.

The protocol of Part IV just guarantees she is using some polynomials L, R, O of the right degree, but not that they were produced from an assignment. This is a point where the formal proof gets a little subtle; here we sketch the solution imprecisely.

Let's combine the polynomials L, R, O into one polynomial F as follows:

$$F = L + X^{d+1} \cdot R + X^{2(d+1)} \cdot O$$

The point of multiplying R by X^{d+1} and O by $X^{2(d+1)}$ is that the coefficients of L, R, O "do not mix" in F: The coefficients of $1, X, \ldots, X^d$ in F are precisely the coefficients of L, the next $d+1$ coefficients of $X^{d+1}, \ldots, X^{2d+1}$ are precisely the coefficients of R, and the last $d+1$ coefficients are those of O.

Let's combine the polynomials in the QAP definition in a similar way, defining for each $i \in \{1, \ldots, m\}$ a polynomial F_i whose first $d+1$ coefficients are the coefficients of L_i, followed be the coefficients of R_i and then O_i. That is, for each $i \in \{1, \ldots, m\}$ we define the polynomial

$$F_i = L_i + X^{d+1} \cdot R_i + X^{2(d+1)} \cdot O_i$$

Note that when we sum two of the F_i's the L_i, R_i, and O_i "sum separately". For example, $F_1 + F_2 = (L_1 + L_2) + X^{d+1} \cdot (R_1 + R_2) + X^{2(d+1)} \cdot (O_1 + O_2)$.

More generally, suppose that we had $F = \sum_{i=1}^{m} c_i \cdot F_i$ for some $(c_1, \ldots, c_m)$. Then we'll also have $L = \sum_{i=1}^{m} c_i \cdot L_i, R = \sum_{i=1}^{m} c_i \cdot R_i, O = \sum_{i=1}^{m} c_i \cdot O_i$ for the same coefficients $(c_1, \ldots, c_m)$. In other words, if F is a linear combination of the F_i's it means that L, R, O were indeed produced from an assignment.

Therefore, Bob will ask Alice to prove to him that F is a linear combination of the F_i's. This is done in a similar way to the protocol for verifiable evaluation:

Bob chooses a random $\beta \in \mathbb{F}_p^*$, and sends to Alice the hidings $E(\beta \cdot F_1(s)), \ldots, E(\beta \cdot F_m(s))$. He then asks Alice to send him the element $E(\beta \cdot F(s))$. If she succeeds, an extended version of the Knowledge of Coefficient Assumption implies she knows how to write F as a linear combination of the F_i's.

Adding the zero-knowledge part – concealing the assignment

In a zk-SNARK Alice wants to conceal all information about her assignment. However the hidings $E(L(s)), E(R(s)), E(O(s)), E(H(s))$ do provide *some* information about the assignment.

For example, given some other satisfying assignment $(c'_1, \ldots, c'_m)$ Bob could compute the corresponding L', R', O', H' and hidings $E(L'(s)), E(R'(s)), E(O'(s)), E(H'(s))$. If these come out different from Alice's hidings, he could deduce that $(c'_1, \ldots, c'_m)$ is not Alice's assignment.

To avoid such information leakage about her assignment, Alice will conceal her assignment by adding a "random T-shift" to each polynomial. That is, she chooses random $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_p^*$, and defines
$L_z := L + \delta_1 \cdot T, R_z := R + \delta_2 \cdot T, O_z := O + \delta_3 \cdot T.$

Assume L, R, O were produced from a satisfying assignment; hence, $L \cdot R - O = T \cdot H$ for some polynomial H. As we've just added a multiple of T everywhere, T also divides $L_z \cdot R_z - O_z$. Let's do the calculation to see this:

$$
\begin{aligned}
L_z \cdot R_z - O_z &= (L + \delta_1 \cdot T)(R + \delta_2 \cdot T) - O - \delta_3 \cdot T \\
&= (L \cdot R - O) + L \cdot \delta_2 \cdot T + \delta_1 \cdot T \cdot R + \delta_1 \delta_2 \cdot T^2 - \delta_3 \cdot T \\
&= T \cdot (H + L \cdot \delta_2 + \delta_1 \cdot R + \delta_1 \delta_2 \cdot T - \delta_3)
\end{aligned}
$$

Thus, defining $H_z = H + L \cdot \delta_2 + \delta_1 \cdot R + \delta_1 \delta_2 \cdot T - \delta_3$, we have that $L_z \cdot R_z - O_z = T \cdot H_z$. Therefore, if Alice uses the polynomials L_z, R_z, O_z, H_z instead of L, R, O, H, Bob will always accept.

On the other hand, these polynomials evaluated at $s \in \mathbb{F}_p$ with $T(s) \neq 0$ (which is all but d s's), reveal no information about the assignment. For example, as $T(s)$ is non-zero and δ_1 is random, $\delta_1 \cdot T(s)$ is a random value, and therefore $L_z(s) = L(s) + \delta_1 \cdot T(s)$ reveals no information about $L(s)$ as it is masked by this random value.

What's left for next time?

We presented a sketch of the Pinocchio Protocol in which Alice can convince Bob she possesses a satisfying assignment for a QAP, without revealing information about that assignment. There are two main issues that still need to be resolved in order to obtain a zk-SNARK:

- In the sketch, Bob needs an HH that "supports multiplication". For example, he needs to compute $E(H(s) \cdot T(s))$ from $E(H(s))$ and $E(T(s))$. However, we have not seen so far an example of an HH that enables this. We have only seen an HH that supports addition and linear combinations.
- Throughout this series, we have discussed *interactive* protocols between Alice and Bob. Our final goal, though, is to enable Alice to send single-message *non-interactive proofs*, that are *publicly verifiable* – meaning that anybody seeing this single message proof will be convinced of its validity, not just Bob (who had prior communication with Alice).

Both these issues can be resolved by the use of pairings of elliptic curves, which we will discuss in the next and final part.

>> Part VII

Technical
#cryptography, explainers, zksnarks

< Release Cycle and Lifetimes | Getting Started Developing with Zcash >

Explaining SNARKs Part VII: Pairings of Elliptic Curves

Ariel Gabizon | June 7, 2017 | Updated: October 1, 2018

[<< Part VI]

In Part VI, we saw an outline of the Pinocchio zk-SNARK. We were missing two things – an HH that supports both addition and multiplication that is needed for the verifier's checks, and a transition from an interactive protocol to a non-interactive proof system.

In this post we will see that using elliptic curves we can obtain a limited, but sufficient for our purposes, form of HH that supports multiplication. We will then show that this limited HH also suffices to convert our protocol to the desired non-interactive system.

We begin by introducing elliptic curves and explaining how they give us the necessary HH.

Elliptic curves and their pairings

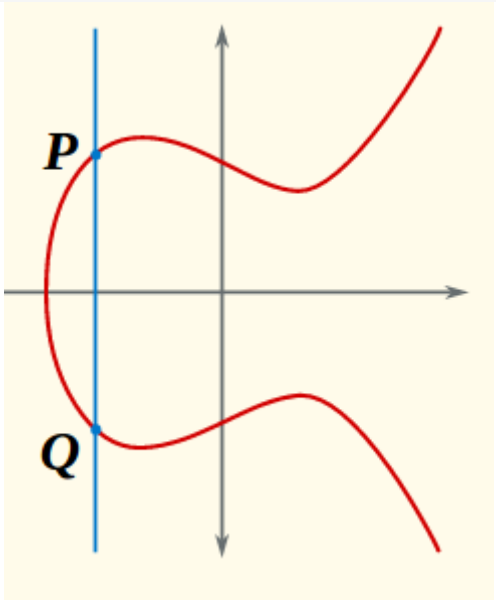
Assume p is a prime larger than 3, and take some $u, v \in \mathbb{F}_p$ such that $4u^3 + 27v^2 \neq 0$. We look at the equation

$$Y^2 = X^3 + u \cdot X + v$$

An elliptic curve $\mathcal{C}$ is the of set of points (x, y) [1] that satisfy such an equation. These curves give us an interesting way to construct groups. The group elements will be the points $(x, y) \in \mathbb{F}_p^2$ that are on the curve, i.e., that satisfy the equation, together with a special point $\mathcal{O}$, that for technical reasons is sometimes refered to as the "point at infinity", and serves as the identity element, i.e. the zero of the group.

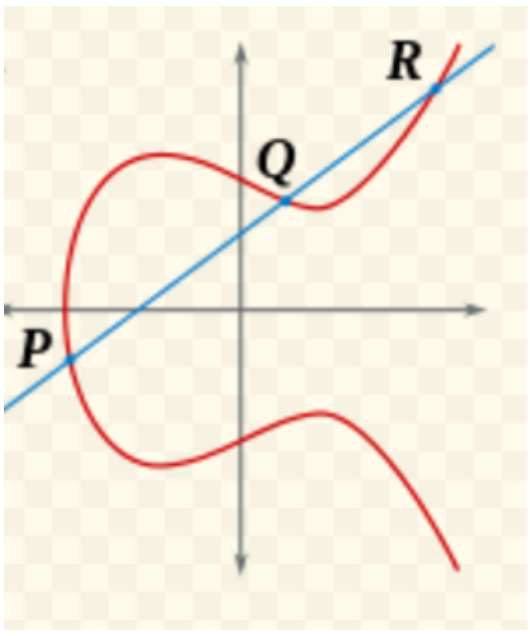
Now the question is how we add two points $P = (x_1, y_1), Q = (x_2, y_2)$ to get a third? The addition rule is derived from a somewhat abstract object called the *divisor class group* of the curve. For our purposes, all you have to know about this divisor class group is that it imposes the following constraint on the definition of addition: The sum of points on any line must be zero, i.e., $\mathcal{O}$.

Let's see how the addition rule is derived from this constraint. Look at a vertical line, defined by an equation of the form $X = c$. Suppose this line intersects the curve at a point $P = (x_1, y_1)$. Because the curve equation is of the form $Y^2 = f(X)$, if (x_1, y_1) is on the curve, so is the point $Q := (x_1, -y_1)$. Moreover, since it's a vertical line and the curve equation is of degree two in Y, we can be sure these are the only points where the line and curve intersect.



Thus, we must have $P + Q = \mathcal{O}$ which means $P = -Q$; that is, Q is the inverse of P in the group.

Now let us look at points P and Q that have a different first coordinate – that is, $x_1 \neq x_2$, and see how to add them. We pass a line through P and Q.



Since the curve is defined by a degree three polynomial in X and already intersects this (non-vertical) line at two points, it is guaranteed to intersect the line at a third point, that we denote $R = (x, y)$, and no other points.

So we must have $P + Q + R = \mathcal{O}$, which means $P + Q = -R$; and we know by now that $-R$ is obtained from R by flipping the second coordinate from y to $-y$.

Thus, we have derived the addition rule for our group: Given points P and Q, pass a line through them, and then take the "mirror" point of the third intersection point of the line as the addition result. [2]

This group is usually called $\mathcal{C}(\mathbb{F}_p)$ – as it consists of points on the curve $\mathcal{C}$ with coordinates in $\mathbb{F}_p$; but let's denote it by G_1 from now on. Assume for simplicity that the number of elements in G_1 is a prime number r, and is different from p. This is many times the case, for example in the curve that Zcash is currently using. In this case, any element $g \in G_1$ different from $\mathcal{O}$ generates G_1.

The smallest integer k such that r divides $p^k - 1$ is called the *embedding degree* of the curve. It is conjectured that when k is not too small, say, at least 6, then the discrete logarithm problem in G_1, i.e. finding α from g and $\alpha \cdot g$, is very hard. (In BN curves [3] currently used by Zcash $k = 12$.)

The multiplicative group of $\mathbb{F}_{p^k}$ contains a subgroup of order r that we denote G_T. We can look at curve points with coordinates in $\mathbb{F}_{p^k}$ and not just in $\mathbb{F}_p$. Under the same addition rule, these points also form a group together with $\mathcal{O}$ called $\mathcal{C}(\mathbb{F}_{p^k})$. Note that $\mathcal{C}(\mathbb{F}_{p^k})$ clearly contains G_1. Besides G_1, $\mathcal{C}(\mathbb{F}_{p^k})$ will contain an additional subgroup G_2 of order r (in fact, $r - 1$ additional subgroups of order r).

Fix generators $g \in G_1, h \in G_2$. It turns out that there is an efficient map, called the *Tate reduced pairing*, taking a pair of elements from G_1 and G_2 into an element of G_T,

such that

1. $\mathrm{Tate}(g, h) = \mathbf{g}$ for a generator $\mathbf{g}$ of G_T, and

2. given a pair of elements $a, b \in \mathbb{F}_r$, we have $\mathrm{Tate}(a \cdot g, b \cdot h) = \mathbf{g}^{ab}$.

Defining Tate is a bit beyond the scope of this series, and relies on concepts from algebraic geometry, most prominently that of *divisors*. Here's a sketch of Tate's definition: [4]

For $a \in \mathbb{F}_p$ the polynomial $(X - a)^r$ has a zero of multiplicity r at the point a, and no other zeroes. For a point $P \in G_1$, divisors enable us to prove there exists a function f_P from the curve to $\mathbb{F}_p$ that also has, in some precise sense, a zero of multiplicity r at P and no other zeroes. $\mathrm{Tate}(P, Q)$ is then defined as $f_P(Q)^{(p^k-1)/r}$.

It may not seem at all clear what this definition has to do with the stated properties, and indeed the proof that Tate has these properties is quite complex.

Defining $E_1(x) := x \cdot g, E_2(x) := x \cdot h, E(x) := x \cdot \mathbf{g}$, we get a weak version of an HH that supports both addition and multiplication: E_1, E_2, E are HHs that support addition, and given the hidings $E_1(x), E_2(y)$ we can compute $E(xy)$. In other words, if we have the "right" hidings of x and y we can get a (different) hiding of xy. But for example, if we had hidings of x, y, z we couldn't get a hiding of xyz.

We move on to discussing non-interactive proof systems. We begin by explaining exactly what we mean by 'non-interactive'.

Non-interactive proofs in the common reference string model

The strongest and most intuitive notion of a non-interactive proof is probably the following. In order to prove a certain claim, a prover broadcasts a single message to all parties, with no prior communication of any kind; and anyone reading this message would be convinced of the prover's claim. This can be shown to be impossible in most cases. [5]

A slightly relaxed notion of non-interactive proof is to allow a common reference string (CRS). In the CRS model, before any proofs are constructed, there is a setup phase where a string is constructed according to a certain randomized process and broadcast to all parties. This string is called the CRS and is then used to help construct and verify proofs. The assumption is that the randomness used in the creation of the CRS is not known to any party – as knowledge of this randomness might enable constructing proofs of false claims.

We will explain how in the CRS model we can convert the verifiable blind evaluation protocol of Part IV into a non-interactive proof system. As the protocol of Part VI consisted of a few such subprotocols it can be turned into a non-interactive proof system in a similar way.

A non-interactive evaluation protocol

The non-interactive version of the evaluation protocol basically consists of publishing Bob's first message as the CRS. Recall that the purpose of the protocol is to obtain the hiding $E(P(s))$ of Alice's polynomial P at a randomly chosen $s \in \mathbb{F}_r$.

Setup: Random $\alpha \in \mathbb{F}_r^*, s \in \mathbb{F}_r$ are chosen and the CRS:

$$(E_1(1), E_1(s), \ldots, E_1(s^d), E_2(\alpha), E_2(\alpha s), \ldots, E_2(\alpha s^d))$$

is published.

Proof: Alice computes $a = E_1(P(s))$ and $b = E_2(\alpha P(S))$ using the elements of the CRS, and the fact that E_1 and E_2 support linear combinations.

Verification: Fix the $x, y \in \mathbb{F}_r$ such that $a = E_1(x)$ and $b = E_2(y)$. Bob computes $E(\alpha x) = \mathrm{Tate}(E_1(x), E_2(\alpha))$ and $E(y) = \mathrm{Tate}(E_1(1), E_2(y))$, and checks that they are equal. (If they are equal it implies $\alpha x = y$.)

As explained in Part IV, Alice can only construct a, b that will pass the verification check if a is the hiding of $P(s)$ for a polynomial P of degree d known to her. The main difference here is that Bob does not need to know α for the verification check, as he can use the pairing function to compute $E(\alpha x)$ only from $E_1(x)$ and $E_2(\alpha)$. Thus, he does

not need to construct and send the first message himself, and this message can simply be fixed in the CRS.

[1] You may ask 'The set of points from where?'. We mean the set of points with coordinates in the algebraic closure of $\mathbb{F}_p$. Also, the curve has an affine and projective version. When we are referring to the projective version we also include the "point at infinity" $\mathcal{O}$ as an element of the curve.

[2] We did not address the case of adding P to itself. This is done by using the line that is tangent to the curve at P, and taking R to be the second intersection point of this line with the curve.

[3] https://eprint.iacr.org/2005/133.pdf

[4] The pairing Zcash actually uses is the optimal Ate pairing, which is based on the Tate reduced pairing, and can be computed more efficiently than Tate.

[5] In computational complexity theory terms, one can show that only languages in BPP have non-interactive zero-knowledge proofs in this strong sense. The type of claims we need to prove in Zcash transactions, e.g. 'I know a hash preimage of this string', correspond to the complexity class NP which is believed to be much larger than BPP.

[6] The images used were taken from the following article and are used under the creative commons license.

Technical
#cryptography, explainers, zksnarks

< New Release: 1.0.9 | Pay-to-sudoku Revisited >