Operational notes

Document updated on **March 27, 2022**.

The following colors are **not** part of the final product, but serve as highlights in the editing/review process:

- text that needs attention from the Subject Matter Experts: Mirco, Anna,& Jan
- terms that have not yet been defined in the book
- things that need to be checked only at the very final typesetting stage
  (and it doesn't make sense to do them before)
- text that needs advice from the communications/marketing team: Aaron & Shane
- text that needs to be completed or otherwise edited (by Sylvia)

Todo list

MoonMath manual

TechnoBob and the Least Scruples crew

March 27, 2022

Contents

Chapter 8

Zero Knowledge Protocols

A so-called **zero-knowledge protocol** is a set of mathematical rules by which one party (usually called **the prover**) can convince another party (usually called **the verifier**) that a given statement is true, while not revealing any additional information apart from the truth of the statement.

As we have seen in chapter 6, given some language L and instance I, the knowledge claim "there is a witness W such that $(I;W)$ is a word in L" is constructively provable by providing W to the verifier. However, the challenge for a zero-knowledge protocol is to prove knowledge of a witness without revealing any information beyond its bare existence.

add reference

In this chapter, we look at various systems that exist to solve this task. We start with an introduction to the basic concepts and terminology in zero-knowledge proving systems and then introduce the so-called Groth_16 protocol as one of the most efficient systems. We plan to update this chapter with new inventions in future versions of this book.

8.1 Proof Systems

From an abstract point of view, a proof system is a set of rules which models the generation and exchange of messages between two parties: a prover and a verifier. Its task is to ascertain whether a given string belongs to a formal language or not.

Proof systems are often classified by certain trust assumptions and the computational capabilities of both parties. In its most general form, the prover usually possesses unlimited computational resources but cannot be trusted, while the verifier has bounded computation power but is assumed to be honest.

Proving the membership statement for some string is executed by the generation of certain messages that are sent between prover and verifier, until the verifier is convinced that the string is an element of the language in consideration.

To be more specific, let Σ be an alphabet, and let L be a formal language defined over Σ. Then a **proof system** for language L is a pair of probabilistic interactive algorithms (P,V), where P is called the **prover** and V is called the **verifier**.

Both algorithms are able to send messages to one another, and each algorithm has its own state, some shared initial state and access to the messages. The verifier is bounded to a number of steps which is polynomial in the size of the shared initial state, after which it stops and outputs either `accept` or `reject` indicating that it accepts or rejects a given string to be in L. In contrast, there are bounds on the computational power of the prover.

When the execution of the verifier algorithm stops the following conditions are required to hold:

- (Completeness) If the tuple $x \in \Sigma^*$ is a word in language L and both prover and verifier follow the protocol, the verifier outputs `accept`.
- (Soundness) If the tuple $x \in \Sigma^*$ is not a word in language L and the verifier follows the protocol, the verifier outputs `reject`, except with some small probability.

In addition, a proof system is called **zero-knowledge** if the verifier learns nothing about x other than $x \in L$.

The previous definition of proof systems is very general, and many subclasses of proving systems are known in the field. The type of languages that a proof system can support crucially depends on the abilities of the verifier (for example, whether it can make random choices) or on the nature and number of the messages that can be exchanged. If the system only requires to send a single message from the prover to the verifier, the proof system is called **non-interactive**, because no interaction other then sending the actual proof is required. In contrast, any other proof system is called **interactive**.

A proof system is usually called **succinct** if the size of the proof is shorter than the witness necessary to generate the proof. Moreover, a proof system is called **computationally sound** if soundness only holds under the assumption that the computational capabilities of the prover are polynomially bound. To distinguish general proofs from computationally sound proofs, the latter are often called **arguments**. Zero-knowledge, succinct, non-interactive arguments of knowledge claims are often abbreviated **zk-SNARKs**.

Example 135 (Constructive Proofs for Algebraic Circuits). To formalize our previous notion of constructive proofs for algebraic circuits, let $\mathbb{F}$ be a finite field, and let $C(\mathbb{F})$ be an algebraic circuit over $\mathbb{F}$ with associated language $L_{C(\mathbb{F})}$. A non-interactive proof system for $L_{C(\mathbb{F})}$ is given by the following two algorithms:

Given some instance I, the prover algorithm P uses its unlimited computational power to compute a witness W such that the pair $(I;W)$ is a valid assignment to $C(\mathbb{F})$ whenever the circuit is satisfiable for I. The prover then sends the constructive proof $(I;W)$ to the verifier.

On receiving a message $(I;W)$, the verifier algorithm V assigns the constructive proof $(I;W)$ to circuit $C(\mathbb{F})$, and decides whether the assignment is valid by executing all gates in the circuit. The runtime is polynomial in the number of gates. If the assignment is valid, the verifier returns `accepts`, if not, it returns `reject`.

To see that this proof system has the completeness and soundness properties, let $C(\mathbb{F})$ be a circuit of the field $\mathbb{F}$, and let I be an instance. The circuit may or may not have a witness W such that $(I;W)$ is a valid assignment to $C(\mathbb{F})$.

If no W exists, I is not part of any word in $L_{C(\mathbb{F})}$, and there is no way for P to generate a valid assignment. If follows that the verifier will not accept any claimed proof sent by P, which implies that the system has **soundness**.

If, on the other hand, W exists and P is honest, P can use its unlimited computational power to compute W and send $(I;W)$ to V, which V will accept in polynomial time. This implies that the system has **completeness**.

The system is non-interactive because the prover only sends a single message to the verifier, which contains the proof itself. Because in this simple system the witness itself is the proof, the proof system is **not** succinct.

8.2 The "Groth16" Protocol

In chapter 6, we have introduced algebraic circuits, their associated rank-1 constraint systems and their induced quadratic arithmetic programs. These models define formal languages, and

check reference

associated memberships as well as knowledge claim can be constructively proofed by executing the circuit to compute a solution to its associated R1CS. The solution can then be transformed into a polynomial such that the polynomial is divisible by another polynomial if and only if the solution is correct.

In Groth [2016], Jens Groth provides a method that can transform those proofs into zero-knowledge succinct non-interactive arguments of knowledge. Assuming that the pairing groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, b)$ are given, the arguments are of constant size and consist of 2 elements from G_1 and a single element from $\mathbb{G}_2$, regardless of the size of the witness. They are zero-knowledge in the sense that the verifier learns nothing about the witness beside the fact that the instance-witness pair is a proper word in the language of the problem.

check reference

Verification is non-interactive and needs to compute a number of exponentiations proportional to the size of the instance, together with 3 group pairings in order to check a single equation.

The generated argument is perfectly zero-knowledge has perfect completeness and soundness in the generic bilinear group model, assuming the existence of a trusted third party that executes a preprocessing phase to generate a common reference string and a simulation trapdoor. This party must be trusted to delete the simulation trapdoor, since everyone in possession of it can simulate proofs.

common reference string

simulation trapdoor

To be more precise, let R be a rank-1 constraint system defined over some finite field $\mathbb{F}_r$. Then **Groth_16 parameters** for R are given by the following set:

$$\texttt{Groth\_16} - \texttt{Param}(R) = (r, \mathbb{G}_1, \mathbb{G}_2, e(\cdot,\cdot), g_1, g_2) \tag{8.1}$$

In the equation above, $\mathbb{G}_1$ and $\mathbb{G}_2$ are finite cyclic groups of order r, g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$ and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate, bilinear pairing for some target group $\mathbb{G}_T$. In real-world applications, the parameter set is usually agreed on in advance.

Given some Groth_16 parameters, a **Groth_16 protocol** is then a quadruple of probabilistic polynomial algorithms $(\textsc{Setup}, \textsc{Prove}, \textsc{Vfy}, \textsc{Sim})$ such that the following conditions hold:

- (Setup-Phase): $(CRS, \tau) \leftarrow \texttt{Setup}(R)$: Algorithm `Setup` takes the R1CS R as input and computes a common reference string CRS and a simulation trapdoor τ.

- (Prover-Phase): $\pi \leftarrow Prove(R, CRS, I, W)$: Given a constructive proof $(I;W)$ for R, algorithm `Prove` takes the R1CS R, the common reference string CRS and the constructive proof (I,W) as input and computes an zk-SNARK π.

- Verify: $\{\texttt{accept}, \texttt{reject}\} \leftarrow Vfy(R, CRS, I, \pi)$: Algorithm `Vfy` takes the R1CS R, the common reference string CRS, the instance I and the zk-SNARK π as input and returns `reject` or `accept`.

- $\pi \leftarrow Sim(R, \tau, CRS, I)$: Algorithm `Sim` takes the R1CS R, the common reference string CRS, the simulation trapdoor τ and the instance I as input and returns a zk-SNARK π.

We will explain these algorithms together with detailed examples in the remainder of this section.

Assuming a trusted third party for the setup, the protocol is able to compute a zk-SNARK from a constructive proof for R, provided that r is sufficiently large, and, in particular, larger than the number of constraints in the associated R1CS.

Example 136 (The 3-Factorization Problem). Consider the 3-factorization problem from 106 and its associated algebraic circuit and rank-1 constraint system from 6.8. In this example,

check reference

check reference

we want to agree on a parameter set $(R, r, \mathbb{G}_1, \mathbb{G}_2, e(\cdot,\cdot), g_1, g_2)$ in order to use the Groth_16 protocol for our 3-factorization problem.

To find proper parameters, first observe that the circuit XXX, as well as its associated R1CS $R_{3.fac\_zk}$ ex:3-factorization-r1cs and the derived QAP 6.14, are defined over the field $\mathbb{F}_{13}$. We therefore have $r = 13$ and need pairing groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order 13.

add reference

check reference

We know from 5.4 that the moon-math curve BLS6_6 has two subgroups $\mathbb{G}_1[13]$ and $\mathbb{G}_2[13]$, which are both of order 13. The associated Weil pairing b (5.45) is a proper bilinear map. We therefore choose those groups and the Weil pairing together with the generators $g_1 = (13, 15)$ and $g_2 = (7v^2, 16v^3)$ of $\mathbb{G}_1[13]$ and $\mathbb{G}_2[13]$, as a parameter:

check reference

check reference

$$\texttt{Groth\_16}-\texttt{Param}(R_{3.fac\_zk}) = (r, \mathbb{G}_1[13], \mathbb{G}_2[13], e(\cdot,\cdot), (13,15), (7v^2, 16v^3))$$

It should be noted that our choice is not unique. Every pair of finite cyclic groups of order 13 that has a proper bilinear pairing qualifies as a Groth_16 parameter set. The situation is similar to real-world applications, where SNARKs with equivalent behavior are defined over different curves, used in different applications.

The Setup Phase To generate zk-SNARKs from constructive knowledge proofs in the Groth16 protocol, a preprocessing phase is required. This has to be executed a single time for every rank-1 constraint system and any associated quadratic arithmetic program. The outcome of this phase is a common reference string that prover and verifier need in order to generate and verify the zk-SNARK. In addition, a simulation trapdoor is produced that can be used to simulate proofs.

To be more precise, let L be a language defined by some rank-1 constraint system R such that a constructive proof of knowledge for an instance $(I_1, \ldots, I_n)$ in L consists of a witness $(W_1, \ldots, W_m)$. Let $QAP(R) = \left\{T \in \mathbb{F}[x], \left\{A_j, B_j, C_j \in \mathbb{F}[x]\right\}_{j=0}^{n+m}\right\}$ be a quadratic arithmetic program associated to R, and let $\{\mathbb{G}_1, \mathbb{G}_2, e(\cdot,\cdot), g_1, g_2, \mathbb{F}_r\}$ be the set of Groth_16 parameters.

The setup phase then samples 5 random, inverible elements α, β, γ, δ and s from the scalar field $\mathbb{F}_r$ of the protocol and outputs the **simulation trapdoor** τ:

"invariable"?

$$\tau = (\alpha, \beta, \gamma, \delta, s) \tag{8.2}$$

In addition, the setup phase uses those 5 random elements together with the two generators g_1 and g_2 and the quadratic arithmetic program to generate a **common reference string** $CRS_{QAP} = (CRS_{\mathbb{G}_1}, CRS_{\mathbb{G}_2})$ of language L:

$$CRS_{\mathbb{G}_1} = \left\{ \begin{array}{c} g_1^{\alpha}, g_1^{\beta}, g_1^{\delta}, \left(g_1^{s^j}, \ldots\right)_{j=0}^{deg(T)-1}, \left(g_1^{\frac{\beta \cdot A_j(s) + \alpha \cdot B_j(s) + C_j(s)}{\gamma}}, \ldots\right)_{j=0}^{n} \\ \left(g_1^{\frac{\beta \cdot A_{j+n}(s) + \alpha \cdot B_{j+n}(s) + C_{j+n}(s)}{\delta}}, \ldots\right)_{j=1}^{m}, \left(g_1^{\frac{s^j \cdot T(s)}{\delta}}, \ldots\right)_{j=0}^{deg(T)-2} \end{array} \right\}$$

$$CRS_{\mathbb{G}_2} = \left\{ g_2^{\beta}, g_2^{\gamma}, g_2^{\delta}, \left(g_2^{s^j}, \ldots\right)_{j=0}^{deg(T)-1} \right\}$$

Common reference strings depend on the simulation trapdoor, and are therefore not unique to the problem. Any language can have more than one common reference string. The size of a common reference string is linear in the size of the instance and the size of the witness.

If a simulation trapdoor $\tau = (\alpha, \beta, \gamma, \delta, s)$ is given, we call the element s a **secret evaluation point** of the protocol, because if $\mathbb{F}_r$ is the scalar field of the finite cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$, then a key feature of any common reference string is that it provides data to compute the evaluation of any polynomial $P \in \mathbb{F}_r[x]$ of degree $deg(P) < deg(T)$ at the point s in the exponent of the generator g_1 or g_2, without knowing s.

To be more precise, let s be the secret evaluation point and let $P(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \ldots a_k \cdot x^k$ be a polynomial of degree $k < deg(T)$ with coefficients in $\mathbb{F}_r$. Then we can compute $g_1^{P(s)}$ without knowing what the actual value of s is:

$$\begin{aligned} g_1^{P(s)} &= g_1^{a_0 \cdot s^0 + a_1 \cdot s^1 + \ldots a_k \cdot s^k} \\ &= g_1^{a_0 \cdot s^0} \cdot g_1 a_1 \cdot s^1 \cdot \ldots \cdot g_1^{a_k \cdot s^k} \\ &= \left(g_1^{s^0}\right)^{a_0} \cdot \left(g_1^{s^1}\right)^{a_1} \cdot \ldots \cdot \left(g_1^{s^k}\right)^{a_k} \end{aligned}$$

In this expression, all group points $g_1^{s^j}$ are part of the common reference string, hence, they can be used to compute the result. The same holds true for the evaluation of $g_2^{P(s)}$, since the $\mathbb{G}_2$ part of the common reference string contains the points $g_2^{s^j}$.

In real-world applications, the simulation trapdoor is often called the **toxic waste** of the setup-phase, while a common reference string is also-called the pair of **prover and verifier key**.

explain why

In order to make the protocol secure, the setup needs to be executed in a way that guarantees that the simulation trapdoor is deleted. Anyone in possession of it can generate arguments without knowledge of a constructive proof. The most simple approach to achieving deletion of the toxic waste is by a so-called **trusted third party**, where the trust assumption is that that the party generates the common reference string precisely as defined and deletes the simulation backdoor afterwards.

However, as trusted third parties are not easy to find, more sophisticated protocols exist in real-world applications. They execute the setup phase as a multi party computation, where the proper execution can be publicly verified and the simulation trapdoor is deleted if at least one participant deletes their individual contribution to the randomness. Each participant only possesses a fraction of the simulation trapdoor, so it can only be recovered if all participants collude and share their fraction.

Example 137 (The 3-factorization Problem). To see how the setup phase of a Groth_16 zk-SNARK can be computed, consider the 3-factorization problem from 106 and the parameters from page 190. As we have seen in 6.14, an associated quadratic arithmetic program is given as follows:

4 examples have the same title. Change it to be distinct

check reference

add reference

check reference

$$\begin{aligned} QAP(R_{3.fac\_zk}) = \{& x^2 + x + 9, \\ & \{0,0,6x+10,0,0,7x+4\}, \{0,0,0,6x+10,7x+4,0\}, \{0,7x+4,0,0,0,6x+10\}\} \end{aligned}$$

To transform this QAP into a common reference string, we choose the field elements $\alpha = 6$, $\beta = 5$, $\gamma = 4$, $\delta = 3$, $s = 2$ from $\mathbb{F}_{13}$. In real-world applications, it is important to sample those values randomly from the scalar field, but in our approach, we choose those non-random values to make them more memorizable, which helps in pen-and-paper computations. Our simulation trapdoor is then given as follows:

$$\tau = (6,5,4,3,2)$$

We keep this secret in order to simulate proofs later on, butwe are careful though to hide τ from anyone who hasn't read this book. Then we instantiate the common reference string XXX

add reference

from those values. Since our groups are subgroups of the `BLS6_6` elliptic curve, we use scalar product notation instead of exponentiation.

To compute the $\mathbb{G}_1$ part of the common reference string, we use the logarithmic order of the group $\mathbb{G}_1$ XXX, the generator $g_1 = (13,15)$, as well as the values from the simulation backdoor. Since $deg(T) = 2$, we get the following:

add reference

$$\begin{aligned}
[\alpha]g_1 &= [6](13,15) = (27,34)\\
[\beta]g_1 &= [5](13,15) = (26,34)\\
[\delta]g_1 &= [3](13,15) = (38,15)
\end{aligned}$$

To compute the rest of the $\mathbb{G}_1$ part of the common reference string, we expand the indexed tuples and insert the secret random elements from the simulation backdoor. We get the following:

$$\begin{aligned}
\Big([s^j]g_1,\ldots\Big)_{j=0}^{1} &= \Big([2^0](13,15),[2^1](13,15)\Big)\\
&= \Big((13,15),(33,34)\Big)\\
\Big([\frac{\beta A_j(s)+\alpha B_j(s)+C_j(s)}{\gamma}]g_1,\ldots\Big)_{j=0}^{1} &= \Big([\frac{5A_0(2)+6B_0(2)+C_0(2)}{4}](13,15),\\
&\quad [\frac{5A_1(2)+6B_1(2)+C_1(2)}{4}](13,15)\Big)\\
\Big([\frac{\beta A_{j+n}(s)+\alpha B_{j+n}(s)+C_{j+n}(s)}{\delta}]g_1,\ldots\Big)_{j=1}^{4} &= \Big([\frac{5A_2(2)+6B_2(2)+C_2(2)}{3}](13,15),\\
&\quad [\frac{5A_3(2)+6B_3(2)+C_3(2)}{3}](13,15),\\
&\quad [\frac{5A_4(2)+6B_4(2)+C_4(2)}{3}](13,15),\\
&\quad [\frac{5A_5(2)+6B_5(2)+C_6(2)}{3}](13,15)\Big)\\
\Big([\frac{s^j\cdot T(s)}{\delta})]g_1\Big)_{j=0}^{0} &= \Big([\frac{2^0\cdot T(2)}{3}](13,15)\Big)
\end{aligned}$$

To compute the curve points on the right side of these expressions, we need the polynomials from the associated quadratic arithmetic program and evaluate them on the secret point $s = 2$.

Since $4^{-1} = 10$ and $3^{-1} = 9$ in $\mathbb{F}_{13}$, we get the following:

$$
\begin{aligned}
[\frac{5A_0(2)+6B_0(2)+C_0(2)}{4}](13,15) =& [(5\cdot 0+6\cdot 0+0)\cdot 10](13,15) = [0](13,14)\\
& \mathcal{O}\\
[\frac{5A_1(2)+6B_1(2)+C_1(2)}{4}](13,15) =& [(5\cdot 0+6\cdot 0+(7\cdot 2+4))\cdot 10](13,15) = [11](13,15) =\\
& (33,9)\\
[\frac{5A_2(2)+6B_2(2)+C_2(2)}{3}](13,15) =& [(5\cdot(6\cdot 2+10)+6\cdot 0+0)\cdot 9](13,15) = [2](13,15) =\\
& (33,34)\\
[\frac{5A_3(2)+6B_3(2)+C_3(2)}{3}](13,15) =& [(5\cdot 0+6\cdot(6\cdot 2+10)+0)\cdot 9](13,15) = [5](13,15) =\\
& (26,34)\\
[\frac{5A_4(2)+6B_4(2)+C_4(2)}{3}](13,15) =& [(5\cdot 0+6\cdot(7\cdot 2+4)+0)\cdot 9](13,15) = [10](13,15) =\\
& (38,28)\\
[\frac{5A_5(2)+6B_5(2)+C_5(2)}{3}](13,15) =& [(5\cdot(7\cdot 2+4)+6\cdot 0+0)\cdot 9](13,15) = [4](13,15) =\\
& (35,28)\\
[\frac{2^0\cdot T(2)}{3}](13,15) =& [1\cdot(2^2+2+9)\cdot 9](13,15) = [5](13,15) =\\
& (26,34)
\end{aligned}
$$

Putting all those values together, we see that the $\mathbb{G}_1$ part of the common reference string is given by the following set of 12 points from the BLS6_6 13-torsion group $\mathbb{G}_1$:

$$
CRS_{\mathbb{G}_1} = \left\{ \begin{array}{c} (27,34),(26,34),(38,15),\Big((13,15),(33,34)\Big),\Big(\mathcal{O},(33,9)\Big) \\ \Big((33,34),(26,34),(38,28),(35,28)\Big),\Big((26,34)\Big) \end{array} \right\}
$$

To compute the $\mathbb{G}_2$ part of the common reference string, we use the logarithmic order of the group $\mathbb{G}_2$ XXX, the generator $g_2 = (7v^2, 16v^3)$, as well as the values from the simulation backdoor. Since $deg(T) = 2$, we get the following:

add reference

$$
\begin{aligned}
[\beta]g_2 &= [5](7v^2,16v^3) = (16v^2,28v^3)\\
[\gamma]g_2 &= [4](7v^2,16v^3) = (37v^2,27v^3)\\
[\delta]g_2 &= [3](7v^2,16v^3) = (42v^2,16v^3)
\end{aligned}
$$

To compute the rest of the $\mathbb{G}_2$ part of the common reference string, we expand the indexed tuple and insert the secret random elements from the simulation backdoor. We get the following:

$$
\begin{aligned}
\Big([s^j]g_2,\ldots\Big)_{j=0}^{1} &= \Big([2^0](7v^2,16v^3),[2^1](7v^2,16v^3)\Big)\\
&= \Big((7v^2,16v^3),(10v^2,28v^3)\Big)
\end{aligned}
$$

Putting all these values together, we see that the $\mathbb{G}_2$ part of the common reference string is given by the following set of 5 points from the BLS6_6 13-torsion group $\mathbb{G}_2$:

$$
CRS_{\mathbb{G}_2} = \Big\{(16v^2,28v^3),(37v^2,27v^3),(42v^2,16v^3),\Big(7v^2,16v^3),(10v^2,28v^3)\Big)\Big\}
$$

Given the simluation trapdoor τ and the quadratic arithmetic program 6.14, the associated common reference string of the 3-factorization problem is as follows:

check reference

$$CRS_{\mathbb{G}_1} = \left\{ \begin{array}{c} (27,34),(26,34),(38,15),\Big((13,15),(33,34)\Big),\Big(\mathcal{O},(33,9)\Big) \\ \Big((33,34),(26,34),(38,28),(35,28)\Big),\Big((26,34)\Big) \end{array} \right\}$$

$$CRS_{\mathbb{G}_2} = \Big\{(16v^2,28v^3),(37v^2,27v^3),(42v^2,16v^3),\Big(7v^2,16v^3),(10v^2,28v^3)\Big)\Big\}$$

We then publish this data to everyone who wants to participate in the generation of zk-SNARKs or the verification of the 3-factorization problem.

To understand how this common reference string can be used to evaluate polynomials at the secret evaluation point in the exponent of a generator, let's assume that we have deleted the simulation trapdoor. In that case, we have no way to know the secret evaluation point anymore, hence, we cannot evaluate polynomials at that point. However, we can evaluate polynomials of smaller degree than the degree of the target polynomial in the exponent of both generators at that point.

To see that, consider e.g. the polynomials $A_2(x) = 6x+10$ and $A_5(x) = 7x+4$ from the QAP of this problem. To evaluate these polynomials in the exponent of g_1 and g_2 at the secret point s without knowing the value of s (which is 2), we can use the common reference string and equation XXX. Using the scalar product notation instead of exponentiation, we get the following:

add reference

$$\begin{aligned} [A_2(s)]g_1 &= [6\cdot s^1 + 10\cdot s^0]g_1 \\ &= [6](33,34)+[10](13,15) && \#\ [s^0]g_1 = (13,15), [s^1]g_1 = (33,34) \\ &= [6\cdot 2](13,15)+[10](13,15) = [9](13,15) && \#\ \text{logarithmic order on } \mathbb{G}_1 \\ &= (35,15) \\ [A_5(s)]g_1 &= [7\cdot s^1 + 4\cdot s^0]g_1 \\ &= [7](33,34)+[4](13,15) \\ &= [7\cdot 2](13,15)+[4](13,15) = [5](13,15) \\ &= (26,34) \end{aligned}$$

Indeed, we are able to evaluate the polynomials in the exponent at a secret evaluation point, because that point is encrypted in the curve point (33,34) and its secrecy is protected by the discrete logarithm assumption. Of course, in our computation, we recovered the secret point $s = 2$, but that was only possible because we have a group of logarithmic order in order to simplify our pen-and-paper computations. Such an order is infeasible for computing in cryptographically secure curves. We can do the same computation on $\mathbb{G}_2$ and get the following:

$$\begin{aligned} [A_2(s)]g_2 &= [6\cdot s^1 + 10\cdot s^0]g_2 \\ &= [6](10v^2,28v^3)+[10](7v^2,16v^3) \\ &= [6\cdot 2](7v^2,16v^3)+[10](7v^2,16v^3) = [9](7v^2,16v^3) \\ &= (37v^2,16v^3) \\ [A_5(s)]g_2 &= [7\cdot s^1 + 4\cdot s^0]g_1 \\ &= [7](10v^2,28v^3)+[4](7v^2,16v^3) \\ &= [7\cdot 2](7v^2,16v^3)+[4](7v^2,16v^3) = [5](7v^2,16v^3) \\ &= (16v^2,28v^3) \end{aligned}$$

Apart from the target polynomial T, all other polynomials of the quadratic arithmetic program can be evaluated in the exponent this way.

The Prover Phase Given some rank-1 constraint system R and instance $I = (I_1, \ldots, I_n)$, the task of the prover phase is to convince any verifier that a prover knows a witness W to instance I such that $(I;W)$ is a word in the language L_R of the system, without revealing anything about W.

To achieve this in the Groth_16 protocol, we assume that any prover has access to the rank-1 constraint system of the problem, in addition to some algorithm that tells the prover how to compute constructive proofs for the R1CS. In addition, the prover has access to a common reference string and its associated quadratic arithmetic program.

In order to generate a zk-SNARK for this instance, the prover first computes a valid constructive proof as explained in XXX, that is, the prover generates a proper witness $W = (W_1, \ldots, W$ add reference such that $(I_1, \ldots, I_n; W_1, \ldots, W_m)$ is a solution to the rank-1 constraint system R.

The prover then uses the quadratic arithmetic program and computes the polynomial $P_{(I;W)}$, as explained in 6.15. They then divide $P_{(I;W)}$ by the target polynomial T of the quadratic arithmetic. check reference Since $P_{(I;W)}$ is constructed from a valid solution to the R1CS, we know from 6.15 that it is divisible by T. check reference This implies that polynomial division of P by T generates another polynomial $H := P/T$, with $deg(H) < deg(T)$.

The prover then evaluates the polynomial $(H \cdot T)\delta^{-1}$ in the exponent of the generator g_1 at the secret point s, as explained in XXX. add reference To see how this can be achieved, let $H(x)$ be the quotient polynomial P/T:

$$H(x) = H_0 \cdot x^0 + H_1 \cdot x^1 + \ldots + H_k \cdot x^k \tag{8.3}$$

To evaluate $H \cdot T$ at s in the exponent of g_1, the prover uses the common reference string and computes as follows:

$$g_1^{\frac{H(s)\cdot T(s)}{\delta}} = \left(g_1^{\frac{s^0\cdot T(s)}{\delta}}\right)^{H_0} \cdot \left(g_1^{\frac{s^1\cdot T(s)}{\delta}}\right)^{H_1} \cdots \left(g_1^{\frac{s^k\cdot T(s)}{\delta}}\right)^{H_k}$$

After this has been done, the prover samples two random field elements $r, t \in \mathbb{F}_r$, and uses the common reference string, the instance variables I_1, …, I_n and the witness variables W_1, …, W_m to compute the following curve points:

$$\begin{aligned}
g_1^W &= \left(g_1^{\frac{\beta\cdot A_{1+n}(s)+\alpha\cdot B_{1+n}(s)+C_{1+n}(s)}{\delta}}\right)^{W_1} \cdots \left(g_1^{\frac{\beta\cdot A_{m+n}(s)+\alpha\cdot B_{m+n}(s)+C_{m+n}(s)}{\delta}}\right)^{W_m}\\
g_1^A &= g_1^\alpha \cdot g_1^{A_0(s)} \cdot \left(g_1^{A_1(s)}\right)^{I_1} \cdots \left(g_1^{A_n(s)}\right)^{I_n} \cdot \left(g_1^{A_{n+1}(s)}\right)^{W_1} \cdots \left(g_1^{A_{n+m}(s)}\right)^{W_m} \cdot \left(g_1^\delta\right)^r\\
g_1^B &= g_1^\beta \cdot g_1^{B_0(s)} \cdot \left(g_1^{B_1(s)}\right)^{I_1} \cdots \left(g_1^{B_n(s)}\right)^{I_n} \cdot \left(g_1^{B_{n+1}(s)}\right)^{W_1} \cdots \left(g_1^{B_{n+m}(s)}\right)^{W_m} \cdot \left(g_1^\delta\right)^t\\
g_2^B &= g_2^\beta \cdot g_2^{B_0(s)} \cdot \left(g_2^{B_1(s)}\right)^{I_1} \cdots \left(g_2^{B_n(s)}\right)^{I_n} \cdot \left(g_2^{B_{n+1}(s)}\right)^{W_1} \cdots \left(g_2^{B_{n+m}(s)}\right)^{W_m} \cdot \left(g_2^\delta\right)^t\\
g_1^C &= g_1^W \cdot g_1^{\frac{H(s)\cdot T(s)}{\delta}} \cdot \left(g_1^A\right)^t \cdot \left(g_1^B\right)^r \cdot \left(g_1^\delta\right)^{-r\cdot t}
\end{aligned}$$

In this computation, the group elements $g_1^{A_j(s)}$, $g_1^{B_j(s)}$ and $g_2^{B_j(s)}$ can be derived from the common reference string and the quadratic arithmetic program of the problem, as we have seen in XXX. add reference In fact, those points only have to be computed once, and can be published and reused for multiple proof generations because they are the same for all instances and witnesses. All other group elements are part of the common reference string.

After all these computations have been done, a valid zero-knowledge succinct non-interactive argument of knowledge π in the Groth_16 protocol is given by the following three curve points:

$$\pi = (g_1^A, g_1^C, g_2^B) \tag{8.4}$$

As we can see, a Groth_16 zk-SNARK consists of 3 curve points, two points from $\mathbb{G}_1$ and 1 point from $\mathbb{G}_2$. The argument is specifically designed this way because, in typical applications, $\mathbb{G}_1$ is a torsion group of an elliptic curve over some prime field, while $\mathbb{G}_2$ is a subgroup of a torsion group over an extension field. Elements from $\mathbb{G}_1$ therefore need less space to be stored, and computations in $\mathbb{G}_1$ are typically faster then in $\mathbb{G}_2$.

Since the witness is encoded in the exponent of a generator of a cryptographically secure elliptic curve, it is hidden from anyone but the prover. Moreover, since any proof is randomized by the occurrence of the random field elements r and t, proofs are not unique to any given witness. This is an important feature because, if all proofs for the same witness would be the same, knowledge of a witness would destroy the zero-knowledge property of those proofs.

Example 138 (The 3-factorization Problem). To see how a prover might compute a zk-SNARK, consider the 3-factorization problem from 106, our protocol parameters from XXX as well as the common reference string from XXX.

check reference

add reference

Our task is to compute a zk-SNARK for the instance $I_1 = 11$ and its constructive proof $(W_1, W_2, W_3, W_4) = (2, 3, 4, 6)$ as computed in XXX. As we know from 6.15, the associated polynomial $P_{(I;W)}$ of the quadratic arithmetic program from XXX is given by the following equation:

add reference

$$P_{(I;W)} = x^2 + x + 9$$

add reference

Since $P_{(I;W)}$ is identical to the target polynomial $T(x) = x^2 + x + 9$ in this example, we know from XXX that the quotient polynomial $H = P/T$ is the constant degree 0 polynomial:

check reference

$$H(x) = H_0 \cdot x^0 = 1 \cdot x^0$$

add reference

We therefore use $[\frac{s^0 \cdot T(s)}{\delta}]g_1 = (26, 34)$ from our common reference string XXX of the 3-factorization problem and compute as follows:

add reference

$$\begin{aligned}[\frac{H(s) \cdot T(s)}{\delta}]g_1 &= [H_0](26, 34) = [1](26, 34) \\ &= (26, 34)\end{aligned}$$

add reference

In the next step, we have to compute all group elements required for a proper Groth16 zk-SNARK. We start with g_1^W. Using scalar products instead of the exponential notation, and $\oplus$ for the group law on the `BLS6_6` curve, we have to compute the point $[W]g_1$:

$$\begin{aligned}[W]g_1 = \; & [W_1]g_1^{\frac{\beta \cdot A_2(s) + \alpha \cdot B_2(s) + C_2(s)}{\delta}} \oplus [W_2]g_1^{\frac{\beta \cdot A_3(s) + \alpha \cdot B_3(s) + C_3(s)}{\delta}} \oplus [W_3]g_1^{\frac{\beta \cdot A_4(s) + \alpha \cdot B_4(s) + C_4(s)}{\delta}} \\ & \oplus [W_4]g_1^{\frac{\beta \cdot A_5(s) + \alpha \cdot B_5(s) + C_5(s)}{\delta}}\end{aligned}$$

To compute this point, we have to remember that a prover should not be in possession of the simulation trapdoor, hence, they do not know what α, β, δ and s are. In order to compute this group element, the prover therefore needs the common reference string. Using the logarithmic order from XXX and the witness, we get the following:

add reference

$$\begin{aligned}[W]g_1 &= [2](33,34)\oplus[3](26,34)\oplus[4](38,28)\oplus[6](35,28)\\ &= [2\cdot 2](13,15)\oplus[3\cdot 5](13,15)\oplus[4\cdot 10](13,15)\oplus[6\cdot 4](13,15)\\ &= [2\cdot 2+3\cdot 5+4\cdot 10+6\cdot 4](13,15) = [5](13,15)\\ &= (26,34)\end{aligned}$$

In a next step, we compute g_1^A. We sample the random point $r = 11$ from $\mathbb{F}_{13}$, using scalar products instead of the exponential notation, and $\oplus$ for the group law on the `BLS6_6` curve. We then have to compute the following expression:

$$\begin{aligned}[A]g_1 = \ & [\alpha]g_1\oplus[A_0(s)]g_1\oplus[I_1][A_1(s)]g_1\oplus[W_1][A_2(s)]g_1\oplus[W_2][A_3(s)]g_1\\ &\oplus[W_3][A_4(s)]g_1\oplus[W_4][A_5(s)]g_1\oplus[r][\delta]g_1\end{aligned}$$

Since we don’t know what α, δ and s are, we look up $[\alpha]g_1$ and $[\delta]g_1$ from the common reference string. Recall from XXX that we can evaluate $[A_j(s)]g_1$ without knowling the secret evaluation point s. According to XXX, we have $[A_2(s)]g_1 = (35,15)$, $[A_5(s)]g_1 = (26,34)$ and $[A_j(s)]g_1 = \mathcal{O}$ for all other indices $0\le j\le 5$. Since $\mathcal{O}$ is the neutral element on $\mathbb{G}_1$, we get the following:

add reference

add reference

$$\begin{aligned}[A]g_1 &= (27,34)\oplus\mathcal{O}\oplus[11]\mathcal{O}\oplus[2](35,15)\oplus[3]\mathcal{O}\oplus[4]\mathcal{O}\oplus[6](26,34)\oplus[11](38,15)\\ &= (27,34)\oplus[2](35,15)\oplus[6](26,34)\oplus[11](38,15)\\ &= [6](13,15)\oplus[2\cdot 9](13,15)\oplus[6\cdot 5](13,15)\oplus[11\cdot 3](13,15)\\ &= [6+2\cdot 9+6\cdot 5+11\cdot 3](13,15) = [9](13,15)\\ &= (35,15)\end{aligned}$$

In order to compute the two curve points $[B]g_1$ and $[B]g_2$, we sample another random element $t = 4$ from $\mathbb{F}_{13}$. Using the scalar product instead of the exponential notation, and $\oplus$ for the group law on the `BLS6_6` curve, we have to compute the following expressions:

$$\begin{aligned}[B]g_1 = \ & [\beta]g_1\oplus[B_0(s)]g_1\oplus[I_1][B_1(s)]g_1\oplus[W_1][B_2(s)]g_1\oplus[W_2][B_3(s)]g_1\\ &\oplus[W_3][B_4(s)]g_1\oplus[W_4][B_5(s)]g_1\oplus[t][\delta]g_1\\ [B]g_2 = \ & [\beta]g_2\oplus[B_0(s)]g_2\oplus[I_1][B_1(s)]g_2\oplus[W_1][B_2(s)]g_2\oplus[W_2][B_3(s)]g_2\\ &\oplus[W_3][B_4(s)]g_2\oplus[W_4][B_5(s)]g_2\oplus[t][\delta]g_2\end{aligned}$$

Since we don’t know what β, δ and s are, we look up the associated group elements from the common reference string. Recall from XXX that we can evaluate $[B_j(s)]g_1$ without knowing the secret evaluation point s. Since $B_3 = A_2$ and $B_4 = A_5$, we have $[B_3(s)]g_1 = (35,15)$, $[B_4(s)]g_1 = (26,34)$ according to XXX, and $[B_j(s)]g_1 = \mathcal{O}$ for all other indices $0\le j\le 5$. Since $\mathcal{O}$ is the neutral element on $\mathbb{G}_1$, we get the following:

add reference

add reference

$$\begin{aligned}[B]g_1 &= (26,34)\oplus\mathcal{O}\oplus[11]\mathcal{O}\oplus[2]\mathcal{O}\oplus[3](35,15)\oplus[4](26,34)\oplus[6]\mathcal{O}\oplus[4](38,15)\\ &= (26,34)\oplus[3](35,15)\oplus[4](26,34)\oplus[4](38,15)\\ &= [5](13,15)\oplus[3\cdot 9](13,15)\oplus[4\cdot 5](13,15)\oplus[4\cdot 3](13,15)\\ &= [5+3\cdot 9+4\cdot 5+4\cdot 3](13,15) = [12](13,15)\\ &= (13,28)\end{aligned}$$

$$\begin{aligned}[B]g_2 &= (16v^2,28v^3)\oplus\mathcal{O}\oplus[11]\mathcal{O}\oplus[2]\mathcal{O}\oplus[3](37v^2,16v^3)\oplus[4](16v^2,28v^3)\oplus[6]\mathcal{O}\oplus[4](42v^2,16v^3)\\
&= (16v^2,28v^3)\oplus[3](37v^2,16v^3)\oplus[4](16v^2,28v^3)\oplus[4](42v^2,16v^3)\\
&= [5](7v^2,16v^3)\oplus[3\cdot 9](7v^2,16v^3)\oplus[4\cdot 5](7v^2,16v^3)\oplus[4\cdot 3](7v^2,16v^3)\\
&= [5+3\cdot 9+4\cdot 5+4\cdot 3](7v^2,16v^3) = [12](7v^2+16v^3)\\
&= (7v^2,27v^3)\end{aligned}$$

In a last step, we combine the previous computations to compute the point $[C]g_1$ in the group $\mathbb{G}_1$ as follows:

$$\begin{aligned}[C]g_1 &= [W]g_1\oplus[\frac{H(s)\cdot T(s)}{\delta}]g_1\oplus[t][A]g_1\oplus[r][B]g_1\oplus[-r\cdot t][\delta]g_1\\
&= (26,34)\oplus(26,34)\oplus[4](35,15)\oplus[11](13,28)\oplus[-11\cdot 4](38,15)\\
&= [5](13,15)\oplus[5](13,15)\oplus[4\cdot 9](13,15)\oplus[11\cdot 12](13,15)\oplus[-11\cdot 4\cdot 3](13,15)\\
&= [5+5+4\cdot 9+11\cdot 12-11\cdot 4\cdot 3](13,15) = [7](13,15)\\
&= (27,9)\end{aligned}$$

Given the instance $I_1 = 11$, we can now combine these computations and see that the following 3 curve points are a zk-SNARK for the witness $(W_1,W_2,W_3,W_4) = (2,3,4,6)$:

$$\pi = ((35,15),(27,9),(7v^2,27v^3))$$

We ca now publish this zk-SNARK, or send it to a designated verifier. Note that, if we had sampled different values for r and t, we would have computed a different SNARK for the same witness. The SNARK, therefore, hides the witness perfectly, which means that it is impossible to reconstruct the witness from the SNARK.

The Verification Phase Given some rank-1 constraint system R, instance $I = (I_1,\ldots,I_n)$ and zk-SNARK π, the task of the verification phase is to check that π is indeed an argument for a constructive proof. Assuming that the simulation trapdoor does not exists anymore and the verification checks the proof, the verifier can be convinced that someone knows a witness $W = (W_1,\ldots,W_m)$ such that $(I;W)$ is a word in the language of R.

To achieve this in the Groth16 protocol, we assume that any verifier is able to compute the pairing map $e(\cdot,\cdot)$ efficiently, and has access to the common reference string used to produce the SNARK π. In order to verify the SNARK with respect to the instance $(I_1,\ldots,I_n)$, the verifier computes the following curve point:

$$g_1^I = \left(g_1^{\frac{\beta\cdot A_0(s)+\alpha\cdot B_0(s)+C_0(s)}{\gamma}}\right)\cdot\left(g_1^{\frac{\beta\cdot A_1(s)+\alpha\cdot B_1(s)+C_1(s)}{\gamma}}\right)^{I_1}\cdots\left(g_1^{\frac{\beta\cdot A_n(s)+\alpha\cdot B_n(s)+C_n(s)}{\gamma}}\right)^{I_n}$$

With this group element, the verifier is able to verify the SNARK $\pi = (g_1^A, g_1^C, g_2^B)$ by checking the following equation using the pairing map:

$$e(g_1^A, e_2^B) = e(g_1^\alpha, g_2^\beta)\cdot e(g_1^I, g_2^\gamma)\cdot e(g_1^C, g_2^\delta) \qquad (8.5)$$

If the equation holds true, the SNARK is accepted. If the equation does not hold, the SNARK is rejected.

Remark 5. We know from chapter 5 that computing pairings in cryptographically secure pairing groups is computationally expensive. As we can see, in the Groth16 protocol, 3 pairings are required to verify the SNARK, because the pairing $e(g_1^{\alpha}, g_2^{\beta})$ is independent of the proof, meaning that can be computed once and then stored as an amendment to the verifier key.

check reference

In Groth [2016], the author showed that 2 is the minimal amount of pairings that any protocol with similar properties has to use. This protocol is therefore close to the theoretical minimum. In the same paper, the author outlined an adaptation that only uses 2 pairings. However, that reduction comes with the price of much more overhead computation. Having 3 pairings is therefore a compromise that gives the overall best performance. To date, the Groth16 protocol is the most efficient in its class.

Example 139 (The 3-factorization Problem). To see how a verifier might check a zk-SNARK for some given instance I, consider the 3-factorization problem from 106, our protocol parameters from XXX, the common reference string from XXX as well as the zk-SNARK $\pi = ((35,15),(27,9),(7v^2,27v^3))$, which claims to be an argument of knowledge for a witness for the instance $I_1 = 11$.

check reference

add reference

add reference

In order to verify the zk-SNARK for that instance, we first compute the curve point g_1^I. Using scalar products instead of the exponential notation, and $\oplus$ for the group law on the `BLS6_6` curve, we have to compute the point $[I]g_1$ as follows:

$$[I]g_1 =[\frac{\beta\cdot A_0(s)+\alpha\cdot B_0(s)+C_0(s)}{\gamma}]g_1 \oplus [I_1][\frac{\beta\cdot A_1(s)+\alpha\cdot B_1(s)+C_1(s)}{\gamma}]g_1$$

To compute this point, we have to remember that a verifier should not be in possession of the simulation trapdoor, which means that they do not know what α, β, γ and s are. In order to compute this group element, the verifier therefore needs the common reference string. Using the logarithmic order from XXX and instance I_1, we get the following:

add reference

$$\begin{aligned}[I]g_1 &= [\frac{\beta\cdot A_0(s)+\alpha\cdot B_0(s)+C_0(s)}{\gamma}]g_1 \oplus [I_1][\frac{\beta\cdot A_1(s)+\alpha\cdot B_1(s)+C_1(s)}{\gamma}]g_1\\ &= \mathcal{O}\oplus[11](33,9)\\ &= [11\cdot 11](13,15) = [4](13,15)\\ &= (35,28)\end{aligned}$$

In the next step, we have to compute all the pairings involved in equation XXX. Using the logarithmic order on $\mathbb{G}_1$ and $\mathbb{G}_2$ as well as the bilinearity property of the pairing map we get the following:

add reference

$$\begin{aligned}
e([A]g_1,[B]g_2) &= e((35,15),(7v^2,27v^3)) = e([9](13,15),[12](7v^2,16v^3))\\
&= e((13,15),(7v^2,16v^3))^{9\cdot 12}\\
&= e((13,15),(7v^2,16v^3))^{108}\\
e([\alpha]g_1,[\beta]g_2) &= e((27,34),(16v^2,28v^3)) = e([6](13,15),[5](7v^2,16v^3))\\
&= e((13,15),(7v^2,16v^3))^{6\cdot 5}\\
&= e((13,15),(7v^2,16v^3))^{30}\\
e([I]g_1,[\gamma]g_2) &= e((35,28),(37v^2,27v^3)) = e([4](13,15),[4](7v^2,16v^3))\\
&= e((13,15),(7v^2,16v^3))^{4\cdot 4}\\
&= e((13,15),(7v^2,16v^3))^{16}\\
e([C]g_1,[\delta]g_2) &= e((27,9),(42v^2,16v^3)) = e([7](13,15),[3](7v^2,16v^3))\\
&= e((13,15),(7v^2,16v^3))^{7\cdot 3}\\
&= e((13,15),(7v^2,16v^3))^{21}
\end{aligned}$$

In order to check equation XXX, observe that the target group $\mathbb{G}_T$ of the Weil pairing is a finite cyclic group of order 13. Exponentiation is therefore done in modular 13 arithmetics. Accordingly, since 108 mod 13 = 4, we evaluate the left side of equation XXX as follows:

add reference

add reference

$$e([A]g_1,[B]g_2) = e((13,15),(7v^2,16v^3))^{108} = e((13,15),(7v^2,16v^3))^4$$

Similarly, we evaluate the right side of equation XXX using modular 13 arithmetics and the exponential law $a^x \cdot a^y = a^{x+y}$:

add reference

$$\begin{aligned}
e([\alpha]g_1,[\beta]g_2)\cdot e([I]g_1,[\gamma]g_2)\cdot e([C]g_1,[\delta]g_2) &=\\
e((13,15),(7v^2,16v^3))^{30}\cdot e((13,15),(7v^2,16v^3))^{16}\cdot e((13,15),(7v^2,16v^3))^{21} &=\\
e((13,15),(7v^2,16v^3))^{4}\cdot e((13,15),(7v^2,16v^3))^{3}\cdot e((13,15),(7v^2,16v^3))^{8} &=\\
e((13,15),(7v^2,16v^3))^{4+3+8} &=\\
e((13,15),(7v^2,16v^3))^{2}
\end{aligned}$$

As we can see, both the left and the right side of equation XXX are identical, which implies that the verification process accepts the simulated proof.

add reference

NOTE: UNFORTUNATELY NOT! :-((HENCE THERE IS AN ERROR SOMEWHERE ... NEED TO FIX IT AFTER VACATION

fix error

Proof Simulation During the execution of a setup phase, a common reference string is generated, along with a simulation trapdoor, the latter of which must be deleted at the end of the setup-phase. As an alternative, a more complicated multi-party protocol like [XXX] can be used to split the knowledge of the simulation trapdoor among many different parties.

add reference

In this paragraph, we will show why knowledge of the simulation trapdoor is problematic, and how it can be used to generate zk-SNARKs for a given instance without any knowledge or the existence of associated witness.

To be more precise, let I be an instance for some R1CS language L_R. We call a zk-SNARK for L_R **forged** or **simulated** if it passes a verification but its generation does not require the existence of a witness W such that $(I;W)$ is a word in L_R.

To see how simulated zk-SNARKs can be computed, assume that a forger has knowledge of proper Groth_16 parameters, a quadratic arithmetic program of the problem, a common reference string and its associated simulation trapdoor τ:

$$\tau = (\alpha, \beta, \gamma, \delta, s) \tag{8.6}$$

Given some instance I, the forger's task is to generate a zk-SNARK for this instance that passes the verification process, without having access to any other zk-SNARKs for this instance and without knowledge of a valid witness W.

To achieve this in the Groth_16 protocol, the forger can use the simulation trapdoor in combination with the QAP and two arbitrary field elements A and B from the scalar field $\mathbb{F}_r$ of the pairing groups to g_1^C compute for the instance $(I_1, \ldots, I_n)$ as follows:

$$g_1^C = g_1^{\frac{A\cdot B}{\delta}} \cdot g_1^{-\frac{\alpha\cdot\beta}{\delta}} \cdot g_1^{-\frac{\beta A_0(s)+\alpha B_0(s)+C_0(s)}{\delta}} \cdot \left(g_1^{-\frac{\beta A_1(s)+\alpha B_1(s)+C_1(s)}{\delta}}\right)^{I_1} \cdots \left(g_1^{-\frac{\beta A_n(s)+\alpha B_n(s)+C_n(s)}{\delta}}\right)^{I_n}$$

The forger then publishes the zk-SNARK $\pi_{forged} = (g_1^A, g_1^C, g_2^B)$, which will pass the verification process and is computable without the existence of a witness $(W_1, \ldots, W_m)$.

To see that the simulation trapdoor is necessary and sufficient to compute the simulated proof π_{forged}, first observe that both generators g_1 and g_2 are known to the forger, as they are part of the common reference string, encoded as $g_1^{s^0}$ and $g_2^{s^0}$. The forger is therefore able to compute $g_1^{A\cdot B}$. Moreover, since the forger knows α, β, δ and s from the trapdoor, they are able to compute all factors in the computation of g_1^C.

If, on the other hand, the simulation trapdoor is unknown, it is not possible to compute g_1^C, since, for example, the computational Diffie-Hellman assumption makes the derivation of $g_1^{\alpha\cdot\beta}$ from g_1^α and g_1^β infeasible.

Example 140 (The 3-factorization Problem). To see how a forger might simulate a zk-SNARK for some given instance I, consider the 3-factorization problem from 106, our protocol parameters from XXX, the common reference string from XXX and the simulation trapdoor $\tau = (6,5,4,3,2)$ of that CRS.

check reference

add reference

add reference

add reference

In order to forge a zk-SNARK for instance $I_1 = 11$, we don't need a constructive proof for the associated rank-1 constraint system, which implies that we don't have to execute the circuit $C_{3.fac}(\mathbb{F}_{13})$. Instead, we have to choose 2 arbitrary elements A and B from $\mathbb{F}_{13}$, and compute g_1^A, g_2^B and g_1^C as defined in XXX. We choose $A = 9$ and $B = 3$, and, since $\delta^{-1} = 3$, we compute as follows:

$$\begin{aligned}
[A]g_1 =& [9](13,15) = (35,15)\\
[B]g_2 =& [3](7v^2, 16v^3) = (42v^2, 16v^3)\\
[C]g_1 =& [\frac{A\cdot B}{\delta}]g_1 \oplus [-\frac{\alpha\cdot\beta}{\delta}]g_1 \oplus [-\frac{\beta A_0(s)+\alpha B_0(s)+C_0(s)}{\delta}]g_1 \oplus\\
& [I_1][-\frac{\beta A_1(s)+\alpha B_1(s)+C_1(s)}{\delta}]g_1\\
=& [(9\cdot 3)\cdot 9](13,15) \oplus [-(6\cdot 5)\cdot 9](13,15) \oplus [0](13,15) \oplus [11][-(7\cdot 2+4)\cdot 9](13,15)\\
=& [9](13,15) \oplus [3](13,15) \oplus [12](13,15) = [11](13,15)\\
=& (33,9)
\end{aligned}$$

This is all we need to generate our forged proof for the 3-factorization problem. We publish the simulated zk-SNARK:

$$\pi_{fake} = ((35,15),(33,9),(42v^2,16v^3))$$

Despite the fact that this zk-SNARK was generated without knowledge of a proper witness, it is indistinguishable from a zk-SNARK that proves knowledge of a proper witness.

To see that, we show that our forged SNARK passes the verification process. In order to verify π_{fake}, we proceed as in XXX, and compute the curve point g_1^I for the instance $I_1 = 11$. Since the instance is the same as in example XXX, we can parallel the computation from XXX:

add reference

add reference

add reference

$$\begin{aligned}[I]g_1 &= [\frac{\beta \cdot A_0(s) + \alpha \cdot B_0(s) + C_0(s)}{\gamma}]g_1 \oplus [I_1][\frac{\beta \cdot A_1(s) + \alpha \cdot B_1(s) + C_1(s)}{\gamma}]g_1 \\ &= (35,28)\end{aligned}$$

In a next step we have to compute all the pairings involved in equation XXX. Using the logarithmic order on $\mathbb{G}_1$ and $\mathbb{G}_2$ as well as the bilinearity property of the pairing map we get

add reference

$$\begin{aligned}
e([A]g_1,[B]g_2) &= e((35,15),(42v^2,16v^3)) = e([9](13,15),[3](7v^2,16v^3)) \\
&= e((13,15),(7v^2,16v^3))^{9\cdot 3} \\
&= e((13,15),(7v^2,16v^3))^{27} \\
e([\alpha]g_1,[\beta]g_2) &= e((27,34),(16v^2,28v^3)) = e([6](13,15),[5](7v^2,16v^3)) \\
&= e((13,15),(7v^2,16v^3))^{6\cdot 5} \\
&= e((13,15),(7v^2,16v^3))^{30} \\
e([I]g_1,[\gamma]g_2) &= e((35,28),(37v^2,27v^3)) = e([4](13,15),[4](7v^2,16v^3)) \\
&= e((13,15),(7v^2,16v^3))^{4\cdot 4} \\
&= e((13,15),(7v^2,16v^3))^{16} \\
e([C]g_1,[\delta]g_2) &= e((33,9),(42v^2,16v^3)) = e([11](13,15),[3](7v^2,16v^3)) \\
&= e((13,15),(7v^2,16v^3))^{11\cdot 3} \\
&= e((13,15),(7v^2,16v^3))^{33}
\end{aligned}$$

In order to check equation XXX, observe that the target group $\mathbb{G}_T$ of the Weil pairing is a finite cyclic group of order 13. Exponentiation is therefore done in modular 13 arithmetics. Using this, we evaluate the left side of equation XXX as follows:

add reference

add reference

$$e([A]g_1,[B]g_2) = e((13,15),(7v^2,16v^3))^{27} = e((13,15),(7v^2,16v^3))^1$$

since 27 mod 13 = 1. Similarly, we evaluate the right side of equation XXX using modular 13 arithmetics and the exponential law $a^x \cdot a^y = a^{x+y}$. We get

add reference

$$\begin{aligned}
e([\alpha]g_1,[\beta]g_2)\cdot e([I]g_1,[\gamma]g_2)\cdot e([C]g_1,[\delta]g_2) &= \\
e((13,15),(7v^2,16v^3))^{30}\cdot e((13,15),(7v^2,16v^3))^{16}\cdot e((13,15),(7v^2,16v^3))^{33} &= \\
e((13,15),(7v^2,16v^3))^{4}\cdot e((13,15),(7v^2,16v^3))^{3}\cdot e((13,15),(7v^2,16v^3))^{7} &= \\
e((13,15),(7v^2,16v^3))^{4+3+7} &= \\
e((13,15),(7v^2,16v^3))^{1} &
\end{aligned}$$

As we can see, both the left and the right side of equation XXX are identical, which implies that the verification process accepts the simulated proof. π_{fake} therefore convinces the verifier that a witness to the 3-factorization problem exists. However, no such witness was really necessary to generate the proof.

add reference

Bibliography

Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, 2016:260, 2016. URL `http://eprint.iacr.org/2016/260`.

P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012. URL `https://web.stanford.edu/class/cs259c/finalpapers/dlp-cdh.pdf`.

Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3. URL `https://fmouhart.epheme.re/Crypto-1617/TD08.pdf`.

Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492, 2016. `https://ia.cr/2016/492`.