
Operational notes

Document updated on **March 19, 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
- text that needs advice from the communications/marketing team: Aaron & Shane
- text that needs to be completed or otherwise edited (by Sylvia)








































NB: This PDF only includes the Zero-Knowledge Protocols chapter

11 Todo list

12	zero-knowledge proofs	12
13	played with	12
14	finite field	12
15	elliptic curve	12
16	Update reference when content is finalized	12
17	methatical	12
18	numerical	12
19	a list of additional exercises	13
20	think about them	13
21	add some more informal explanation of absolute value	14
22	We haven't really talked about what a ring is at this point	14
23	What's the significance of this distinction?	15
24	reverse	15
25	Turing machine	15
26	polynomial time	15
27	sub-exponentially, with $\mathcal{O}((1 + \varepsilon)^n)$ and some $\varepsilon > 0$	15
28	Add text	16
29	\mathbb{Q} of fractions	16
30	Division in the usual sense is not defined for integers	16
31	Add more explanation of how this works	17
32	pseudocode	18
33	modular arithmetics	18
34	actual division	18
35	multiplicative inverses	18
36	factional numbers	18
37	exponentiation function	20
38	See XXX	20
39	once they accept that this is a new kind of calculations, its actually not that hard	20
40	perform Euclidean division on them	20
41	This Sage snippet should be described in more detail.	21
42	prime fields	23
43	residue class rings	23
44	Algorithm sometimes floated to the next page, check this for final version	23
45	Add a number and title to the tables	25
46	(-1) should be $(-a)$?	26
47	we have	28
48	rephrase	32
49	subtrahend	33
50	minuend	33

51	what does this mean?	37
52	Chapter 1?	126
53	"rigorous"?	126
54	"proving"?	126
55	Add example	127
56	Add more explanation	127
57	I'd delete this, too distracting	127
58	binary tuples	127
59	add reference	128
60	add reference	128
61	check reference	128
62	check reference	128
63	Are we using w and x interchangeably or is there a difference between them?	129
64	check reference	129
65	jubjub	129
66	Edwards form	129
67	add reference	129
68	add reference	129
69	check wording	129
70	add reference	129
71	check references	130
72	add reference	130
73	add reference	130
74	preimage	131
75	check reference	131
76	add reference	131
77	check reference	132
78	check reference	132
79	add reference	133
80	Can we reword this? It's grammatically correct but hard to read	133
81	add reference	134
82	Schur/Hadamard product	134
83	add reference	134
84	check reference	134
85	check reference	135
86	add reference	136
87	check reference	137
88	check reference	137
89	check reference	137
90	check reference	137
91	check reference	138
92	add reference	138
93	add reference	139
94	check reference	139
95	check reference	139
96	add reference	140
97	add reference	140
98	add reference	141

99	■ We already said this in this chapter	143
100	■ check reference	143
101	■ add reference	143
102	■ check reference	144
103	■ add reference	144
104	■ check reference	144
105	■ Should we refer to R1CS satisfiability (p. 137 here?	145
106	■ add reference	146
107	■ add reference	146
108	■ add reference	146
109	■ add reference	147
110	■ check reference	147
111	■ check reference	148
112	■ check reference	150
113	■ add reference	151
114	■ "by"?	151
115	■ add reference	151
116	■ check reference	151
117	■ add reference	151
118	■ add reference	151
119	■ check reference	151
120	■ add reference	151
121	■ clarify language	153
122	■ add reference	154
123	■ add reference	154
124	■ add reference	154
125	■ add reference	154
126	■ add reference	184
127	■ add reference	185
128	■ add reference	186
129	■ add reference	186
130	■ add reference	186
131	■ add reference	187
132	■ add reference	187
133	■ add reference	187
134	■ add reference	187
135	■ add reference	187
136	■ "invariable"?	187
137	■ add reference	188
138	■ add reference	188
139	■ add reference	188
140	■ add reference	189
141	■ add reference	189
142	■ add reference	190
143	■ add reference	191
144	■ add reference	191
145	■ add reference	192
146	■ add reference	192

147		add reference	192
148		add reference	192
149		add reference	192
150		add reference	193
151		add reference	193
152		add reference	193
153		add reference	193
154		add reference	193
155		add reference	193
156		add reference	193
157		add reference	193
158		add reference	193
159		add reference	194
160		add reference	194
161		add reference	194
162		add reference	194
163		add reference	196
164		add reference	196
165		add reference	196
166		add reference	196
167		add reference	196
168		add reference	196
169		add reference	197
170		add reference	197
171		add reference	197
172		add reference	197
173		add reference	197
174		add reference	198
175		add reference	198
176		add reference	198
177		add reference	198
178		add reference	199
179		add reference	199
180		add reference	199
181		add reference	199
182		add reference	199
183		add reference	199
184		add reference	199
185		add reference	199

186

MoonMath manual

187

TechnoBob and the Least Scruples crew

188

March 19, 2022

Contents

190	1 Introduction	5
191	1.1 Target audience	5
192	1.2 The Zoo of Zero-Knowledge Proofs	6
193	To Do List	8
194	Points to cover while writing	8
195	2 Preliminaries	9
196	2.1 Preface and Acknowledgements	9
197	2.2 Purpose of the book	9
198	2.3 How to read this book	10
199	2.4 Cryptological Systems	10
200	2.5 SNARKS	10
201	2.6 complexity theory	10
202	2.6.1 Runtime complexity	10
203	2.7 Software Used in This Book	11
204	2.7.1 Sagemath	11
205	3 Arithmetics	12
206	3.1 Introduction	12
207	3.1.1 Aims and target audience	12
208	3.1.2 The structure of this chapter	13
209	3.2 Integer Arithmetics	13
210	Euclidean Division	16
211	The Extended Euclidean Algorithm	18
212	3.3 Modular arithmetic	19
213	Congurency	20
214	Modular Arithmetics	20
215	The Chinese Remainder Theorem	23
216	Modular Inverses	26
217	3.4 Polynomial Arithmetics	29
218	Polynomial Arithmetics	33
219	Euklidean Division	34
220	Prime Factors	36
221	Lange interpolation	37
222	4 Algebra	40
223	4.1 Groups	40
224	Commutative Groups	41
225	Finite groups	43

226		Generators	43
227		The discrete Logarithm problem	43
228	4.1.1	Cryptographic Groups	44
229		The discret logarithm assumption	45
230		The decisional Diffi Hellman assumption	47
231		The computational Diffi Hellman assumption	47
232		Cofactor Clearing	48
233	4.1.2	Hashing to Groups	48
234		Hash functions	48
235		Hashing to cyclic groups	50
236		Hashing to modular arithmetics	51
237		Pederson Hashes	55
238		MimC Hashes	55
239		Pseudo Random Functions in DDH-A groups	55
240	4.2	Commutative Rings	55
241		Hashing to Commutative Rings	58
242	4.3	Fields	58
243		Prime fields	60
244		Square Roots	61
245		Exponentiation	63
246		Hashing into Prime fields	63
247		Extension Fields	63
248		Hashing into extension fields	66
249	4.4	Projective Planes	67
250	5	Elliptic Curves	69
251	5.1	Elliptic Curve Arithmetics	69
252	5.1.1	Short Weierstraß Curves	69
253		Affine short Weierstraß form	70
254		Affine compressed representation	74
255		Affine group law	75
256		Scalar multiplication	80
257		Projective short Weierstraß form	83
258		Projective Group law	85
259		Coordinate Transformations	85
260	5.1.2	Montgomery Curves	85
261		Affine Montgomery Form	87
262		Affine Montgomery coordinate transformation	88
263		Montgomery group law	90
264	5.1.3	Twisted Edwards Curves	90
265		Twisted Edwards Form	91
266		Twisted Edwards group law	92
267	5.2	Elliptic Curves Pairings	93
268		Embedding Degrees	93
269		Elliptic Curves over extension fields	95
270		Full Torsion groups	96
271		Torsion-Subgroups	98
272		The Weil Pairing	100

273	5.3	Hashing to Curves	103
274		Try and increment hash functions	103
275	5.4	Constructing elliptic curves	106
276		The Trace of Frobenius	106
277		The j -invariant	107
278		The Complex Multiplication Method	108
279		The <i>BLS6_6</i> pen& paper curve	117
280		Hashing to the pairing groups	124
281	6	Statements	126
282	6.1	Formal Languages	126
283		Decision Functions	127
284		Instance and Witness	130
285		Modularity	133
286	6.2	Statement Representations	133
287	6.2.1	Rank-1 Quadratic Constraint Systems	133
288		R1CS representation	134
289		R1CS Satisfiability	136
290		Modularity	138
291	6.2.2	Algebraic Circuits	138
292		Algebraic circuit representation	138
293		Circuit Execution	143
294		Circuit Satisfiability	145
295		Associated Constraint Systems	146
296	6.2.3	Quadratic Arithmetic Programs	151
297		QAP representation	151
298		QAP Satisfiability	153
299	7	Circuit Compiler	157
300	7.1	A Pen and Paper Language	157
301	7.1.1	The Grammar	157
302	7.1.2	The Execution Phases	159
303		The Setup Phase	159
304		The Proofer Phase	161
305	7.2	Common Programing concepts	161
306	7.2.1	Primitive Types	161
307		The Basefield type	162
308		The Subtraction Constraints System	165
309		The Inversion Constraint System	166
310		The Division Constraint System	167
311		The Boolean Type	168
312		The Boolean Constraint System	168
313		The AND operator constraint system	169
314		The OR operator constraint system	169
315		The NOT operator constraint system	170
316		Modularity	171
317		Arrays	174
318		The Unsigned Integer Type	174

319		The uN Constraints System	175
320		The Unsigned Integer Operators	176
321	7.2.2	Control Flow	177
322		The Conditional Assignment	177
323		Loops	179
324	7.2.3	Binary Field Representations	180
325	7.2.4	Cryptographic Primitives	182
326		Twisted Edwards curves	182
327		Twisted Edwards curves constraints	182
328		Twisted Edwards curves addition	183
329	8	Zero Knowledge Protocols	184
330	8.1	Proof Systems	184
331	8.2	The “Groth16” Protocol	185
332		The Setup Phase	187
333		The Proofer Phase	192
334		The Verification Phase	195
335		Proof Simulation	197
336	9	Exercises and Solutions	200

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 XXX, 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.

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

add reference

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 it 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.

Proofing the membership statement for some string is then 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 L 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 output 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.

After 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 sub-classes of proofing systems are known in the field. The type of languages any proof system can support, crucially depends on the abilities of the verifier, for example to make random choices, or not, 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 than 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 polynomial 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 called **zk-SNARKs**.

Example 135 (Constructive Proofs for Algebraic Circuits). To formalize our previous notion of constructive proof for algebraic circuits, let \mathbb{F} be a finite field and $C(\mathbb{F})$ 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 if 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 property, let $C(\mathbb{F})$ be a circuit of the field \mathbb{F} and I 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. It 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 and since in this simple system the witness itself is the proof, the proof system is **not** succinct.

8.2 The “Groth16” Protocol

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

add reference

and associated membership as well as knowledge claim can be constructively proofed by executing the circuit in order 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 [XXX] Jens Groth provides a method that can transform those proofs into zero-knowledge succinct non interactive arguments of knowledge. Assuming that 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 \mathbb{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, besides the fact that the instance, witness pair is a proper word in the language of the problem.

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 has perfect completeness, perfect zero-knowledge and soundness in the generic bilinear group model, assuming that a trusted third party exists, 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.

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

$$\text{Groth_16} - \text{Param}(R) = (r, \mathbb{G}_1, \mathbb{G}_2, e(\cdot, \cdot), g_1, g_2) \quad (8.1)$$

where \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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerate, bilinear pairing for some target group \mathbb{G}_T . In 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 (SETUP, PROVE, VFY, SIM) such that

- (Setup-Phase): $(CRS, \tau) \leftarrow \text{Setup}(R)$: Algorithm Setup takes the R1CS R as input and computes a common reference string CRS and a simulation trapdoor τ .
- (Prover-Phase): $\pi \leftarrow \text{Prove}(R, CRS, I, W)$: 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: $\{\text{accept}, \text{reject}\} \leftarrow \text{Vfy}(R, CRS, I, \pi)$: 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`.
- $\pi \leftarrow \text{Sim}(R, \tau, CRS, I)$: 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 those algorithms together with examples in detail in the appropriate paragraphs of this section.

Assuming a trusted third party for the setup, the protocol is then able to compute a zk-SNARK from a constructive proof for R , assuming 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 XXX and its associated algebraic circuit and rank-1 constraints system from XXX. In this example,

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} XXX and the derived QAP XXX 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.

From XXX we know, that the moon-math curve BLS6 6 has two subgroups $\mathbb{G}_1[13]$ and $\mathbb{G}_2[13]$, that are both of order 13. The associated Weil pairing b XXX 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 parameter

$$\text{Groth_16} - \text{Param}(R_{3, fac_zk}) = (r, \mathbb{G}_1[13], \mathbb{G}_2[13], e(\cdot, \cdot), (13, 15), (7v^2, 16v^3))$$

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 that has to be executed a single time for every rank-1 constraints system and any associated quadratic arithmetic program. The outcome of this phase is a common reference string, that proofer and verifier need 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 constraints system R such that, a constructive proof of knowledge for an instance (I_1, \dots, I_n) in L consists of a witness (W_1, \dots, W_m) . Let $QAP(R) = \{T \in \mathbb{F}[x], \{A_j, B_j, C_j \in \mathbb{F}[x]\}_{j=0}^{n+m}\}$ be a quadratic arithmetic program associated to R and $\{\mathbb{G}_1, \mathbb{G}_2, e(\cdot, \cdot), g_1, g_2, \mathbb{F}_r\}$ be the set of Groth_16 parameters.

The setup phase then samples 5 random, **inverible** elements $\alpha, \beta, \gamma, \delta$ and s from the scalar field \mathbb{F}_r of the protocol and outputs the **simulation trapdoor**

$$\tau = (\alpha, \beta, \gamma, \delta, s) \quad (8.2)$$

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 :

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

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

Common reference strings depend on the simulation trapdoor and are therefor 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 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 $P(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_k \cdot x^k$ 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:

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

In this expression all the group points $g_1^{s^j}$ are part of the common reference string and hence 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 **toxic waste** of the setup-phase, while a common reference string is also called a pair of **proofer and verifier key**.

In order to make the protocol secure the setup needs to be executed in a way such that, it is guaranteed 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 achieve 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 in real world application more sophisticated protocols exists that 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 participants deletes their individual contribution to the randomness. Each participant only possesses a fraction of the simulation trapdoor and the toxic waste 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 XXX and the parameters from XXX. As we have seen in XXX an associated quadratic arithmetic program is given by

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

To transform this QAP into a common reference string, we choose the following 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 memorable, which helps in pen and paper computations. Our simulation trapdoor is then given by

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

and we keep this secret in order to simulate proofs later on. We are careful though to hide τ from anyone who hasn't read this book. From those values we then instantiate the common

add reference

add reference

add reference

5196 reference string XXX. Since our groups are subgroups of the BLS6_6 elliptic curve, we use
 5197 scalar product notation instead of exponentiation.

add refer-
ence

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

add refer-
ence

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

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

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

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

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

$$\left\lfloor \frac{5A_0(2) + 6B_0(2) + C_0(2)}{4} \right\rfloor(13, 15) = [(5 \cdot 0 + 6 \cdot 0 + 0) \cdot 10](13, 15) = [0](13, 15) =$$

\emptyset

$$\left\lfloor \frac{5A_1(2) + 6B_1(2) + C_1(2)}{4} \right\rfloor(13, 15) = [(5 \cdot 0 + 6 \cdot 0 + (7 \cdot 2 + 4)) \cdot 10](13, 15) = [11](13, 15) =$$

$(33, 9)$

$$\left\lfloor \frac{5A_2(2) + 6B_2(2) + C_2(2)}{3} \right\rfloor(13, 15) = [(5 \cdot (6 \cdot 2 + 10) + 6 \cdot 0 + 0) \cdot 9](13, 15) = [2](13, 15) =$$

$(33, 34)$

$$\left\lfloor \frac{5A_3(2) + 6B_3(2) + C_3(2)}{3} \right\rfloor(13, 15) = [(5 \cdot 0 + 6 \cdot (6 \cdot 2 + 10) + 0) \cdot 9](13, 15) = [5](13, 15) =$$

$(26, 34)$

$$\left\lfloor \frac{5A_4(2) + 6B_4(2) + C_4(2)}{3} \right\rfloor(13, 15) = [(5 \cdot 0 + 6 \cdot (7 \cdot 2 + 4) + 0) \cdot 9](13, 15) = [10](13, 15) =$$

$(38, 28)$

$$\left\lfloor \frac{5A_5(2) + 6B_5(2) + C_5(2)}{3} \right\rfloor(13, 15) = [(5 \cdot (7 \cdot 2 + 4) + 6 \cdot 0 + 0) \cdot 9](13, 15) = [4](13, 15) =$$

$(35, 28)$

$$\left\lfloor \frac{2^0 \cdot T(2)}{3} \right\rfloor(13, 15) = [1 \cdot (2^2 + 2 + 9) \cdot 9](13, 15) = [5](13, 15) =$$

$(26, 34)$

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 :

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

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

$$[\beta]g_2 = [5](7v^2, 16v^3) = (16v^2, 28v^3)$$

$$[\gamma]g_2 = [4](7v^2, 16v^3) = (37v^2, 27v^3)$$

$$[\delta]g_2 = [3](7v^2, 16v^3) = (42v^2, 16v^3)$$

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

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

Putting all those 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 :

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

add reference

Given the simulation trapdoor τ and the quadratic arithmetic program XXX, the associated common reference string of the 3-factorization problem is given by

add reference

$$\begin{aligned} CRS_{\mathbb{G}_1} &= \left\{ (27, 34), (26, 34), (38, 15), \left((13, 15), (33, 34) \right), \left(\mathcal{O}, (33, 9) \right) \right\} \\ &\quad \left\{ \left((33, 34), (26, 34), (38, 28), (35, 28) \right), \left((26, 34) \right) \right\} \\ CRS_{\mathbb{G}_2} &= \left\{ (16v^2, 28v^3), (37v^2, 27v^3), (42v^2, 16v^3), \left(7v^2, 16v^3 \right), (10v^2, 28v^3) \right\} \end{aligned}$$

5198 We then publish this data to everyone who wants to participate in the zk-SNARK generation or
5199 verification of the 3-factorization problem.

5200 To understand how this common reference string can be used, to evaluate polynomials at
5201 the secret evaluation point in the exponent of a generator, let's assume that we have deleted the
5202 simulation trapdoor. In that case we have no way to know the secret evaluation point anymore
5203 and hence can not evaluate polynomials at that point. However, we can evaluate polynomials
5204 of degree smaller than the degree of the target polynomial in the exponent of both generators at
5205 that point.

To see that consider for example 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

add reference

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

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 logarithmic ordering of the group to simplify our pen and paper computations. Such an order is infeasible to compute in cryptographically secure curves. We can do the same computation on \mathbb{G}_2 and get

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

5206 Except for the target polynomial T all other polynomials of the quadratic arithmetic program
 5207 can be evaluated in the exponent this way.

5208 **The Proofer Phase** Given some rank-1 constraints system R and instance $I = (I_1, \dots, I_n)$, the
 5209 task of the proofer phase is to convince any verifier, that a proofer knows a witness W to instance
 5210 I such that, $(I; W)$ is a word in the language L_R of the system, without revealing anything about
 5211 W .

5212 To achieve this in the Groth_16 protocol, we assume that any proofer has access to the rank-
 5213 1 constraints system of the problem in addition with some algorithm, that tells the proofer how
 5214 to compute constructive proofs for the R1CS. In addition the proofer has access to a common
 5215 reference string and its associated quadratic arithmetic program.

5216 In order to generate a zk-SNARK for this instance, the proofer first computes a valid
 5217 constructive proof as explained in XXX, that is the proofer generates a proper witness $W =$
 5218 (W_1, \dots, W_m) such that, $(I_1, \dots, I_n; W_1, \dots, W_m)$ is a solution to the rank-1 constraints system R .

5219 The proofer then uses the quadratic arithmetic program and computes the polynomial $P_{(I;W)}$
 5220 as explained in XXX. They then divide $P_{(I;W)}$ by the target polynomial T of the quadratic arith-
 5221 metic. Since $P_{(I;W)}$ is constructed from a valid solution to the R1CS we know from XXX that it
 5222 is divisible by T . This implies that polynomial division of P by T generates another polynomial
 5223 $H := P/T$, with $\deg(H) < \deg(T)$.

5224 The proofer then evaluates the polynomial $(H \cdot T)\delta^{-1}$ in the exponent of the generator g_1 at
 5225 the secret point s as explained in XXX. To see how this can be achieved, let

$$H(x) = H_0 \cdot x^0 + H_1 \cdot x^1 + \dots + H_k \cdot x^k \quad (8.3)$$

be the quotient polynomial P/T . To evaluate $H \cdot T$ at s in the exponent of g_1 , the proofer uses
 the common reference string and computes

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

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

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

5226 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
 5227 reference string and the quadratic arithmetic program of the problem, as we have seen in XXX.
 5228 In fact those points only have to be computed once and can be published and reused for multiple
 5229 proof generations as they are the same for all instances and witnesses. All other group elements
 5230 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

$$\pi = (g_1^A, g_1^C, g_2^B) \quad (8.4)$$

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 than 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 for 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 XXX, our protocol parameters from XXX as well as the common reference string from XXX.

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 XXX the associated polynomial $P_{(I;W)}$ of the quadratic arithmetic program from XXX is given by

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

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

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

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

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

In a 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

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

To compute this point, we have to remember that a prover should not be in possession of the simulation trapdoor and hence does 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

add reference

add reference

add reference

add reference

add reference

add reference

add reference

add reference

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} , use 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

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

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 and recall from XXX that we can evaluate $[A_j(s)]g_1$ without knowledge of 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 $0 \leq j \leq 5$. Since \mathcal{O} is the neutral element on \mathbb{G}_1 , we get

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

add reference

add reference

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

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

Since we don't know what β , δ and s are we look up the associated group elements from the common reference string and recall from XXX that we can evaluate $[B_j(s)]g_1$ without knowledge of the secret evaluation point s . Since $B_3 = A_2$ as well as $B_4 = A_5$, 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 $0 \leq j \leq 5$. Since \mathcal{O} is the neutral element on \mathbb{G}_1 , we get

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 can combine the previous computations, to compute the point $[C]g_1$ in the group \mathbb{G}_1 . we get

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

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

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

5247 We can publish this zk-SNARK or send it to a designated verifier. Note that if we had sam-
 5248 pled different values for r and t , we would have computed a different SNARK for the same
 5249 witness. The SNARK therefore hides the witness perfectly, which means that it is impossible
 5250 to reconstruct the witness from the SNARK.

5251 **The Verification Phase** Given some rank-1 constraints system R , instance $I = (I_1, \dots, I_n)$
 5252 and zk-SNARK π , the task of the verifier phase is to check that π is indeed an argument for
 5253 a constructive proof. Assuming that the simulation trapdoor does not exist anymore and the
 5254 verification checks the proof, the verifier can be convinced, that someone knows a witness
 5255 $W = (W_1, \dots, W_m)$ 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, \dots, I_n) , the verifier computes the following curve point:

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

5256 With this group element the verifier is then able to verify the SNARK $\pi = (g_1^A, g_1^C, g_2^B)$ by
 5257 checking the following equation using the pairing map:

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

5258 If the equation holds true, the SNARK is accepted and if the equation does not hold, the SNARK
 5259 is rejected.

Remark 5. We know from XXX 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 and can be computed once and then stored as an amendment to the verifier key.

In [GROTH16] the author showed that 2 pairings is the minimal amount of pairings that any protocol with similar properties has to use. This protocol is therefore close to the theoretic 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. 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 XXX, 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 $I_1 = 11$.

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

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

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

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

In a next step, we have to compute all the pairings involved in equation XXX. Using the loga-

rithmic order on \mathbb{G}_1 and \mathbb{G}_2 as well as the bilinearity property of the pairing map we get

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

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. Using this we evaluate the left side of equation XXX as

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

since $108 \bmod 13 = 4$. Similarly, we evaluate the right side of equation XXX using modular 13 arithmetics and the exponential law $a^x \cdot a^y = a^{x+y}$. We get

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

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.

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

Proof Simulation During the execution of a setup phase, a common reference string is generated accompanied by 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.

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 given instances without any knowledge or the existence of associated witnesses.

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 any 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

$$\tau = (\alpha, \beta, \gamma, \delta, s) \quad (8.6)$$

Given some instance I the forgers task is to generate a zk-SNARK for this instance that passes the verification process, without access to any other zk-SNARK 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 compute

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

for the instance (I_1, \dots, 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, \dots, 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 XXX, our protocol parameters from XXX, the common reference string from XXX and the simulation trapdoor $\tau = (6, 5, 4, 3, 2)$ of that CRS.

In order to forge a zk-SNARK for instance $I_1 = 11$ we don't need a constructive proof for the associated rank-1 constraints 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

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

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

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

add reference

add reference

add reference

add reference

5310 Despite the fact that this zk-SNARK was generated without knowledge of a proper witness, it
 5311 is indistinguishable from a zk-SNARK that proofs 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 and get

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

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

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

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

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

since $27 \bmod 13 = 1$. 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

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

5312 As we can see both the left and the right side of equation XXX are identical, which implies that
 5313 the verification process accepts the simulated proof. π_{fake} therefore convince the verifier that
 5314 a witness to 3-factorization problem exists, However, no such witness was really necessary to
 5315 generate the proof.

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>.
- 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>.