




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)

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

51	add reference	41
52	Add real-life example of 0?	41
53	add reference	41
54	check reference	42
55	check references to previous examples	43
56	RSA crypto system	43
57	size 2048-bits	43
58	check reference	43
59	add reference: 27?	43
60	check reference	44
61	polynomial time	44
62	exponential time	44
63	TODO: Fundamental theorem of finite cyclic groups	44
64	check reference	44
65	runtime complexity	45
66	add reference	45
67	S: what does “efficiently” mean here?	45
68	computational hardness assumptions	45
69	check reference	45
70	check reference	46
71	explain last sentence more	46
72	“equation”?	47
73	check reference	47
74	what’s the difference between \mathbb{F}_p^* and \mathbb{Z}_p^* ?	47
75	Legendre symbol	47
76	Euler’s formular	47
77	These are only explained later in the text, “	47
78	are these going to be relevant later?	48
79	TODO: theorem: every factor of order defines a subgroup...	48
80	Is there a term for this property?	49
81	a few examples?	51
82	check reference	51
83	TODO: DOUBLE CHECK THIS REASONING.	51
84	Mirco: We can do better than this	53
85	check reference	54
86	add reference	55
87	pseudorandom	55
88	oracle	55
89	check reference	55
90	add text on this	55
91	check reference	57
92	check reference	57
93	check reference	57
94	check reference	58
95	add more examples protocols of SNARK	58
96	check reference	58
97	add reference	58
98	Abelian groups	58

99	■ codomain	58
100	■ Check change of wording	59
101	■ add reference	60
102	■ Expand on this?	60
103	■ check reference	60
104	■ unify \mathbb{Z}_5 and \mathbb{F}_5 across the board?	61
105	■ S: are we introducing elliptic curves in section 1 or 2?	61
106	■ check reference	62
107	■ check reference	62
108	■ add reference	62
109	■ check reference	62
110	■ write paragraph on exponentiation	63
111	■ add reference	63
112	■ check reference	63
113	■ add reference	63
114	■ group pairings	63
115	■ add reference	64
116	■ check reference	64
117	■ check reference	67
118	■ add reference	68
119	■ TODO: Elliptic Curve asymmetric cryptography examples. Private key, generator,	
120	public key.	70
121	■ add reference	70
122	■ maybe remove this sentence?	70
123	■ affine space	70
124	■ cusps	71
125	■ self-intersections	71
126	■ check reference	72
127	■ check reference	73
128	■ jubjub	73
129	■ check reference	73
130	■ affine plane	73
131	■ check reference	74
132	■ check reference	74
133	■ check reference	75
134	■ sign	75
135	■ more explanation of what the sign is	75
136	■ check reference	75
137	■ S: I don't follow this at all	76
138	■ check reference	76
139	■ add explanation of how this shows what we claim	76
140	■ should this def. be moved even earlier?	77
141	■ chord line	77
142	■ tangential	77
143	■ tangent line	77
144	■ remove Q ?	77
145	■ where?	78
146	■ check reference	78

















































147	 check reference	78
148	 check reference	78
149	 check reference	79
150	 check reference	79
151	 check reference	80
152	 check reference	80
153	 check reference	81
154	 add term	81
155	 add term	81
156	 add reference	81
157	 cofactor clearing	81
158	 add reference	81
159	 check reference	81
160	 check reference	82
161	 add reference	82
162	 add reference	82
163	 check reference	82
164	 check reference	82
165	 check reference	83
166	 check reference	83
167	 check reference	83
168	 Explain how	83
169	 write example	84
170	 check reference	84
171	 add reference	84
172	 check reference	84
173	 add reference	85
174	 check reference	85
175	 add reference	85
176	 check reference	85
177	 add reference	85
178	 check reference	85
179	 add reference	85
180	 add reference	85
181	 add reference	85
182	 check reference	85
183	 check reference	85
184	 Check if following Alg is floated too far	86
185	 add reference	86
186	 add reference	86
187	 write up this part	86
188	 is the label in L ^A T _E X correct here?	88
189	 check reference	88
190	 check reference	88
191	 check reference	88
192	 check reference	89
193	 check reference	89
194	 check reference	90

















































195	check reference	90
196	check reference	90
197	check reference	90
198	check reference	90
199	add reference	90
200	check reference	92
201	check reference	92
202	check reference	92
203	check reference	92
204	check reference	92
205	change “tiny-jubjub” to “pen-jubjub” throughout?	93
206	check reference	94
207	check reference	94
208	check reference	95
209	either expand on this or delete it	95
210	add reference	95
211	check reference	95
212	check reference	95
213	check reference	95
214	check reference	95
215	check reference	95
216	check reference	96
217	check reference	96
218	check reference	97
219	check reference	97
220	add reference	97
221	add reference	97
222	This needs to be written (in Algebra)	97
223	add reference	97
224	add reference	97
225	check reference	97
226	towers of curve extensions	98
227	check reference	98
228	check reference	98
229	check reference	98
230	check reference	99
231	add reference	99
232	is “huge” a technical term?	99
233	check reference	100
234	S: either add more explanation or move to a footnote	100
235	type 3 pairing-based cryptography	100
236	add references?	100
237	check reference	101
238	check reference	101
239	check floating of algorithm	102
240	add references	102
241	check reference	103
242	add reference	103
























243	check reference	103
244	check reference	103
245	add reference	104
246	should all lines of all algorithms be numbered?	104
247	check reference	105
248	check reference	105
249	check reference	105
250	check if the algorithm is floated properly	105
251	check reference	105
252	again?	107
253	check reference	107
254	circuit	107
255	signature schemes	107
256	this was called “pen-jubjub”.	107
257	check reference	107
258	add reference	108
259	check reference	108
260	add references	108
261	add reference	108
262	reference text to be written in Algebra	108
263	check reference	108
264	check reference	108
265	check reference	109
266	add reference	109
267	algebraic closures	109
268	check reference	109
269	check reference	110
270	check reference	110
271	check reference	110
272	check reference	111
273	disambiguate	111
274	add reference	111
275	unify terminology	111
276	check reference	112
277	actually make this a table?	112
278	exercise still to be written?	113
279	add reference	113
280	check reference	113
281	check reference	113
282	add reference	114
283	check reference	115
284	check reference	115
285	check reference	115
286	add reference	116
287	check reference	116
288	check reference	116
289	check reference	117
290	what does this mean?	118

291	write up this part	119
292	add reference	119
293	check reference	119
294	cyclotomic polynomial	119
295	Pholaard-rho attack	119
296	todo	119
297	why?	120
298	check reference	120
299	check reference	120
300	what does this mean?	120
301	add reference	120
302	add reference	120
303	check reference	120
304	check reference	121
305	add reference	122
306	add exercise	122
307	check reference	123
308	add reference	123
309	add reference	123
310	add reference	123
311	check reference	124
312	check reference	124
313	add reference	124
314	add reference	124
315	add reference	125
316	check reference	125
317	add reference	125
318	add reference	125
319	finish writing this up	126
320	add reference	126
321	correct computations	126
322	fill in missing parts	126
323	add reference	127
324	check equation	127
325	Chapter 1?	128
326	"rigorous"?	128
327	"proving"?	128
328	Add example	129
329	Add more explanation	129
330	I'd delete this, too distracting	129
331	binary tuples	129
332	add reference	130
333	add reference	130
334	check reference	130
335	check reference	130
336	Are we using w and x interchangeably or is there a difference between them?	131
337	check reference	131
338	jubjub	131

339	check reference	131
340	check reference	131
341	check wording	131
342	check reference	131
343	check references	132
344	add reference	132
345	add reference	132
346	check reference	133
347	add reference	133
348	check reference	134
349	check reference	134
350	add reference	135
351	add reference	136
352	Schur/Hadamard product	136
353	add reference	136
354	check reference	136
355	check reference	137
356	add reference	138
357	check reference	139
358	check reference	139
359	check reference	139
360	check reference	139
361	check reference	140
362	add reference	140
363	add reference	141
364	check reference	141
365	check reference	142
366	check reference	142
367	check reference	142
368	add reference	143
369	check reference	145
370	add reference	145
371	check reference	146
372	check reference	146
373	check reference	146
374	Should we refer to R1CS satisfiability (p. 139 here?	147
375	check reference	148
376	add reference	148
377	check reference	148
378	check reference	149
379	check reference	149
380	check reference	150
381	check reference	152
382	add reference	153
383	"by"?	153
384	check reference	153
385	check reference	153
386	add reference	153

387	 add reference	153
388	 check reference	153
389	 add reference	153
390	 clarify language	155
391	 check reference	156
392	 add reference	156
393	 check reference	156
394	 add reference	156
395	 check references	159
396	 add references to these languages?	159
397	 check reference	162
398	 check reference	163
399	 check reference	163
400	 check reference	164
401	 check reference	165
402	 check reference	165
403	 check reference	167
404	 check reference	167
405	 check reference	168
406	 add reference	168
407	 check reference	168
408	 add reference	168
409	 add reference	168
410	 check reference	169
411	 check reference	169
412	 check reference	169
413	 check reference	169
414	 add reference	169
415	 check reference	170
416	 check reference	171
417	 "constraints" or "constrained"?	171
418	 check reference	172
419	 "constraints" or "constrained"?	172
420	 add reference	172
421	 "constraints" or "constrained"?	172
422	 add reference	173
423	 check references	173
424	 check reference	173
425	 add reference	174
426	 can we rotate this by 90°?	174
427	 check reference	175
428	 add reference	175
429	 add reference	175
430	 shift	177
431	 bishift	178
432	 add reference	179
433	 check reference	180
434	 add reference	182

435		add reference	183
436		check reference	184
437		add reference	184
438		add reference	184
439		check reference	185
440		add reference	185
441		add reference	185
442		add reference	187
443		check reference	188
444		check reference	189
445		common reference string	189
446		simulation trapdoor	189
447		check reference	189
448		check reference	189
449		add reference	190
450		check reference	190
451		check reference	190
452		check reference	190
453		"invariable"?	190
454		explain why	191
455		4 examples have the same title. Change it to be distinct	191
456		check reference	191
457		add reference	191
458		check reference	191
459		add reference	191
460		add reference	192
461		add reference	193
462		check reference	194
463		add reference	194
464		add reference	195
465		check reference	195
466		check reference	195
467		add reference	195
468		add reference	195
469		check reference	196
470		add reference	196
471		add reference	196
472		add reference	196
473		check reference	196
474		add reference	196
475		add reference	196
476		add reference	196
477		add reference	196
478		add reference	197
479		add reference	197
480		add reference	197
481		add reference	197
482		check reference	199

483	 check reference	199
484	 add reference	199
485	 add reference	199
486	 add reference	199
487	 add reference	199
488	 add reference	200
489	 add reference	200
490	 add reference	200
491	 add reference	200
492	 fix error	200
493	 add reference	200
494	 check reference	201
495	 add reference	201
496	 add reference	201
497	 add reference	201
498	 add reference	202
499	 add reference	202
500	 add reference	202
501	 add reference	202
502	 add reference	202
503	 add reference	202
504	 add reference	202
505	 add reference	203

506

MoonMath manual

507

TechnoBob and the Least Scruples crew

508

March 27, 2022

Contents

510	1	Introduction	5
511	1.1	Target audience	5
512	1.2	The Zoo of Zero-Knowledge Proofs	6
513		To Do List	8
514		Points to cover while writing	8
515	2	Preliminaries	9
516	2.1	Preface and Acknowledgements	9
517	2.2	Purpose of the book	9
518	2.3	How to read this book	10
519	2.4	Cryptological Systems	10
520	2.5	SNARKS	10
521	2.6	complexity theory	10
522	2.6.1	Runtime complexity	10
523	2.7	Software Used in This Book	11
524	2.7.1	Sagemath	11
525	3	Arithmetics	12
526	3.1	Introduction	12
527	3.1.1	Aims and target audience	12
528	3.1.2	The structure of this chapter	13
529	3.2	Integer Arithmetics	13
530		Euclidean Division	16
531		The Extended Euclidean Algorithm	18
532	3.3	Modular arithmetic	19
533		Congurency	20
534		Modular Arithmetics	20
535		The Chinese Remainder Theorem	23
536		Modular Inverses	26
537	3.4	Polynomial Arithmetics	29
538		Polynomial Arithmetics	33
539		Euklidean Division	34
540		Prime Factors	36
541		Lange interpolation	37
542	4	Algebra	40
543	4.1	Groups	40
544		Commutative Groups	41
545		Finite groups	43

546		Generators	43
547		The discrete Logarithm problem	43
548	4.1.1	Cryptographic Groups	44
549		The discrete logarithm assumption	45
550		The decisional Diffie–Hellman assumption	47
551		The computational Diffie–Hellman assumption	47
552		Cofactor Clearing	48
553	4.1.2	Hashing to Groups	48
554		Hash functions	48
555		Hashing to cyclic groups	50
556		Hashing to modular arithmetics	51
557		Pedersen Hashes	55
558		MimC Hashes	55
559		Pseudorandom Functions in DDH-A groups	55
560	4.2	Commutative Rings	55
561		Hashing to Commutative Rings	58
562	4.3	Fields	58
563		Prime fields	60
564		Square Roots	61
565		Exponentiation	63
566		Hashing into prime fields	63
567		Extension Fields	63
568		Hashing into extension fields	67
569	4.4	Projective Planes	67
570	5	Elliptic Curves	70
571	5.1	Elliptic Curve Arithmetics	70
572	5.1.1	Short Weierstraß Curves	70
573		Affine short Weierstraß form	71
574		Affine compressed representation	75
575		Affine group law	76
576		Scalar multiplication	81
577		Projective short Weierstraß form	84
578		Projective Group law	85
579		Coordinate Transformations	86
580	5.1.2	Montgomery Curves	86
581		Affine Montgomery Form	88
582		Affine Montgomery coordinate transformation	89
583		Montgomery group law	91
584	5.1.3	Twisted Edwards Curves	92
585		Twisted Edwards Form	92
586		Twisted Edwards group law	94
587	5.2	Elliptic Curve Pairings	95
588		Embedding Degrees	95
589		Elliptic Curves over extension fields	96
590		Full torsion groups	97
591		Torsion subgroups	100
592		The Weil pairing	102

593	5.3	Hashing to Curves	104
594		Try-and-increment hash functions	105
595	5.4	Constructing elliptic curves	107
596		The Trace of Frobenius	108
597		The j -invariant	109
598		The Complex Multiplication Method	110
599		The <i>BLS6_6</i> pen-and-paper curve	119
600		Hashing to pairing groups	126
601	6	Statements	128
602	6.1	Formal Languages	128
603		Decision Functions	129
604		Instance and Witness	132
605		Modularity	135
606	6.2	Statement Representations	135
607	6.2.1	Rank-1 Quadratic Constraint Systems	136
608		R1CS representation	136
609		R1CS Satisfiability	138
610		Modularity	140
611	6.2.2	Algebraic Circuits	140
612		Algebraic circuit representation	140
613		Circuit Execution	145
614		Circuit Satisfiability	147
615		Associated Constraint Systems	148
616	6.2.3	Quadratic Arithmetic Programs	153
617		QAP representation	153
618		QAP Satisfiability	155
619	7	Circuit Compilers	159
620	7.1	A Pen-and-Paper Language	159
621	7.1.1	The Grammar	159
622	7.1.2	The Execution Phases	161
623		The Setup Phase	161
624		The Prover Phase	163
625	7.2	Common Programing concepts	163
626	7.2.1	Primitive Types	163
627		The base-field type	164
628		The Subtraction Constraint System	167
629		The Inversion Constraint System	168
630		The Division Constraint System	169
631		The boolean Type	170
632		The boolean Constraint System	170
633		The AND operator constraint system	171
634		The OR operator constraint system	171
635		The NOT operator constraint system	172
636		Modularity	173
637		Arrays	176
638		The Unsigned Integer Type	176

639		The uN Constraint System	177
640		The Unsigned Integer Operators	178
641	7.2.2	Control Flow	179
642		The Conditional Assignment	179
643		Loops	181
644	7.2.3	Binary Field Representations	182
645	7.2.4	Cryptographic Primitives	184
646		Twisted Edwards curves	184
647		Twisted Edwards curve constraints	184
648		Twisted Edwards curve addition	185
649	8	Zero Knowledge Protocols	187
650	8.1	Proof Systems	187
651	8.2	The “Groth16” Protocol	188
652		The Setup Phase	190
653		The Prover Phase	195
654		The Verification Phase	198
655		Proof Simulation	200
656	9	Exercises and Solutions	202

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.

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.

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 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 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 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. 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. 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 \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 beside 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 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.

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:

$$\text{Groth}_{16} - \text{Param}(R) = (r, \mathbb{G}_1, \mathbb{G}_2, e(\cdot, \cdot), g_1, g_2) \quad (8.1)$$

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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 (SETUP, PROVE, VFY, SIM) such that the following conditions hold:

- (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 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

common
reference
string

simulation
trapdoor

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.

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:

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

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. 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, \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 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 $\alpha, \beta, \gamma, \delta$ and s from the scalar field \mathbb{F}_r of the protocol and outputs the **simulation trapdoor** τ :

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

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 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 $P(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_k \cdot x^k$ 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:

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

In this expression, all 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**.

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:

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

To transform this QAP into a common reference string, we choose the 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, but we are careful though to hide τ from anyone who hasn't read this book. Then we instantiate the common reference string XXX

explain why

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

check reference

add reference

check reference

add reference

5747 from those values. Since our groups are subgroups of the BLS_{6_6} elliptic curve, we use scalar
 5748 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:

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

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

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

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

\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, 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:

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

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

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

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 the following:

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

Putting all 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 :

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

add reference

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

check
reference

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

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 refer-
ence

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

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:

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

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 $I = (I_1, \dots, I_n)$, 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 $W = (W_1, \dots, W_m)$ such that $(I_1, \dots, I_n; W_1, \dots, W_m)$ 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. Since $P_{(I;W)}$ is constructed from a valid solution to the R1CS, we know from 6.15 that it is divisible by T . 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. To see how this can be achieved, let $H(x)$ be the quotient polynomial P/T :

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

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

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

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, \dots, I_n and the witness variables W_1, \dots, W_m to compute the following curve points:

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

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. 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:

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

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 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.

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:

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

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:

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

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 as follows:

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

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 $[W]g_1$:

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

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

check reference

add reference

add reference

add reference

check reference

add reference

add reference

add reference

add reference

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

5802 In a next step, we compute g_1^A . We sample the random point $r = 11$ from \mathbb{F}_{13} , using scalar
5803 products instead of the exponential notation, and \oplus for the group law on the BLS6_6 curve.
5804 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. Recall from XXX that we can evaluate $[A_j(s)]g_1$ without knowing 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 the following:

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

5805 In order to compute the two curve points $[B]g_1$ and $[B]g_2$, we sample another random element
5806 $t = 4$ from \mathbb{F}_{13} . Using the scalar product instead of the exponential notation, and \oplus for the group
5807 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. 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 $0 < j < 5$. Since \mathcal{O} is the neutral element on \mathbb{G}_1 , we get the following:

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

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

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

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

Given 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 $(W_1, W_2, W_3, W_4) = (2, 3, 4, 6)$:

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

5808 We can now publish this zk-SNARK, or send it to a designated verifier. Note that, if we had
 5809 sampled different values for r and t , we would have computed a different SNARK for the same
 5810 witness. The SNARK, therefore, hides the witness perfectly, which means that it is impossible
 5811 to reconstruct the witness from the SNARK.

5812 **The Verification Phase** Given some rank-1 constraint system R , instance $I = (I_1, \dots, I_n)$ and
 5813 zk-SNARK π , the task of the verification phase is to check that π is indeed an argument for
 5814 a constructive proof. Assuming that the simulation trapdoor does not exist anymore and the
 5815 verification checks the proof, the verifier can be convinced that someone knows a witness $W =$
 5816 (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:

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

5817 With this group element, the verifier is able to verify the SNARK $\pi = (g_1^A, g_1^C, g_2^B)$ by checking
 5818 the following equation using the pairing map:

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

5819 If the equation holds true, the SNARK is accepted. If the equation does not hold, the
 5820 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

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:

add refer-
ence

add refer-
ence

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

To compute this point, we have to remember that a verifier should not be in possession of the simulation trapdoor, 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 refer-
ence

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

In 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 refer-
ence

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

5846 In order to check equation XXX, observe that the target group \mathbb{G}_T of the Weil pairing is
 5847 a finite cyclic group of order 13. Exponentiation is therefore done in modular 13 arithmetics.
 5848 Accordingly, since $108 \bmod 13 = 4$, we evaluate the left side of equation XXX as follows:

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

5849 Similarly, we evaluate the right side of equation XXX using modular 13 arithmetics and the
 5850 exponential law $a^x \cdot a^y = a^{x+y}$:

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

5851 As we can see, both the left and the right side of equation XXX are identical, which implies
 5852 that the verification process accepts the simulated proof.

5853 **NOTE: UNFORTUNATELY NOT! :-((HENCE THERE IS AN ERROR SOMEWHERE ...**
 5854 **NEED TO FIX IT AFTER VACATION**

5855 **Proof Simulation** During the execution of a setup phase, a common reference string is gen-
 5856 erated, along with a simulation trapdoor, the latter of which must be deleted at the end of the
 5857 setup-phase. As an alternative, a more complicated multi-party protocol like [XXX] can be
 5858 used to split the knowledge of the simulation trapdoor among many different parties.

5859 In this paragraph, we will show why knowledge of the simulation trapdoor is problematic,
 5860 and how it can be used to generate zk-SNARKs for a given instance without any knowledge or
 5861 the existence of associated witness.

5862 To be more precise, let I be an instance for some R1CS language L_R . We call a zk-SNARK
 5863 for L_R **forged** or **simulated** if it passes a verification but its generation does not require the
 5864 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 :

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

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, \dots, 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 $\pi_{\text{forged}} = (g_1^A, g_1^C, g_2^B)$, 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 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.

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 $C_{3.\text{fac}}(\mathbb{F}_{13})$. 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:

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

check
reference

add refer-
ence

add refer-
ence

add refer-
ence

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:

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

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 follows:

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

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

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

5898 As we can see, both the left and the right side of equation XXX are identical, which implies
5899 that the verification process accepts the simulated proof. π_{fake} therefore convinces the veri-
5900 fier that a witness to the 3-factorization problem exists. However, no such witness was really
5901 necessary to generate the proof.

add refer-
ence

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>.