Language-Agnostic Detection of Computation-Constraint Inconsistencies in ZKP Programs via Value Inference

Arman Kolozyan[*†‡], Bram Vandenbogaerde[‡], Janwillem Swalens[†], Lode Hoste[†], Stefanos Chaliasos[§‖], Coen De Roover[‡]
* MPI-SP † Nokia Bell Labs ‡ VUB § UCL ‖ zkSecurity

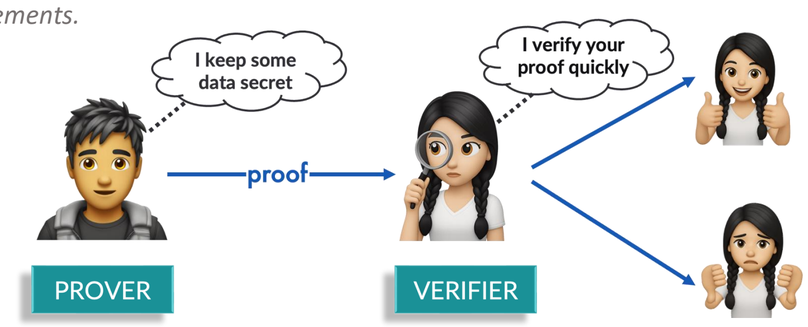
MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

SPRING

Zero-knowledge proofs (ZKPs) enable us to prove a statement is true (e.g., "I am over 18") without revealing any additional information (e.g., my name). However, ZKP development is error-prone, and bugs can lead to financial loss or broken (privacy) guarantees. Therefore, we introduce a language-agnostic formal model and a corresponding scalable detection tool that catches bugs existing approaches miss.

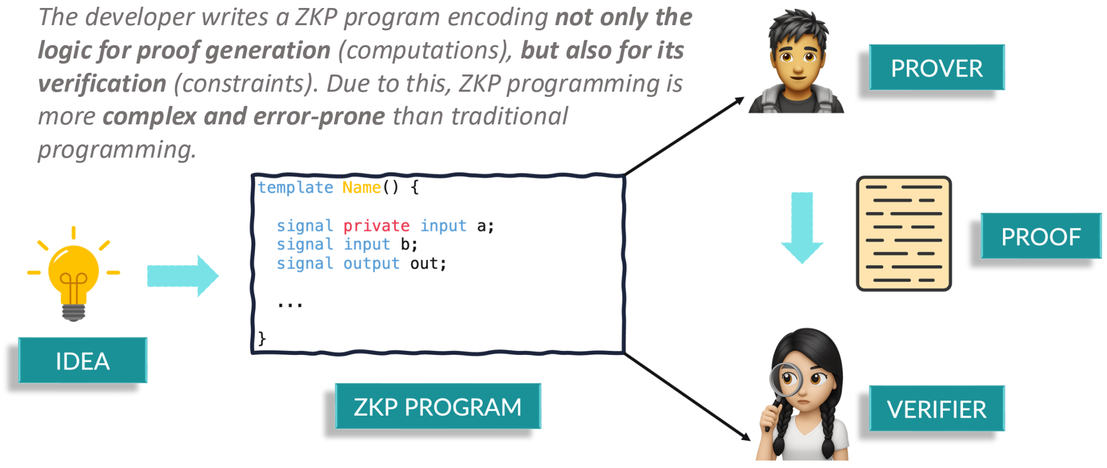
Zero-Knowledge Proofs (ZKPs)

*A prover convinces a verifier that a **statement is true without revealing any additional information**. The proof is compact and can be verified quickly, even for complex statements.*



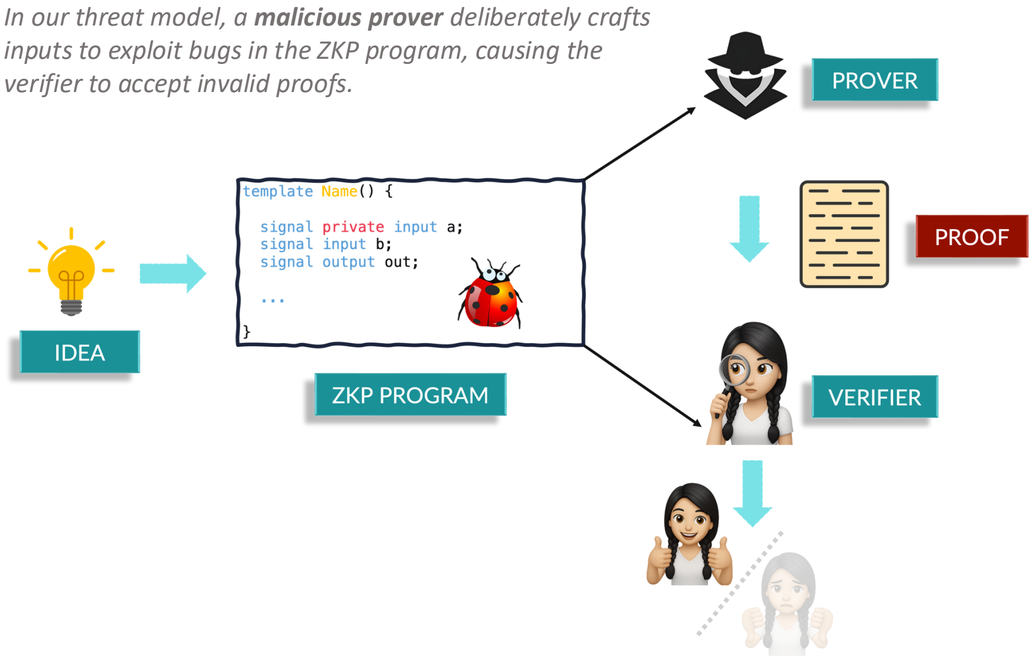
ZKP Pipeline

*The developer writes a ZKP program encoding **not only the logic for proof generation** (computations), **but also for its verification** (constraints). Due to this, ZKP programming is more **complex and error-prone** than traditional programming.*



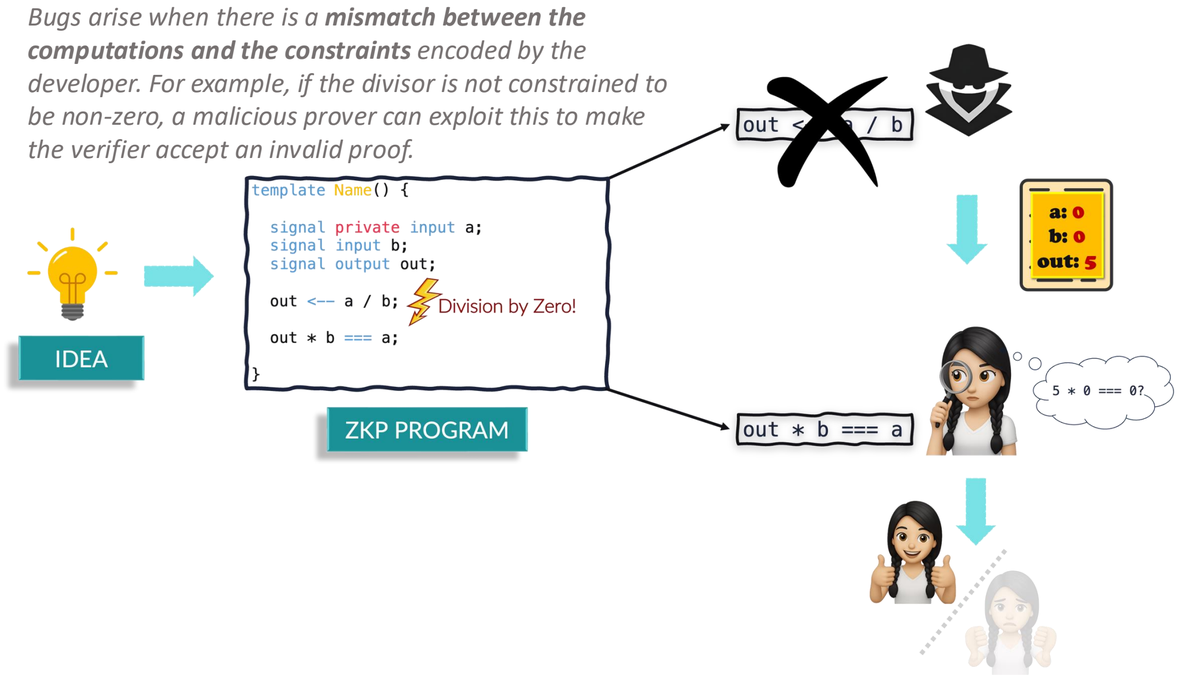
Threat Model

*In our threat model, a **malicious prover** deliberately crafts inputs to exploit bugs in the ZKP program, causing the verifier to accept invalid proofs.*



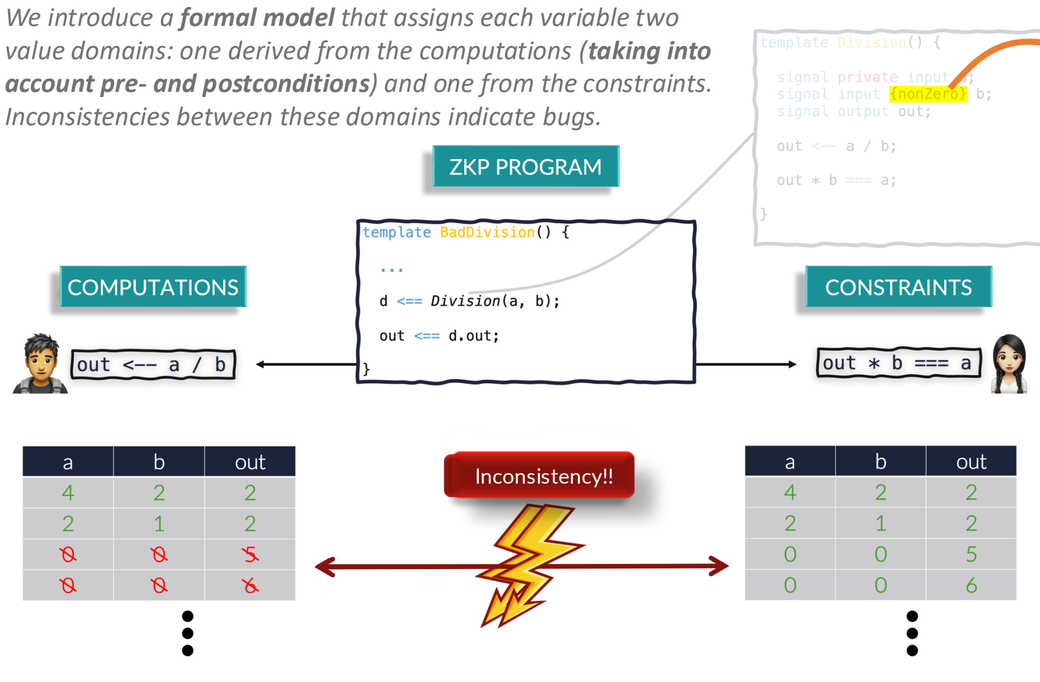
Prover-Verifier Mismatch Bugs

*Bugs arise when there is a **mismatch between the computations and the constraints** encoded by the developer. For example, if the divisor is not constrained to be non-zero, a malicious prover can exploit this to make the verifier accept an invalid proof.*



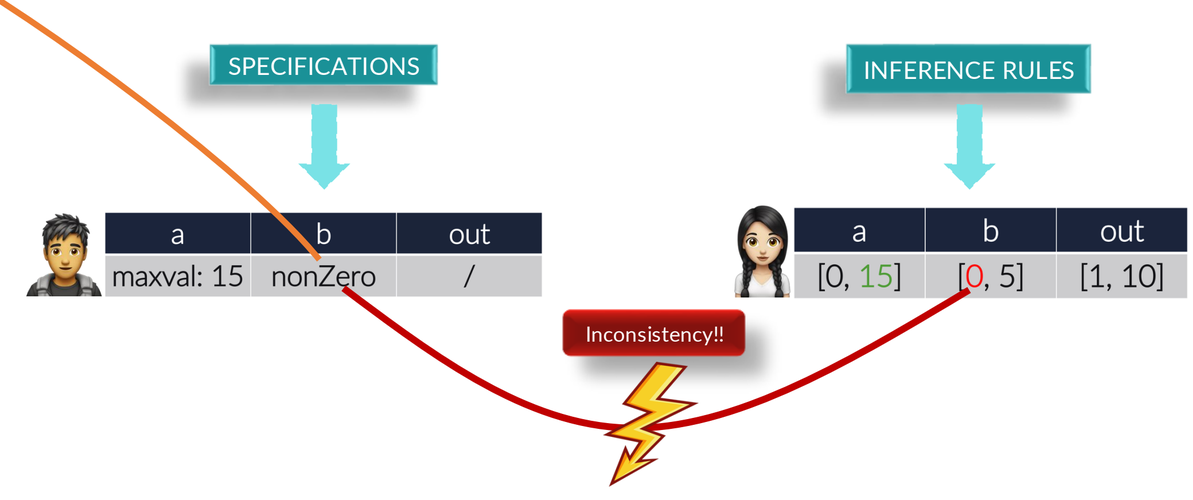
Domain-Consistency Model (DCM)

*We introduce a **formal model** that assigns each variable two value domains: one derived from the computations (**taking into account pre- and postconditions**) and one from the constraints. Inconsistencies between these domains indicate bugs.*



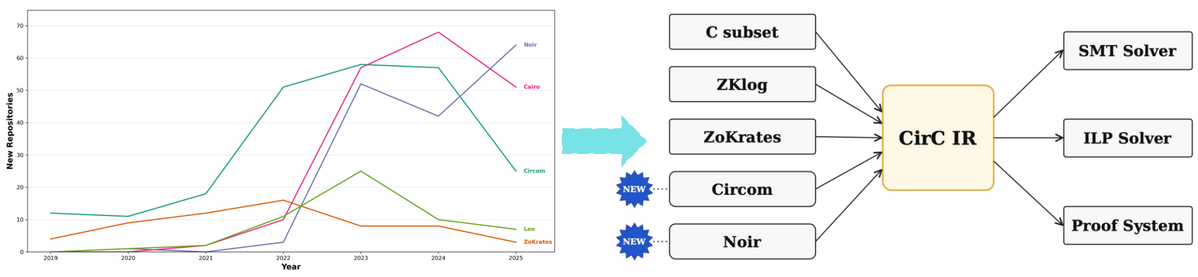
Approximations for Efficient Detection

*Iterating over all possible variable values is infeasible over large finite fields. We therefore **over-approximate** them using **specifications and inference rules**.*



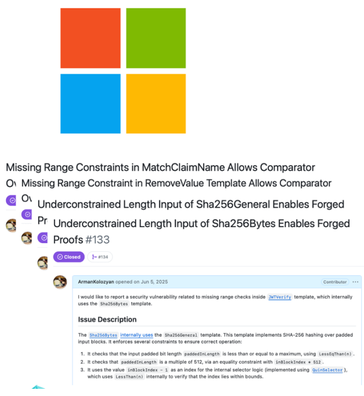
Multi-Language Support

*Our tool **operates on the CirC intermediate representation** (IR), enabling **support for multiple ZKP languages**.*



Evaluation

*We **analyzed real-world projects** and **reported zero-day vulnerabilities**, resulting in a bug bounty. Compared to an SMT-based tool, our approach is on average **two orders of magnitude faster**, with few **false positives** due to over-approximation.*



Program	LOC	#It	CCC-Check (ms)	Civer (ms)	Speedup	Verified? Tags	Array-Bounds	Divisions
escalarmulfx	284	4,831	12.152	36.58	3×	✓	✓	✗
escalarmul	145	1,531	0.870	47.58	55×	✓	✓	✓
pedersen	181	3,379	1.870	79.98	43×	✓	✓	✓
mux1	27	15	0.002	5.75	3,194×	✓	✓	✓
mux1_1	27	83	0.004	5.63	1,522×	✓	✓	✓
mux2	59	17	0.002	3.55	2,088×	✓	✓	✓
mux2_1	59	146	0.005	3.50	745×	✓	✓	✓
mux3	72	27	0.003	3.65	1,141×	✓	✓	✓
mux3_1	72	207	0.008	3.60	444×	✓	✓	✓
mux4	103	49	0.007	3.95	573×	✓	✓	✓
mux4_1	103	390	0.014	3.84	270×	✓	✓	✓
pointbits_loopback	136	6,113	0.653	257.85	395×	✓	✓	✓
babypbk	344	5,827	9.003	131.70	15×	✓	✓	✗
pedersen2	276	2,441	0.908	42.03	46×	✓	✓	✓
escalarmul_min	141	1,095	0.975	103.65	106×	✓	✓	✓
constants	24	67	0.003	3.85	1,426×	✓	✓	✓
check_bitify	37	91	0.005	23.89	4,594×	✓	✓	✓
escalarmulany	208	3,125	6.901	27.34	4×	✓	✓	✗
binsub	43	20	0.002	17.65	7,674×	✓	✓	✓
sum	51	20	0.003	20.47	7,059×	✓	✓	✓
aliascheck	63	1,320	0.121	110.98	920×	✓	✓	✓
sign	66	1,315	0.120	170.92	1,423×	✓	✓	✓
check_comps	79	236	0.258	37.33	145×	✓	✓	✗
decoder	18	9	0.018	19.86	1,128×	✓	✓	✗
bigmod_32	337	435	0.468	5,104.89	10,920×	✓	✓	✗
bigmod_22	337	410	0.388	5,101.05	13,140×	✓	✓	✗
bigsubmodp_32	221	423	0.029	87.81	3,018×	✓	✓	✓
bigmult_21	142	56	0.004	31.32	8,242×	✓	✓	✓
bigmult_22	142	77	0.007	184.75	25,660×	✓	✓	✓
bigmult_23	142	97	0.012	5,044.67	434,885×	✓	✓	✓
bigadd_15	93	129	0.011	34.64	3,268×	✓	✓	✓
bigadd_23	93	132	0.009	46.37	5,210×	✓	✓	✓
bigadd_2030	90	159	0.106	146.26	1,385×	✓	✓	✓
bigsub_23	133	71	0.005	57.62	11,081×	✓	✓	✓
bigsub_15	133	71	0.008	58.10	7,645×	✓	✓	✓

Paper

Code