# Private Sudoku solution verification from polynomial set representation

Vadym Fedyukovych
Infopulse
Kiev Ukraine

***Abstract*—We introduce private Sudoku solution verification from polynomial set representation, evaluate it's gate complexity, and present a libsnark-based implementation.**

***Index Terms*—zk-SNARK proof, polynomial set representation.**

## I. Introduction

As a simple well-known puzzle, Sudoku become a classroom example in computer science[1]. Private verification idea was explained with Sudoku and playing cards[2] [1] such that Verifier is convinced solution is valid and fits the puzzle, but do not learn it. Sudoku solution was privately verified and sold[3] with an advanced Bitcoin transaction and a SNARK proof implemented[4] with libsnark[5] [2].

In this report we suggest efficient set equality verification from polynomial set representation and introduce an entry-level SNARK proof verification example. We achieve $5n^4$ gate complexity for Sudoku $(n^2 \times n^2)$-size solution verification.

## II. Sudoku solution verification

We elaborate on Pinkas et.al. approach [1] that basically compares $3n^2$ sets of cards to decide solution is valid, in particular each set of $n^2$ cards matches a pre-defined set. We suggest polynomial set representation, as introduced in the context of set reconciliation [3] and later used for a signature scheme tolerating noisy private keys [4] and graph colorability proof [5].

### A. Sudoku characteristic polynomials

Let $\mathbb{F}_q$ be a finite field for a prime $q$, and $\{v_{x,y}\}$ be a set of field elements that is a solution to an $n^2 \times n^2$ Sudoku puzzle.

**Definition 1.** *Sudoku row, column and block characteristic polynomials for a particular row $y$, column $x$ or block $(i,j)$ of solution are*

$$f_r(y,s) = \prod_{x=1}^{n^2} (1 + s v_{x,y}) \tag{1}$$

$$f_c(x,s) = \prod_{y=1}^{n^2} (1 + s v_{x,y}) \tag{2}$$

$$f_b(i,j,s) = \prod_{x=1}^{n} \prod_{y=1}^{n} (1 + s v_{x+n(i-1),y+n(j-1)}) \tag{3}$$

To compare sets, we evaluate set characteristic polynomials at an argument $s$ chosen at random unpredictable to proving party. We illustrate row characteristic polynomial evaluation with a circuit shown on Figure 1, used later to produce and verify a proof.

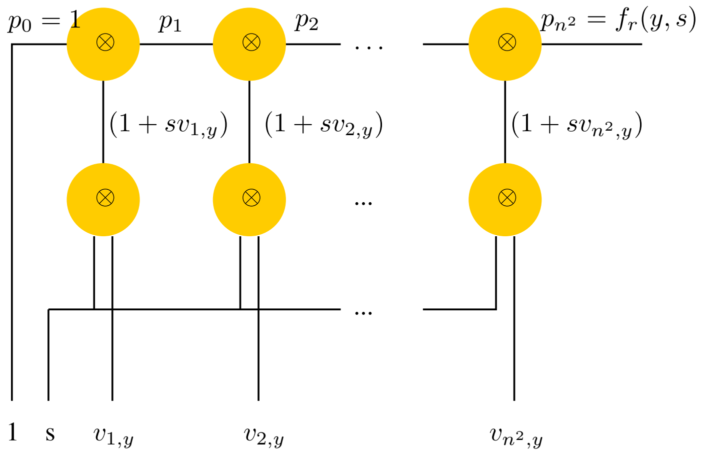

Fig. 1. Arithmetic circuit for row characteristic polynomial

### B. SNARK proof system

SNARK [7], [8] stands for succinct non-interactive argument of knowledge that rely on 'knowledge of exponent' assumption and results in a short constant-size constant-verification-time proof. Statement to be proved is expected in the form of an R1CS system of equations (constraints), with each equation of a fixed 'single multiplication' form. Such a system can be shown equivalent to an QAP instance, verifiable with polynomial division. Verification can actually be done by evaluating polynomials at a random point chosen while setup. Divisibility is verified with bilinear operation on elliptic curve

[1] Computational complexity blog, Fortnow, 2006
[2] http://www.wisdom.weizmann.ac.il/~naor/PAPERS/SUDOKU_DEMO/
[3] Transaction description on Bitcoincore, Maxwell, 2016
[4] https://github.com/zcash-hackworks/pay-to-sudoku
[5] https://github.com/scipr-lab/libsnark

points, argument value is only available the proving party as a group element (a private-coins proof system).

### C. Introductory R1CS example

To facilitate SNARK proofs and libsnark library adoption, we give a simple working example shown on Figure 2. This example illustrates how to create and verify a SNARK proof of satisfiability of a system of 3 equations and 7 variables:

$$\begin{cases} a \cdot b_1 = c_1 \\ a \cdot b_2 = c_2 \\ a \cdot b_3 = c_3 \end{cases} \tag{4}$$

Variables $b_1, b_2, b_3, c_1, c_2, c_3$ are available to verifying party (public, primary), just like Sudoku puzzle to solve. Variable $a$ is not available to verifying party (secret, witness):

$$\begin{cases} a \cdot 4 = 12 \\ a \cdot 5 = 15 \\ a \cdot 6 = 18 \end{cases} \tag{5}$$

Variables are declared and allocated at lines 11–14; public-witness separation is at line 15. System of equations is specified at lines 16–18. Proving and verifying public keys are produced at line 20. Public and witness are assigned at lines 21-23, SNARK proof is produced at line 32 and verified at lines 36 and 51. Groth16 proof system [6] and an 'alternative' BN elliptic curve with bilinear pairing are chosen at lines 3 and 1.

```
 1 #include <libff/algebra/curves/alt_bn128/alt_bn128_pp.hpp>
 2 #include <libsnark/gadgetlib1/protoboard.hpp>
 3 .../ppzksnark/r1cs_gg_ppzksnark/r1cs_gg_ppzksnark.hpp>
 4 using namespace libsnark;
 5 using namespace std;
 6 #define ppT libff::alt_bn128_pp
 7 #define FieldT libff::Fr<ppT>
 8 int main() {
 9   libff::alt_bn128_pp::init_public_params();
10   protoboard<FieldT> pb;
11   pb_variable<FieldT> a, b1, b2, b3, c1, c2, c3;
12   b1.allocate(pb, "b1"); b2.allocate(pb, "b2"); b3.allocate(pb, "b3");
13   c1.allocate(pb, "c1"); c2.allocate(pb, "c2"); c3.allocate(pb, "c3");
14   a.allocate(pb, "a");
15   pb.set_input_sizes(6);
16   pb.add_r1cs_constraint(r1cs_constraint<FieldT>(a, b1, c1), "eq1");
17   pb.add_r1cs_constraint(r1cs_constraint<FieldT>(a, b2, c2), "eq2");
18   pb.add_r1cs_constraint(r1cs_constraint<FieldT>(a, b3, c3), "eq3");
19   r1cs_gg_ppzksnark_keypair<ppT> kp =
20     r1cs_gg_ppzksnark_generator<ppT>(pb.get_constraint_system());
21   pb.val(a) = 3;
22   pb.val(b1) = 4;  pb.val(b2) = 5;  pb.val(b3) = 6;
23   pb.val(c1) = 12; pb.val(c2) = 15; pb.val(c3) = 18;
24   if(pb.is_satisfied()) {
25     cout << "SAT␣ok" << endl;
26   } else {
27     cout << "SAT␣NOT␣ok" << endl;
28   }
29   cout << "Primary␣input:" << endl << pb.primary_input() << endl;
30   cout << "Aux␣input:" << endl << pb.auxiliary_input() << endl;
31   r1cs_gg_ppzksnark_proof<ppT> pi =
32     r1cs_gg_ppzksnark_prover<ppT>(kp.pk,
33                                   pb.primary_input(),
34                                   pb.auxiliary_input());
35   bool isvalid =
36     r1cs_gg_ppzksnark_verifier_strong_IC<ppT>(kp.vk,
37                                               pb.primary_input(),
38                                               pi);
39   if(isvalid) { cout << "SNARK␣proof␣is␣valid" << endl; }
40   pb.val(c2) -= 8;
41   if(pb.is_satisfied()) { cout << "modified␣SAT␣ok" << endl; }
42   bool is_mod_valid =
43     r1cs_gg_ppzksnark_verifier_strong_IC<ppT>(kp.vk,
44                                               pb.primary_input(),
45                                               pi);
46   if(is_mod_valid) { cout << "modified␣instance␣+␣SNARK␣are␣ok" << endl; }
47   return 0;
48 }
```

Fig. 2. 'Secret multiplication by 3' implemented

## III. Implementation of Sudoku solution verification

This verification was implemented https://github.com/vadym-f/Sudoku_solvability_proof/, circuit complexity is $5n^4$ multiplication gates. Original libsnark-based circuit includes verification of correct ciphertext and proper encryption key, so direct complexity comparison is infeasible.

## References

[1] R. Gradwohl, M. Naor, B. Pinkas, and G. N. Rothblum, "Cryptographic and physical zero-knowledge proof systems for solutions of sudoku puzzles," in *Proceedings of the 4th International Conference on Fun with Algorithms*, ser. FUN'07, 2007, pp. 166–182. [Online]. Available: http://dl.acm.org/citation.cfm?id=1760607.1760623

[2] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 781–796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[3] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory*, vol. 49, no. 9, pp. 2213–2218, 2003.

[4] V. Fedyukovych, "A signature scheme with approximate key matching (in Russian)," in *Information Technology and Systems Conference*, 2009, pp. 396–400. [Online]. Available: http://www.iitp.ru/ru/conferences/539.htm

[5] G. Di Crescenzo and V. Fedyukovych, "Zero-knowledge proofs via polynomial representations," in *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, ser. MFCS'12, 2012, pp. 335–347. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32589-2_31

[6] J. Groth, "On the size of pairing-based non-interactive arguments," Cryptology ePrint Archive, 2016. [Online]. Available: https://eprint.iacr.org/2016/260

[7] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," Cryptology ePrint Archive, 2012. [Online]. Available: https://eprint.iacr.org/2012/215

[8] B. Parno, C. Gentry, J. Howell, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," Cryptology ePrint Archive, 2013. [Online]. Available: https://eprint.iacr.org/2013/279