Primer on NIZK Proofs for Secure Computation [*]

Stéphane Vincent

Sikoba Research

December 2018

Abstract

Since their conception in the 1980s, ZKPs have emerged as a fundamental object in modern cryptography. They have found numerous applications as a building block in other cryptographic constructions such as: identification schemes [1], group signature schemes [2], anonymous credentials [3], voting [4], and secure computation [5].

In this paper, we provide an overview of ZKPs for secure computations. We introduce circuit representation, provide a description of Pinocchio, a C-to-arithmetic-circuit compiler, and present two NIZP systems based on different cryptographic assumptions.

1 Introduction

Herein, we will use the terminologies "secure computation" and "smart contracts" interchangeably.

The Ethereum system uses highly expressive smart contracts to enable complex transactions. Smart contracts, like any other blockchain transactions, are public and provide no inherent privacy; yet smart contracts face the issue of scalability. To bring privacy to smart contracts and to maintain good performance of the platform, zero-knowledge proofs have been proposed as a tool to enable complex smart contract executions.

Zero-knowledge proofs (ZKPs) provide the ability to convince a verifier that a statement is true without revealing the secrets involved. A ZKP allows a prover P with an input x to persuade a verifier V that $f(x) = y$. The prover should be unable to cheat, and the verifier should learn nothing more from receiving the message, than the mere fact that the statement involved is true.

In 1989, Goldwasser, Micali, and Rackoff [6] raised a question that turned out to be of great interest for cryptography: what else is revealed by the proof of statement? Suppose you enter a bar on a Friday night and anyone who wishes to buy anything from that establishment has to show his ID to prove his age. Providing your document ID gives the bartender the opportunity to know your DOB, your name, and your address. Could you avoid this pitfall, showing him that you are over the legal drinking age without revealing detailed personal information? This boils down to the following general question:

[*]Research supported by Fantom Foundation

Is it possible to produce a proof of the truth of a statement (i.e. a proof that you are over the legal drinking age?), that yields nothing (not even the DOB) but the validity of this statement?

Goldwasser, Micali, and Rackoff's positive answer to this question has laid the foundations of cryptographic primitives known as zero-knowledge proofs.

ZKP systems are usually interactive: they involve exchanging several messages between the prover and the verifier. In some scenarios, interactions are undesirable; communication channels can be vulnerable and each round of communication adds extra cost in bandwidth. In these situations, a specific type of zero-knowledge proofs, called non-interactive zero-knowledge proofs (NIZKPs), are used which consist of a single flow from the prover to the verifier. As communications over the blockchain are expensive and the computational power of smart contract on blockchains is limited, NIZKPs are favoured over ZKPs for applications using blockchain technologies.

Since the introduction of NIZKPs by Blum, Feldman and Micali [7] much effort has been spent on reducing their size. The author in [8] suggests protocols on circuit satisfiability since computer operations are naturally represented as circuits, hence a central component of a secure computation protocol is the circuit representation for the function (resp. the smart contract) to be evaluated (resp. to be executed).

The rest of the paper is organized as follows: in section 1, we introduce the necessary background on circuit representation. In section 2, we provide a description of Pinocchio, a C-to-arithmetic-circuit compiler. In section 3, we present two NIZP systems based on different cryptographic assumptions. In section 4, we make some concluding remarks.

2 Secure Computation & Circuit Representation

A natural way for a programmer to express a proof is in the form of an ordinary computer program written in a language like C. The program takes the data as input, tests them in various ways, and returns either success or failure. But cryptographic algorithms typically need problems to be expressed as a set of equations and circuits are the standard model for representing them. Thus finding ways to represent problems as circuits are of central importance in secure computation.

The world of generic secure computation protocols is split into protocols that allow to evaluate Boolean circuits and protocols that evaluate arithmetic circuits. Arithmetic circuits are a more expressive variant of binary circuits: instead of restricting the values carried by wires to 1s and 0s, arithmetic circuits consider values in the range $0, ..., p-1$ with p a large prime number. Both models can express the same kind of functions (arithmetic operations can be performed by a Boolean circuit and vice versa).

The number of cryptographic operations, that dominates the complexity of secure computation protocols, are usually proportional to the number of gates in the circuit. Thus, one can gain in efficiency by choosing the right model of computation: for instance computing a multiplication between l-bit numbers with a Boolean circuit might be up to l^2 more expensive than with an arithmetic one. On the other hand, non-linear operations, like comparisons, can be computed more efficiently using the Boolean representation. In the following, we focus on the case of arithmetic circuits.

An arithmetic circuit (AC) is a virtual construction of arithmetic gates that are connected by wires (forming a directed acyclic graph). Each gate has two input wires and one output wire and perform either a multiplication ($\times$) or addition ($+$) operation on the inputs. A complete circuit has free input wires and free output values. A legal assignment of the values of the wires as those which satisfy the circuit, i.e. each wire is assigned a value where the output of each gate correctly corresponds to the product or sum of the inputs (i.e. the gate is consistent).

We consider the equation $f(w_1, w_2, w_3) = w_1 \cdot w_2 \cdot (w_2 + w_3)$ where the circuit representation is shown in Fig.1. Given the input values (w_1, w_2, w_3), the aim of secure computations and ZKP systems is to evaluate the intermediate values (w_4, w_5), the output value (w_6), and to generate a proof that $w_6 = f(w_1, w_2, w_3)$

3 From Source Code to Arithmetic Circuit

In the following, we briefly discuss the procedure to convert the original code that can be written in domain-specific language or in general purpose language into a circuit. We do this by explaining the functionality of the compiler Pinocchio [9]. Pinocchio transforms an input C program f into an arithmetic circuit formula C.

The C-to-arithmetic-expression compiler consists of a python program that takes as input a function `void compute (struct In *in, struct Out *out)` whose parameters identify the set of input and output values. The compiler understands a subset of C, including global, function and block-scoped variables; arrays, structs, and pointer; function calls, conditional, loops. It also understands arithmetic and bitwise operators. Similarly, loops with statically-evaluate termination conditions are automatically unrolled completely and the only scalar type supported is **int**.

Pinocchio is not for all C programs, it is not meant to turn arbitrary C code into verifiable computation schemes: computations are statically bounded; loops are unfolded; and circuitry is needed for all branches.

Consider the function `void compute` that evaluates $f(w_1, w_2, w_3) = w_1 \cdot w_2 \cdot (w_2 + w_3)$

```
void compute (struct In *in, struct Out *out)
{
 out->ou1 = in->in1 * in->in2 * ( in->in2 + in->in3 );
}
```

This function corresponds to the following arithmetic circuit:

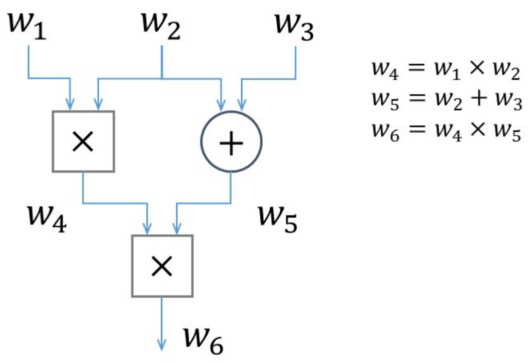

Figure 1: A simple arithmetic circuit with three gates, three input wires (w_1, w_2, w_3), one output wire (w_6), and two internal wires (w_4, w_5)

The output from the compilation process is given in Table 1. Based on this representation, the goal is to evaluate the circuit and to produce a proof of correct execution.

Left Input Gate	Right Input Gate	Gate Type	Output Gate
w_1	w_2	mul	w_4
w_2	w_3	add	w_5
w_4	w_5	mul	w_6

Table 1: Pinocchio's compiler converts a C program into an arithmetic circuit.

4 Proofs of Arithmetic Circuit Satisfiability in Zero-Knowledge

The last few years saw the emergence of new ZK proof systems based on different cryptographic assumptions from discrete logarithm approach to pairing based cryptography. In the following sections, we briefly discuss two different algorithms for circuit satisfiability.

4.1 Discrete Logarithm Based ZK Proofs for Arithmetic Circuit

Currently the most efficient discrete logarithm based zero-knowledge proofs for arithmetic circuits are the ones by Groth [10], Seo [11], and Bootle [12]. The soundness of the proof relies on the well-established discrete logarithm assumption in prime order groups and Fiat-Shamir transformations are applied to the proofs to make them non-interactive, as well as full zero-knowledge.

At the heart of the proof system is an efficient zero-knowledge argument of knowledge of opening of Pedersen commitments satisfying inner product relations. The inner product argument requires linear computation for both the prover and the verifier. The authors in [12] handle addition gates with linear equations thus commitments to the input and output wires of addition gates are not required yielding performance improvement proportional to the number of addition gates. This parallels Quadratic Arithmetic Program constructions from circuits (see section 4.2).

The authors in [12] also develop a scheme to commit to a polynomial and later reveal the evaluation at an arbitrary point. However, current implementations have proofs that for modest circuit sizes range into the hundreds of kilobytes or have verification times that are linear of the circuit and make verification of large statements onerous to applications such as blockchains where space and bandwidth are constrained and proofs are expected to be verified many times in a performance-critical process.

4.2 Pairing Based ZK Proofs for Arithmetic Circuit

Zero-Knowledge Succint Non-interative Arguments of Knowledge (zk-SNARK) refer to an implementation of a general purpose proof system based on bilinear maps. The authors in [13] manage to produce a verifiable computation system with constant-size proofs where verification times for a computation are faster than running the computation locally.

In the SNARK framework, a statement encoded as an arithmetic circuit is converted into a construct called a Quadratic Arithmetic Program (QAP) which consists of a set of polynomial equations. The statement can then be proved by demonstrating the validity of this set of equations at a single point. The main advantage of the SNARK method is that the verifier only needs to perform a few elliptic curve (pairing) operations (taking a few milli-seconds) and the proof is very small (288 bytes) and independent of the circuit size.

To instantiate proofs of arithmetic circuit satisfiability based on bilinear pairings requires the use of special pairing friendly elliptic curves. This precludes the use of standard cryptographic elliptic curve such as secp256k1.

The authors in [14] present a protocol employing zk-SNARK method to execute smart contracts as part of a blockchain transaction validation.

5 Summary

The growing demand for blockchain and smart contract technologies sets the challenge of protecting them from intellectual property theft and attacks: security, confidentiality and privacy are the key issues holding back their adoption. The solution is inter-disciplinary (i.e. cryptography and formal verification technique).

In this paper, we introduced the key elements for outsourcing secure computations. The methods described in the previous sections enable the zero-knowledge proof of statements that involve elliptic curve public-private key relationship simultaneously with general arithmetic circuit satisfiability providing a compiler to translate a source code to an arithmetic circuit description.

References

[1] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptol.*, 1(2):77–94, 1988.

[2] Z. Fucai, Xu Jian, Li Hui, and W. Lanlan. Group signature based on non-interactive zero-knowledge proofs. *China Communications*, 8:34–41, 2011.

[3] C. Garman, M. Green, and I. Miers. Decentralized anonymous credentials. *ACR Cryptology ePrint Archive, 2013:622*, 2013.

[4] H. Lipmaa, N. Asokan, and V. Niemi. Secure vickrey auctions without threshold trust. In *Proceedings of the 6th International Conference on Financial Cryptography*, FC'02, pages 87–101. Springer-Verlag, 2003.

[5] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008*, pages 192–206. Springer Berlin Heidelberg, 2008.

[6] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[7] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112. ACM, 1988.

[8] I. Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with preprocessing. In *Advances in Cryptology — EUROCRYPT' 92*, pages 341–355. Springer Berlin Heidelberg, 1993.

[9] `https://github.com/amiller/pinocchio`.

[10] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In *Advances in Cryptology - CRYPTO 2009*, pages 192–208. Springer Berlin Heidelberg, 2009.

[11] Jae Hong Seo. Round-efficient sub-linear zero-knowledge arguments for linear algebra. In *Public Key Cryptography – PKC 2011*, pages 387–402. Springer Berlin Heidelberg, 2011.

[12] J Bootle, A Cerulli, P Chaidos, J Groth, and C Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. volume 9666 of *Lecture Notes in Computer Science*, pages 327–357. Springer, Berlin, Heidelberg, 2016.

[13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology – EUROCRYPT 2013*, pages 626–645. Springer Berlin Heidelberg, 2013.

[14] A. Covaci, Madeo S., Motylinski P., and Vincent S. NECTAR: non-interactive smart contract protocol using blockchain technology. *CoRR*, abs/1803.04860, 2018.