State of the Art in Verifiable Computation [*]

Dmitry Khovratovich

Sikoba Research

21st December 2018

1 Introduction

We consider the following scenario. There are several parties $P_1, P_2, \dots, P_n$ that want to conduct computation $\mathcal{C}$ on a dataset $\mathcal{D}$. They also desire that alleged result $\mathcal{D}' = \mathcal{C}(\mathcal{D})$ can be publicly verified. We distinguish the following cases:

- If $\mathcal{D}$ is public, then the verification $\mathcal{D}' = \mathcal{C}(\mathcal{D})$ should take less time than repeating $\mathcal{C}$. Ideally, the verification time should be independent of the size of $\mathcal{C}$.
- If $n = 1$ (i.e. there is only one party) and $\mathcal{D}$ is partly private, then we look for a zero-knowledge proof of computation and knowledge, i.e. the proof that P_1 knows D such that $\mathcal{D}' = \mathcal{C}(\mathcal{D})$.
- If $n > 1$ and each party knows a subset of $\mathcal{D}$, we call it secure multiparty computation (MPC). However, by default the MPC protocols do not provide publicly verifiable results, so a special modification is required. Such protocols are called MPC-based and will be considered in the report.

In a particular application we may require the following additional properties:

- Prover scalability: the overhead incurred to proving parties should be linear in the size of $\mathcal{C}$ (denoted by $|\mathcal{C}|$), and the factor should be reasonably small.
- Verification scalability: that the verification time scales with $|\mathcal{C}|$ better than a naive approach (recompute $\mathcal{C}$). We look for the methods with verification time being proportional to $\log |\mathcal{C}|$ and ideally being a constant time.
- Post-quantum security: the cryptographic primitives used in the protocol are not vulnerable to quantum computers. This covers most of symmetric cryptography, as blockciphers and hash functions lose only half of their security level. Among asymmetric cryptography, lattice- and code-based cryptosystems are the most promising in this regard, and the recent competition run by NIST [nis18] should produce a number of efficient schemes.
- Transparency: the protocol does not require any privately generated randomness for the setup, which must be kept secret. Instead, all the parameters needed for public verification should be also created in public.

[*]Research supported by Fantom Foundation

Over time, several computation models have been proposed. However, we identify the following properties that are shared by most models:

- Computation is typically represented as a circuit, where each node represents a single computational item, incoming wires stand for input variables of the item, and the outcoming wire stands for the result of the item execution. The simplest circuits are *boolean* where each item is a boolean operation. More advanced computation can be represented as arithmetic circuits, where every wire corresponds to a field element, and the item is a field operation. A recent generalization of the arithmetic circuits is the algebraic representation (AIR) [BBHR18], where items are replaced by polynomial equations over the internal variables.

- Generic computation models do not follow the von Neumann architecture with random access memory. In the circuit or AIR model all the variables that can be potentially referenced contribute to the complexity. As a result, the amount of memory is typically limited, and a special setup is needed to handle very large datasets [ZGK$^+$17a].

The first notable results in verifiable computation date back to 1990s. A series of results called Probabilistically Checkable Proofs (PCPs) [BFLS91] were the first that permitted polylogarithmic verification, i.e. the verification time was proportional to $\log^c |\mathcal{C}|$ for some positive c. However, the proving time for reasonable values of c was so high that no practical implementation appeared.

The researchers continued to look at different approaches, which we summarize below:

- Proof systems using homomorphic public-key cryptosystems, notable for compact proofs. The most famous example is ZK-SNARK [BCG$^+$13].

- Proof systems based on the hardness of discrete logarithm problem (DLP), notable for no need in the trusted setup. The recent example is Bulletproofs [BBB$^+$18].

- Those based on efficient PCPs and conversions of interactive proofs into zero-knowledge ones. Examples: ZK-STARKs [BBHR18], Hyrax [WTS$^+$18].

- Proofs derived from secure multiparty computation protocols: ZKB++ [CDG$^+$17], Ligero [AHIV17].

- Proofs based on the aggregation of smaller proofs: incrementally verifiable computation (IVC).

2 Definitions

The following notation is common in the zero-knowledge proof systems:

- The proof-of-knowledge protocol is *sound* if the adversary can not create a fake proof if he does not have knowledge. We distinguish between *computational* (adversaries with reasonable capabilities) and *perfect* (all adversaries) soundness. This is usually proven by showing that anyone who can create acceptable proofs necessarily has the knowledge.

- The proof-of-knowledge protocol is *complete* if the honest prover can convince an honest verifier. This is proven by showing that for every correct statement there is a protocol execution that makes the verifier to accept.

- The proof-of-knowledge protocol is *zero-knowledge* if the verifier learns nothing from the protocol execution. This is proven by showing that the protocol transcript can be created by a simulator without knowledge.
- The communication complexity of the protocol is the amount of data passed among the parties during the execution.
- *Prover complexity* is the amount of work to be done by the executor of computation who wants to prove its correctness.
- *Verifier complexity* is the amount of work to be done to verify the proof. It is lower bounded by the communication complexity.
- *Commitment* is a non-invertible function F of input X and possibly random value r. We say it is binding if it is infeasible to find another X', r' such that $F(X', r') = F(X, r)$. We say it is hiding if X can not be deduced from $F(X, r)$.

3 Approaches

One may think that the verifiable computation problem when input $\mathcal{D}$ is public is easier, but apparently it is not the case. Since most proofs of knowledge can be turned into zero-knowledge protocols, the vast majority of the approaches we consider below work in both settings with about the same performance. Unless there is an exception we do not stress this property for each method.

3.1 Interactive (Oracle) Proofs

It has been known since the seminal result by Shamir [Sha92] that the correctness of quite rich statements can be proven interactively. In the context of verifiable computation, we can define the language Λ as a set of pairs $\{(\mathcal{D}, \mathcal{D}')\}$ such that $\mathcal{C}(\mathcal{D}) = \mathcal{D}'$ for deterministic computation $\mathcal{C}$. Then one can assert that $\mathcal{D}'$ is the result of $\mathcal{C}$ applied to $\mathcal{D}$ by proving that $(\mathcal{D}, \mathcal{D}') \in \Lambda$. Most interactive protocols can be turned into zero knowledge versions where $\mathcal{D}$ or $\mathcal{D}'$ are available only as commitments, but in general this process can lead to very expensive solutions. In the first papers the amount of computations the prover had to do was exponential in $|\mathcal{C}|$.

In Interactive Oracle Proofs (IOP) [BCS16] the verifier does not read the proof in its entirety. Instead, he queries random parts of the proof to be convinced. This approach yields more efficient provers. The IOP protocol can be turned into non-interactive one, for example, using the Fiat-Shamir technique.

The IOP framework first inspired the verifiable computation without zero knowledge [BBC+17] with reasonable but large proving time. The version of this protocol enhanced with zero knowledge and other optimizations is called **ZK-STARKs** [BBHR18], where verifier and communication complexity are logarithmic of $|\mathcal{C}|$, proving time is linear of $|\mathcal{C}|$. STARKs operate on algebraic represenation of computation, where there is a chain of internal states $A_1, A_2, \ldots, A_n$ and a number of polynomial relations among those: $p_i(A_{i_1}, A_{i_2}, \ldots, A_{n_i}) = 0$. It is then asserted that the computation is correct if and only if it satisfies the relations. One can trade the communication complexity for the verification time, and obtain a scheme with constant proof/communication size and linear verification time [RRR16]. When polynomials are of degree 2 and follow the R1CS structure (like for SNARKs), the performance can be improved by a constant factor [BCR+18]: 15x faster and smaller proofs and the same improvement on the verifier side, the proof system called **Aurora**.

When the computation is a set of small executions that can be done in parallel, a technique called **Hyrax** [WTS+18] can be applied, which provides $\log|\mathcal{C}|$-size proof for circuits of small depth. The constants are low enough so the proof size and time are among the lowest (except for **Bulletproofs**).

3.2 Homomorphic public-key cryptosystems

The approaches based on homomorphic PKC use the following property of certain function F: $F(x)^y = F(x \cdot y)$. It allows computing F on the xy product without knowing x. Examples of such functions are scalar multiplication of elliptic curve points and exponentiations in a prime field. This property has been used to create constant-size zero-knowledge proofs of computation integrity starting with [Gro10], where the proving time was quadratic of $|\mathcal{C}|$. This was prohibitive for most applications till the emergence of the Pinocchio scheme [PHGR13] and its derivatives [BCG+13, Gro16], mostly known as ZK-SNARKs. All these schemes require the computation to be described as an arithmetic circuit with additions and multiplications in a prime field, or alternatively in the algebraic form as a system of polynomial constraint of degree 2 and form

$$V(X) \cdot W(X) = U(X) \tag{1}$$

where V, W, U are linear functions of internal and input variables. The proofs are extremely small (less than 1KB) and are independent of circuit size, but the downside is the setup. Concretely, for public verifiability of proofs the trusted party must compute the (specially prepared) polynomial constraints on a secretly chosen input, commit to the results (called proving key), and destroy the secret. If the secret is leaked, an adversary can forge a proof.

ZK-SNARKs gained popularity from their use in the privacy-preserving Zcash cryptocurrency [Zca18], where the sender, the receiver, and the amount of the transaction are all hidden from public but correctness is guaranteed by ZK-SNARKs. The trusted setup for the ZK-SNARKs was organized as a secure multiparty computation of the proving key. Assuming that at least one participant of the procedure destroyed his secret, the result is secure.

In the same trust model one can create compact proofs for statements that cover large datasets. For example, consider certain class of computation such as selection and join queries in the SQL model. Schemes IntegriDB [ZKP15] and vSQL [ZGK+17a, ZGK+17b] allow compact proofs of correctness for responses to these queries, but the client must first make a trusted setup for the entire database. The advantage is that the proof generation, communication and verification time do not depend on the database size, only on the query, and verification is almost instant.

3.3 Discrete logarithm problem

This method is somewhat similar to the one based on homomorphic PKC, but does not require a trusted setup. The idea is again to consider an algebraic computation with its internal registers linked by special degree-2 polynomials (Equation (1)). The technique is different though, which results in larger proofs: originally [Gro11] sublinear ($|\mathcal{C}|^{1/3}$), later logarithmic ($2\log|\mathcal{C}|$). The most recent version of the technique is called **Bulletproofs** [BBB+18]. There is no need in a trusted setup but the verifier has to do amount of work equal to the original computation or more. As a result, a natural application of this technique are small but repetitive computations such as range proofs in cryptocurrencies, where a single range proof takes less than a KB and is fast to verify. This has just been implemented for Monero [mon18] and is in consideration for Bitcoin.

3.4 MPC derivative

Secure Multiparty Computation (MPC) is a vast area of research where multiple parties participate in a common protocol to compute a certain function F. Even though each party can be assured that the result is computed correctly, this can not be used for public verification per se. To be eligible for the public verification, the protocol is transformed so that the prover executes it alone, that's why it got called *MPC in its head* paradigm. In this concept, the prover sets up virtual participants of MPC and executes the protocol among them. Then he commits to the transcript of the protocol and to the view from one of the parties. Most recent examples are ZKB++ [CDG+17] and Ligero [AHIV17]. The prover and verifier time are linear in circuit size $|C|$, whereas the communication complexity varies: also linear for ZKB++ and $\sqrt{|\mathcal{C}|}$ in Ligero. Concrete implementations can be quite efficient.

3.5 Incrementally Verifiable Computation

The idea to decompose the computation $\mathcal{C}$ into a series of computations $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$, run them all, produce proofs $P_1, P_2, \ldots, P_k$, and then combine them together. The final proof P states that (1) all proofs P_i are correct for computations $\mathcal{C}_i$; (2) the computations $\mathcal{C}$ is a composition of $\mathcal{C}_i$. This approach has been known for a while, but was effectively implemented only recently with ZK-SNARKs as the underlying proof system [BCTV17]. The resulting proof system inherits all the properties of the underlying one, except that the verifier becomes more efficient.

4 Implementations

The existing implementations for verifiable computation systems can be classified into the following groups by maturity and activity:

1. Production-ready code, which has been in active development and use for long time, has been used multiple times in other projects. Currently we can name only libsnark [lib18], the implementation of ZK-SNARKs based on the Pinocchio system. However, libsnark has not been in active development in recent months partly due to the migration of Zcash developers to the bellman library [bel18].

2. Implementations that are in active use but not old enough to speak about long-term support and development. Here we name dalek-cryptography [dal18] as the implementation of Bulletproofs system, ZoKrates as a SNARK compiler for Ethereum [zok18].

3. Implementations that were prepared and published for one particular paper, but have not been updated since then. Examples: ZKB++ [zkb18], Hyrax [WTS+18].

5 Comparison

In this section we compare existing methods for verifiable computation.

5.1 Properties

In Table 1 we provide a comparison on various mechanisms for verifiable computation according to the list compiled in Section 1. The exception is IVC, as its properties are determined by the underlying proof system.

We note that in all examples the prover time is linear in the circuit size, except for a specially crafted SQL statements in vSQL [ZGK+17a]. However, the setup time is still linear in the database size in vSQL with a significant overhead. We also stress that the actual constant or logarithmic factors over the circuit size for the prover are quite different, both in concrete examples and asymptotically. The actual factors may differ by a few orders of magnitude.

All the methods are different in the verifier complexity. While HPKC/SNARK methods are the easiest for the verifier, the DLT- and MPC-based approaches incur for the verifier about the same effort as for the prover. For Bulletproofs this property is not explicit, as this approach has been promoted primarily for range proofs, which are quite efficient.

Most of the approaches are not post-quantum secure, except for STARKs and MPC-based. This might not be a big problem, as the post-quantum insecurity actually means that quantum-equipped adversaries will be able to forge proofs. However, they won't be able to break the zero-knowledge property of existing ones. Therefore, if proofs are rather short-lived, it should be possible to switch to a quantum-secure method around the time when the quantum computers become powerful.

The trusted setup requirement is present in HPKC-based methods. Thus the compactness of proofs comes together with the increased privacy assumptions. In real-world instantiations of HPKC proofs, such as Zcash, an extra procedure was carried on to limit the risk of setup compromise. Concretely, several independent parties were chosen to carry the setup, and a dedicated multiparty computation protocol [BGM17] was used to generate the parameters of the trusted setup.

	Prover	Verifier	Post-quantum	Trusted setup
I(O)P: ZK-STARK	Linear	Logarithmic	Yes	No
Hyrax	Linear	Logarithmic	No	No
HPKC: Pinocchio/libsnark/ bellman	Linear	Constant	No	Yes
vSQL	Linear in query size	Constant	No	Yes
DLT Bulletproofs	Linear	Linear	No	No
MPC Ligero, ZKB++	Linear	Linear with sublinear proofs	Yes	No

Table 1: Properties of verifiable computation systems

5.2 Performance

In these section we present comparative benchmarks. First we note that it is really difficult to compare "apples to apples" here, as ideally all the proof systems should be run on the same machine and applied to the same computation. Unfortunately, there is no such benchmark in a single paper, so we have tried to aggregate data from different sources. We have selected 1 mln multiplication gates as a reference point for a medium-size computation. This is roughly equivalent to 40 calls of the SHA-256 hash function or 600 calls of the Pedersen hash function, thus being slightly less than the circuit needed to be verified in the first version of the Zcash cryptocurrency.

We have scaled all available benchmarks to the proof of 1 mln-gate computation and to a machine equipped with a 8-core CPU with 16 GB of RAM. Such computation is typical for set membership proofs in large Merkle trees (based on SHA-256 and with many layers); for smaller computations the numbers should be reduced. The performance figures were scaled down in [BBHR18], where 32 cores were used, but scaled up in all others. We take the libsnark/Pinoccio performance from [BBHR18], the Ligero performance from [AHIV17], and the Aurora benchmark from [BCR+18], as 1 mln gate circuit was benchmarked there explicitly. For ZK-STARKs we use an estimate of 600 Pedersen hashes from the very recent talk at DevCon4 (privately communicated). For ZKBoo [CDG+17], Bulletproofs [BBB+18], and Hyrax [WTS+18] we extrapolate the performance of single SHA-256 to 40 SHA-256 invocations in their papers.

Depending on the application requirements, one can select the proof system based on the table 2. Where the proof generation is the bottleneck, Ligero, ZK-STARKs and ZKBoo demonstrate the highest performance. The smallest proof size is in ZK-SNARKs (libsnark or bellman), followed by Bulletproofs. Verification time is the smallest in ZK-SNARKs and ZK-STARKs, whereas Aurora, Ligero and ZKBoo are in the next tier.

1 mln multiplication gates			
	Proof time	Proof size	Verification time
libsnark Pinocchio	32 s	250 B	40 ms
Ligero	5 s	200 KB	2 s
ZK-STARK	1 s	78 KB	18ms
ZKBoo	5 s	100 MB	2 s
Hyrax	100 min	100 KB	1 min
Aurora	7 mins	250 KB	1.5 sec
Bulletproofs	5 mins	1.7 KB	15 sec

Table 2: Performance of notable VC implementations.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE, 2018.

[BBC+17] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. In *EUROCRYPT (3)*, volume 10212 of *Lecture Notes in Computer Science*, pages 551–579, 2017.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.

[BCR$^+$18] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. *IACR Cryptology ePrint Archive*, 2018:828, 2018.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *TCC (B2)*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016.

[BCTV17] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017.

[bel18] bellman: a zk library., 2018. `https://github.com/zkcrypto/bellman`.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, pages 21–31. ACM, 1991.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. `https://eprint.iacr.org/2017/1050`.

[CDG$^+$17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *ACM Conference on Computer and Communications Security*, pages 1825–1842. ACM, 2017.

[dal18] dalek-cryptography: Bulletproofs, 2018. `https://github.com/dalek-cryptography/bulletproofs`.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments, 2010.

[Gro11] Jens Groth. Efficient zero-knowledge arguments from two-tiered homomorphic commitments. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 431–448. Springer, 2011.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT (2)*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[lib18] Scipr lab. libsnark: a c++ library for zksnark proofs., 2018. `https://github.com/scipr-lab/libsnark`.

[mon18] Monero becomes bulletproof, 2018. `https://medium.com/digitalassetresearch/monero-becomes-bulletproof-f98c6408babf`.

[nis18] NIST: Post-quantum cryptography standardization, 2018. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, pages 49–62. ACM, 2016.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[WTS$^+$18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *IEEE Symposium on Security and Privacy*, pages 926–943. IEEE, 2018.

[Zca18] ZCash protocol specification, 2018. `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`.

[ZGK$^+$17a] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE Symposium on Security and Privacy*, pages 863–880. IEEE Computer Society, 2017.

[ZGK$^+$17b] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vsql. *IACR Cryptology ePrint Archive*, 2017:1146, 2017.

[zkb18] Generalized ZKB++, 2018. `https://github.com/IAIK/gzkbpp`.

[ZKP15] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. Integridb: Verifiable SQL for outsourced databases. In *ACM Conference on Computer and Communications Security*, pages 1480–1491. ACM, 2015.

[zok18] ZoKrates: toolbox for zkSNARKs on Ethereum., 2018. `https://github.com/Zokrates/ZoKrates`.