**International Journal of Civil Engineering and Technology (IJCIET)**
Volume 9, Issue 7, July 2018, pp. 45–60, Article ID: IJCIET_09_07_005
Available online at http://www.iaeme.com/ijciet/issues.asp?JType=IJCIET&VType=9&IType=7
ISSN Print: 0976-6308 and ISSN Online: 0976-6316



# VERIFIABLE COMPUTATIONS: APPROACHES AND PERSPECTIVES

**Babenko L. K., Burtyka F. B., Trepacheva A. V.**
Southern Federal University, Bolshaya Sadovaya ul. 105 / 42,
Rostov-on-Don, 344006, Russia

**ABSTRACT**

*The paper reviews various existing approaches to construct verifiable computing schemes. Such schemes allow a weak client to delegate the computation of function F(X) to the powerful untrusted worker (server). The worker returns the result of the function evaluation y=F(X) to the client and provides a proof that F was computed correctly. We compare different verifiable computing schemes with each other and discuss their flaws and vulnerabilities. The most attention is focused on approaches using some cryptographic primitives. In particular we concentrate on those that exploit fully homomorphic encryption. Finally, we outline open problems and predict the most perspective directions.*

**Key words:** verifiable computations, untrusted environment, cloud computations, fully homomorphic encryption, functional encryption, threshold homomorphic encryption, interactive proof.

**Cite this Article:** Babenko L.K., Burtyka F.B and Trepacheva A.V., Verifiable Computations: Approaches and Perspectives, International Journal of Civil Engineering and Technology, 9(7), 2018, pp. 45–60.
http://www.iaeme.com/IJCIET/issues.asp?JType=IJCIET&VType=9&IType=7

## 1. INTRODUCTION

Cloud computing is a very popular service allowing outsourcing data and computations to the powerful cloud server. Low-performance clients like smartphones or netbooks are forced to work today with computationally complex applications such as cryptography or working with photos. And proper cloud services can help them to cope with their tasks.

The reduction in the cost of cloud computing services led to the fact that many companies refused to buy and maintain their own computing system. Instead they prefer to buy working time in clouds. Critical applications are often transferred to the cloud servers. Thus it is important to avoid errors in the computation.

However, usually cloud server is an untrusted computational environment. It may be controlled by the adversary that wants to steal the client's data or corrupt the result of computations. So it's desirable to have some tools for protecting the data and verifying the correctness of any outsourced computation.

Protecting sensitive data from theft in clouds refers to *confidentiality issue*. It's extensively studied nowadays [1]. A variety of cryptographic tools is proposed for its solving. One of the most powerful and impressive primitive is *fully homomorphic encryption (FHE)* [2]. It gives an ability to conduct an arbitrary computation over encrypted data without knowledge of decryption key.

Correctness of the computation result is related to another security issue: *integrity*. Traditionally, information integrity means that the data has not been tampered with, altered or damaged by the adversary. But in the case of delegated computations the concept of integrity acquires a new meaning: the consistency of the computation result with the input data and the user-defined computation algorithm. The adversary should not be able to falsify the result.

To provide the correctness of the outsourced computations *verifiable computations schemes (VCS)* have been introduced. This notion dates back to Babai's work [3]. It has been studied under various terms, like for example: checking computations [3], delegating computations [4], certified computation [5] and verifiable computing. The term *verifiable computing* itself was formalized by Gennaro et al [6].

In a nutshell VCS solves the following problem. The client specifies a function $F$ and input $x$ and requests a worker to evaluate $y = F(x)$ and return $y$. The worker has to provide a proof $\rho$ that $F$ was computed correctly. And if indeed it was, the client must accept $y$. Otherwise he rejects $y$ with overwhelming probability. The key ingredient here is that verification of $\rho$ should require a less computational work than the computation of $F$. If it's not true it is easier for the client to compute $F$ locally.

To date back a huge variety of VCS has been proposed. The majority of them can be divided into two classes*: interactive* and *non-interactive.* In the first case the client interactively asks the worker about the computation performed. The number of communications between client and worker is greater than two. In non-interactive case the worker sends a proof $\rho$ to the client and this proof is checked locally. So here there are only two communications in contrast to interactive case.

In this work we give a brief review of existing VCS. Mainly we are interested in the approaches based on some cryptographic primitives. The particular attention is given to non-interactive VCS exploiting FHE. We discuss and analyze the existing constructions, their properties, vulnerabilities and practical applicability.

## 2. DEFINITION AND BASIC PROPERTIES OF VCS

Usually verifiable computing protocol has three basic steps:

1) Preprocessing;

2) Input Preparation;

3) Computation and Verification.

At the first step the client computes some auxiliary (public and private) information related to function *F*. This phase can take an essential time comparable to the time required for computing of *F*. But it is performed only once. At the second step the client computes auxiliary (public and private) information about input *x* for function *F*. The public information associated with *F* and *x* is given to the worker. At the last step the worker computes a string $\omega_x$ encoding the value *F(x)* and returns it to the client. In non-interactive case the client

computes the value *F(x)* from $\omega_x$ and verifies its correctness locally. In interactive case the client and the worker are involved into special interactive protocol to verify the result.

The notion of VCS was firstly strictly formalized by Gennaro et al [6]. They put forward the following definition of non-interactive VCS.

***Definition 1 ([6]):*** A *non-interactive verifiable computing scheme* is a tuple of the following probabilistic polynomial-time algorithms:

- $\text{KeyGen}(F,\lambda) \to (SK, VK, PK)$, λ): For security parameter $\lambda$ and function $F$ algorithm KeyGen generates private key $SK$, verification key $VK$ and public key $PK$. $PK$ encodes the function $F$ and is given to the worker.
- $\text{ProbGen}(SK, x) \to (\sigma_x, \tau_x)$: Using $SK$ the input data $x$ is transformed into two values: decoding value $\sigma_x$ and public value $\tau_x$. The public value $\tau_x$ is sent to the executor to evaluate $F(x)$.
- $\text{Compute}(PK, \tau_x) \to \tau_y$: Using $PK$ and public encoding $\tau_x$ of input $x$ the worker calculates $\tau_y$ that encodes value $y = F(x)$.
- $\text{Verify}(VK, \sigma_x, \tau_y) \to y \cup \perp$: The verification algorithm converts the encoded result $\tau_y$ into the real value $F(x)$ using $VK$, $\sigma_x$. If $y = F(x)$ holds it outputs $y$. Otherwise it returns $\perp$ indicating that $\tau_y$ does not represent a valid output of $F$ on input $x$.

***Definition 2 ([6]).*** A verifiable computing scheme is *correct* if for any $F$ and $(SK, VK, PK) \leftarrow \text{KeyGen}(F,\lambda)$ it holds that for $\forall x \in Domain(F)$, $(\sigma_x, \tau_x) \leftarrow \text{ProbGen}(SK, x)$, $\tau_y \leftarrow \text{Compute}(PK, \tau_x)$ there is $y = F(x) \leftarrow \text{Verify}(VK, \sigma_x, \tau_y)$

There are several basic properties of VCS that characterize it and indicate the practical importance. These properties were strictly formalized in [6]. Let's briefly enumerate them.

*Security:* It determines what an adversary is allowed to do. Usually it's required that a malicious worker cannot convince a client of the computation correctness if the result is not correct.

*Input privacy:* It means that the server cannot see the data on which he is computing, i.*e. input* $x$ for function $F$. It's of key importance for applications dealing with sensitive data, like medical records for instance.

*Output privacy:* The client requesting the computation and the verifier of the computation correctness can be not the same entities. Client may not trust the verifier. In this case it may be desirable the verifier checking the correctness of result $y$ is not able to learn anything about the input $x$ for $F$.

*Efficiency:* The work required for verification must not exceed the complexity of local computing of $F$ by the client. Preprocessing phase also may be required to have the complexity comparable to the complexity of $F$ or less (if the client is going to run VCS for given $F$ only once).

*Expressiveness of the computations:* VCS may handle different function classes. For example, some approaches deal with arbitrary C code. Others may handle arbitrary arithmetic circuits or arithmetic circuits of bounded degree.

*Public/private verifiability:* Verification may require some secret information (private verifiability) or can be performed by any party (public verifiability).

Analysis of any modern VCS is reduced to analysis of these properties. And it shows if the scheme could be used in practice and its possible practical applications.

## 3. ANCIENT HISTORY

The basic idea underlying the historically first VCS is to conduct *redundant overlapping computations*. This is an approach relying on replication. The same computation is outsourced to several independent servers and then voting is used to determine the correctness. This general method was implemented for example in *audit-based* VCS [7,8]. In such schemes a client or a randomly selected worker is asked to recalculate the result of some work performed by another participant. All participants of the computation are ranked. The participant rating increases if he returns the correct result, and decreases otherwise. In addition, the participant who finds a mistake of another participant is rewarded. Such approach is not suitable for resource-constrained customers and supposes that some fraction of the workers is honest, or at least non-colluding.

The other old fashioned approach is using *secure co-processors* [9,10]. These devices provide an isolated execution environment and high resistance against unauthorized access. The disadvantage is a high price of such processors.

## 4. INTERACTIVE PROOF BASED VCS

The basic idea is that worker (prover) must convince the client (verifier) of the validity of some NP statement which represents the correctness of the computation. This statement the verifier could not compute by himself. So the prover does it. Such cryptographic primitives as *interactive proof systems (IPS)* [11], *probabilistically checkable proofs (PCP)* [12] and *interactive argument systems (IAS)* [13] are exploited for convincing the client.

Historically, IPS and PCP were the first potential candidate to build VCS. The prover was supposed to be super-polynomial and verifier – polynomial. The prover prepares a proof $\rho$ that the verifier should check it only in few places. But the whole proof may be too long for the verifier to process. So using of the first IPS and PCP [11, 12] was not suitable for verifying the computation of arbitrary function in practice. These tools are rather theoretical than practical. But for some restricted classes of functions researchers proposed quite efficient IPS with polynomial provers. For example, in [4] Goldwasser et al put forward an efficient IPS for any function $F$ that can be represented as a log-space circuit.

Several subsequent works described VCS using IPS with polynomial provers (like in [4]). They are quite efficient, but their expressiveness of computations is limited. In particular in [14] Thaler et al. described VCS exploiting parallel IPS. This approach uses parallel processing based on GPU to speed up the calculation. It permits to compute arithmetic circuits of polylogarithmic depth, does not provide input/output privacy and is publicly verifiable. Its main advantage is that *it is the only existing to date scheme providing both precomputation and verification (see section 2) to be efficient*. It means the total time required for these two steps is less than the time required to evaluate $F$. So it makes sense to use it even if the client is going to run VCS for some function $F$ at once. For other existing VCS in this case it's easier to evaluate $F$ locally.

In [15] Vu et al. extended work [14]. They built a system "Allspice" additionally supporting comparisons and inequality checks. It allows verifying straight-line programs not containing loops and branches. This VCS has better server's overhead for non-regular functions. But its precomputation phase is more expensive. The scheme also does not provide input/output privacy and is publicly verifiable.

Later in order to make interactive approach more efficient IAS were introduced in [13]. The prover is polynomial here. The main idea is to combine PCPs with linear commitments with local openings. In such systems the prover sends the verifier a short commitment to the proof using a Merkle tree. The prover can then interactively open the bits requested by the verifier. But although this approach is interesting and has attractive asymptotic complexity in the case of short PCP, the constants used by it are too large constants for practice.

Another line of research within IAS is usage of linear PCP [16]. Such PCP has exponential size, but the prover needn't to write it down. PCP is represented by some linear function and the verifier may use homomorphic encryption to commit to the function. However, preprocessing stage of VCS using such approach becomes expensive.

A variety of VCS using IAS has been proposed. In [17] the authors presented a system "Pepper". It supports only a limited class of functions represented by arithmetic constraints. For some configurations Pepper can be used in practice. In [18] "Pepper" was extended. The proposed system called "Ginger" supports a larger class of computations such as floating point arithmetics, inequality tests, conditional branching. Setty et al. [19] improved "Pepper" and "Ginger". Their system "Zataar" eliminates the restrictions of the previous schemes. It exploits a new PCP by using the representation of arithmetic circuits as quadratic arithmetic programs (QAP) [20]. It supports a wider class of functions and has the better efficiency than "Ginger". Systems "Pepper", "Ginger" and "Zataar" do not provide input/output privacy and are privately verifiable. Finally in [21] "Zataar" was reworked in order to permits verification of programs with general loops that form the most general class of functions. *It should be stressed out that other existing VCS now cannot handle functions from this class.*

The essential disadvantage of interactive proof based VCS is that their security is not analyzed thoroughly. And also almost all VCS in this class does not provide input/output privacy.

# 5. NON-INTERACTIVE VCS

## 5.1. Non-Interactive Argument based VCS

All interactive approaches require multiple communications between the worker and the client. So the server's overhead is essential. To make argument based VCS more efficient and practical researchers proposed to transform them into non-interactive. For this purpose the concept of *succinct non-interactive argument of knowledge (SNARG)* as introduced in [22]. The main technical tool for constructing SNARG is quadratic arithmetic program (QAP) described by Gennaro et al also in [22]. The basic idea of QAP is to write a circuit as a set of degree-2 constraints over a large finite field. It is known that this primitive is secure against the adaptive adversary. But for non-interactive VCS based on QAP the same level of security must be proven. And for now it has not been done for any VCS.

Now let's review briefly a list of existing VCS based on SNARG. In [23] a system named "Pinocchio" was developed. It supports computation of arithmetic circuits. The complexity of its preprocessing phase is proportional to the complexity of function $F$. This system is nearly practical. In [24] the authors proposed a versatile VCS "Geppetto". It also deals with arithmetic circuits. And it reduces the overhead of the worker by decomposing circuits into a set of subcircuits. In [25] Ben-Sasson et al. presented a system SNARK allowing to verify any program written in programming language C. It is based on non-interactive zero knowledge protocol and QAP. All listed in this paragraph schemes are publicly verifiable and do not provide input/output privacy.

Backes et al. described a non-interactive VCS ADSNARK [26] for straight-line computations on authenticated data. ADSNARK permits public and private verification. And it has input privacy property.

More information about non-interactive argument based VCS can be found in [27].

## 5.2. VCS based on Functional Encryption

*Functional encryption (FE)* is a novel type of encryption formalized by Boneh et al [28]. Secret key $SK_F$ in FE is associated with some function $F$. Decryption of ciphertext $c$ encrypting plaintext $m$ does not reveal $m$. Instead there is $Dec_{SK_F}(c) = F(m)$. And it is required that the owner of $SK_F$ is not able to recover $m$. Such encryption allows organizing a fine-grained access to data and has many practical application.

There are several subtypes of FE having narrower functionality. For example there is a *key-policy attribute-based encryption (KP-ABE)* [29]. Any ciphertext $c$ in KP-ABE scheme is associated with attribute vector $\bar{w} = (w_1,..., w_n) \in \{0,1\}^n$. And secret key $SK$ is associated with Boolean formula $\varphi(\bar{z})$, $\bar{z} = (z_1,.., z_n), z_i \in \{0,1\}$. Decryption algorithm decrypts ciphertext $c$ correctly only if $\varphi(\bar{w}) = 1$. Otherwise it returns $\perp$.

In [30] the authors show how to construct a publicly verifiable VCS based on KP-ABE for a family of functions $\Omega$ being closed under complement. The construction is capable to verify the correctness of computation of Boolean function $F : \{0,1\}^n \to \{0,1\}$ that can be represented by a formula of polynomial size. One can extend this for the case of $F$ having the output of arbitrary bit length by decomposing it into Boolean subfunctions $F_1,..., F_n$. The complexity of client's work does not depend on $F$. This scheme does not hold input-output privacy property. The security has not been analyzed yet.

In [31] the scheme from [30] was extended to embrace multiple clients. The authors employ an ABE to obtain verifiability for several clients simultaneously. The keys of ABE are associated with the function $F$ that is outsourced. The client has not to do an excessive work and a single interaction with the server is enough.

In [32] one can find VCS based on more generic primitive: predicate encryption (PE) for general predicates (PE is also a subtype of FE). This VCS is publicly verifiable and capable to provide input/output privacy. The authors prove it is secure against adaptive adversary. But this scheme unfortunately is not practical for now since suitable efficient instance of PE has not been proposed yet to the base of our knowledge. Of course there is FE scheme supporting any circuit [33] that could be a candidate. But no efficient embodiments are known.

Finally we would like to mention the recent works [34, 35]. The authors put forward a new concept of *verifiable functional encryption (VFE)*. It captures the requirement of output correctness. Even if ciphertext, setup or key generation procedures are corrupted the decryptor still has some guarantee of correctness. To achieve this VFE has public verification procedures for ciphertexts and decryption keys. VFE has the following essential property. For each (possibly maliciously generated) ciphertext $c$ that is passed verification procedure, a unique plaintext $m$ exists such that: for any allowed function description $F$ and decryption key $SK_F$ that is also passed verification procedure, it must be that the decryption algorithm given $c, SK_F$ and $F$ is guaranteed to produce $F(m)$. This implies that if two decryptions

corresponding to functions $F_1$ and $F_2$ of the same ciphertext $c$ give $y_1$ and $y_2$, then a single message $m$ exist such that $y_1 = F_1(m)$ and $y_2 = F_2(m)$.

The authors present a generic compiler from any public-key FE scheme to a public-key VFE scheme, with the only additional assumption being Decision Linear Assumption over Bilinear Groups (DLIN).For the secret-key setting the compiler is also developed.

For now real instances of VFE have not been described. But possibly in future it will be exploited in some specific VCS.

To conclude this section we would like to note that to date this direction of research is poorly studied. A very few VCS based on FE were proposed. The existing schemes are not analyzed thoroughly and the majority of them have no practical instantiations.

## 5.3. VCS based on Homomorphic Encryption

Homomorphic encryption (HE) is a powerful cryptographic primitive giving an ability to conduct some computations over encrypted data without knowledge of decryption key [2]. In this section we discuss an approach based on threshold homomorphic encryption (THE), several VCS exploiting fully homomorphic encryption and also a couple of VCS based on Residue Number System (RNS).

### *5.3.1. Multiparty Verifiable Computation based on THE*

Multiparty computation (MPC) is a cryptographical tool allowing outsourcing a computation to several workers, i.e. so called “computation parties”. They get the private input from multiple “input parties”, perform the computation and finally send the result to a “result party”. Usually if no more than half workers are corrupted they cannot recover the input and produce an incorrect result. But sometimes it is not enough.

In [36] the authors introduced the notion of *universally verifiable MPC*. Such MPC gives an ability to verify the correctness of result even if all “computational” parties have been corrupted. And also the result can be verified even by the parties not participating in the computation (public verifiability). The key ingredient is usage of so-called threshold homomorphic encryption (THE) [37].

THE allows conducting some operations (addition or multiplication) over encrypted data. And also it may permit to multiply a ciphertext by a constant. It gives an encryption of the product of the plaintext with the constant. Anybody can encrypt a plaintext using the public key. The decryption procedure has a specific property. There are $n$ decryption keys corresponding to a given public key. And decryption is possible only if at least $t$ decryption keys are known, where $t \le n$.

Construction proposed in [36] supposes performing computations over encrypted values using homomorphism of underlying cryptosystem. So it holds input/output privacy property. Security against active adversary is achieved by letting parties prove the correctness of their actions by the means of interactive zero-knowledge proofs. As a result the proof’s size can be quite large and depends on the number of computation parties. Also there is the following disadvantage. There is no rigorous security model for universal verifiability and analysis of the proposed construction.

In [38] the construction from [36] was reworked. The authors propose a new security model for universally verifiable MPC and a practical construction achieving it. Their protocols are secure in the random oracle model “without dependent auxiliary input”. They

eliminated interactive proofs of correctness. Instead they proposed their own *non-interactive zero-knowledge proof*, which may be of independent interest. Consequently in [38] the proof size doesn't depend on the number of parties performing the computation and has moderate size.

In [38] the authors used a threshold variant of well-known additively homomorphic Paillier cryptosystem [37] based on hard problem of big numbers factorization. Hence the protocol is able to evaluate a wide set of arithmetic circuits. Also they exploit such cryptographic primitive as $\sum$-protocol and cryptographic library SCAPI [39].

The authors of [38] state that their universally verifiable MPC provides a practical alternative to all recent single-party techniques for verifiable computing (which we review in all other sections). They stress out that some papers devoted to verifiable computation focus on efficiency, but do not cover the privacy. From the other hand there are constructions providing strong privacy and security. But they achieve this by computationally costly primitives like fully homomorphic encryption, functional encryption and garbled circuits. In contrast to this, method from [38] relies on simple basic cryptographic primitives. It is quite efficient and provides security and privacy.

### *5.3.2. VCS based on FHE*

In this subsection we discuss approaches to build VCS using FHE. All these constructions are privately verifiable and hold input-output privacy. Before describing them let's briefly recall the notion of FHE.

Fully homomorphic cryptosystem (FHC) is a cryptosystem allowing computing an arbitrary function $F$ over encrypted data without knowledge of decryption key. This computation must be efficient. It means that its computational complexity should be polynomial in security parameter and the size of representation of $F$. Functions are usually supposed to be represented by Boolean circuits or some arithmetic circuits. So basically FHE must permit to carry out efficiently an unlimited number of addition and multiplication over ciphertexts. The result of homomorphic computation is ciphertexts encrypting the result of computations over underlying plaintexts.

Historically the first FHC was proposed by Gentry [2]. His construction is based on lattices. Gentry provides a strict proof of security of the cryptosystem. But it is not efficient and cannot be used in practice. So cryptographers proposed a variety of modifications of original Gentry's scheme [40]. But even the latest constructions in Gentry-style suffer from inefficiency in the general case. Their usage could be practical only in some use cases.

Also there are alternative FHC not using Gentry's approach [41-44]. The majority of them are more efficient and conceptually simpler than Gentry-style construction. But their security is not analyzed thoroughly. And some of them were attacked in different settings [45-47].

#### *5.3.2.1. Gennaro's Construction based on Gentry-Style FHE and Yao's Garbled Circuits*

The first VCS exploiting FHE was proposed by Gennaro et al [6]. It also uses Yao's garbled circuits [48]. Before describing the Gennaro's construction let's in a nutshell recall what is Yao's garbled circuits.

Let's suppose Alice and Bob, wish to compute a function $F$ over their private inputs $a$ and $b$. Alice transforms $F$ into a Boolean circuit $C$, prepares a garbled circuit $G(C)$ and a garbled input $G(a)$. She sends $G(C)$ and $G(a)$ to Bob. Then Alice and Bob conduct several oblivious transfers. And Bob gets the garbled input $G(b)$. And Alice does not learn anything

about $b$. Finally Bob applies $G(C)$ to $G(a), G(b)$ and get the garbled output $G(F(a,b))$. Alice can transform $G(F(a,b))$ into $F(a,b)$ and share it with Bob.

Alice prepare $G(C)$ as follows. For each wire $w$ of $C$ she generates two random labels ($l$-bit long numbers) $k_w^0, k_w^1$ representing bits 0 and 1 respectively. Then she garbles each gate $g$ in $C$. Let $g$ has input wires $w_a$ and $w_b$ and output wire $w_z$. The garbled gate $G(g)$ is represented by four ciphertexts $\gamma_{i,j} = Enc_{k_a^i}(Enc_{k_b^j}(k_z^{g(i,j)})), i \in \{0,1\}, j \in \{0,1\}$, where $Enc$ is a secure symmetric encryption algorithm, $Enc_k(x)$ is encryption of $x$ on the secret key $k$.

Alice sends all garbled truth tables (i.e. the shuffled list of all $\gamma_{i,j}$ for each gate) to Bob. Also Bob needs input labels to open the garbled tables. So Alice gives him the wire labels corresponding to the bit representation of $a$. Using oblivious transfer Bob obtains the wire labels for $b$. Bob learns exactly one value for each wire. Finally he uses these input wire labels to recursively decrypt the gate ciphertexts. He goes through all gates and decrypts the rows in garbled tables. He can open only one row for each table and get the corresponding output label. He continues the evaluation until he arrives at the output labels. Then he sends them to Alice and she can map them back to 0 or 1 values. The privacy of this protocol assumes an honest-but-curious adversary model.

Now we briefly review VCS from [6]. First of all let's note that the scheme is privately verifiable, so here $SK = EK$ (see definition 1). At the preprocessing step the client garbles a circuit $C$ representing $F$ and gets $G(C)$. The public key $PK$ for this scheme is the set of ciphertexts $\gamma_{i,j}$ representing $G(C)$ and $SK$ consists of all wire labels $k_w^0, k_w^1$.

The input $x$ for $F$ is encoded in two steps:

1) Fresh public/secret key pair for FHE is generated;
2) Wire labels representing bits of $x$ are encrypted by FHE.

These ciphertexts form the public encoding $\tau_x$ for $x$ that is given to worker for processing. Using homomorphism the worker is able to compute all steps of Yao's protocol over ciphertexts. Finally the worker will get the encryptions of the labels of the correct output wires. He returns them to the client. The client decrypts this data and computes the output.

It's easy to see that the described scheme holds input/output privacy. Any Boolean function can be computed using it. Its security and privacy are analyzed thoroughly. It offers security only against a weak adversary. The main problem of this construction is efficiency. It strictly depends on FHE performance. Gennaro supposes usage of provably secure FHE in Gentry-style. But such FHE as we've said already is not practical. So it makes it not practically today.

***5.3.2.2. VCS based on Redundant Computation of $F$ on Random Point***

In [49] Chung et al. described the method to verify the correctness of computation using FHE. The main idea is to compute $F$ on some random point $r$ at the preprocessing step. Value $y_r = F(r)$ is stored by the client. The client asks the worker to evaluate $F$ on desirable input $x$ and $r$ in a random order. Finally he verifies if the worker's output contains $y_r$. If it does, the worker is assumed to be honest. However, in this case the value $y_r$ can be used only once. To overcome it the probabilistic FHE is used to encrypt both $x$ and $r$. And the server is asked

to evaluate $F(r)$ and $F(x)$ homomorphically. The client can decrypt both results and verify the correctness.

The scheme is obviously privately verifiable and provides input/output privacy. It offers only weak security. Any Boolean circuit can be evaluated using it. But the worker's overhead depends on the underlying FHE scheme. In [49] the authors meant using a provably secure FHE following Gentry's approach. In this case it would be impractical today.

Independently of Chung [49] two other research groups [41, 42] proposed the similar approach to construct VCS. In these works they developed their own symmetric FHE schemes not using Gentry's approach. And they described how the client can verify the result of homomorphic computation of function $F$ over encrypted data. Their FHE schemes are conceptually simpler and more efficiently. So at a glance they could be used in practice. Let's briefly describe them and make some short analysis.

In [41] the authors proposed a simple and efficient symmetric FHE scheme based on univariate polynomials over the residue ring $Z_n$, where $n = p \cdot q$ is a number that is hard to factorize, i.e. $p, q$ are big primes. For simplicity let's suppose that the client needs to compute a univariate polynomial $F(z) \in \mathrm{Z}_n[z]$ over his data. At the precomputation step he does the following:

- Choose a random point $u \in \mathrm{Z}_n$ and compute $v = F(u)$.
- Generate random values $t_1, t_2 \in \mathrm{Z}_n$, where $t_1$ – decryption key, $t_2$ – verification key.
- Generate polynomial $f(z) = (z - t_1) \cdot (z - t_2) \cdot (z + t_1 + t_2) \in \mathrm{Z}_n[z]$ and send it to worker.

To encrypt input $x \in \mathrm{Z}_n$ the client generates polynomial $X(z) = z^2 + a \cdot z + b \in \mathrm{Z}_n[z]$ such that $X(t_1) \equiv x (\mathrm{mod} n)$, $X(t_2) \equiv u (\mathrm{mod} n)$. The worker receives $X(z)$ and $F$ from the client and computes $Y(z) = F(X(Z))$ in the ring $\mathrm{Z}_n[z]/(f(z))$, i.e. all operations are performed modulo $f(z)$. For $Y(z)$ there is an equality modulo $n$: $Y(t_2) \equiv v (\mathrm{mod} n)$. So the client verifies this property. If it's correct he computes $Y(t_1)$ that should be equal to $F(x)$.

This VCS is privately verifiable, permits the computation of arbitrary arithmetic circuit over $Z_n$ and at a glance holds input/output privacy. However privacy is doubtful. The security of this FHE is not proved thoroughly. The authors do not provide a strong reduction to the hard problem of big numbers factoring. In [45] an efficient known-plaintext attack on this cryptosystem was described and implemented. It requires an adversary to intercept at least two pairs (plaintext, ciphertext) made on the same key.

Holding of security property is also questionable here even in the settings of cipher-text only attack. Assume the adversary knows two ciphertexts $X_i(z) = z^2 + a_i \cdot z + b_i \in Z_n[z], i = 1,2$ made on the same key ($t_1$, $t_2$). And let's suppose $X_i(t_2) = u, i = 1,2$ (the same checkpoint was used). In this case one may compute $H(z) = X_1(z) - X_2(z) = (a_1 - a_2) \cdot z + b_1 - b_2$. Obviously there is $H(t_2) = 0$. So an adversary may find $t_2$ that equals $(b_2 - b_1) \cdot (a_1 - a_2)^{-1}$ modulo $n$. The last formula requires to compute the inverse modulo $n$. But it is not a hard task that can be solved by Euclidean algorithm without knowledge of factorization of $n$. The complexity is $O((\log_2 n)^2)$.

After recovering $t_2$ the adversary may get a check point $(u,v)$. Hence he is able to falsify the result of computation by generating a random monic quadratic polynomial $R(z) = z^2 + r_1 \cdot z + r_2$ such that $R(t_2) = v$. For this he may generate $r_1$ randomly and then compute $r_2 = v - t_2^{\,2} - r_1 \cdot t_2$.

Thus the practical applicability of VCS [41] is questionable. One must conduct a detailed analysis to understand in what situations it's reasonable to use this VCS.

Finally we would like to note that VCS [41] has the following limitation. It is impossible to recognize if the element of $\mathrm{Z}_n[z]/(f(z))$ is positive, negative or zero. So computation of branching programs such as "If argument is positive, do step *A*, else do step *B*" is impossible. And also there is no division in $\mathrm{Z}_n[z]/(f(z))$. Therefore for example, exponentiation in prime finite field or exponentiation on elliptic curve over prime finite field cannot be realized using this FHE.

In [42] the authors proposed another symmetric FHE based on matrices over $\mathrm{Z}_n$. Here again $n = p \cdot q$ is a number that is hard to factorize. Let's briefly review how it works. Plaintext $m \in \mathrm{Z}_n$ is encrypted in two steps:

- $\mathbf{D} = diag(m, r) \in \mathrm{Z}_{\mathrm{n}}^{2\times 2}$, where $r \xleftarrow{\$} \mathrm{Z}_{\mathrm{n}}^{2\times 2}$ – random element, $diag(m,r)$ – diagonal matrix;
- $\mathbf{C} = \mathbf{K}^{-1} \cdot \mathbf{D} \cdot \mathbf{K}$, where $\mathbf{K} \xleftarrow{\$} GL(\mathrm{Z}_{\mathrm{n}}^{2\times 2})$ – secret key.

Obviously thisscheme is additively and multiplicatively homomorphic and allows to conduct an unlimited number of homomorphic operations in $\mathrm{Z}_n$.

Let's the client wants to compute $F(x_1,...x_t) \in \mathrm{Z}_n[x_1,...,x_t]$. At the preprocessing step he chooses values $r_1,...,r_t$ and computes $y_r = F(r_1,...r_t)$. Then if he want to compute $F$ on plaintexts $m_1,...,m_t$ he prepares ciphertexts $\mathbf{C}_1,...,\mathbf{C}_t$ using $r_1,...,r_t$ at the step 1 of encryption procedure and the same secret key $\mathbf{K}$. The worker computes $\mathbf{C}_* = F(\mathbf{C}_1,...\mathbf{C}_t)$. The following equality holds:

$$\mathbf{C}_* = \mathbf{K}^{-1} \cdot \begin{bmatrix} F(m_1,...m_t) & 0 \\ 0 & F(r_1,...r_t) \end{bmatrix} \cdot \mathbf{K} = \mathbf{K}^{-1} \cdot \begin{bmatrix} F(m_1,...m_t) & 0 \\ 0 & y_r \end{bmatrix} \cdot \mathbf{K}.$$

The client decrypts $\mathbf{C}_*$ and verifies the presence of $y_r$ inside it. If it is the client assumes that server is honest.

The described VCS is privately verifiable and permits the computation of arbitrary arithmetic circuit in $\mathrm{Z}_n$. Input/output privacy holds according to the above description. However in fact it's questionable. The security of this FHE is not investigated thoroughly. In [46,47] the authors described an efficient known-plaintext attack on this cryptosystem. It requires an adversary to know at least one pair (plaintext, ciphertext) to recover the key.

Another problem is that to be able to verify the computation of $F(x_1,...x_t)$ we are forced to fix the randomness $r_1,...,r_t$ at the precomputing step. For each computing of $F$ the same random values will be used. If the client needs to compute $F$ multiple times over different

inputs then encryption procedure loses its probabilistic character and becomes deterministic in some sense. In this case it may be vulnerable even to ciphertexts-only attack. For example frequency analysis may help to break it.

From the other hand if $F$ is supposed to compute only once over client's data then it doesn't make any sense. It's easier for client to make the computation locally.

Security *property* was not analyzed by the authors of [42]. Recall that it indicates if the worker can convince a client of the correctness of a computation although the result is not correct. At a glance it seems that for example in known plaintext attack settings the worker is able recover the random value $r$ of ciphertext $\mathbf{C}$. Indeed let's adversary has a pair $(m, \mathbf{C})$. He can compute a characteristic polynomial $\delta(x) \in \mathrm{Z}_n[x]$ of $\mathbf{C}$. Its roots are $m$ and $r$. So he may divide $\delta(x)$ by $x - m$ and get $r$. It may be dangerous in the case we are forced to fix the randomness in ciphertexts to be able to compute $F$ multiple times and verify the result.

According to all these facts the practical importance of VCS from [42] is questionable now. Its security and privacy properties must be analyzed more thoroughly to understand in what cases it's safe to use this scheme.

### *5.3.3. VCS based on RNS*

In [50] the authors offer a simple HE scheme based on RNS. In a nutshell RNS works as follows. Let there is a set of modulus $P = \{p_1, ..., p_n\}, p_i \in \mathrm{Z}$, such that $GCD(p_i, p_j) = 1$ for $i \neq j$. To encrypt $x \in \mathrm{Z}$ one computes a vector $\bar{x} = ([x]_{p_1}, ..., [x]_{p_n})$, where $[x]_{p_i} = x \bmod p_i$. The decryption may be done by applying Chinese reminder theorem (CRT) to $\bar{x}$. Obviously there are additive and multiplicative homomorphisms.

To date RNS has attracted attention of several research groups. In particular in [51, 52] the authors also proposed constructions based on RNS. This interest can be explained by simplicity and efficiency. Such construction can be easily parallelized according to CRT.

But RNS has limitations. First of all it's not fully homomorphic (over $\mathrm{Z}$). If the result of computation over some integers really exceeds the modulus $M_P = \prod_{i=1}^{n} p_i$ the result produced by RNS will be incorrect. It will be reduced modulo $M_P$. Another important issue is security and privacy. Obviously this system is vulnerable to many types of attack. If untrusted executor knows the set $P$ he can recover all plaintexts using CRT. If $P$ is hidden (like in [51]) and the server performs computation over integers without modulo reduction it still can be attacked. For example a simple known-plaintext attack on RNS using Euclidean algorithm is clearly possible.

In [50] the authors enumerated several possible tricks to improve basic RNS. In particular they discussed how verification of correctness by the client can be done. For RNS there are two possible sources of errors: overflow of computation process that we addressed in the previous paragraph and malicious server corrupting the result.

To struggle with overflow it was proposed to use the redundant RNS. It means that there are two set of modulus $P = \{p_1, ..., p_n\}$ and $Q = \{q_1, ..., q_n\}$ are used. Computation is done twice using $P$ and $Q$. The results obtained $y_P$ and $y_Q$ are compared in the end. They are

valid only if $y_P = y_Q$ or $M_P - y_P = M_Q - y_Q$, where $M_Q = \prod_{i=1}^{n} q_i$. This method allows detecting overflow, but not correcting it.

To prevent the corruption of computation result by malicious server the authors proposed to add a random noise to plaintext. Before reducing $x \in \mathrm{Z}$ by system of modulus $P$ it is transformed by the formula $x' = x + |\Re| \cdot r_x$, where $r_x \in \Re$ – random value taken from the range $\Re$. Next $x'$ is encrypted by RNS and homomorphically processed. The result of any computation must be converted to the original data domain by reducing modulo $|\Re|$.

The system described in [50] does not look like a complete VCS. In fact only a set o tricks and recommendations are presented showing how the original RNS can be changed in order to achieve better confidentiality and integrity. The authors haven't analyzed security and privacy properties thoroughly (see section 2). So the further work is required to understand better in what cases such VCS could be used safely for protecting real data and computations.

In [52] the authors also improved the original RNS. They offer to use modulus of special form, in particular numbers looking like $2^l - c$, $2^l + c$. In this setting more efficient decryption procedure is possible. For moduli set $\Lambda = \{2^l - 1, 2^l + 1, 2^l - 3, 2^l + 3\}$ an algorithm for error detection, localization and correction was developed. This algorithm was inspired by distributed storage systems in the clouds with erasure codes and Byzantine protocol. In fact it is error correction code with a moduli set $\Lambda$. It allows detecting and correcting errors in vector $\bar{x} = ([x]_{2^l-1}, [x]_{2^l+1}, [x]_{2^l-3}, [x]_{2^l+3})$ by computing special syndrome for each coordinate.

The system from [52] looks interesting since it's efficient and allows not only detecting the corruption of result, but possibly correcting it. It's a very useful property that was not met in others VCS. But still it looks unsecure. The detailed analysis of security and privacy properties is needed for safely using this system.

## 6. CONCLUSIONS

We described the main lines of research in the field of verifiable computing and discussed existing instantiations. This branch of cryptography is not old, but there is a variety of different directions. In recent years the invention of such cryptographic primitives as fully homomorphic encryption, functional encryption, attribute-based encryption etc gave birth to a plenty of verifiable computing schemes. These schemes allow to compute functions from different classes, even programs with general loops. But the majority of existing constructions deal with arithmetic circuits.

The practicality of verifiable computing scheme depends on the efficiency of primitives exploited by it. So at the present moment the schemes based on functional encryption or provably secure fully homomorphic encryption in Gentry style are not practical. Since it's well known that these primitives are rather theoretical than practical today. Such VCS have too expensive precomputation phase and the outsourced computation itself may be too complex. And only advances in aforementioned branches will change the state of the art in verifiable computing. But, for example all FHE-based VCS hold such important property as input/output privacy. In some practical applications it may be very important.

At the present moment the most efficient instances of VCS can be found in the class of proof based systems. In particular, this class contains the only to date VCS where both the

time required for setup and verification is less than the time required to compute the function. But security of proof based constructions is poorly investigated. And another disadvantage is that almost all proof based VCS do not provide input/output privacy.

Also there are conceptually simple and efficient VCS from alternative FHE not using Gentry's approach. However their security and privacy are questionable and must be analyzed thoroughly in a future. But VCS exploiting provably secure FHE as their building blocks also have some problems with security. The majority of them are proved to be secure against weak adversary. There were several attempts to make VCS holding privacy property and secure against adaptive adversary. However all obtained instances are not practical. For many applications such VCS would be very valuable. Hence, its developing is an important task for future work.

To conclude we would like to note that each class of VCS has its own unresolved to date problems. But nevertheless a variety of the mentioned in this review VCS has been implemented and used in some applications. Of course it was done with some limitations, for some restricted sets of functions. The further work should be devoted to constructing more universal, efficient and provably secure verifiable computing schemes.

## ACKNOWLEDGMENTS

This research is supported by the Russian Ministry of Education and Science according to the project part of the state funding assignment No. 2.6264.2017/8.9.

## REFERENCES

[1] Gholami, A. and Erwin L. (2016). Security and privacy of sensitive data in cloud computing: a survey of recent developments. arXiv preprint arXiv:1601.01498.

[2] Gentry, C. (2009). A fully homomorphic encryption scheme. Stanford University.

[3] Babai, László, et al. (1991). Checking computations in polylogarithmic time. Proceedings of the twenty-third annual ACM symposium on Theory of computing. ACM.

[4] Shafi, G., Kalai, Y.T. and Rothblum, G.N. (2008). Delegating computation: interactive proofs for muggles." Proceedings of the fortieth annual ACM symposium on Theory of computing. ACM.

[5] Micali, S. (2000). Computationally sound proofs. SIAM Journal on Computing, 30(4), 1253-1298.

[6] Gennaro R., Gentry C. and Parno B. (2010). Non-interactive verifiable computing: Outsourcing computation to untrusted workers. Advances in Cryptology, 465-482.

[7] Monrose, F., Wyckoff, P. and Rubin, A.D. (1999). Distributed Execution with Remote Audit. Ndss, 99.

[8] Belenkiy, M. et al. (2008). Incentivizing outsourced computation. Proceedings of the 3rd international workshop on Economics of networked systems. ACM, 85-90.

[9] Smith, S.W. and Weingart, S. (1999). Building a high-performance, programmable secure coprocessor. Computer Networks, 31(8), 831-860.

[10] Yee, B. (1994). Using secure coprocessors, CMU-CS-94-149. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.

[11] Shafi, G., Micali, S. and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. SIAM Journal on computing, 18(1), 186-208.

[12] Babai, L. et al. (1991). Checking computations in polylogarithmic time. Proceedings of the twenty-third annual ACM symposium on Theory of computing. ACM.

[13] Kilian, Joe. (1992). A note on efficient zero-knowledge proofs and arguments. Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. ACM.

[14] Thaler, J. et al. (2012). Verifiable Computation with Massively Parallel Interactive Proofs. HotCloud.

[15] Vu, V. et al. (2013). A hybrid architecture for interactive verifiable computation. Security and Privacy (SP), 2013 IEEE Symposium on. IEEE.

[16] Yuval I., Kushilevitz, E. and Ostrovsky, R. (2007). Efficient arguments without short PCPs. Computational Complexity, 2007. CCC'07. Twenty-Second Annual IEEE Conference on. IEEE.

[17] Setty, T.V.S. et al. (2012). Making argument systems for outsourced computation practical (sometimes). NDSS, 1(9).

[18] Setty, T.V.S. et al. (2012). Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. USENIX Security Symposium.

[19] Setty, T.V.S. et al. (2013). Resolving the conflict between generality and plausibility in verified computation. Proceedings of the 8th ACM European Conference on Computer Systems. ACM.

[20] Gennaro, Rosario, et al. (2013). Quadratic span programs and succinct NIZKs without PCPs. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg.

[21] Wahby, R.S. et al. (2015). Efficient RAM and control flow in verifiable outsourced computation. NDSS.

[22] Gennaro, R. et al. (2013). Quadratic span programs and succinct NIZKs without PCPs. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg.

[23] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. Proceedings of 2013 IEEE Symposium on Security and Privacy (SP). IEEE, 238-252.

[24] Costello, C. et al. (2015). Geppetto: Versatile verifiable computation. Security and Privacy (SP), 2015 IEEE Symposium on. IEEE.

[25] Ben-Sasson, E. et al. (2013). SNARKs for C: Verifying program executions succinctly and in zero knowledge. Advances in Cryptology-CRYPTO 2013. Springer, Berlin, Heidelberg, 90-108.

[26] Backes, M. et al. (2015). ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. Security and Privacy (SP), 2015 IEEE Symposium on. IEEE.

[27] Buchmann, J. et al. (2016). Overview of Verifiable Computing Techniques Providing Private and Public Verification.

[28] Boneh, D., Sahai, A. and Waters, B. (2011). Functional encryption: Definitions and challenges. Theory of Cryptography Conference. Springer, Berlin, Heidelberg.

[29] Goyal, V., et al. (2006). Attribute-based encryption for fine-grained access control of encrypted data. Proceedings of the 13th ACM conference on Computer and communications security. Acm.

[30] Parno, B., Raykova, M. and Vaikuntanathan, V. (2012). How to delegate and verify in public: Verifiable computation from attribute-based encryption. Theory of Cryptography Conference. Springer, Berlin, Heidelberg.

[31] Alderman, J. (2016). On the use of Attribute-based Encryption in Publicly Verifiable Outsourced Computation.

[32] Barbosa, M. and Pooya, F. (2012). Delegatable homomorphic encryption with applications to secure outsourcing of computation. Cryptographers' Track at the RSA Conference. Springer, Berlin, Heidelberg.

[33] Garg, S. et al. (2016). Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing, 45(3), 882-929.

[34] Badrinarayanan, S. et al. (2016). Verifiable functional encryption. International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg.

[35] Badrinarayanan, S. et al. (2017). A note on VRFs from Verifiable Functional Encryption. IACR Cryptology ePrint Archive 2017, 51.

[36] de Hoogh, S. (2012). Design of large scale applications of secure multiparty computation: secure linear programming. Diss. PhD thesis, Eindhoven University of Technology.

[37] DamgardI, J.M. (2001). A generalisation, asimplificationand someapplicationsofPaillier! sprobabilisticpublic-keysystem. Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptosystems.

[38] Schoenmakers, B. and Veeningen, M. (2015). Universally verifiable multiparty computation from threshold homomorphic cryptosystems. International Conference on Applied Cryptography and Network Security, 3-22.

[39] Ejgenberg, Y. et al. (2012). SCAPI: The Secure Computation Application Programming Interface. IACR Cryptology EPrint Archive 2012, 629.

[40] Rostovtsev, A., Bogdanov, A. and Mikhaylov, M. (2011). Secure evaluation of polynomial using privacy ring homomorphisms. IACR Cryptology ePrint Archive, 24.

[41] Kipnis, A. and Hibshoosh, E. (2012). Efficient Methods for Practical Fully Homomorphic Symmetric-key Encrypton, Randomization and Verification. IACR Cryptology ePrint Archive, 637.

[42] Xiao, L., Bastani, O. and Yen, I.L. (2012). An Efficient Homomorphic Encryption Protocol for Multi-User Systems. IACR Cryptology ePrint Archive, 193.

[43] Zhirov, A., Zhirova, O. and Krendelev, S.F. (2013). Practical fully homomorphic encryption over polynomial quotient rings. Internet Security (WorldCIS), 70-75.

[44] Trepacheva, A. and Babenko, L. (2014). Known plaintexts attack on polynomial based homomorphic encryption. Proceedings of the 7th International Conference on Security of Information and Networks. ACM, 157.

[45] Vizár, D. and Vaudenay, S. (2014). Analysis of Chosen Symmetric Homomorphic Schemes. Central European Crypto Conference, EPFL-CONF-198992.

[46] Tsaban, B. and Lifshitz, N. (2014). Cryptanalysis of the MORE symmetric key fully homomorphic encryption scheme. Journal of Mathematical Cryptology.

[47] Yao, A. C-C. (1986). How to generate and exchange secrets. Foundations of Computer Science, 1986, 27th Annual Symposium on. IEEE.

[48] Chung, K-M., Yael, K. and Salil, V. (2010). Improved delegation of computation using fully homomorphic encryption. Annual Cryptology Conference. Springer, Berlin, Heidelberg.

[49] Gomathisankaran, M., Tyagi, A. and Namuduri, K. (2011). HORNS: A homomorphic encryption scheme for Cloud Computing using Residue Number System. Information Sciences and Systems (CISS), 2011 45th Annual Conference on. IEEE.

[50] Vishnevskij, A.K. and Knjazev, V.V. (2015). Complex application of homomorphic cryptographic transformations for solving systems of linear algebraic equations. High technology, 16(11), 28-35.

[51] Babenko, M. et al. Development of a Control System for Computations in BOINC with Homomorphic Encryption in Residue Number System.