prisma cloud

Overview of Verifiable Computing Techniques Providing Private and Public Verification

D5.8

Document Identification	
Date	February 1, 2016
Status	Final
Version	1.0

Related WP	WP5	**Document Reference**	prismacloud.eu
Related Deliverable(s)		**Dissemination Level**	PU
Lead Participant	TUDA	**Lead Author**	Denise Demirel (TUDA) Lucas Schabhüser (TUDA)
Contributors	Johannes Buchmann (TUDA) Denise Demirel (TUDA) David Derler (TU Graz) Lucas Schabhüser (TUDA) Daniel Slamanig (TU Graz)	**Reviewers**	Daniel Slamanig (TU Graz) David Derler (TU Graz) Thomas Gross (UNEW)

D5.8: Overview of Verifiable Computing Techniques Providing Private and Public Verification

Johannes Buchmann[†], Denise Demirel[†], David Derler[‡], Lucas Schabhüser[†], and Daniel Slamanig[‡]

[†] Technische Universität Darmstadt, Germany
[‡] Graz University of Technology, Austria

Abstract. In this deliverable, we investigate the state-of-the-art in cryptographic approaches to verifiable computing. Verifiable computing encompasses methods that allow to delegate the computation of a function f on outsourced data x to third parties, such that the data owner and/or other third parties can verify that the outcome $y = f(x)$ has been computed correctly by the third party. Thereby, approaches that are of prime interest are those providing an efficient verification process, i.e., it requires significantly lower computational costs to verify the correctness of the result than to perform the computation locally. In addition to presenting on overview of the state-of-the-art, we also highlight some interesting directions for future work.

This document is issued within the frame and for the purpose of the PRISMACLOUD project. This project has received funding from the European Union's Horizon 2020 Programme for research, technological development and demonstration under grant agreement no. 644962.

Executive Summary

Cloud computing is an increasing trend within IT outsourcing that allows vendors to offer traditional IT facilities, such as storage and/or computational systems, via the Internet. Clearly, such a business model brings many benefits and often allows customers to increase efficiency, flexibility and/or cost efficiency. However, cloud computing also raises many (novel) security and privacy related issues. This is mainly due to the fact that cloud providers who store and process data of their tenants cannot be considered fully trustworthy or immune to attacks. Thus, a very important and relevant research question is how one can outsource data and computations to a non-trusted third party such that this party can process the data and at the same time provide guarantees that integrity and confidentiality has been preserved. This question lead to a new and interesting research field called verifiable computing.

Within PRISMACLOUD, we (among others) aim at developing tools that counter problems related to integrity, authenticity, and confidentiality in the context of cloud computing. In the field of verifiable computing, many solutions for different types of computations, e.g., described by a subset of some programming language like C, using various different approaches have been presented in recent years. Nevertheless, all existing solutions come with several trade-offs and so far it is not clear if there is a comprehensive solution that provides both the security and privacy level needed for sensitive data and the flexibility and efficiency to be used in practice. One example for such sensitive data are health and medical data of individuals as encountered within the PRISMACLOUD eHealth use-case.

Thus, the main purpose of this deliverable (D5.8) is to present the state-of-the-art in verifiable computing and to analyze to what extent the constructions provide security, privacy, and efficiency. This allows identifying which approaches are the most promising candidates to be adapted and potentially improved for our use cases and integrated in the toolbox developed within PRISMACLOUD.

Document information

Contributors

Name	**Partner**
Johannes Buchmann	TUDA
Denise Demirel	TUDA
David Derler	TU Graz
Lucas Schabhüser	TUDA
Daniel Slamanig	TU Graz

History

0.01	2015-10-15	Denise Demirel Lucas Schabhüser	structure
0.02	2015-11-26	Denise Demirel Lucas Schabhüser	added schemes based on proofs, FHE, homomorphic authentication, specific applications
0.03	2015-11-27	All	proof reading, bugfixing
0.04	2015-11-30	Denise Demirel Lucas Schabhüser	added verifiable computing from attribute based encryption
0.05	2015-12-01	David Derler Daniel Slamanig	added verifiable computation from functional signatures
0.06	2015-12-02	All	proof reading, bugfixing
0.07	2015-12-03	Denise Demirel Lucas Schabhüser	added signature based verifiable computing using homomorphic encryption
0.08	2015-12-09	All	proof reading, bugfixing
0.09	2015-12-14	Denise Demirel Lucas Schabhüser	abstract, introduction, preliminaries, analysis, conclusion, assumptions
0.10	2015-12-15	All	proof reading, bugfixing

Table of Contents

List of Tables

List of Acronyms

ABE	Attribute Based Encryption
CNF	Conjunctive Normal Form
DNF	Disjunctive Normal Form
EUF-CMA	Existential Unforgeability under Adaptively Chosen Message Attacks
FE	Functional Encryption
FHE	Fully Homomorphic Encryption
FS	Functional Signature
GPU	Graphics Processing Unit
HE	Homomorphic Encryption
HEA	Homomorphic Encrypted Authenticator
MAC	Message Authentication Code
NIZKAoK	Non-interactive Zero Knowledge Argument of Knowledge
OWF	One-Way Function
PCP	Probabilistically Checkable Proof
PE	Predicate Encryption
PPT	Probabilistic Polynomial Time
QAP	Quadratic Arithmetic Program
QPP	Quadratic Polynomial Program
QSP	Quadratic Span Program
SCC	Signature of Correct Computation
SNARG	Succinct Non-Interactive Argument
SNARK	Succinct Non-Interactive Argument of Knowledge
VHE	Verifiable Homomorphic Encryption

1 Introduction

Due to the increasing popularity and prevalence of cloud computing, there is an increasing market for solutions that allow to outsource data and computations to the cloud. However, since the servers performing these computations might be malicious or infected (and can thus not be fully trusted), it is a desirable feature that clients can verify the correctness of any outsourced computation. A naïve solution to this problem is to reobtain the outsourced data from the cloud, check its integrity, and reexecute the computation locally. However, in this case the client needs to have enough space to store the data and enough power to perform the computation. This is not a viable solution especially for weak devices such as smartphones, or huge amounts of data as well as time-consuming computations. Moreover, it negates many of the benefits of using the cloud in the first place. Thus, an interesting research question is how verification of correctness of computation can be performed while *requiring less computational work* than a local computation and optimally without needing access to the data locally.

The field of *verifiable computing* aims to give solutions to this problem. Originally, researchers developed the idea of *super-polynomial servers* (provers) convincing a *computationally bounded client* (verifier) of the validity of some statement in an **NP**-language using by then rather theoretical tools such as *interactive proof systems* [GMR89] and *probabilistically checkable proofs* [BFLS91, AS98]. While the application to verifiable computing scenarios have already been mentioned in very early works, the solutions from these theoretical tools were not suitable for any practical application. Later, work relaxed these potentially super-polynomial provers to polynomially bounded provers to obtain (typically more efficient) *argument systems* and within the last few years, motivated by the potential of cloud computing, research in making such approaches practical yielded many different solutions. In 2009, Genarro et al. [GGP10] provided the first definition of a non-interactive *verifiable computing* scheme and since then many solutions for different types of computations using vastly different approaches have been presented. From these approaches, many have already been implemented and can be termed nearly practical today[1]. As this field has significantly grown in the last few years and is still growing quite fast, this work aims at providing an exhaustive overview of the current state of the art in verifiable computing. The existing constructions are described and their properties, e.g., level of security, efficiency, and privacy, are analysed.

1.1 Roadmap

In this work we will be concerned with a setting where a verifier specifies a function f and some input x and requests a prover to compute $y = f(x)$ and return y to the verifier. If the prover indeed computed $y = f(x)$ correctly, then the prover should accept y, but otherwise the verifier should reject y with high probability. In order establish the correctness guarantee, either the verifier interactively asks the prover questions about the computation performed by the prover or the prover returns a certificate of correct computation (a proof) to the verifier which can be locally checked. The former approach is an interactive approach to verifiable computation, whereas the latter approach is non-interactive (only requires two moves). We work in the

[1]For very restricted classes of computations there are entirely practical solutions available.

predominant model which considers a single client and a single server. Works dealing with a more complicated setting of multiple clients or servers are beyond the scope of this article.

Now, let us briefly informally discuss what we mean when we are talking about security, privacy, and efficiency of verifiable computing solutions.

Security: Security basically means what an adversary is allowed to do in such a scheme and will be formally defined in Section 2.1. Basically, we require that a malicious server (that may have different capabilities) will not be able to convince a verifier of the correctness of a computation although the result is not correct.

Privacy: Privacy comes in different flavours. Firstly, privacy can be related to the input of the computation from a servers point of view (so called input privacy). It requires that the server performing the computation does not learn the data on which the server is computing. Such a feature is often desirable when dealing with sensitive data the server should not see, e.g., medical data of a patient that should be analysed in the cloud, but not disclosed to the cloud. Secondly, privacy can be related to the verifier (so called output privacy). It requires that verifier checking the correctness of the computation does not learn anything about the input to the computation. For an example think of a query to a database that reveals the average income of all employees which should not leak the individual incomes to the verifier.

Efficiency: Efficiency considers the work required by the verifier in comparison to locally performing the computation. The efficiency is often considered in an amortized sense, i.e., the verifier has some setup costs which are performed once and then by verifying computations on different inputs the costs of this setup are amortized over time.

Another distinction is whether verification requires some secret information (private verifiability) or can be performed by any party (public verifiability).

Besides the above measures, another important measure is the expressiveness of the computations that can be handled by some approach. While some approaches deal with function classes, such as arithmetic circuits of fixed degree others can handle arbitrary arithmetic circuits or even arbitrary C code.

In this work we do not consider approaches that require quite strong assumption (not in the cryptographic sense). In particular, we do not consider approaches that rely on replication, i.e., to outsource the same computation to n independent servers and use majority voting on the results to determine the correctness. Since this assumes uncorrelated failures, this assumptions seems to strong. Also, we do not consider the use of trusted hardware at the server, remote attestation or auditing. The former two approaches requires trusted hardware assumptions and additionally trusted hardware is usually strongly limited in scalability. The latter uses a spot-checking approach, but needs to assume that failures, if they occur, are very frequent, which also seems to be a quite unreasonable assumption.

1.2 Organisation

We first give some general definitions in Section 2. Afterwards, in Section 3 we present the *proof* and *argument* based systems that have been developed in the recent years and for which a variety of tools are already available. Section 4 covers schemes based on *fully homomorphic encryption* and in Section 5 we give an overview of schemes based on *homomorphic authenticators* (message authentication codes and signatures). Section 6 covers solutions from functional cryptography, while in Section 7 individual schemes that allow for the verification of specific computations are presented. Finally, Section 8 provides a summary and comprehensive analysis of the different types of verifiable computing schemes followed by a conclusion and planned future work in Section 9.

2 Preliminaries

In this section we provide a formal definition of verifiable computing schemes and their relevant properties. For the hardness assumptions underlying the individual constructions in the subsequent sections we refer the reader to the original papers.

2.1 Verifiable Computation

In this work we will always consider the following scenario. A client $\mathcal{C}$ wants a server $\mathcal{S}$ to evaluate a function f on some input x. Therefore $\mathcal{C}$ gives (encodings of) f and x to $\mathcal{S}$. $\mathcal{S}$ will do the computation and then return a result y to $\mathcal{C}$. To prove the correctness of the result to the client, i.e., to prove that y is indeed equal to $f(x)$, a verifiable computing scheme can be used. In the following we recall the definition of a non-interactive verifiable computing scheme [GGP10].

Definition 2.1 (Verifiable Computing Scheme)**.** *A Verifiable Computing Scheme $\mathcal{VC}$ is a tuple of the following probabilistic polynomial-time (PPT) algorithms:*

$\mathsf{KeyGen}(1^\lambda, f)$: *The probabilistic key generation algorithm takes a security parameter λ and the description of a function f. It generates a secret key* sk*, a corresponding verification key* vk *and a public evaluation key* ek *(that encodes the target function f) and returns all these keys.*

$\mathsf{ProbGen}(\mathsf{sk}, x)$: *The problem generation algorithm takes a secret key* sk *and data x. It outputs a decoding value ρ_x and a public value σ_x which encodes the data x.*

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: *The computation algorithm takes the evaluation key* ek *and the encoded input σ_x. It outputs an encoded version σ_y of the function's output $y = f(x)$.*

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$ *The verification algorithm obtains a verification key* vk *and the decoding value ρ_x. It converts the encoded output σ_y into the output of the function y. If $y = f(x)$ holds, it returns y or outputs $\perp$ indicating that σ_y does not represent a valid output of f on x.*

Definition 2.2 (Correctness)**.** *A verifiable computing scheme $\mathcal{VC}$ is correct if for any choice of f and output* $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$ *of the key generation algorithm it holds that* $\forall\ x \in \mathsf{Domain}(f)$*, if* $(\sigma_x, \rho_x) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x)$ *and* $y \leftarrow \mathsf{Compute}(\mathsf{ek}, \sigma_x)$*, then* $y = f(x) \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_\mathsf{x}, \sigma_\mathsf{y})$.

In the original work on non-interactive verifiable computing Gennaro et al. [GGP10] only considered privately verifiable computing schemes as defined below.

Definition 2.3 (Privately Verifiable Computing Scheme)**.** *If* $\mathsf{sk} = \mathsf{vk}$ *and $\mathcal{C}$ needs to keep ρ_x private, $\mathcal{VC}$ is called a* privately verifiable computing scheme*.*

Clearly, such a scheme only allows $\mathcal{C}$ to run the verification algorithm. Later in [PRV12], Parno et al. introduced the notion of publicly verifiable computing schemes.

Definition 2.4 (Publicly Verifiable Computing Scheme)**.** *If* $\mathsf{sk} \neq \mathsf{vk}$*,* $\mathcal{VC}$ *is called a* publicly verifiable computing scheme.

It allows to hand out vk to third parties without revealing sk, therefore everyone with knowledge of vk and ρ_x can verify the correctness of the server's computation.

Intuitively the difference between the two notions is that in privately verifiable computing the client keeps its verification key secret. It follows that only the client can verify the correctness of a computation. Note that revealing the verification key to the server would allow it to break the security, i.e. computing a wrong result leading to a correct verification proof.

In publicly verifiable computing, on the other hand, the verification key can be published since knowledge of it does not help a malicious server to forge an incorrect result. This allows not only the client but anyone to check the correctness of a performed computation.

2.2 Properties of Verifiable Computing Schemes

In this section a definition for security, privacy, and efficiency is given. We will mainly follow the approach of Gennaro et al. [GGP10], who were the first to define verifiable computing schemes, but also integrate some later proposals [BGV11a] to obtain stronger security definitions (i.e., adaptive security).

2.2.1 Security

Intuitively a verifiable computing scheme $\mathcal{VC}$ is secure, if a malicious server cannot persuade the verification algorithm to output $y^* \neq f(x)$ except with negligible probability. Formally, we define the following experiments. We distinguish between two types of adversaries, a weak adversary and an adaptive adversary. The weak adversary [GGP10] can try only once to have an incorrect result verified as correct (and is not allowed to call Verify in the privately verifiable computing setting). An adaptive adversary [BGV11a] can run $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}$ multiple times, by calling $\mathbf{EXPadapt}_{\mathcal{A}}^{\mathsf{Verify}}$, and learn about the client's acceptance bit.

```
Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}[\mathcal{VC}, f, \lambda]$ :
  $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$
  for $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$ do
    $x_i \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots x_{i-1}, \sigma_1, \ldots, \sigma_{i-1})$
    $(\sigma_i, \rho_i) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$
  end for
  $(i, \sigma_y^*) \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots, x_\ell, \sigma_1, \ldots, \sigma_\ell)$
  $y^* \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_i, \sigma_y^*)$
  if $y^* \neq \perp \ \wedge \ y^* \neq f(x)$ then
    return 1
  else
    return 0
  end if
```

```
Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathcal{VC}, f, \lambda]$:
   $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$
   for $j = 1, \ldots, m = \mathsf{poly}(\lambda)$ do
      for $i = 1, \ldots, \ell = \mathsf{poly}(\lambda)$ do
            $x_i \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots x_{i-1}, \sigma_1, \ldots, \sigma_{i-1}, \delta_1, \ldots, \delta_{j-1})$
            $(\sigma_i, \rho_i) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_i)$
      end for
      $(i, \sigma_y^*) \leftarrow \mathcal{A}(\mathsf{ek}, x_1, \ldots, x_\ell, \sigma_1, \ldots, \sigma_\ell, \delta_1, \ldots, \delta_{j-1})$
      $y^* \leftarrow \mathsf{Verify}(\mathsf{vk}, \rho_i, \sigma_y^*)$
      if $y^* \neq \bot \ \wedge \ y^* \neq f(x)$ then
            $\delta_j := 1$
      else
            $\delta_j := 1$
      end if
   end for
   if $\exists\ j$ such that $\delta_j = 1$ then
      return 1
   else
      return 0
   end if
```

In the non-adaptive case the adversaries $\mathcal{A}$'s advantage is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Verify}}(\mathcal{VC}, f, \lambda) = \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Verify}}[\mathcal{VC}, f, \lambda] = 1\right].$$

So in practice this type of adversary is acceptable if a client aborts the protocol once it detects an incorrect result.

An adaptive adversaries $\mathcal{A}$'s advantage is defined as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AdaptVerify}}(\mathcal{VC}, f, \lambda) = \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{AdaptVerify}}[\mathcal{VC}, f, \lambda] = 1\right].$$

From this the security definition for verifiable computing schemes follows.

Definition 2.5 (Security)**.** *A verifiable computing scheme $\mathcal{VC}$ is (weakly) secure if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Verify}}(\mathcal{VC}, f, \lambda) \leq \mathsf{negl}(\lambda)$$

and adaptively secure if

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{AdaptVerify}}(\mathcal{VC}, f, \lambda) \leq \mathsf{negl}(\lambda).$$

2.2.2 Privacy

Verifiable computing can guarantee the integrity of a computation. Another desirable property is to protect the secrecy of the client's inputs. To formally define *input privacy* we define the following experiment. We use the oracle $\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk},x)}$ which calls $\mathsf{ProbGen}(\mathsf{sk}, x)$ to obtain (σ_x, ρ_x) and only returns the public part σ_x.

```
Experiment $\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Privacy}}[\mathcal{VC}, f, \lambda]$
    $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$
    $(x_0, x_1) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk}, \cdot)}}(\mathsf{ek})$
    $(\sigma_0, \rho_0) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_0)$
    $(\sigma_1, \rho_1) \leftarrow \mathsf{ProbGen}(\mathsf{sk}, x_1)$
    $b \overset{\$}{\leftarrow} \{0, 1\}$
    $b^* \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{ProbGen}(\mathsf{sk}, \cdot)}}(\mathsf{ek}, x_0, x_1, \sigma_b)$
    if $b^* = b$ then
        return 1
    else
        return 0
    end if
```

In this experiment, the adversary first receives the public evaluation key for the scheme. Then, it selects two inputs x_0, x_1 and is given the encoding one of the two inputs chosen at random. The adversary then must determine which input has been encoded. Note that during this process the adversary is allowed to request the encoding of any input of its choice. We define an adversaries $\mathcal{A}$'s advantage as

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Privacy}}(\mathcal{VC}, f, \lambda) = \left| \Pr\left[\mathbf{EXP}_{\mathcal{A}}^{\mathsf{Privacy}}[\mathcal{VC}, f, \lambda] = 1 \right] - 1/2 \right| .$$

Definition 2.6 (Input Privacy). *A verifiable computing scheme $\mathcal{VC}$ provides input privacy if*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Privacy}}(\mathcal{VC}, f, \lambda) \leq \mathsf{negl}(\lambda).$$

Besides *input privacy* a verifiable computing scheme can also provide privacy with respect to the data output. This so called *output privacy* can be defined by an analogous experiment and is omitted here.

2.2.3 Efficiency

Finally we are interested in using verifiable computing schemes by means of delegating computations. For this we want the client's work to be less than computing the function on its own.

Definition 2.7 (Efficiency). *A verifiable computing scheme provides* efficiency *if for any x and any σ_y, the time required for $\mathsf{KeyGen}(1^\lambda, f)$ plus the time required for $\mathsf{ProbGen}(\mathsf{sk}, x)$ plus the time required for $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.*

A slightly relaxed definition is the following.

Definition 2.8 (Amortized Efficiency). *A verifiable computing scheme provides* amortized efficiency *if it permits efficient verification. This implies that for any x and any σ_y, the time required for $\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$ is $o(T)$, where T is the time required to compute $f(x)$.*

Note that in literature amortized efficiency has been defined ambiguously. We use here a broad version that ensures that the minimal requirements for outsourceability are met.

Intuitively the difference between efficiency and amortized efficiency is the cost of the preprocessing phase. Efficient verifiable computing schemes allow a client to verify the correctness of a computation more efficiently than performing the computation by itself, including the preprocessing phase. Some verifiable computing schemes have an expensive preprocessing phase, but still provide an efficient verification phase. Since the preprocessing phase only has to be performed once and might not be time critical in some applications, we classify them as verifiable computing schemes providing amortized efficiency.

One aspect that also impacts the practicality of all verifiable computing schemes is the server's overhead to evaluate a computation using $\mathsf{Compute}$ versus natively executing it. Note that this does not affect the computation complexity for the client and is therefore not considered in our efficiency analysis.

3 Proof Based Verifiable Computing

In this section we discuss approaches to verifiable computation that rely on the concept of (non-)interactive proof (IP) or argument systems and probabilistically checkable proofs (PCPs). In this setting a (super-)polynomial-time *prover* wants to convince a polynomial-time *verifier* of the truth of some NP statement, which in context of verifiable computing represents the correctness of a given computation. While the use of the theoretical tools of IPs [GMR89] and PCPs [BFLS91, AS98] as is, is highly unsuited for practical applications, in recent years this line of work has been significantly improved by either theoretical improvements or the introduction of clever tricks such as using suitable encodings or preprocessing. To verify the correctness of an outsourced program or function by using proof (or argument) based systems, the program or function has to be encoded. A suitable encoding is to write it as a circuit or to express it as a set of arithmetic constraints, i.e., polynomials who simultaneously evaluate to 0 iff the circuit is evaluated correctly. For the latter case, Genarro et al. [GGPR13] introduced two new notions, one called quadratic span program (QSP) for boolean circuits and another one for arithmetic circuits called quadratic arithmetic program (QAP). These constructions have been developed specifically for the verifiable computing use case. The basic idea is to write a circuit as a set of degree-2 constraints over some large finite field. As later shown in [BCI$^+$13] the QAP approach in [GGPR13] implicitly uses a PCP structure.

We refer the interested reader also to a recent article by Walfish and Blumberg [WB15] that provides a good overview of proof based approaches to verifiable computations as well as the available tools basing on different approaches and using different sophisticated tricks.

3.1 Interactive Proof Based Approaches

Subsequently, we consider interactive proof systems. Let us therefore define all required concepts and let $L \subseteq \{0,1\}^*$ be an **NP**-language.

Definition 3.1 (Interactive Proof System (IPS))**.** *An interactive proof system for a language L is an interactive protocol between an unrestricted prover P and a PPT verifier V such that the following conditions hold:*

Completeness. $\forall\, x \in L : \Pr[\,(\mathsf{P},\mathsf{V})(x) = 1] = 1,$

Soundness. $\forall\, x \notin L\ \forall\, \mathsf{P}^* : \Pr[\,(\mathsf{P}^*,\mathsf{V})(x) = 1] \leq \frac{1}{2}.$

where we use $(\mathsf{P},\mathsf{V})(x) = 1$ to denote that V accepts the interaction with P on common input x.

Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time (witness) relation, i.e., a relation such that membership of (x, w) in $\mathcal{R}$ can be decided in polynomial time in $|x|$. Subsequently, for an **NP**-language L we may explicitly index it with its witness relation and write $L_{\mathcal{R}}$ where $L_{\mathcal{R}} = \{x \mid \exists\, w : (x, w) \in \mathcal{R}\}$. Now, we can define the concept of proofs of knowledge which define IPs with a stronger notion of soundness.

While the traditional definition of IPS does not put any restrictions on the prover P and in particular allows P to run in super-polynomial time, this is clearly not meaningful for the application to verifiable computation. In [GKR08] Goldwasser et al. present efficient interactive proofs with polynomial provers for any function representable as a log-space uniform circuit that has communication complexity being the depth of the circuit.

Several subsequent work build verifiable computing frameworks based on such IPS with polynomial provers, which are more efficient than the ones discussed later in this section, but whose expressibility in terms of functions is rather limited.

Verifiable Computation with Massively Parallel Interactive Proofs. In [TRMP12] Thaler et al. proposed the first verifiable computing protocol with a reasonable server's overhead. The approach is to use parallel processing, i.e., running parts of the protocol in parallel using a GPU, to speed up the evaluation process. Their protocol supports arithmetic circuits of polylogarithmic depth. In [Tha13] the authors define the notion of *regularity* of a function. This contains for instance to what extend output bits depend on input bits and therefore to what extend the computation of the function can be parallelised. For circuits that are *regular* in this sense the server's overhead is just a factor of approximately 10. This construction does not depend on any cryptographic assumptions.

Allspice: A Hybrid Architecture for Interactive Verifiable Computation. In [VSBW13] Vu et al. generalized [TRMP12] to functions beyond arithmetic circuits. They build a system called *Allspice* that also supports comparisons and inequality checks. This allows to verify computations expressed as straight-line programs (i.e., programs that do neither branch nor loop). Furthermore, their schemes improves with respect to the server's overhead for *non-regular* functions. On the other hand, they require a computationally more expensive setup phase. Therefore, this scheme only achieves amortized efficiency.

Now, we define probabilistically checkable proofs (PCPs), which are proofs that can be verified by a randomized algorithm using a bounded number of random coins and inspecting a bounded number of bits in the proof.

Definition 3.2 (Probabilistically Checkable Proof (PCP))**.** *A probabilistically checkable proof (PCP) for a language* $L \in \mathbf{PCP}(r(n), q(n))$ *is a string* π *such that there exists a PPT alorithm* V *(the verifier) that, on input* $a \in \{0,1\}^n$ *uses* $O(r(n))$ *random coins and inspects* $O(q(n))$ *locations in* π*, after which it outputs* 1 *(accept) or* 0 *(reject) such that:*

Completeness. *If* $a \in L$*, then there exists a* π *such that* $\Pr[\mathsf{V}^{\pi}(a) = 1] = 1$.

Soundness. *If* $a \notin L$*, then for all* π^* *it holds that* $\Pr\left[\mathsf{V}^{\pi^*}(a) = 1\right] < 1/2$.

Here V^{π} *denotes that* V *has oracle access to the string* π*.*

While asymptotically short PCPs [BGH+05, BGH+06] are interesting in theory, for their application to verifiable computing the length of the PCP (needed to be retrieved by the verifier) is still longer than the execution trace of any function. Thus, this does not yield solutions that provide verification that is more efficient as the local evaluation of the function.

3.2 Interactive Argument Based Approaches

While in the IPS setting, the soundness guarantees are unconditional, i.e., hold with respect to an all-powerful prover, one may reduce the soundness guarantees to computational soundness, i.e., computationally bounded provers. The resulting systems are no longer denoted as proofs but as arguments.

The idea of computationally sound proofs dates back to Kilian [Kil92], who proposed to combine PCPs with linear commitments with local openings (such as those obtained from collision resistant hash functions and generally known as Merkle Trees). The idea is to commit to a PCP string π and the prover needs to send the commitment to π to the verifier. Then, the verifier can ask the prover to open the commitment on various positions (determined by the random coins of the PCP verifier). So, one obtains four-move argument systems. Later Micali in [Mic00] has shown how this approach can be turned into a one-move scheme secure in the random oracle model by applying the Fiat-Shamir heuristic [FS86]. The basic idea is simply to let the prover compute the random coins (of the PCP verifier) by computing them from the output of a random oracle on input the commitment to π. Although this is an interesting approach and yields good asymptotic complexity when used with short PCPs, the constants within this approach are intricate and seem to yield too large constants for any practical applications (although there are no experimental results available).

Another direction within interactive arguments is the use of what is called linear PCPs [IKO07]. These PCPs are exponnentially long, but the prover does not need to write them down but the PCP string is implicity represented as a linear function and the verifier uses additively homomorphic encryption to commit to a function of this form (cf. [IKO07, SMBW12]). However, this comes at the cost of an expensive preprocessing stage which can be amortized over a batch of verifications of the same function over different inputs.

Pepper: Making Argument Systems for Outsourced Computation Practical (Sometimes). In 2012 Setty et al. [SMBW12] presented an interactive argument system named *Pepper*. Here functions are not represented as circuits but as arithmetic constraints. These are algebraic equations that hold simultaneously iff the function f is evaluated correctly. Pepper only supports a very limited class of functions and only achieves amortized efficiency, while having a setup phase, whose computational cost is proportional to $\mathcal{O}(|f|)$.

Ginger: Taking Proof-Based Verified Computation a Few Steps Closer to Practicality. Setty et. al. further improved on [SMBW12] in a system called Ginger [SVP+12] that supports a larger class of computations such as inequality tests, floating point arithmetics, and conditional branching.

Zataar: Resolving the Conflict Between Generality and Plausibility in Verified Computation. Setty et al. [SBV+13] developed an improvement over Pepper and Ginger called Zaatar. It uses a new PCP by using the algebraic representation of computations as QAPs from [GGPR13] and removes the restrictions of the previous schemes yielding a richer class of supported functions and it is also shown that Zaatar improves on the efficiency of Ginger.

Pantry: Verifying Computations with State. Braun et al. introduced within their construction Pantry [BFR+13b] an expansion of Zataar which allows verification of stateful computations.

River: Verifiable Computation with Reduced Informational Costs and Computational Costs. In [XAG14] Xu et al. presented a QAP based verifiable computing system named River. Compared to Zataar, River reduces the client's computational costs while only marginally increasing the server's overhead. This scheme supports arithmetic circuits and achieves amortized efficiency.

Buffet: Efficient RAM and control flow in verifiable outsourced computation. In [WSR+15] Wahby et al. improve upon the functionality supported by Zataar and Pantry by supporting programs with general loops.

3.3 Non-Interactive Argument Based Approaches

All proof based schemes presented so far are interactive protocols. In order to provide a non-interactive solution, Gennaro et al. show in [GGPR13] how to construct succinct non-interactive arguments of knowledge (SNARKs) using QSPs and QAPs. Like for all QAP based schemes even though this primitive is secure against the adaptive adversary it has to be proven that also the verifiable computing techniques using this primitive provide the same level of security. Furthermore, all these schemes are only secure under an assumption that is *non-falsifiable.*

Definition 3.3 (Succinct Non-Interactive Argument (SNARG) [BCCT12])**.** *A SNARG for the relation $\mathcal{R} \subset \mathcal{R}_\mathcal{U}$ is a triple of the following probabilistic, polynomial-time algorithms:*

- $\mathsf{Gen}_\mathcal{V}(1^\lambda) \to (\mathsf{vgrs}, \mathsf{priv})$. *Takes the security parameter λ as input and outputs a verifier-generated reference string* vgrs *and corresponding private verification coins* priv.
- $\mathsf{P}(y, w, \mathsf{vgrs}) \to \pi$. *Takes a statement $y = (M, x, t)$, a witness w, and the reference string* vgrs *and outputs a proof π.*
- $\mathsf{V}\ (\mathsf{priv}, \mathsf{y}, \pi) \to \{0, 1\}$ *verifies the validity of π for y using the private verification coins* priv *and returns '1' if the input is correct and '0' otherwise.*

These algorithms have to satisfy the following conditions.

Completeness. *For any $(y, w) \in \mathcal{R}$*

$$\Pr\left[\mathsf{V}(\mathsf{priv}, y, \pi) = 1 \;\middle|\; (\mathsf{vgrs}, \mathsf{priv}) \leftarrow \mathsf{Gen}_\mathcal{V}(1^\lambda), \pi \leftarrow \mathsf{P}(y, w, \mathsf{vgrs})\right] = 1$$

Succinctness. *The length of* π *that* $\mathsf{P}(y, w, \mathsf{vgrs})$ *outputs as well as the running time of* $\mathsf{V}(\mathsf{priv}, y, \pi)$ *is bounded by*

$$p(\lambda + |y|) = p(\lambda + |M| + |x| + log(t))$$

where p *is a universal polynomial that does not depend on* $\mathcal{R}$

Adaptive Soundness. *For all poly-size prover* P^* *and large enough* $\lambda \in \mathbb{N}$

$$\Pr\left[\mathsf{V}(\mathsf{priv}, y, \pi) = 1 \;\middle|\; (\mathsf{vgrs}, \mathsf{priv}) \leftarrow \mathsf{Gen}_{\mathcal{V}}(1^\lambda), (\mathsf{y}, \pi) \leftarrow \mathsf{P}^*(\mathsf{vgrs}), \mathsf{y} \notin \mathcal{L}_{\mathcal{R}}\right] \leq \mathsf{negl}(\lambda).$$

For our purpose we need an even stronger definition.

Definition 3.4 (SNARG of Knowledge (SNARK) [BCCT12])**.** *A SNARK is a SNARG* $(\mathsf{Gen}_{\mathcal{V}}, \mathsf{P}, \mathsf{V})$ *where soundness is replaced by the following stronger condition.*

Adaptive Proof of Knowledge. *For any poly-size prover* P^* *there exists a poly-size extractor* E_{P^*} *such that for all large enough* $\lambda \in \mathbb{N}$ *and all auxiliary inputs* $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$

$$\Pr\left[\begin{array}{c}(\mathsf{vgrs}, \mathsf{priv}) \leftarrow \mathsf{Gen}_{\mathcal{V}}(1^\lambda)\\ (y, \pi) \leftarrow \mathsf{P}^*(z, \mathsf{vgrs})\\ \mathsf{V}(\mathsf{priv}, \mathsf{y}, \pi) = 1\end{array} \;\wedge\; \begin{array}{c}(y, w) \leftarrow \mathsf{E}_{\mathsf{P}^*}(z, \mathsf{vgrs})\\ w \notin \mathcal{R}(y)\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Pinocchio: Nearly Practical Verifiable Computation. In [PHGR13] Parno et al. developed a system named Pinocchio that supports arithmetic circuits (that are turned into QAPs). Pinocchio offers public verifiability, but no input-output privacy. Due to its preprocessing phase that runs in time proportional to a one time execution of function f it only achieves amortized efficiency.

Geppetto: Versatile Verifiable Computation. Costello et al. generalized the QAPs to MultiQAPs and use them to build a verifiable computing system called Geppetto [CFH+15]. They show how to reduce the server's overhead by decomposing circuits into a collection of subcircuits. Geppetto offers public verifiability, but no input-output privacy. Due to its preprocessing phase that runs in time proportional to a one time execution of function f it only achieves amortized efficiency.

SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In [BCG+13] Ben-Sasson et al. presented a system that can verify all operations in programming language C albeit at an increased server's overhead compared to [PHGR13]. It is also based on QAPs. This system also offers public verifiability without input-output privacy and achieves amortized efficiency.

Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In [BCTV14] Ben-Sasson et al. presented a new QAP based SNARK for arithmetic circuits that allows for more efficient verification and proof generation compared to [PHGR13] and [BCG+13]. They also presented a universal circuit generator further broadening the class of admitted programs. Furthermore, they show that their approach provides zero-knowledge, i.e.

if the statement is true, no cheating verifier learns anything other than this. Their construction is publicly verifiable. However, they do not provide input-output privacy.

ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data. Backes et al. presented ADSNARK [BBFR15] a non-interactive proof system for straight-line computations on authenticated data. Following the generic construction presented in [CRR12] one obtains a verifiable computing system. ADSNARK includes both a publicly verifiable and a more efficient privately verifiable proof. It achieves amortized efficiency and input privacy.

Block Programs: Improving Efficiency of Verifiable Computation for Circuits with Repeated Substructures. Xu et al. showed in [XAG15] a new and more efficient way to handle loops in a program. This improvement can be used together with all other approaches listed here that support loops.

4 Verifiable Computing from Fully Homomorphic Encryption

In this section we briefly discuss approaches to verifiable computing that use fully homomorphic encryption (FHE) as a building block. All these schemes are privately verifiable and provide input-output privacy, but due to the practical inefficiency of current FHE schemes, they do not yield practical solutions. First, we define fully homomorphic encryption schemes and afterwards describe the verifiable computing schemes using this primitive.

Definition 4.1 (Homomorphic Encryption (HE) Scheme [GGP10])**.** *A* homomorphic encryption scheme *is a tuple of the following probabilistic, polynomial-time algorithms:*

$\mathsf{KeyGen}(1^\lambda)$: *This algorithm takes a security parameter λ as input and outputs a public key pk and a secret key sk. The public key pk implicitly defines a message space $\mathcal{M}$, and a ciphertext space $\mathcal{C}$.*

$\mathsf{Encrypt}(\mathsf{pk}, m)$: *The encryption algorithm takes a public key pk and message $m \in \mathcal{M}$ as input and outputs a ciphertext c.*

$\mathsf{Decrypt}(\mathsf{sk}, c)$: *The decryption takes a secret key sk and a ciphertext c as input and outputs a message $m \in \mathcal{M} \cup \perp$.*

$\mathsf{Eval}(\mathsf{pk}, f, \vec{c_i})$: *The evaluation algorithm takes a public key pk, a description of a function f, and a vector of ciphertexts $\vec{c_i}$ as input and outputs a new ciphertext c.*

A homomorphic encryption scheme *is homomorphic for a class $\mathcal{F}$ of functions, if*

$$\forall f \in \mathcal{F}, \{m_i\} \subset \mathcal{M}$$
$$\Pr[\mathsf{Decrypt}(\mathsf{Eval}(\mathsf{pk}, f, \{\mathsf{Encrypt}(m_i, \mathsf{pk})\}), \mathsf{sk}) = f(m_1, \ldots, m_n)] = 1.$$

Besides the above property of evaluating correctness (which may also allow a negligible evaluation error), one requires the usual correctness property of an encryption scheme as well as at least $\mathsf{IND\text{-}CPA}$ security.

Now, informally, a HE scheme is called fully homomorphic (is an FHE scheme) if the class $\mathcal{F}$ of functions represents the class of all circuits. We stress that one requires an additional compactness property, which basically means that the ciphertext output by the Eval algorithm does only depend on the security parameter (and not on the function). This rules out trivial constructions of FHE, e.g., ones where the Eval algorithm simply applies the identity function (or does nothing) and the Decrypt evaluates the function on the decrypted ciphertext(s) and then returns the result. We do not require a formal treatment of properties of FHE here and refer the reader to [ABC+15] for an overview.

Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. The verifiable computing scheme presented in [GGP10] by Gennaro et al. achieves verifiability by combining Yao's garbled circuits ([Yao82], [Yao86]) with FHE. This combination allows to reuse a garbled circuit multiple times while still preserving security. The idea is that during the setup, the client once generates a garbled version of a circuit C representing a function f and sets the public key to the garbled circuit and the secret key to the secret random wire labels. If the clients wants to outsource a computation of f on some input x, its generates a fresh key pair $(\mathsf{sk}, \mathsf{pk})$ of an FHE scheme, sets the public value σ_x to pk and ciphertexts to all wire values of the binary expression of the input x and the decoding value ρ_x as sk. Then, the server can use the homomorphic property of the FHE scheme to evaluate a garbled circuit and sends the encrypted output wires back to the client. The client can then decrypt and map the wires to the output $y = f(x)$.

Besides providing privacy, for this construction the authors were the first to formally introduce the notion of verifiable computation (see Section 2.1). This scheme is in the amortized model and the server's overhead depends on the efficiency of the underlying FHE scheme (making it not practically efficient today). It only offers security against a weak adversary as no verification queries are allowed.

Improved Delegation of Computation Using Fully Homomorphic Encryption. In [CKV10] Chung et al. presented another way to verify the correctness of a result by using FHE. The underlying idea is to evaluate f on some random point r in the preprocessing, store $y_r = f(r)$ and then in the online phase ask Compute to return f evaluated on x and r in a random order and check if y_r equals what is returned for the computation corresponding to r. If this result is correct, the server is assumed to behave honestly. However, with this naive approach firstly the soundness error is too large, i.e., $1/2$, and secondly the precomputed value y_r can only be used once. In order to overcome these issues, the client precomputes $f(r_1, \ldots, r_n)$ for some random r_i for large enough n (to make the soundness error small enough) and target function f. In addition, it uses an FHE scheme to compute encryptions $\hat{x_i}$ of the inputs x_i and the encryptions $\hat{r_i}$ of the random values r_i. The server will then be asked to homomorphically evaluate both $f(\hat{x_1}, \ldots, \hat{x_n})$ and $f(\hat{r_1}, \ldots, \hat{r_n})$. The client can decrypt both results and accepts the computation as correct if one of them matches his precomputed result.

This scheme achieves amortized efficiency while the server's overhead depends on the underlying FHE scheme. This scheme offers only weak security.

In [TC14] a similar scheme is presented, that reduces the preprocessing stage. It offers weak security. However it should be noted that its security against an adaptive adversary has not been analysed yet.

5 Homomorphic Authenticators

Homomorphic authenticators are cryptographic primitives that allow to evaluate a certain class of functions on *authenticated* data, preserving the *authenticity* of any function of the class applied to authenticated data. There exist constructions both in the secret key setting in the form of *homomorphic message authentication codes (MACs)* and in the public key setting in the form of *homomorphic signatures*. These solutions can be used to respectively construct *privately verifiable computing schemes* and *publicly verifiable computing schemes*. We emphasize that there are homomorphic MAC and signature schemes that are not known to allow verification faster than computing the function, like for example [GW13] or [Fre12]. Such schemes are not considered in this survey. For an overview of homomorphic signatures in general we refer to Deliverable 4.4.

5.1 Message Authentication Codes

5.1.1 Definitions for Message Authentication Codes

First, we provide the definitions for homomorphic message authentication codes (MACs), their correctness, and their security. To do so we will use multi-labels and multilabeled programs, which we will briefly explain here.

A multi-label $L = (\Delta, \tau)$ consists of a data set identifier Δ and an input identifier τ. Given some function $f : \mathcal{M}^n \to \mathcal{M}$ that takes n inputs $\tau_1, \dots \tau_n$ label the different input columns while Δ labels the set from which we take our data. This allows to both identify the dataset a server is supposed to work on and restrict the server to this data. A labeled program $\mathcal{P} = (f, \tau_1, \dots, \tau_n)$ consists of a function $f : \mathcal{M}^n \to \mathcal{M}$ on n variables and each $\tau_i \in \{0,1\}^*$ is the label of the i-th input to f. A multi-labeled program $\mathcal{P}_\Delta$ is a pair $(\mathcal{P}, \Delta)$ where $\mathcal{P} = (f, \tau_1, \dots, \tau_n)$ is a labeled program and $\Delta \in \{0,1\}^*$ is the data set identifier.

Definition 5.1 (Homomorphic Message Authentication Code)**.** *A homomorphic MAC scheme is a tuple of the following probabilistic, polynomial-time algorithms:*

$\mathsf{KeyGen}(1^\lambda, \mathcal{L})$: *The key generation algorithm takes as input a security parameter λ and the description of the label space $\mathcal{L}$, and outputs a secret key* sk *and a public evaluation key* ek *(we omit to make the message space $\mathcal{M}$ explicit).*

$\mathsf{Auth}(\mathsf{sk}, L, m)$: *The tag computation algorithm takes a secret key* sk*, a multi-label $L = (\Delta, \tau)$, and a message m as input, and outputs a tag σ.*

$\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$: *The verification algorithm takes as input a secret key* sk*, a message m, a multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \dots, \tau_n), \Delta)$ with $f \in \mathcal{F}$, and a tag σ. It outputs '1' if σ is a valid tag for m under $\mathcal{P}_\Delta$ and '0' otherwise.*

$\mathsf{Eval}(\mathsf{ek}, \mathcal{P}_\Delta, \vec{\sigma})$: *The evaluation algorithm takes as input an evaluation key* ek*, a multi-labeled program $\mathcal{P}_\Delta$, and a vector of tags $\vec{\sigma}$ of length n (assuming f takes n inputs) and outputs a new tag σ.*

Definition 5.2 (Authentication Correctness [BFR13a]). *A homomorphic MAC satisfies authentication correctness if for any message $m \in \mathcal{M}$, all keys* $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$*, any multi-label* $(\Delta, \tau) \in (\{0,1\}^*)^2$*, and any tag* $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathsf{L}, \mathsf{m})$*, we have that*

$$\Pr[\mathsf{Ver}(\mathsf{sk}, Id_L, m, \sigma) = 1] = 1,$$

where Id_L is the identity program with respect to L.

Definition 5.3 (Evaluation Correctness [BFR13a]). *We fix a pair of keys* $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$*, a function $f : \mathcal{M}^n \rightarrow \mathcal{M}$, and any set of (message, program, tag) triples* $\{(m_i, \mathcal{P}_{\Delta,i}, \sigma_i)\}_{i=1}^n$ *such that all multi-labeled programs $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ share the same data set identifier Δ and* $\mathsf{Ver}(\mathsf{sk}, P_{\Delta,i}, m_i, \sigma_i) = 1$*. If $m = f(m_1, \ldots, m_n), \mathcal{P} = f(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and* $\sigma = \mathsf{Eval}(\mathsf{ek}, f, (\sigma_1, \ldots, \sigma_n))$*, then*

$$\Pr[\mathsf{Ver}(\mathsf{sk}, P_\Delta, m, \sigma) = 1] = 1.$$

To formally define security we look at the following security experiment (due to [BFR13a]).

Setup. The challenger generates $(\mathsf{sk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and gives ek to the adversary $\mathcal{A}$.

Authentication Queries. The adversary can adaptively ask for tags on multi-labels and messages of its choice. Given a query (L, m) where $L = (\Delta, \tau)$, the challenger proceeds as follows: If (L, m) is the first query with data set identifier Δ, then the challenger initializes an empty list $T_\Delta = \emptyset$ for the data set identifier Δ. If T_Δ does not contain a tuple $(\tau, \cdot)$ (i.e. the multi-label (Δ, τ) was never queried), the challenger computes $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, L, m)$, returns σ to $\mathcal{A}$ and updates the list $T_\Delta \leftarrow T_\Delta \cup (\tau, m)$. If $(\tau, m) \in T_\Delta$ (i.e. the query was previously made), then the challenger replies with the same tag generated before. If T_Δ already contains a tuple for label τ, i.e. (τ, m'), for some $m \neq m'$ then the challenger ignores the query.

Verification Queries. The adversary has access to a verification oracle as follows: Given a query $(\mathcal{P}_\Delta, m, \sigma)$ from $\mathcal{A}$, the challenger replies with the output of $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}_\Delta, m, \sigma)$.

Forgery. The adversary terminates the experiment by sending $(\mathcal{P}^*_{\Delta^*}, m^*, \sigma^*)$ for some $\mathcal{P}^*_{\Delta^*} = (\mathcal{P}^*, \Delta^*)$ and $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ to the challenger. Notice that also during the verification query $\mathcal{A}$ sends such tuples to the challenger and asks for verification. Thus, if this query is accepted this allows it to terminates the experiment successfully.

We say a labeled program $\mathcal{P}^* = (f^*, \tau_1^*, \ldots, \tau_n^*)$ is well-defined with regards to T_Δ if one of the following conditions hold.

- There exist messages $m_1, \ldots, m_n$ such that T_{Δ^*} contains all tuples $(\tau_1^*, m_1), \ldots, (\tau_n^*, m_n)$. Intuitively, this means that the entire input space of f for the data set Δ^* has been authenticated.
- There exist indices $i \in \{1, \ldots, n\}$ such that $(\tau_i^*, \cdot) \notin T_{\Delta^*}$. This happens when $\mathcal{A}$ never asks authentication queries with multi-label (Δ^*, τ_i^*) and the function $f(\cdot)$ outputs the same value for all possible unauthenticated inputs.

The experiment outputs '1' if and only if $\mathsf{Ver}(\mathsf{sk}, \mathcal{P}^*_{\Delta^*}, m^*, \sigma^*) = 1$ and one of the following conditions holds:

- *Type 1 Forgery:* no list T_{Δ^*} was created during the game, i.e., no message m has been authenticated with respect to data set identifier Δ^* during the experiment.
- *Type 2 Forgery:* $\mathcal{P}^*$ is well-defined with regards to T_{Δ^*} and $m^* \neq f^*(\{m_j\}_{(\tau_j, m_j) \in T_{\Delta^*}})$, i.e. m^* is not the correct output of the labeled program $\mathcal{P}^*$ when executed on previously authenticated messages $(m_1, \ldots, m_n)$.
- *Type 3 Forgery:* $\mathcal{P}^*$ is not well-defined with regards to T_{Δ^*}.

Definition 5.4 (Security). *A homomorphic MAC scheme is adaptively secure if $\mathcal{A}$ wins the experiment above with probability* $\mathsf{negl}(\lambda)$. *A homomorphic MAC scheme is (weakly) secure if $\mathcal{A}$ wins with probability* $\mathsf{negl}(\lambda)$ *the above experiment without asking verification queries.*

5.1.2 Verifiable Computing Schemes Based on MACs

Based on these homomorphic MACs several verifiable computing schemes have been proposed. All of them are based on bilinear or multilinear maps and for a definition of the respective assumptions, we refer to the original papers.

The schemes described below are all privately verifiable and only the work in [FGP14] addresses privacy.

Verifiable Delegation of Computation on Outsourced Data. In [BFR13a] Backes et al. construct a homomorphic MAC for arithmetic circuits f of degree 2. It is based on bilinear maps (pairings) and pseudo-random functions with so called *closed-form efficiency*. After a preprocessing stage of complexity $\mathcal{O}(|f|)$ the client can verify the correctness in constant time. This paper presents a generic way to turn homomorphic MACS with efficient verification into verifiable computing schemes and thus achieves amortized efficiency. Furthermore, it is secure against adaptive adversaries.

Generalized Homomorphic MACs with Efficient Verification. In [ZS14] Zhang and Safavi-Naini generalized the verifiable computing scheme presented in [BFR13a]. Using ℓ-linear maps, their homomorphic MAC supports arithmetic circuits of depth ℓ. Using the generic transformation of [BFR13a], one can thus obtain a verifiable computing scheme for depth ℓ circuits. It also achieves amortized efficiency, while offering security against adaptive adversaries.

Efficiently Verifiable Computation on Encrypted Data. In [FGP14] Fiore et al. show how to combine the homomorphic MACs of [BFR13a] with a FHE scheme to construct a verifiable computing scheme for multivariate polynomials of degree 2 that offers input privacy. They furthermore improve on the efficiency by using a homomorphic hash function. Likewise this scheme achieves amortized efficiency and remains secure against adaptive adversaries.

5.2 Homomorphic Signatures

5.2.1 Definitions for Homomorphic Signatures

In this section we provide the definitions for homomorphic signatures, their correctness, and their security.

Definition 5.5. *A* homomorphic signature scheme *is a tuple of the following probabilistic, polynomial-time algorithms:*

$\mathsf{KeyGen}(1^\lambda, \mathcal{L})$: *The algorithm takes a security parameter λ and the description of the label space $\mathcal{L}$ as input and outputs a public key* vk *and a secret key* sk. *The public key* vk *implicitly defines a message space $\mathcal{M}$ and a set $\mathcal{F}$ of admissible functions.*

$\mathsf{Sign}(\mathsf{sk}, L, m)$: *The signing algorithm takes a secret key* sk*, a multi-label $L = (\Delta, \tau)$, and a message $m \in \mathcal{M}$ as input and outputs a signature σ.*

$\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma)$: *The verification takes a public key* vk*, a message $m \in \mathcal{M}$, a multi-labeled program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$ with $f \in \mathcal{F}$, and a signature σ as input. It outputs '1' if σ is a valid signature for m under $\mathcal{P}_\Delta$ and outputs '0' otherwise.*

$\mathsf{Eval}(\mathsf{vk}, \mathcal{P}_\Delta, \overrightarrow{\sigma})$: *The evaluation algorithm takes a public key* vk*, a program $\mathcal{P}_\Delta = ((f, \tau_1, \ldots, \tau_n), \Delta)$, and a vector of signatures $\vec{\sigma}$ of length n (assuming f takes n inputs). It outputs a new signature σ.*

As in case of homomorphic MACs, the labels are used to tag the respective dataset.

Definition 5.6 (Authentication Correctness)**.** *A homomorphic signature scheme satisfies authentication correctness if for any message $m \in \mathcal{M}$, all keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, any multi-label $L = (\Delta, \tau) \in (\{0,1\}^*)^2$, and any signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, L, m)$, it holds that*

$$\Pr[\mathsf{Ver}(\mathsf{vk}, Id_L, m, \sigma) = 1] = 1,$$

where Id_L is the identity program with respect to L.

Definition 5.7 (Evaluation Correctness)**.** *We fix a pair of keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$, a function $f : \mathcal{M}^n \to \mathcal{M}$, and any set of message/program/signature triples $\{m_i, \mathcal{P}_{\Delta,i}, \sigma_i\}_{i=1}^n$ such that all programs $\mathcal{P}_{\Delta,i} = (\mathcal{P}_i, \Delta)$ share the same data set identifier Δ and $\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_{\Delta,i}, m_i, \sigma_i) = 1$. If $m = f(m_1, \ldots, m_n), \mathcal{P} = f(\mathcal{P}_1, \ldots, \mathcal{P}_n)$, and $\sigma = \mathsf{Eval}(\mathsf{vk}, f, (\sigma_1, \ldots, \sigma_n))$, then*

$$\Pr[\mathsf{Ver}(\mathsf{vk}, \mathcal{P}_\Delta, m, \sigma) = 1] = 1.$$

The formal definition of unforgeability can be given analogously to Definition 5.4. The only difference is that in the experiment the verification algorithm Ver takes public key vk instead of private key sk as input.

5.2.2 Signature Based Verifiable Computing on Linear Functions

For certain classes of functions, it is possible to straightforwardly build a verifiable computing scheme from homomorphic signatures. For linear functions

$$\begin{aligned} f:\ \mathbb{F}^{k^N} &\to \mathbb{F}^n \\ v_1, \dots v_N &\mapsto \sum_{i=1}^{N} c_i v_i \end{aligned}$$

linearly homomorphic signatures have been proposed, originally in the context of *network coding* (see [BFKW09]). Below we sketch a verifiable computing protocol between a client $\mathcal{C}$ and a server $\mathcal{S}$. Note that without combining this with a suitable homomorphic encryption scheme to encrypt the input data, this construction does not provide input-output privacy.

Setup: $\mathcal{C}$ generates the keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ for a linearly homomorphic signature scheme.

Data Outsourcing: To outsource vectors $v_1, \dots, v_N$, $\mathcal{C}$ first signs $w_i = (e_i, v_i)^T$, where e_i is the i-th canonical basis vector of $\mathbb{F}^k$ with regards to some label τ, i.e. $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, w_i, \tau)$ for $i = 1, \dots, N$ and sends all (w_i, σ_i) to $\mathcal{S}$.

Delegation: $\mathcal{C}$ sends $f = (c_1, \dots, c_N)$ to $\mathcal{S}$.

Computation: $\mathcal{S}$ computes $y = \sum_{i=1}^{N} c_i w_i$ and $\sigma \leftarrow \mathsf{Eval}(f, \tau, v_1, \dots, v_N, \sigma_1, \dots, \sigma_N)$ and sends (y, σ) to $\mathcal{C}$.

Verification: $\mathcal{C}$ checks, whether σ is a valid signature for y and whether the result is of the form $y = (c_1, \dots c_N, \tilde{y})^T$. If both is true it accepts $\tilde{y}$ as the result.

In 2015 Catalano et al. [CFN15] presented the first construction that is signature based and allows to verify faster than computing target function f. This solution is based on bilinear maps and uses asymmetric programmable hash functions (see the paper for a formal definition). These signatures are also secure against adaptive adversaries. It should be noted that the preprocessing stage in this paper is divided into the two algorithms KeyGen and EffVerPrep. This however is still only dependent on f and therefore fits our criteria of amortized efficiency.

5.2.3 Signature Based Verifiable Computing for Polynomial Functions

A broader class of admissible functions are multivariate polynomials of fixed degree. The following works provide signature schemes for polynomial functions. Furthermore, they sketch how these schemes can be used to support verifiable computing. Note that these schemes do not address input-output privacy.

Homomorphic Signatures with Efficient Verification for Polynomial Functions. Catalano et al. constructed homomorphic signatures in [CFW14] based on multilinear maps. Their signatures support arithmetic circuits of fixed depth. By using the techniques of [BFR13a], one is able to realize a verifiable computing scheme secure against adaptive adversaries that is efficient in an amortized sense.

Algebraic (Trapdoor) One-Way Functions and their Applications. Catalano et al. showed in [CFGV13] how to use OWFs to outsource (multivariate) polynomial evaluations of fixed degree. In particular they construct a OWF based on the RSA assumption and use this to build a verifiable computing scheme. It is the first one that achieves public verifiability without bilinear maps. The scheme is efficient in an amortized sense and offers adaptive security.

5.3 Signature Based Verifiable Computing Using Homomorphic Encryption

Verifiable computing schemes based on homomorphic signatures or homomorphic MACs do not provide data confidentiality. Therefore, Lai et al. showed in [LDPW14] how to generically construct a verifiable homomorphic encryption (VHE) scheme which allows for verifiable computation on outsourced encrypted data.

The constructed VHE combines a homomorphic encryption (HE) scheme, as defined in Section 4, and a so called homomorphic encrypted authenticator (HEA). The latter is basically a homomorphic signature scheme, as defined in Section 5.2.1, providing *semantic security*.

To formally define semantic security we look at the following security experiment between a challenger and an adversary $\mathcal{A}$ (due to [LDPW14]).

Setup. The challenger runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{L})$ and gives vk to $\mathcal{A}$ and initializes a list $T_\Delta = \emptyset$.

Authentication Queries. $\mathcal{A}$ can adaptively ask the challenger for authenticators of its choice. Given a query (L, m) by $\mathcal{A}$, where $L = (\Delta, \tau)$, the challenger proceeds as follows: If $(L, m) \in T_\Delta$, the challenger computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, L, m)$. If T_Δ does not contain a tuple (L, m) (i.e., the multi-label (Δ, τ) was never queried), the challenger chooses a fresh multi-label $L = (\Delta, \tau) \in (\{0,1\}^*)^2$, computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, L, m)$, returns σ to $\mathcal{A}$ and updates the list $T_\Delta \leftarrow T_\Delta \cup (L, m)$.

Challenge. The adversary submits a multi-label $L = (\Delta, \tau) \in (\{0,1\}^*)^2$ and two messages $m_0, m_1 \in \mathcal{M}$. The challenger selects a random bit $\beta \in \{0,1\}$, computes $\sigma^* \leftarrow \mathsf{Sign}(\mathsf{sk}, L, m_\beta)$, and sends σ^* to the adversary.

Guess. The adversary $\mathcal{A}$ outputs its guess $\beta^* \in \{0,1\}$ for β and wins the game if $\beta = \beta^*$.

The advantage of the adversary in this game is defined as $|\Pr[\beta = \beta^*] - \frac{1}{2}|$ where the probability is taken over the random bits used by the challenger and the adversary.

Definition 5.8 (Homomorphic Encrypted Authenticator (HEA)). *A HEA is a signature scheme as defined in Section 5.2.1 that is* semantically secure, *i.e. where all probabilistic polynomial* *time adversaries have at most a negligible advantage in the security game described above.*

Definition 5.9 (Verifiable Homomorphic Encryption (VHE))**.** *Let* $\mathsf{HE} = (\mathsf{HE.KeyGen}, \mathsf{HE.Encrypt}, \mathsf{HE.Decrypt}, \mathsf{HE.Eval})$ *be a homomorphic encryption scheme and let* $\mathsf{HEA} = (\mathsf{HEA.KeyGen}, \mathsf{HEA.Sign}, \mathsf{HEA.Ver}, \mathsf{HEA.Eval})$ *be a HEA. Then, a* VHE *scheme is a tuple of the following PPT algorithms:*

$\mathsf{KeyGen}(1^\lambda, \mathcal{L})$: *The algorithm takes a security parameter* λ *and the description of the label space* $\mathcal{L}$ *as input and outputs a public key* $\mathsf{pk} = (\mathsf{pk}_{\mathsf{HE}}, \mathsf{pk}_{\mathsf{HEA}})$ *and a secret key* $\mathsf{sk} = (\mathsf{sk}_{\mathsf{HE}}, \mathsf{sk}_{\mathsf{HEA}})$, *where* $(\mathsf{pk}_{\mathsf{HE}}, \mathsf{sk}_{\mathsf{HE}}) \leftarrow \mathsf{HE.KeyGen}(1^\lambda)$ *and* $(\mathsf{pk}_{\mathsf{HEA}}, \mathsf{sk}_{\mathsf{HEA}}) \leftarrow \mathsf{HEA.KeyGen}(1^\lambda, \mathcal{L})$. *The public key* pk *implicitly defines a message space* $\mathcal{M}$ *and a set* $\mathcal{F}$ *of admissible functions.*

$\mathsf{EncSign}(\mathsf{sk}_{\mathsf{HEA}}, \mathsf{pk}_{\mathsf{HE}}, L, m)$: *This algorithm takes a secret key* $\mathsf{sk}_{\mathsf{HEA}}$, *a public key* $\mathsf{pk}_{\mathsf{HE}}$, *a multi-label* $L = (\Delta, \tau)$, *and a message* $m \in \mathcal{M}$ *as input. It runs* $c_{\mathsf{HE}} \leftarrow \mathsf{HE.Encrypt}(\mathsf{pk}_{\mathsf{HE}}, m)$ *and* $c_{\mathsf{HEA}} \leftarrow \mathsf{HEA.Sign}(\mathsf{sk}_{\mathsf{HEA}}, L, m)$ *and returns* $c = (c_{\mathsf{HE}}, c_{\mathsf{HEA}})$.

$\mathsf{VerDec}(\mathsf{sk}_{\mathsf{HE}}, \mathsf{pk}_{\mathsf{HEA}}, \mathcal{P}_\Delta, m, c)$: *This algorithm takes a secret key* $\mathsf{sk}_{\mathsf{HE}}$, *a public key* $\mathsf{pk}_{\mathsf{HEA}}$, *a message* $m \in \mathcal{M}$, *a multi-labeled program* $\mathcal{P}_\Delta = ((f, \tau_1, \dots, \tau_n), \Delta)$ *with* $f \in \mathcal{F}$, *and a ciphertext* c *as input. If* $\mathsf{HEA.Ver}(\mathsf{pk}_{\mathsf{HEA}}, \mathcal{P}_\Delta, c_{\mathsf{HEA}}) = 1$ *it runs* $m \leftarrow \mathsf{HE.Decrypt}(\mathsf{sk}_{\mathsf{HE}}, c_{\mathsf{HE}})$ *and outputs* m. *It outputs '0' otherwise.*

$\mathsf{Eval}(pk, \mathcal{P}_\Delta, \overrightarrow{c})$: *This algorithm takes a public key* pk, *a multi-labeled program* $\mathcal{P}_\Delta = ((f, \tau_1, \dots, \tau_n), \Delta)$ *with* $f \in \mathcal{F}$, *and a vector of ciphertexts vecc of length* n *(assuming* f *takes* n *inputs). It runs* $c_{\mathsf{HE}} \leftarrow \mathsf{HE.Eval}(\mathsf{pk}_{\mathsf{HE}}, f, \overrightarrow{c})$ *and* $c_{\mathsf{HEA}} \leftarrow \mathsf{HEA.Eval}(pk_{\mathsf{HEA}}, \mathcal{P}_\Delta, \vec{c})$ *and outputs the new ciphertext* $c = (c_{\mathsf{HE}}, c_{\mathsf{HEA}})$.

The authors used in their work standard homomorphic signature schemes to build the homomorphic encrypted authenticator. This instantiation has the shortcoming that it does not provide an efficient verification process. However, the construction indicates that a verifiable computing scheme that provides not only privacy, but also amortized efficiency can be built, e.g., using the signature scheme proposed by Catalano et al. [CMP14]. Another important requirement for a successful instantiation, which has not been explicitly mentioned by the authors, is that the homomorphic encryption scheme and the homomorphic encrypted authenticator must be homomorphic over the same message space $\mathcal{M}$. Thus, it should be analysed for which pairs of encryption and signature schemes this is provided.

6 Verifiable Computing Frameworks From Functional Encryption and Functional Signatures

In addition to proof based verifiable computing schemes and constructions that rely on homomorphic encryption or homomorphic authenticators, verifiable computing schemes can also be constructed using functional encryption or functional signatures.

6.1 Verifiable Computation from Functional Encryption

There are basically two approaches that use functional encryption (FE) to build a verifiable computing scheme. One uses (key-policy) attribute-based encryption (ABE), a specific instantiation of functional encryption, and the other one is constructed from FE schemes. Key-policy ABE (KP-ABE) [SW05, GPSW06] is a rather recent public key encryption paradigm, where a public key is associated to a universe of attributes A and secret keys are associated to Boolean functions f. A holder of a secret key corresponding to f can only decrypt a message encrypted with respect to a subset A' of the attributes iff $f(A') = 1$. FE [BSW11] is a very generic definition of various types of public key encryption concepts, such as IBE, ABE and many other classes. Basically, in such schemes secret keys are associated to a function f and given a ciphertext of a message m under the corresponding public key, the holder of a secret key corresponding to f will only learn $f(m)$ during decryption, instead of learning the full plaintext m. Assuming that the plaintext space has an additional structure and in particular plaintexts are pairs of some (public) index and message space, then one can define FE on predicates over the index space and the key space. In doing so, one obtains KP-ABE as a so called predicate encryption (PE) scheme with a public index.

Verifiable Computation from Attribute Based Encryption. In [PRV12] Parno et al. showed how to build a publicly verifiable computation scheme from any key-policy ABE scheme for function family $\mathcal{F}$ (that is closed under complement). Their construction verifies the correct output of a function $f : \{0,1\}^n \to \{0,1\}$ that can be computed by a polynomial sized boolean formula. They use the fact that a message encrypted under an attribute x can only be decrypted if $f(x) = 1$ holds. One can extend this to functions f with outputs of arbitrary bitlength by decomposing f into boolean subfunctions $f_1, \ldots, f_n$. The client's computation is independent of f. This approach does not provide input-output privacy and the security has not been analysed yet.

Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation. Barbosa and Farshim showed in [BF12] how to create a verifiable computing scheme from a FE scheme, a FHE scheme, and a special type of MACs denoted as *MACs with chameleon keys*. For the relevant definitions and properties of their construction we refer to the original paper. By combining these primitives, this scheme achieves amortized efficiency, while offering public verifiability, and security against adaptive adversaries. It should however be noted that one of the necessary building blocks for this construction, a so called *predicate encryption (PE) scheme for general predicates*, does not exist to the authors knowledge. Today we have functional encryption for any circuit (see, e.g., [GGH$^+$13]). We, however, note that no efficient instantiations are known. Furthermore, they assume that the auxiliary information is transferred not only authentically but also confidentially from the client to the verifier. Their proposed scheme can however handle functions of arity one that can be expressed as k CNF/DNF (conjunctive/disjunctive normal forms) formulas for fixed k. Note that for $k \geq 3$ constructing such formulas is NP hard (see [Coo71]).

6.2 Verifiable Computation from Functional Signatures

In [BGI14] Boyle et al. introduced the concept of functional signature (FS) schemes. In such a scheme, in addition to a master signing key msk, which allows to compute signatures on arbitrary messages, there are secondary signing keys sk_f, which are parametrized by a particular function f. Such a key sk_f restricts the signing capabilities to messages in the range of f, i.e., given any m the key sk_f only allows to produce signatures for $f(m)$. Before discussing the application of FS to verifiable computing, we briefly introduce the concept of FS.

Definition 6.1 (Functional Signature (FS) Scheme [BGI14])**.** *A functional signature (FS) scheme for a message space $\mathcal{M}$ and function family $\mathcal{F} = \{f : \mathcal{D}_f \to \mathcal{M}\}$ consists of the following polynomial time algorithms:*

$\mathsf{Setup}(1^\lambda)$ *The setup algorithm takes as input the security parameter λ and outputs the master signing key msk and master verification key mvk.*

$\mathsf{KeyGen}(\mathsf{msk}, f)$: *The key generation algorithm takes as input the master signing key msk and a function $f \in \mathcal{F}$ (represented as a circuit) and outputs a signing key sk_f for f.*

$\mathsf{Sign}(\mathsf{sk}_f, f, m)$: *The signing algorithm takes as input the signing key sk_f, a function $f \in \mathcal{F}$ and a message $m \in \mathcal{D}_f$ and outputs $f(m)$ and a signature σ for $f(m)$.*

$\mathsf{Verify}(\mathsf{mvk}, m, \sigma)$: *The verification algorithm takes as input a master verification key mvk, a message m and a signature σ and outputs 1 if the signature is valid or 0 otherwise.*

An FS scheme needs to provide the usual correctness property as well as unforgeability. Unforgeability is defined with respect to adaptively chosen signing keys for functions and adaptive signature queries and requires that under such queries it is infeasible to produce a valid signature for a message that is outside the range of the queried functions and is not the image of any function and message queried to the signing oracle (cf. [BGI14] for formal definitions). Additionally FS schemes may provide the properties of function privacy and succinctness. Informally, the former means that the distributions of signatures on a message generated via different signing keys are computationally indistinguishable and the latter means that the signature size is independent of the size of the message m as well as the size of the description of the function f.

In [BGI14] the authors propose three generic constructions. The first is a naive construction and just requires an adaptively secure (EUF-CMA secure) signature scheme. However, it does neither achieve function privacy nor succinctness. The second construction uses the first one, but additionally requires a zero-knowledge succinct non-interactive argument of knowledge (SNARK) system in order to achieve function privacy and succinctness. Finally, the third construction drops the succinctness requirement but still preserves function privacy. This is achieved using a non-interactive zero-knowledge arguments of knowledge (NIZKAoK) system instead of SNARKs. Unfortunately, neither of these three construction can be considered practically efficient.

We now sketch the application of FS to verifiable computing, where we align our description with the general definition of a verifiable computing scheme (cf. Definition 2.1). Therefore, let $(\mathsf{FS.Setup}, \mathsf{FS.KeyGen}, \mathsf{FS.Sign}, \mathsf{FS.Verify})$ be a secure, i.e., correct and unforgeable, FS scheme.

$\mathsf{KeyGen}(1^\lambda, f)$: Based on security parameter λ, run $(\mathsf{msk}, \mathsf{mvk}) \leftarrow \mathsf{FS.Setup}(1^\lambda)$. Set the evaluation key $\mathsf{ek} := \mathsf{sk}_{f'}$ with $\mathsf{sk}_{f'} \leftarrow \mathsf{FS.KeyGen}(\mathsf{msk}, f')$ where $f'(x) := f(x)||x$. It sets the verification key $\mathsf{vk} := \mathsf{mvk}$ and $\mathsf{sk} := \perp$ and returns $(\mathsf{sk}, \mathsf{vk}, \mathsf{ek})$.

$\mathsf{ProbGen}(\mathsf{sk}, x)$: The problem generation algorithm does not need to do any preprocessing. It sets $\sigma_x := x$ and $\rho_x := x$. The value σ_x is given to the server $\mathcal{S}$ to compute with it while the decoding value ρ_x is kept by the client $\mathcal{C}$ (but could be made public).

$\mathsf{Compute}(\mathsf{ek}, \sigma_x)$: Using the evaluation key $\mathsf{ek} := \mathsf{sk}_{'f}$ and the (encoded) input $\sigma_x := x$, $\mathcal{S}$ computes and returns an encoded version $\sigma_y := (y, \sigma)$ with $(\cdot, \sigma) \leftarrow \mathsf{FS.Sign}(\mathsf{sk}_{f'}, f', x)$ and with $y := f(x)$.

$\mathsf{Verify}(\mathsf{vk}, \rho_x, \sigma_y)$: Using the verification key $\mathsf{vk} := \mathsf{mvk}$, the decoding value $\rho_x := x$ and the encoded result $\sigma_y := (y, \sigma)$ the verification algorithm computes $b \leftarrow \mathsf{FS.Verify}(\mathsf{mvk}, y||x, \sigma)$ and if $b = 1$ it outputs y and $\perp$ otherwise.

The correctness of this construction follows from the correctness of the FS scheme. Moreover, it is obvious that the so obtain verifiable computing scheme is a publicly verifiable computing scheme according to Definition 2.4.

The above construction provides security in the non-adaptive model (weakly secure) (cf. Definiton 2.5 and Theorem 4.4 in [BGI14]). Moreover, it is clear that the so obtained scheme does trivially not provide input privacy (cf. Defintion 2.6). The efficiency of the above construction is directly related to the underlying FS scheme. In particular, the runtime of the verification is that of *FS.Verify* and the proof size is equal to the size of the signature of the FS scheme. As the definition of FS does not put any restriction on the time it requires to verify a signature (apart from being polynomial in the security parameter) it depends on the concrete FS scheme used in the construction if the efficiency definitions for verifiable computing are satisfied (cf. Definition 2.7 and 2.8).

7 Verifiable Computing for Specific Applications

Beyond the families of schemes we have seen so far, there exist verifiable computing schemes for specific functions, which we present here. Works like [CRR12], [ZSL14], or [CSL+15] which consider multiple clients or servers are beyond the scope of this work.

Signatures of Correct Computation. Papamanthou et al. presented in [PST13] the first, and to our knowledge only, framework for signatures of correct computation (SCC), which implies publicly verifiable computing. In particular they construct two SCC schemes, one for multivariate polynomials of fixed degree d and one for computing the derivations of multivariate polynomials. For multivariate polynomials f they use the fact that one can always write

$$f(x_1, \dots, x_n) - f(a_1, \dots, a_n) = \sum_{i=1}^{n} (x_i - a_i) q_i(x_1, \dots, x_n),$$

where $f, q_i \in \mathbb{F}[x_1, \dots, x_n]$ and the a_i are fixed inputs. Working over a symmetric bilinear group generated by g with bilinear map (or pairing) e one can compute $FK(f) = g^{f(t_1,\dots,t_n)}$ for some random t_i. The server evaluates the function for the given input $a_1, \dots, a_n$ and writes $f(x_1, \dots, x_n) - f(a_1, \dots, a_n)$ like above. It computes $w_i = g^{q_i(t_1,\dots,t_n)}$ and gives the w_i as well as the claimed result v to the client. The client can then check, whether

$$e(FK(f) \cdot g^{-v}, g) = \prod_{i=1}^{n} e(g^{t_i - a_i}, w_i)$$

holds. If it does, it accepts the result.

Using similar techniques they also construct a scheme to verify the computation of derivations. Both schemes are set in the standard model, offer adaptive security, but do not address input-output privacy.

Efficient Computation Outsourcing for Inverting a Class of Homomorphic Functions. In [ZML14] Zhang et al. present a scheme for verifying the inversion of a class of functions, namely group homomorphisms ϕ where computing a preimage under ϕ is computationally much more expensive than evaluating ϕ. In this case the server's evaluation of ϕ^{-1} can efficiently be verified by computing ϕ. This scheme offers only private verifiability, but is secure against an adaptive adversary. It also does not depend on any computational assumption and thus provides security in an information-theoretic sense. However, it does not provide input-output privacy.

Secure Delegation of Elliptic-Curve Pairing. A further scheme for outsourcing a concrete function, in this case a cryptographic bilinear map e, was introduced by Chevallier-Mames et al. in [CCM+10]. To compute $e(A, B)$ the client asks the server to compute

$$\begin{aligned} a_1 &= e(A + g_1 G_1, G_2) \\ a_2 &= e(G_1, B + g_2 G_2) \\ a_3 &= e(A + g_1 G_1, B + g_2 G_2) \\ a_4 &= e(s_1 A + r_1 G_1, s_2 B + r_2 G_2) \end{aligned}$$

with random points G_1, G_2 and random integers $g_1, g_2, r_1, r_2, s_1, s_2$. The client computes

$$e_{AB} = a_1^{-g_2} \cdot a_2^{-g_1} \cdot a_3 \cdot e(G_1, G_2)^{g_1 g_2}$$

and accepts the result as correct if

$$a_4 = (e_{AB})^{s_1 s_2} \cdot a_1^{r_2 s_1} \cdot a_2^{r_1 s_2} \cdot e(G_1, G_2)^{r_1 r_2 - g_1 r_2 s_1 - g_2 r_1 s_2}$$

holds. This scheme provides private verifiability, is efficient, and offers input and output privacy. In addition, it is unconditionally secure, so in particular secure against an adaptive adversary.

Efficiently Verifiable Computation on Encrypted Data. In [FGP14] Fiore et al. presented a way to verify univariate polynomial evaluations over encrypted data. The resulting scheme offers private verifiability, input privacy, and adaptive security while providing amortized efficiency.

TrueSet: Nearly Practical Verifiable Set Computations. In [KPP$^+$14] Kosba et al. presented a system named TrueSet that allows to verify *set operations*. This scheme supports *set circuits* built on union, intersection, and set difference gates. They presented a variant of [GGPR13]'s QAPs called *quadratic polynomial programs* (QPP). It achieves amortized efficiency and decreases the server's overhead by a factor of more than 150 compared to [GGPR13]. This scheme does not provide input-output privacy and security against adaptive adversaries is not mentioned.

Verifiable Delegation of Computation over Large Datasets. In [BGV11b] Benabbas et al. presented a verifiable computing scheme for multivariate polynomials of fixed degree d. Their scheme allows amortized verification while offering security against adaptive adversaries. It does not consider input-output privacy.

8 Analysis of the State of the Art

In this section, all verifiable computing schemes discussed in this survey are summarized and their properties are highlighted. The first property examined is which *function class* the scheme supports. Some support (subsets of) arithmetic circuits, while others can also deal with stateful operations or general loops, i.e., without needing to know the length of the loop during the preprocessing stage. Furthermore, we specify which type of *adversary* the solution can cope with. Some schemes are secure against a strong adversary (S), some are only secure against a weak adversary (W), and for some approaches the security level has not been analysed yet ($\emptyset$). In addition, we show which *primitives* the construction rely on, since most of them come with further assumptions regarding security. Furthermore, in some scenarios it might be preferable that the scheme provides a certain level of *privacy*. Depending on the type of data, a scheme may either ensure input privacy (I), output privacy (O), input-output privacy (I/O), or no privacy at all ($\times$). To be usable in practice, a scheme also needs to provide *efficiency* as described in Section 2.7. We define a verifiable computing scheme as efficient (E), if the time required for preprocessing and verification is $o(T)$, where T is the time required to compute the function. If only the verification can be performed in $o(T)$, then the computing scheme only provides amortized efficiency (A). Note that verifiable computing schemes that do not provide any of these two types of efficiency have not been discussed in this work. Finally, most solutions are tailored to *private verification*, i.e., where the verification can only be performed by the data owner. However, in some scenarios the verification must be performed by a party different to the owner, requiring the scheme to be *publicly verifiable*. Sometimes it is not possible to make a general statement about a scheme's attributes as they are dependent on choices of primitives (D). These abbreviations are summarized in table 1.

Category	Abbreviation	Explanation
Adversary	S	Strong adversary
	W	Weak adversary
Privacy	I	Input privacy
	O	Output Privacy
	I/O	Input-output privacy
	$\times$	no privacy
Efficiency	E	Efficient
	A	Amortized efficient
General	D	Dependent on primitives

Table 1: Used abbreviations

As shown by Table 2 the only proof based approach that provides an efficient generation and verification process is the one proposed in [TRMP12]. This scheme, however, only supports a very restricted class of circuits. The other PCP or linear PCP based constructions support larger classes of programs, but only achieve amortized efficiency. In addition, all these approaches are interactive, i.e., require multi round interaction between the server and the client. To reduce the server's overhead later solutions are non-interactive. The latest proposal [BBFR15] even

Scheme	**Function Class**	**P**	**E**	**PV**
[TRMP12]/[Tha13]	Circuits of polylog. depth	×	E	✓
[VSBW13]	Arithm. Circuits	×	A	✓
[SMBW12]	Arithm. Circuits	×	A	×
[SVP+12]	Arithm Circuits + more	×	A	×
[BFR+13b]	Stateful	×	A	×
[SBV+13]	Arithm. Circuits + more	×	A	×
[XAG14]	Arithm. Circuits	×	A	✓
[PHGR13]	Arithm. Circuits + more	×	A	✓
[CFH+15]	Arithm. Circuits + more	×	A	✓
[BCG+13]	General Loops	×	A	✓
[BCTV14]	General Loops	×	A	✓
[BBFR15]	Arithm. Circuits	I	A	✓

Table 2: Proof Based Verifiable Computation Schemes. Legend: **P**. . . privacy, **E**. . . efficiency, **PV**. . . public verifiability

achieves input privacy and provides public verifiability. However, all non-interactive proof based schemes use QAPs and are therefore based on *non-falsifiable* assumptions of knowledge. As shown in [GW11] it is actually impossible to build a SNARG (e.g., using QAPs) that is based solely on falsifiable assumptions. This raises some questions on the security of these schemes. In fact, although it has been shown that PCPs and QAPs are secure against an adaptive adversary, it has not been proven that the same holds true for the verifiable computing scheme using this primitive.

Scheme	**Function Class**	$\mathcal{A}$	**P**	**E**	**PV**
[GGP10]	Arithm. Circuits	W	I/O	A	×
[CKV10]	Arithm. Circuits	W	I/O	A	×
[TC14]	Arithm. Circuits	W	I/O	A	×

Table 3: FHE Based Verifiable Computation Schemes. Legend: $\mathcal{A}$. . . adversary, **P**. . . privacy, **E**. . . efficiency, **PV**. . . public verifiability

Constructions based on fully homomorphic encryption naturally offer input-output privacy, because the inputs and correspondingly the outputs are encrypted. However, they do not provide public verifiability. Furthermore, as shown in Table 3 all constructions available are proven secure against a weak adversary only and provide amortized efficiency. Thus, how to build efficient solutions that are secure against strong adversaries is still an open question. Moreover, currently FHE cannot be considered a practical tool.

The schemes using homomorphic authentication (see Table 4) are more restrictive with respect to the supported function class. Furthermore, all schemes only provide amortized efficiency. This is due to an expensive preprocessing stage which is computationally dominated by the costs for authenticating or signing data. Note that in some use cases only authenticated data is processed anyway. Here the preprocessing stage can be omitted making these schemes much more efficient. The solutions using homomorphic signature schemes even provide public verifi-

Scheme	Function Class	$\mathcal{A}$	P	Primitives	E	PV
[BFR13a]	Poly. of Degree 2	S	×	Bilinear Maps	A	×
[CKV10]	Poly of Fixed Degree	S	×	Multilinear Maps	A	×
[CFGN14]	Poly of Fixed Degree	S	×	Multilinear Maps	A	×
[FGP14]	Poly of Degree 2	S	I	Bilinear Maps	A	×
[CFN15]	Linear	S	×	Bilinear Maps	A	✓
[CFW14]	Poly of Fixed Degree	S	×	Multilinear Maps	A	✓
[CFGV13]	Poly of Fixed Degree	S	×	RSA	A	✓
[LDPW14]	D	D	I/O	HE/HEA	D	D

Table 4: Authenticator Based Verifiable Computation Schemes. Legend: $\mathcal{A}$. . . adversary, **P**. . . privacy, **E**. . . efficiency, **PV**. . . public verifiability

ability. Furthermore, the generic construction proposed by Lai et al. [LDPW14] allows to combine authentication based verifiability with encryption gaining a verifiable computing scheme preserving input-output privacy. Nevertheless, the function class, security, and efficiency depends on the underlying primitives and further research is required for identifying promising instantiations for different applications.

Scheme	Function Class	$\mathcal{A}$	P	Primitives	E	PV
[PRV12]	Boolean Functions	∅	×	ABE	A	×
[BF12]	D	S	I/O	FE,MAC,FHE,PE	A	✓
[BGI14]	Arithm. Circuits	W	×	FS	D	✓

Table 5: FE and FS based Verifiable Computation Schemes. Legend: $\mathcal{A}$. . . adversary, **P**. . . privacy, **E**. . . efficiency, **PV**. . . public verifiability

Another line of research are verifiable computing schemes based on functional encryption or functional signatures (cf. Table 5). The authors of [PRV12], for instance, introduced a primitive built on attribute based encryption. However, for this construction the security has not been analysed yet. The other FE based approach introduced by Barbosa and Farshin [BF12] is generic but requires FE for general predicates and can thus not be considered practical yet. Another very interesting approach is to build verifiable computing schemes from functional signatures. Also in this direction more research can be done to allow for a scheme that is efficient and secure against the strong adversary.

Besides the main research directions, i.e., proof based, FHE based, authenticator based, and functional encryption/signature based verifiable schemes, there are also several solutions for specific applications, see Table 6. If their properties meet the requirements of the application to be implemented these constructions can also be considered.

The summary shows that the only verifiable computing scheme that achieves efficiency over a single instantiation is a proof based solution. In addition, this line of research has produced constructions that support the most general classes of functions, e.g., general loops and stateful operations. On the downside, their security has not been proven yet, some solutions rely on non-falsifiable assumptions, and privacy is not addressed. Thus, if a verifiable computing scheme

Scheme	**Function Class**	$\mathcal{A}$	**P**	**Primitives**	**E**	**PV**
[PST13]	Poly. + Derivations	S	$\times$	Bilinear Maps	A	✓
[ZML14]	Inversions of Homomorphismss	S	$\times$	$\emptyset$	A	$\times$
[CCM+10]	Bilinear Maps	S	$\times$	Bilinear Maps	A	$\times$
[FGP14]	Univariate Poly.	S	I	Bilinear Maps	A	$\times$
[BGV11b]	Poly of Fixed Degree	S	$\times$	Bilinear Map	A	$\times$
[KPP+14]	Set Operations	W	$\times$	QPP	A	✓

Table 6: Other Verifiable Computation Schemes. Legend: $\mathcal{A}$...adversary, **P**...privacy, **E**...efficiency, **PV**...public verifiability

providing input-output privacy for a wide class of functions is needed, one currently has to rely on inefficient approaches using fully homomorphic encryption. Their additional shortcomings are, however, that they are only proven secure against the weak adversary and that they come with an expensive preprocessing phase. However, use cases that allow the termination of a protocol as soon as one single input is rejected and that do not require an efficiency preprocessing phase may still use this line of research. Verifiable computing schemes that are authenticator based are proven secure against an adaptive adversary and some solutions even provide public verifiability and/or input-output privacy. On the downside they are currently very restricted with respect to the function class provided. Note however that the operations supported include a huge amount of statistical operations and can therefore be of interest for many concrete instantiations. Furthermore, with respect to the approach providing input-output privacy, more research has to be done regarding an instantiation gaining (amortized) efficiency. Apart from the numerous solutions based on proofs, homomorphic encryption, and homomorphic authentication, also other promising approaches, e.g., based on functional encryption, based on functional signatures, and tailored to specific applications, have been proposed. The properties of the construction using functional signatures, for instance, depend on the signature scheme used. Thus, with developing efficient functional signatures also the potential of this approach will increase.

9 Conclusion

This work shows that the field of verifiable computing, although not very old, has made huge improvements over the last years. Various solutions have been found for different function classes. The concrete practicality of schemes depends on the server's computational overhead, which in turn often depends on the efficiency of the primitives used in the verifiable computing scheme. So advances in fields like FHE, pairings, multilinear maps, circuit generation, or garbled circuits will each be beneficial for the state of the art in verifiable computing. Note that so far there is only one scheme where both, the time required for generation and verification is $o(T)$, where T is the time required to compute the function.

Another requirement that is very important, but only sparely provided in a strong adversary setting, is privacy. There are several attempts to combine verifiable computing schemes secure against adaptive adversaries with privacy preserving ones. However, there are no instantiations so far that allow to build a construction that at the same time is secure in the strong adversary model and provides efficiency and privacy. For many applications such a primitive would be very valuable. Thus, developing a corresponding solution is an interesting task for future work.

In PRISMACLOUD we aim at providing verifiable computing solutions for an e-Health use case. After identifying the function class that needs to be supported we will collect all candidates that fulfil the requirements, select the best approach, and integrate it into our toolbox. Furthermore, we aim at improving the state of the art. Possible directions are to identify and improve suitable instantiations for the generic construction proposed by Lai et al. [LDPW14] and Boyle et al. [BGI14].

References

[ABC$^+$15] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A. Reuter, and Martin Strand. A guide to fully homomorphic encryption. Cryptology ePrint Archive, Report 2015/1192, 2015. http://eprint.iacr.org/.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BBFR15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 271–286, 2015.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349, 2012.

[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 *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 90–108, 2013.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 781–796, 2014.

[BF12] Manuel Barbosa and Pooya Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. In *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, pages 296–312, 2012.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, pages 68–87, 2009.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BFR13a] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 863–874, 2013.

[BFR+13b] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pages 341–357, 2013.

[BGH+05] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Short pcps verifiable in polylogarithmic time. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005)*, pages 120–134, 2005.

[BGH+06] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust pcps of proximity, shorter pcps, and applications to coding. *SIAM J. Comput.*, 36(4):889–974, 2006.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 501–519, 2014.

[BGV11a] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 111–131, 2011.

[BGV11b] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:111, 2011.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011*, pages 253–273, 2011.

[CCM+10] Benoît Chevallier-Mames, Jean-Sébastien Coron, Noel McCullagh, David Naccache, and Michael Scott. Secure delegation of elliptic-curve pairing. In *Smart Card Research and Advanced Application, 9th IFIP WG 8.8/11.2 International Conference, CARDIS 2010, Passau, Germany, April 14-16, 2010. Proceedings*, pages 24–35, 2010.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, pages 538–555, 2014.

[CFGV13] Dario Catalano, Dario Fiore, Rosario Gennaro, and Konstantinos Vamvourellis. Algebraic (trapdoor) one-way functions and their applications. In *TCC*, pages 680–699, 2013.

[CFH+15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 253–270, 2015.

[CFN15] Dario Catalano, Dario Fiore, and Luca Nizzardo. Programmable hash functions go private: Constructions and applications to (homomorphic) signatures with shorter public keys. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 254–274, 2015.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 371–389, 2014.

[CKV10] Kai-Min Chung, Yael Tauman Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 483–501, 2010.

[CMP14] Dario Catalano, Antonio Marcedone, and Orazio Puglisi. Authenticating computation on groups: New homomorphic primitives and applications. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 193–212, 2014.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[CRR12] Ran Canetti, Ben Riva, and Guy N. Rothblum. Two protocols for delegation of computation. In *Information Theoretic Security - 6th International Conference, ICITS 2012, Montreal, QC, Canada, August 15-17, 2012. Proceedings*, pages 37–61, 2012.

[CSL+15] Xiaofeng Chen, Willy Susilo, Jin Li, Duncan S. Wong, Jianfeng Ma, Shaohua Tang, and Qiang Tang. Efficient algorithms for secure outsourcing of bilinear pairings. *Theoretical Computer Science*, 562(0):112 – 121, 2015.

[FGP14] Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 844–855, 2014.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, pages 697–714, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 465–482, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 626–645, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 113–122, 2008.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 89–98, 2006.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 99–108, 2011.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, pages 301–320, 2013.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 278–291, 2007.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 723–732, 1992.

[KPP$^+$14] Ahmed E. Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, Mahmoud F. Sayed, Elaine Shi, and Nikos Triandopoulos. TRUESET: faster verifiable set computations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 765–780, 2014.

[LDPW14] Junzuo Lai, Robert H. Deng, HweeHwa Pang, and Jian Weng. Verifiable computation on outsourced encrypted data. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, pages 273–291, 2014.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 2013.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, pages 422–439, 2012.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[SBV$^+$13] Srinath T. V. Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 71–84, 2013.

[SMBW12] Srinath T. V. Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[SVP$^+$12] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 253–268, 2012.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology - EUROCRYPT 2005*, pages 457–473, 2005.

[TC14] Chunming Tang and Yuenai Chen. Efficient non-interactive verifiable outsourced computation for arbitrary functions. *IACR Cryptology ePrint Archive*, 2014:439, 2014.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 71–89, 2013.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. In *4th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'12, Boston, MA, USA, June 12-13, 2012*, 2012.

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 223–237, 2013.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2):74–84, 2015.

[WSR$^+$15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[XAG14] Gang Xu, George T. Amariucai, and Yong Guan. Verifiable computation with reduced informational costs and computational costs. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I*, pages 292–309, 2014.

[XAG15] Gang Xu, George T. Amariucai, and Yong Guan. Block programs: Improving efficiency of verifiable computation for circuits with repeated substructures. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, pages 405–416, 2015.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

[ZML14] Fangguo Zhang, Xu Ma, and Shengli Liu. Efficient computation outsourcing for inverting a class of homomorphic functions. *Inf. Sci.*, 286:19–28, 2014.

[ZS14] Liang Feng Zhang and Reihaneh Safavi-Naini. Generalized homomorphic macs with efficient verification. In *ASIAPKC'14, Proceedings of the 2nd ACM Wookshop on ASIA Public-Key Cryptography, June 3, 2014, Kyoto, Japan*, pages 3–12, 2014.

[ZSL14] Liang Feng Zhang, Reihaneh Safavi-Naini, and Xiao Wei Liu. Verifiable local computation on distributed data. In *Proceedings of the Second International Workshop on Security in Cloud Computing, SCC@ASIACCS '14, Kyoto, Japan, June 3, 2014*, pages 3–10, 2014.