On Encrypted Functions and Verifiable Computing

Hunter Leath
University of Virginia
jhl4qf@virginia.edu

Abstract

Much like exploring the notion of zero-knowledge proofs through extensions of the typical notions of "proof," we can take one step further into the world of verifiable computation by extending our notion of proof again. This paper will discuss recent advances towards verifiable delegation of computation while also exploring the relevant intersection with secure function evaluation.

Verifiable computation represents the notion of "outsourcing" computation to more powerful workers and ensuring that the results returned are correct despite the worker's actions. On the other hand, secure function evaluation is the attempt to prevent a worker from learning anything about the function that it is computing.

With a combination of these two approaches, we can create a system that allows us to "outsource" proprietary code to an untrusted worker while remaining confident that the proprietary algorithms will remain secure and the results of the computation are correct.

1. Introduction

With the rise in popularity of mobile devices over the past decade, it has become clear that people enjoy devices that are small in size, long in battery life, and powerful. However, it is immediately obvious that a single device cannot possible incorporate all of these features because of their interplay. For that reason, hardware manufacturers and consumers have grown accustomed to using mobile devices that are unable to compete with desktops in terms of power. Developers for these devices must then make trade offs between features and available processing power (assuming they desire a minimal impact on battery life).

This situation has become less important with the advances of the "cloud". Developers for mobile devices are increasingly moving computation-heavy algorithms onto servers to lighten the load on the individual mobile client. This model makes the necessary assumption that the servers are controlled by the developer and not malicious; however, as can be seen from the constant flurry of news articles about high-profile companies being "hacked", this is not a realistic assumption in today's world.

Instead, we can turn to cryptography to provide these assurances. Verifiable computing is a topic in security with a renewed vigor due in part to Gentry's recent contributions towards Fully Homomorphic Encryption [G09]. It allows the secure "outsourcing" of computation to untrusted workers while ensuring that results of the computation are correct.

Figure 1. A typical content producing website with advertising.

In this paper, we will survey recent advances in the field of verifiable computation as well as the assumptions they are founded on, attempt to integrate secure function evaluation as a metric of these schemes in order to protect proprietary algorithms from the untrusted worker, and present a unique usage pattern for these schemes to evaluate their practicality.

2. Overview

This paper will be organized by the topicality of the assumptions that verifiable computing schemes utilize. We will begin with recent advances in primitives, then build the larger schemes that utilize theses new primitives. We will also discuss the runtime cost of each scheme.

2.1 Motivating Example

We will begin first, however, with the discussion of the motivating example. This will help in comparing the runtime of current algorithms with the "ideal" to gauge the distance to practicality.

A typical website that produces content (as opposed to selling a service) is often seen with advertisements surrounding the main text of the webpage like the *New York Times* in Figure 1. The interaction between advertisements and journalistic integrity has been the subject of much controversy over the years. With the advent of impression-based advertising on the internet, the desire for journalists to bend headlines to be attention-grabbing or "clickbait" has only increased.

Using verifiable computation and secure function evaluation, we can introduce a new model of monetization for content producing websites. In the new model, when a user visits the website, a small amount of Javascript is loaded onto the client to fetch jobs from a central server. The users computer is then designated as the

UVa SIGCRYPT, Dec 15, 2014, Charlottesville, VA, USA.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

q	p	z	$Computation$
k_q^0	k_p^0	k_z^{00}	$Enc_{k_q^0}(Enc_{k_p^0}(k_z^{00}))$
k_q^0	k_p^1	k_z^{01}	$Enc_{k_q^0}(Enc_{k_p^1}(k_z^{01}))$
k_q^1	k_p^0	k_z^{10}	$Enc_{k_q^1}(Enc_{k_p^0}(k_z^{10}))$
k_q^1	k_p^1	k_z^{11}	$Enc_{k_q^1}(Enc_{k_p^1}(k_z^{11}))$

Table 1. A simplified example of Yao's Garbled Circuits on a binary function taking two inputs (q and p) and returning a single output (z).

"worker" and completes computational jobs from the central queue so long as the user does not navigate away from the site.

The content producing website is then paid by organizations who wish to run computationally-intensive applications on users machines in a distributed manner. Rather than being paid based merely on impressions, the site is rewarded for the amount of time users spend consuming their content. This aligns the goals of the benefactors who wish to see their computations completed and the content producers who wish for their content to be consumed.

Of course, the missing piece of the scheme is the assurance that the distributed workers do not cheat or attempt to change the results of the computation. Verifiable computation is able to fill this hole nicely. Additionally, the utilization of secure function evaluation could convince companies that proprietary algorithms run on these platforms would not be leaked to consumers or competing organizations.

If a practical scheme supporting both verifiable computation and secure function evaluation existed, then the construction of such a monetization strategy could quickly follow. The rest of this paper will be focused on examining recent progress towards these goals in the context of this web-based usage.

3. Definitions

In this section, we will cover the definitions of primitives utilized by the full verfiable computation schemes.

Garbled Circuits. One of the first strong primitives utilized for verifiable computation is the "Garbled Circuit" pioneered by Yao in the 1980s. The Garbled Circuit was a generalization of previous specific secure computation protocols. With Garbled Circuits, any computation that could be represented as a binary circuit could be computed securely by multiple parties (keeping personal inputs private). One party would serve as the circuit constructor and another as the evaluator. For each gate in the circuit, the constructor picks random numbers representing the state of the input wires and the output wire. He then computes the double encryption of the output wire random numbers under the keys of the random numbers corresponding to the input wires. When the evaluator eventually computes the output of the gate, he will only be able to decrypt the answer corresponding to the values of the input wires he actually has. An example of a "Garbled Gate" can be seen in Table 1.

Fully Homomorphic Encryption. Up until recently, "Fully Homomorphic Encryption" (FHE) was a primitive often wished for by cryptographers while remaining out of reach without the promise of ever being constructed. In 2009, Gentry [G09] was able to fulfill hopes by constructing the first FHE scheme based on ideal lattices.

As defined by him, a fully homomorphic encryption scheme $\mathcal{E}$ has all of the functions as a usual asymmetric encryption primitive (KeyGen, Encrypt, Decrypt) with the addition of the function Evaluate. Such that

$$\mathsf{Evaluate}_{\mathcal{E}}(pk, C, \psi_0, ...\psi_n) = \mathsf{Encrypt}_{\mathcal{E}}(pk, C(\pi_0, ...\pi_n))$$

where $\psi_i = \mathsf{Encrypt}_{\mathcal{E}}(pk, \pi_i)$.

Therefore, anyone with the public key is able to do arbitrary computation over the ciphertext and receive an encrypted answer. It should be clear that for this reason, it is impossible to achieve CCA security of a Fully Homomorphic Encryption scheme. However, in general, Gentry's scheme is able to provide semantic security and is very useful for verfiable computation.

Attribute Based Encryption. Attribute Based Encryption (ABE) as refined by Goyal et. al. [G06] is an encryption scheme where the ciphertext is based on a *public* attribute x such that $\psi = \mathsf{Enc}_{pk}(\pi; x)$. Using the master secret key (msk) created with KeyGen, the encryptor can create a functional secret key sk_f that will allow a user to reconstruct the plaintext if $f(x) = 1$.

Often one talks about Attribute Based Encryption with "collusion-resistance" - a property implying that many users with different secret keys cannot collude (by combining their secret keys) to get access to the plaintext.

Functional Encryption. Functional Encryption (FE) as formalized in [BSW11] is able to generalize the Attribute Based Encryption scheme by changing how the secret keys interact with the ciphertext. In FE, there is no public attribute x encoded into the ciphertext. Instead, when secret keys sk_f are created to give users "access" to a function f, the only property of the plaintext that users can compute is $f(\pi)$.

Up until recently, results in Functional Encryption were only able to calculate the inner product of messages. However, recent advances by Garg et. al. [G13a] have given candidate encryption schemes that support arbitrary computations.

Witness Encryption. In the same year, Garg et. al. [G13b] proposed the concept of Witness Encryption (WE). A WE scheme is defined on an NP language $\mathcal{L}$. The Encrypt function takes a member x of the language $\mathcal{L}$ as one of the parameters. To decrypt the ciphertext, one must provide the witness w that proves x's membership in $\mathcal{L}$. If $x \notin \mathcal{L}$, then no poly-time adversary can decrypt the ciphertext with non-negligible probability.

Oblivious Turing Machine. The standard Turing Machine has been the basis for computational complexity for most of modern history. During that time, it is natural for slightly modified versions of the Turing Machine to be proposed. One such modification, useful for our purposes here, is the idea of an *Oblivious* Turing Machine. For these machines, the movement of the tape head is *exactly* the same for all input of the same length. The advantage conferred from this model is that an adversary watching the Turing Machine learns nothing about the input from the movement of its heads.

In the late 1970s, Pippenger and Fischer created the most efficient known reduction to Oblivious Turing Machines. They showed that any computation run on a Turing Machine running in $\mathcal{O}(n)$ time can be made oblivious and run in $\mathcal{O}(n \log n)$ time.

4. Verifiable Computation

Armed with the background knowledge of important primitives for creating Verifiable Computation, we are ready to analyze recent work in the area.

Verifiable Computation is a protocol between two parties: the delegator and the worker. The delegator has a difficult computation F that he wants outsourced to the worker, but doesn't trust the worker to perform the calculation correctly. A verifiable computation scheme gives the assurance that the delegator will not be fooled by an incorrect result with a non-negligible probability.

4.1 Circuit Based Approaches

Since most approaches to Verifiable Computation utilize a variant of Yao's Garbled Circuits, they implicitly model computation as Boolean Circuits. For all of the works presented below, algorithms must be compiled into Boolean Circuits before evaluation.

Yao's Circuits. The first observation for utilizing Yao's Circuits for a Verifiable computation is changing the protocol from two-parties calculating a function to just one-party. In this simple modification, the delegator prepares the circuit for computation, and the worker actually evaluates the circuit. Given that the random wire values are drawn from a large enough space, it is infeasible for the worker to "guess" an incorrect wire value that he did not decrypt.

The problem with this approach lies in the runtime cost. Yao's circuits are only secure for a single evaluation, otherwise an adversary can learn a portion of the random wire values. Since Garbling the circuit for each input is very slow, we hope that a more advanced scheme will give a faster running time for subsequent evaluations.

Verifiable Computation with FHE. One of the first papers to take advantage of the renewed focus on FHE utilizes the construct to create Verifiable Computation [GGP10]. The motivating factor behind the work was from looking at the traditional properties that FHE was unable to provide. Namely, the FHE scheme proposed by Gentry was unable to ensure that the computations undertaken using Evaluate were correct. An adversary could return a malformed ciphertext that did not contain the result of a computation. This work added these features to FHE.

This work builds directly off of the scheme utilizing only Yao's circuits, with the goal of *amortizing* the cost of building the circuit over many evaluations. Their scheme involves running the entire circuit (double decryption and all) under a FHE scheme. This allows the worker to compute the encryption of the random number representing the state of the output wire, but not see it directly.

Because of this jump, the Garbled Circuit may now be used multiple times for different inputs. However, they note an important weakness in their scheme - the worker may not learn whether the delegator accepted the computation or not. Given a sufficiently advanced adversary that perturbs the garbled circuit for each execution, the decision of the delegator to accept the computation may leak a bit of information about the circuit itself.

4.2 Turing Machine Based Approaches

The works below are able to break away from the standard model of computation in terms of Boolean Circuits and are able to create results that operate directly on Turing Machines.

Running Turing Machines on Encrypted Data. Unlike previous approaches, Goldwasser et. al. [GK13a] provides a result that models computation off of Turing Machines. The first major benefit of this approach is avoiding the "worst case runtime curse" in which Turing Machines compiled into Boolean Circuits take the same amount of time on every input - the worst case runtime. They are able to build off of their previous results [GK13b] that provide the first *succint* functional encryption scheme to create reusable garbled circuits.

In this work, they create many cryptographic objects of value:

1. Attribute Based Encryption for Turing Machines
2. Functional Encryption for Turing Machines
3. Reusable Garbling for Turing Machines
4. Fully Homomorphic Encryption for Turing Machines

The work relies on Witness Encryption to provide the reductions between these different esoteric forms of encryption.

It is with this result that we are able to generalize our Verifiable Computation schemes to protocols that necessarily hide the underlying function being evaluated. Now, rather than communicating the function to the worker as a garbled circuit, the delegator can actually encrypt the function itself to evaluate it. Using this result, the function can be encoded as a Turing Machine, encrypted using their FHE scheme for Turing Machines, and then evaluated under the Universal Turing Machine.

In their work, they give the option of utilizing the Peppinger-Fischer transformation [PF79] to hide the runtime of the Turing Machine by making it oblivious. We can utilize this to add another layer of security to our scheme with the trade off of runtime.

Blind Turing Machines. The work by Rass [R13] is able to successfully combine the notions of hiding the inputs to the computation and the algorithm itself from the worker in the Turing Machine model. The work focuses on showing that in order to evaluate a Turing Machine, one does not need a *Fully* Homomorphic Encryption scheme. Rather, using ElGamal as an example, only requires multiplicative Homomorphisms.

The motivation behind this work is the speed of current Verifiable Computation schemes and their heavily utilization of the slow FHE system. It is certainly an encouraging result for exploring Verifiable Computation in real world systems.

4.3 Pure Function Approach

In this approach, rather than compiling the function to any intermediary representation, the delegator can simply give the function to the worker.

Improving the FHE-Based Scheme. A follow-up work to the [GGP10] scheme of Verifiable Computation attempts to alleviate the inefficiencies present in the circuit-generation stage - the large public key length and the runtime [CKV10].

The work draws on the intuition about the knowledge of the worker. If the delegator were to ask the worker to compute the function for *two* identically distributed inputs (only not-knowing the result for one of the two), the worker would be unable to distinguish between the input the delegator "cared about" and the random input. Therefore, the delegator could check that the computation was performed correctly for the random input and use that as a signal for whether the worker attempted to cheat for the other input.

They are able to utilize several layers of FHE to allow for arbitrary inputs with reusable precomputed pairs. In effect, each input is encrypted under FHE so that the worker cannot distinguish between an input drawn from a random distribution an input chosen purposefully. FHE is used to evaluate the function "under the encryption". Much like [GGP10], they utilize FHE again over the entire scheme to make it reusable. This ensures that the worker learns nothing about the randomized, pre-computed inputs regardless of malicious aciton.

Note that because this scheme requires transporting the function directly to the worker, it cannot provide secure function evaluation and hide the function from the worker. However, this technique is general enough that it can be applied to other schemes that hide the function being evaluated.

5. Practicality

As discussed in previous sections, one of the main obstacles to practicality of Verifiable Computation schemes is their heavy reliance on Fully Homomorphic Encryption. While much work is being done to improve the results of the original Gentry [G09] scheme, the most complete *implementation* test of the scheme [GH11] showed than many operations were on the verge of taking *minutes* on modern computers for appropriately sized keys.

Since many of these schemes rely on utilizing one (or two) rounds of FHE at each step of evaluation, it is certainly difficult to envision a situation in which a real-world computation could be performed in this manner without being an order of magnitude more expensive than simply doing the calculation.

Fortunately, with the renewed interest in Fully Homomorphic Encryption, improvements are being created. Since all of these schemes utilize FHE as a primitive, any speedups of the underlying encryption will increase the speed of the scheme itself.

Additionally, avenues with Turing Machines are promising due to their use of other encryption primitives (FE, ABE) which may see speed increases in the future.

6. Conclusion

Verifiable Computation has many immediate uses in a world where being reliant on the "cloud" for processing power is commonplace. Much like Fully Homomorphic Encryption before its proof by construction, the applications are clearly present.

In this paper, we have discussed a novel use case for the Verifiable Computation primitive as well as several schemes that attempt to create Verifiable Computation. This paper has addressed important primitives that are used to create Verifiable Computation including the approaches that are used.

Unfortunately, under the current limits of Fully Homomorphic Encryption, these schemes are not fit for use in a real-world environment, but will hopefully see improvement over time. Assuming that this obstacle is eventually overcome, it would be interesting to see the creation of compilers like Zahur's [Z14] that effectively allow normal developers to create Verifiable Computations by translating their high-level code into Boolean Circuits or Turing Machines.

As can be seen with the Motivating Example, a fast Verifiable Computation scheme could be used to change the economics behind content producing websites today changing users' computers into a distributed version of the large data centers that we rely on so heavily.

References

[BSW11] Boneh, Dan, Amit Sahai, and Brent Waters. "Functional encryption: Definitions and challenges." Theory of Cryptography. Springer Berlin Heidelberg, 2011. 253-273.

[CKV10] Chung, Kai-Min, Yael Kalai, and Salil Vadhan. "Improved delegation of computation using fully homomorphic encryption." Advances in CryptologyCRYPTO 2010. Springer Berlin Heidelberg, 2010. 483-501.

[G13a] Garg, Sanjam, et al. "Candidate indistinguishability obfuscation and functional encryption for all circuits." Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on. IEEE, 2013.

[G13b] Garg, Sanjam, et al. "Witness encryption and its applications." Proceedings of the forty-fifth annual ACM symposium on Theory of computing. ACM, 2013.

[G09] Gentry, Craig. "Fully homomorphic encryption using ideal lattices." STOC. Vol. 9. 2009.

[G06] Goyal, Vipul, et al. "Attribute-based encryption for fine-grained access control of encrypted data." Proceedings of the 13th ACM conference on Computer and communications security. ACM, 2006.

[GGP10] Gennaro, Rosario, Craig Gentry, and Bryan Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers." Advances in CryptologyCRYPTO 2010. Springer Berlin Heidelberg, 2010. 465-482.

[GH11] Gentry, Craig, and Shai Halevi. "Implementing gentrys fully-homomorphic encryption scheme." Advances in CryptologyEUROCRYPT 2011. Springer Berlin Heidelberg, 2011. 129-148.

[GK13a] Goldwasser, Shafi, et al. "How to run turing machines on encrypted data." Advances in CryptologyCRYPTO 2013. Springer Berlin Heidelberg, 2013. 536-553.

[GK13b] Goldwasser, Shafi, et al. "Reusable garbled circuits and succinct functional encryption." Proceedings of the forty-fifth annual ACM symposium on Theory of computing. ACM, 2013.

[P13] Parno, Bryan, et al. "Pinocchio: Nearly practical verifiable computation." Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013.

[PF79] Pippenger, Nicholas, and Michael J. Fischer. "Relations among complexity measures." Journal of the ACM (JACM) 26.2 (1979): 361-381.

[R13] Rass, Stefan. "Blind Turing-Machines: Arbitrary Private Computations from Group Homomorphic Encryption." arXiv preprint arXiv:1312.3146 (2013).

[Z14] Zahur, Samee. "Obliv-c: A lightweight compiler for data-oblivious computation." Workshop on Applied Multi-Party Computation. Microsoft Research, Redmond. 2014.