Probabilistic Verification of Outsourced Computation Based on Novel Reversible PUFs

Hala Hamadeh, Abdallah Almomani, Akhilesh Tyagi

Probabilistic Verification of Outsourced Computation Based on Novel Reversible PUFs

Hala Hamadeh[1], Abdallah Almomani[2], and Akhilesh Tyagi[1]

[1] Iowa State University, Ames IA 50010, USA
[2] Jordan University of Science and Technology, Irbid 22110, Jordan

Abstract. With the growing number of commercial cloud-computing services, there is a corresponding need to verify that such computations were performed correctly. In other words, after a weak client outsources computations to an untrusted cloud, it must be able to ensure the correctness of the results with less work than re-performing the computations. This is referred to as verifiable computation. In this paper we present a new probabilistic verifiable computation method based on a novel Reversible Physically Unclonable Function (PUF) and a binomial Bayesian Inference model. Our scheme links the outsourced software with the cloud-node hardware to provide a proof of the computational integrity and the resultant correctness of the results with high probability. The proposed Reversible SW-PUF is a two-way function capable of computing partial inputs given its outputs. Given the random output signature of a specific instruction in a specific basic block of the program, only the computing platform that originally computed the instruction can accurately regenerate the inputs of the instruction correct within a certain number of bits. To explore the feasibility of the proposed design, the Reversible SW-PUF was implemented in HSPICE using 45 nm technology. The probabilistic verifiable computation scheme was implemented in C++, and the Bayesian Inference model was utilized to estimate the probability of correctness of the results returned from the cloud service. Our proof-of-concept implementation of Reversible SW-PUF exhibits good uniqueness compared to other types of PUFs and exhibits perfect reliability and acceptable randomness. Finally, we demonstrate our verifiable computation approach on a matrix computation. We show that it enables faster verification than existing verification techniques.

1 Introduction

Verifiable computations (VC) have attracted enormous interest and attention with the recent growth in cloud computing. The concept of verifiable computation allows a lower-resource client to outsource the computation of a program to an untrusted cloud. With a proof provided by the cloud, the client can verify that the results produced are consistent with the program specification and that the computations were performed correctly. To be viable, the effort of performing the verification must be negligible compared to the actual computation. Three main solutions were proposed to support verifiable computation: VC based on

Trusted Computing [2]. The main drawback of this approach was the assumption that the physical protections cannot be defeated. A second method, VC with a Non-Interactive Argument, is described in [8]. This approach is not practical because it relies on complex Probabilistically Checkable Proofs (PCPs) or fully-homomorphic encryption (FHE). Finally, VC with Interactive Proofs [9] has been propose. While this approach is often efficient, it applies to only a narrow class of computations.

In recent years, interest in physically-unclonable functions (PUFs) has evolved. PUFs have been deployed in different applications because of their ability to generate digital fingerprints of unique identities for a physical system. SW-PUF [6] is a specific type of PUF that binds software execution to the exact hardware platform and produces unique signatures at various points in the softwares execution. The SW-PUF signature is a promising candidate for providing a proof that a specific computation was performed on a specific platform. By expanding the capabilities of a SW-PUF to include invertibility and commutativity, we achieve elements of verifiable computation. Invertibility is achieved by capturing a physical attribute such as time when an output bit settles using reversible functions. Reversibility is obtained with transmission gates.

2 The Reversible SW-PUF

The design of the Reversible SW-PUF is an extension of our previous work on the SW-PUF [6]. As in the original SW-PUF, the ALU signatures of an instruction on the reversible SW-PUF are generated from an early sampling of the ALU results. However, in the reverse mode, the roles of inputs and outputs are reversed, and the early sampling is done on the original input end. Reversible SW-PUF has two modes: forward and reverse. The forward mode is similar to the SW-PUF where it generates a unique signature by capturing the delay variations of carry propagation in ripple-carry adders (which is a basic component in an ALU). The delay variation is caused by instruction input values and the silicon fabrication foundry variations. The reverse mode computes the partial inputs from the signature and the instruction output. Early sampling captures a subset of original input bits correctly in a platform specific manner, which itself is a platform specific secret. Only the computing platform that originally computed the instruction can regenerate the inputs of the instruction accurate within a certain number of bits. In this design, Reversible SW-PUF is implemented in reversible logic. Fredkin gate [4] is used as the Boolean basis for conservative logic because it is universal.

Since Fredkin gates are based on transmutation gates, and TGs are slow compare to a regular gate, we propose to use two ALUs (fast-ALU, rev-ALU). The actual computation values consumed by the following program instructions occur at the fast-ALU. The rev-ALU is used only for verification.

3 Verifiable Computation Scheme

In this section an efficient Verifiable Computation Scheme based on Reversible SW-PUF is proposed. The proposed scheme fits with a probabilistic consistency guarantee. In this scheme, we are interested in estimating the probability of a cloud service to return a correct result for the outsourced function. The main idea is to bind the verification scheme to the cloud service hardware by entangling the computation with the SW-PUF. When the cloud computes the function, an instruction sequence for each instruction generates relevant attributes which are the two data inputs for the lth instruction - X_0^l, X_1^l, the instruction output Y^l, and the PUF output P_l. Effectively, the cloud node generates a signature (response) for each instruction (challenge) in the execution path. This entire sequence of challenge-response pairs will be returned to the client as a proof of computational consistency.

For the verification process, the client can verify the behavior of a program of variable granularities. Most straightforward granularity is to verify an individual instruction behavior. Pick a random challenge-response(C_k, R_k) pair of an instruction I_k to verify. The client needs to send the response part (the instruction output Y^k, and the PUF output P_k) to the cloud node. The cloud instantiates the reversible SW-PUF to re-compute the challenge from the response (the data inputs of the instruction $(X_0'^k, X_1'^k)$. Only the cloud node that computed the original signature will be able to compute the inverse PUF, so that $(X_0'^k, X_1'^k)$ is consistent with the (X_0^k, X_1^k) in the original computation's proof of consistency within a large number of bits. We assume that over all the clients and programs, the amount of data is too large to be archived by the cloud node preventing a look-up based response to the verification step.

Repeating this verification process for all the n instructions is not feasible because a large number of instructions could be executed during a program run. As we discuss later, an alternative approach to pick a subset of instructions is used to increase the confidence interval for the verification while maintaining an efficient verification.

Static program slices raise this granularity naturally. It is a technique for reducing a program to a minimal form that still retains the original program computation for a given variable at a chosen point. Merging the program slicing technique with our verification scheme leads to a more efficient Verifiable Computation. A program slice's input/output consistency can be established with the Reversible SW-PUF method. For a program slice, all of the instructions in its execution flow can be verified leading to a deterministic verification. The program slices can be extracted to maximize certain static properties.

Figure 1 describes an example of a client that wants to run the program on a cloud server using the proposed protocol:

For choosing the slice set in our scheme, two elements are critical: the size of the slice, and the number of slices. Since small slices result in more efficient verification, we propose to use a selection method based on the super-node [10] algorithm to reduce the verification effort. However, certain types of program control flow graphs may not be amenable to small slices, and in such a case,

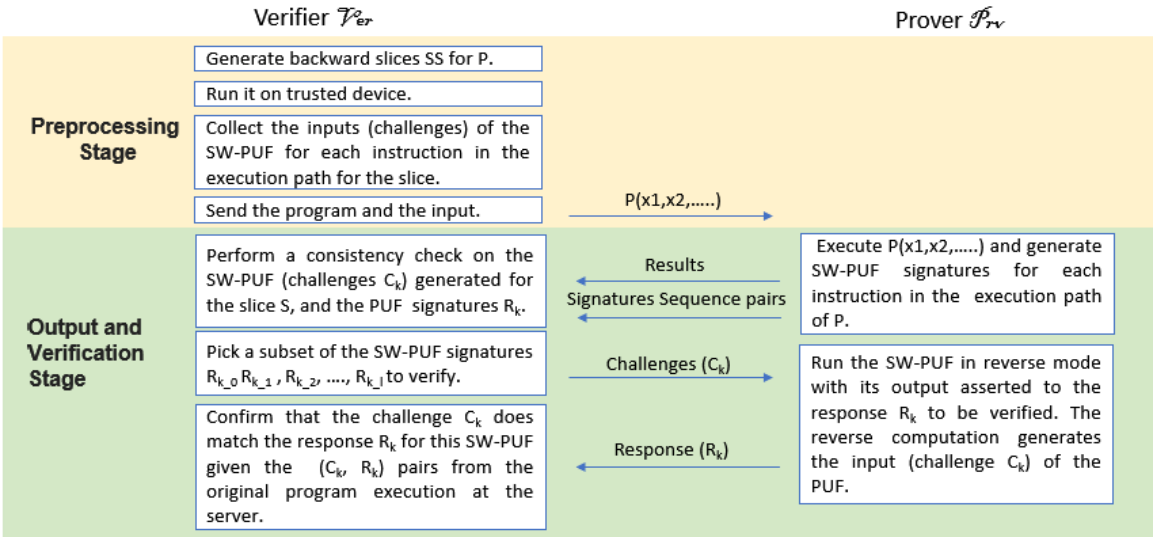

Fig. 1. The proposed protocol

different methods could be applied. As in any interactive proof system, increasing the number of slices will increase confidence in the computed results. To provide a desired probabilistic proof about the servers results, we propose to use a Bayesian Inference [3] model to determine the appropriate number of static slices required.

Slices Selection:

Given a program P that contains a set of instructions S, our goal is to find a subset of S called M such that M exhibits the same behavior as S with respect to one of the program outputs. Once we find M, while we want to generate static slices SS that go through M, the selection of M must be based in some randomized algorithm to prevent an adversary from producing the same M to cheat. For choosing M, we used the algorithm in [10], for selecting all the super-nodes in P as our set of desired nodes. A super-node is formed from a strict dominator-post-dominator pair. A node X is defined as a dominator to a node Y if every path from the start node to Y goes through X. Similarly, a node X is defined as a post-dominator to a node Y if all paths to the exit node of the graph starting at Y go through X. The super-node method will reduce the proposed verification scheme overhead. Verifying at least one instruction from each super-node block will be sufficient to verify the entire slice.

Probabilistic Verification Algorithms:

In this section, we propose use of a Bayesian inference on a binomial proportion method to verify the outsourced computation statistically. Bayesian inference is a statistical technique to update our subjective beliefs as new evidence or data becomes available. Our objective here is to characterize the probability density function for the outsourced computation correctness given that a set of

slices were run correctly. In particular, we are interested in estimating confidence in verifying the correctness of the calculation results returned by an untrusted cloud server. Bayesian computation of probability distributions starts with a prior belief about a model parameter, then updates this distribution based on observed data to produce new posterior beliefs. The mathematical definition of the Bayesian method is as follows:

$$p(H|D) = \frac{p(D|H) \times p(H)}{p(D)} \tag{1}$$

4 Evaluation of the Reversible SW-PUF

We evaluate 32-bit Reversible SW-PUF in HSPICE using predictive technology model. We studied three metrics: uniqueness, randomness, and reliability.

Uniqueness: Uniqueness measures the capability to distinguish between different devices. Hamming distances (HD) between PUF responses are used to measure uniqueness. An ideal HD between any two PUF responses is 50% (16-bit). To evaluate the uniqueness of the Reversible SW-PUF on the same ALU under different data inputs (Intra-chip), we measured the average HD distribution between a pair of output data on the same device PUF instance with different set of input data. The uniqueness of the forward signature for the Reversible SW-PUF has been measured the same way as the regular SW-PUF [6]. Figure 2 (A) shows the HD in Forward mode. For the reverse computation, both ALU inputs were measured on ten different PUF instances with identical output (response, which constitutes the input for a reversible PUF in reverse mode). The HD between each pair of different ALUs was calculated. Figure 2 (B) shows the HD in Reverse mode.

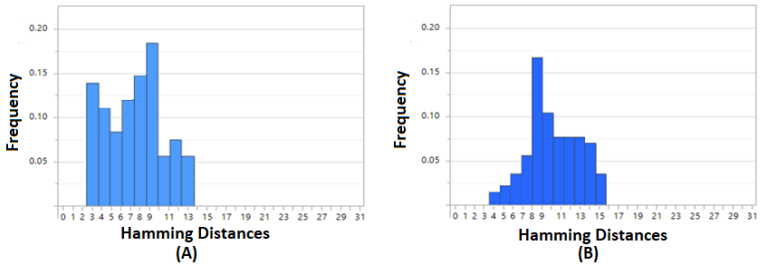

Fig. 2. Hamming distance distribution of reversible SW-PUF: (A) Forward mode; (B) Reverse mode

Randomness:

Randomness evaluates a PUF signature by analyzing the distribution of 0s and 1s. The standard statistical test suite of the National Institute of Standard

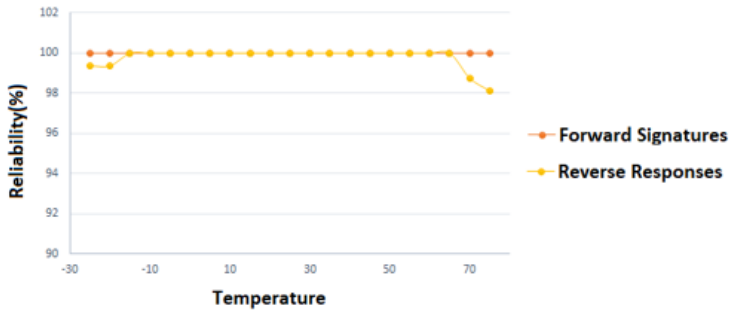

Fig. 3. The reliability of reversible SW-PUF against temperature variations.

and Technology (NIST) was used to evaluate the responses of the reversible SW-PUF. We have applied the NIST tests to 512-bit stream that was produced from the 16 PUF instances. Only two categories (rank and linear complexity), out of fifteen statistical tests, failed.

Reliability: Reliability measures robustness of a PUF in the presence of environmental variations. Temperature variations are the main factor that affect the stability of a PUF response. Figure 3 shows the reliability results for the responses of the Reversible SW-PUF for both the forward and backward computations. The reversible PUF is very stable under the temperature variation from -10°C to 65°C.

Case Study: Verification of Matrix Multiplication

We evaluate the proposed method thorough a matrix multiplication experiment, a widely-used example in Verifiable Computation Systems. We considered the following scenario: a client C needs to multiply two large scale matrices $A(n \times n)$and $B(n \times n)$ using a cloud service S. However, since the client C does not completely trust the cloud S to return the correct results for multiplication, the client C could verify the results in many ways. A naive algorithm could replicate the multiplication using another cloud service and compare the results, but this method is expensive, e.g., multiplying $n \times n$ matrices execute $O(n^3)$ time using the standard method. A faster check could use Freivalds' algorithm [5], a probabilistic randomized algorithm that verifies matrix multiplication in $O(kn^2)$ with a probability of failure less than 2^{-k}. Our approach improves Freivalds' algorithm by reducing the running time of the verification process by a factor of $O(n)$. Finally, we compare the execution time of our approach with the Verifiable Computation method proposed in [11].

Experimental Setup We implemented a C++ tool to generate the random slices and perform the verification, and a LLVM compiler framework to compile the matrix multiplication program into LLVM Immediate Representation (IR). We used the Symbiotic 3 tool [1] to obtain the backward static slice for the program. Symbiotic 3 linked with C++ code to generate the random slices in

which the slicing criterion was one element of the output matrix. The number of slices was chosen based on the Bayesian Inference model. For simplicity, we assumed that client C challenges must completely match the server signatures, and any failure will result in rejection of the verification. Finally, a Pin tool was used to generate the desired instruction traces, while HSPICE was used to represent the Reversible SW-PUF to generate the signatures.

Performance evaluation

We performed the experiments for evaluating our scheme and present the computation time cost for each of its elements. The resultant time cost was obtained by averaging the outcomes of testing 10 different randomly generated inputs of the matrix multiplication code for matrix sizes ranging from 1000 to 7000. Table 1 shows a computational cost comparison between the server S (i.e. Matrix Multiplication, Reverse computations) and the client C (i.e. Slices Generation "the number of slices was picked to produce a probability of more than 0.97", Signatures Verification) sides.

Table 1. Computation cost of proposed scheme for different problem size.

Dimension	Verification at Client Side		Computations at Server Side	
	Slices Generation	Signatures Verification	Matrix Multiplication	Reverse computations
n= 1000	10.025 ms	0.570 ms	0.201 s	0.008 s
n= 2000	11.504 ms	0.684 ms	2.129 s	0.078 s
n= 3000	12.753 ms	0.746 ms	6.372 s	0.183 s
n= 4000	15.025 ms	0.866 ms	12.479 s	0.366 s
n= 5000	16.875 ms	0.925 ms	20.692 s	0.675 s
n= 6000	17.752 ms	0.990 ms	29.668 s	1.065 s
n= 7000	20.057 ms	1.136 ms	44.050 s	1.523 s

We evaluate the advantage of our scheme by comparing our experiment with the PVCBMM scheme proposed in [11]. Both of the experiments are performed on the same computer properties. However, we used the Strassen's algorithm [7] to reduce the time required to multiply matrices. We studied seven dimensions size ranging from 1000 to 7000. As shown in Table 2, the experimental results reveal that our scheme is more efficient than the PVCBMM scheme.

5 Conclusions

We present reversible SW-PUF, a novel PUF design for computing partial inputs given a set of outputs. We implemented the reversible SW-PUF in HSPICE and established its desirable properties (uniqueness, randomness, and reliability). We then provided an efficient interactive verifiable computation scheme based on the proposed PUF and based on the Bayesian method. Our approach links outsourced computation with server cloud node hardware to provide proof of correctness of the results with high probability.

Table 2. Computation and Verification cost between two schemes.

Dimension	The proposed scheme		PVCBMM scheme [11]	
	Computations cost	Verification cost	Computations cost	Verification cost
n= 1000	0.201 s	0.018 s	1.75 s	6.94 s
n= 2000	2.12 s	0.090 s	4.36 s	14.86 s
n= 3000	6.37 s	0.19 s	8.35 s	32.26 s
n= 4000	12.47 s	0.38 s	24.62 s	61.37 s
n= 5000	20.69 s	0.69 s	36.31 s	85.03 s
n= 6000	29.66 s	1.08 s	65.16 s	178.54 s
n= 7000	44.05 s	1.54 s	105.28 s	193.86 s

References

1. Chalupa, M., Jonáš, M., Slaby, J., Strejček, J., and Vitovská, M. Symbiotic 3: New slicer and error-witness generation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (2016), Springer, pp. 946–949.
2. Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A.-R., and Stüble, C. A protocol for property-based attestation. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing* (New York, NY, USA, 2006), STC '06, ACM, pp. 7–16.
3. Dempster, A. P. A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological) 30*, 2 (1968), 205–232.
4. Fredkin, E., and Toffoli, T. Conservative logic. *International Journal of Theoretical Physics 21*, 3 (Apr 1982), 219–253.
5. Freivalds, R. Fast probabilistic algorithms. In *International Symposium on Mathematical Foundations of Computer Science* (1979), Springer, pp. 57–69.
6. Hamadeh, H., and Tyagi, A. Physical unclonable functions (pufs) entangled trusted computing base. In *2019 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS)* (2019), IEEE.
7. Huss-Lederman, S., Jacobson, E. M., Johnson, J. R., Tsao, A., and Turnbull, T. Implementation of strassen's algorithm for matrix multiplication. In *Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing* (1996), IEEE, pp. 32–32.
8. Parno, B., Howell, J., Gentry, C., and Raykova, M. Pinocchio: Nearly practical verifiable computation. *Commun. ACM 59*, 2 (Jan. 2016), 103–112.
9. Vu, V., Setty, S. T. V., Blumberg, A. J., and Walfish, M. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy* (2013), IEEE Computer Society, pp. 223–237.
10. Zhang, M., Gu, Z., Li, H., and Zheng, N. Wcet-aware control flow checking with super-nodes for resource-constrained embedded systems. *IEEE Access 6* (2018), 42394–42406.
11. Zhang, X., Jiang, T., Li, K.-C., Castiglione, A., and Chen, X. New publicly verifiable computation for batch matrix multiplication. *Information Sciences* (2017).