Modular Verification of Probabilistic Constant-Time

Xingyu Xie
xingyu.xie@mpi-sp.org
MPI-SP
Bochum, Germany

Abstract

While microarchitectural constant-time is a golden standard for cryptographic implementations and probability plays an essential role in cryptography, program analysis techniques for probabilistic constant-time are still under developed.

Our work aims to provide a call-by-contract-style modular verification technique for probabilistic constant-time, including code with public outputs, i.e., benign leakages. The technique is inspired by the correspondence between probabilistic constant-time and non-adaptive multiparty computation. Inspired by Theorem 5 in Canetti (2000), a modular logical verification approach emerges.

Keywords: Probabilistic Constant-Time, Modular Verification, Side-Channel Security, Program Verification

1 Introduction

Microarchitectural constant-time has become a golden discipline of cryptographic implementation [7, 9, 10]. While the techniques of enforcing constant-time have been developed over a long time, most of them still only consider deterministic programs. However, randomness plays a core role in cryptography — particularly, probability allows a rigorous way to discuss information and leakage of information [12]. Lifting the techniques around constant-time into probabilistic settings seems to be a necessary next step.

This work focuses on modular verification for probabilistic constant-time (PCT), so that the verification can be done layer by layer along the call hierarchy. Treating programs modularly requires an annotation for each function, which labels inputs and outputs as secret or public. Intuitively, PCT with respect to an annotation says that the microarchitectural leakage of the secrets is no more than public outputs. The difficulty of proving PCT modularly lies on the use of conditional distributions at the contract level, which falls outside of the existing probabilistic program verification frameworks such as pRHL [6] and Bluebell [5].

The solution we provide is based on an interesting observation between probabilistic constant-time and multiparty computation (MPC). The basic security definition of multiparty computation has the same form as probabilistic constant-time, in the sense that part of the inputs and outputs are allowed to be known by adversaries and intermediate values should not leak more secret information than these parts of input and output. With the help of this observation, we port a modular composition lemma by Ran Canetti [8, Theorem 5], which is widely used in MPC literature for building protocols and proving security modularly [3, 13, 14].

Contemporary MPC protocols are built in the following modular way: small protocols are proved to be secure with respect to "ideal functionalities", which specify the input-output relations; with these small protocols as building blocks, large protocols are defined by invoking small protocols, which are treated as ideal functionalities in security analysis; and so on for more complex protocols.

Inspired from the above, our approach is the following. First, we borrow the concept of "ideal function", which is a function semantically equivalent to the implementation, but leaks nothing. Then, when the PCT of a callee has already been proved, we can replace the function call by the ideal function and leaking its public inputs and public outputs, and continue analyzing the caller while being oblivious to the leakages in the callee. See Fig. 1 for an illustrating example.

The current technical contributions of this ongoing work are as follows. The main contribution is a modular verification approach for probabilistic constant-time and a proof sketch of its soundness (Thm. 3.1). Also, we provide fully fledged definitions for probabilistic constant-time, which allow specifying public outputs (benign leakages). Two equivalent definitions are provided: non-interference-based (Def. 2.1) and simulation-based (Def. 2.2).

2 Definition: Probabilistic Constant-Time

This section provides the notion of probabilistic constant-time (PCT). The notion is defined with respect to annotations: labeling which inputs and outputs are public or secret. We will denote the public parts of input/output as $\cdot_{\mathsf{pub}}$.

As preliminary of PCT, we take standard syntax and semantics for probabilistic imperative programs, and we explicitly mark constant-time leakages with a specific primitive `leak`$(\cdot)$ (See Fig. 1 as examples). The semantics of a probabilistic program is a function from a state to a distribution over states and leakages, denoted as $P(i) \Downarrow_O^L$, where P is a program, i is an input state, O is a random variable of output state, and L is a random variable of leakages (which is a list of booleans and addresses). We refer readers to [1, 4, 11] for detailed explanations.

Our definitions are as follows. The intuition is to say that the leakages are entirely determined by public inputs and public outputs. We first provide a non-interference-based definition, which is a direct lift of classical deterministic

```
// input: bound (public)
// output: r (secret)
int reject_sampling(int bound) {
  int r;
  bool b = false;
  leak(!b);
  while (!b) {
    r = $ [0..32767];
    b = r < bound;
    leak(!b);
  }
  return r;
}


// input: x (secret)
// output: y (public)
int otp(int x) {

  int r = reject_sampling(2048);
  int y = (x + r) mod 2048;
  return y;
}
```

```
// input: bound (public)
// output: r (secret)
int reject_sampling(int bound) {
  int r;
  bool b = false;
  leak(!b);
  while (!b) {
    r = $ [0..32767];
    b = r < bound;
    leak(!b);
  }
  return r;
}
```

```
// input: x (secret)
// output: y (public)
int otp(int x) {
    leak(2048);
    int r = $ [0..2047];
  int y = (x + r) mod 2048;
  return y;
}
```

Figure 1. One-time pad with a key generated by rejection sampling (which keeps drawing samples until hitting the range). This is a two-level program with a single function call. The left-hand side is the original verification task of PCT. The right-hand side shows how the task can be decomposed following the program structure: (1) probabilistic constant-time of the callee; (2) probabilistic constant-time of a transformed caller, in which the function call is substituted with, as highlighted, an ideal callee (directly sampling from the target range), and leakage of public arguments of the callee (and public results if there exists).

constant-time definition [2]. However, the modularity is not obvious from the perspective of this definition. Therefore, we present an equivalent simulation-based definition, which reveals modularity more naturally as function composition.

Definition 2.1 (Probabilistic constant-time, non-interference-based)**.** *A program P is* probabilistic constant-time *if for any two inputs i_1 and i_2 and two outputs o_1 and o_2 s.t. $i_{\mathsf{pub}_1} = i_{\mathsf{pub}_2}$ and $o_{\mathsf{pub}_1} = o_{\mathsf{pub}_2}$, then the two leakages are equally distributed conditioned on the outputs, i.e.,*

$$\forall l,\ \Pr[\, L_1 = l \mid O_1 = o_1 \,] = \Pr[\, L_2 = l \mid O_2 = o_2 \,],$$

where $P(i_1) \Downarrow_{O_1}^{L_1}$ and $P(i_2) \Downarrow_{O_2}^{L_2}$.

Definition 2.2 (Probabilistic constant-time, simulation-based)**.** *A program P is* probabilistic constant-time *if there is a probabilistic function, $\mathcal{S}$ (called "simulator"), s.t., for any input i and output o, $\mathcal{S}$ can simulate the leakage of P given public part of inputs and outputs, i.e.,*

$$\exists \mathcal{S}, \forall l\ o, \Pr\big[(\mathcal{S}(i_{\mathsf{pub}}, o_{\mathsf{pub}}), O) = (l, o)\big] = \Pr[(L, O) = (l, o)],$$

where $P(i) \Downarrow_O^L$.

In previous works [4], PCT has been formulated as *leakage-freeness*. Unfortunately, leakage-freeness considers all outputs only as secret, which misses important cases such as public key generators. Compared with leakage-freeness, our PCT notion takes full annotations into consideration: public/secret input/output.

3 Modular Composition

Now we are ready to formulate our modular verification technique as follows. See Fig. 1 for an illustrating example.

Theorem 3.1 (Modular Composition)**.** *Consider a program P including function calls in the form of $\mathtt{res}_k = Q_k(\mathtt{arg}_k)$. Let $(\!|\, P \,|\!)$ be the program in which every function call $\mathtt{res}_k = Q_k(\mathtt{arg}_k)$ is replaced by*

$$\mathtt{leak}(\mathtt{arg}_{\mathsf{pub}_k})\ ;\ \mathtt{res}_k = Q_k^{\mathsf{ideal}}(\mathtt{arg}_k)\ ;\ \mathtt{leak}(\mathtt{res}_{\mathsf{pub}_k}),$$

where Q_k^{ideal} is a program which is functionally equivalent with Q_k but has no leakage.

If all callees are PCT and $(\!|\, P \,|\!)$ is PCT, then P is PCT.

Proof Sketch. If any callee, Q_k, is PCT, then there is a simulator $\mathcal{S}_k$ from public inputs and public outputs to leakages in Q_k. If $(\!|\, P \,|\!)$ is PCT, then there is a simulator $\mathcal{S}$ from public inputs and public outputs to leakages in $(\!|\, P \,|\!)$. As the leakages of $(\!|\, P \,|\!)$ includes all public inputs and public outputs of any callee Q_k, we can compose $\mathcal{S}$ and all $\mathcal{S}_k$ to get a full simulator for the original program P. □

4 Conclusion and Future Works

We provide a call-by-contract-style modular verification approach for probabilistic constant-time allowing benign leakages. The natural next step is a thorough theoretical proof and comprehensive practical evaluation.

References

[1] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. 2016. Verifiable Side-Channel Security of Cryptographic Implementations: Constant-Time MEE-CBC. In *Revised Selected Papers of the 23rd International Conference on Fast Software Encryption - Volume 9783* (Bochum, Germany) *(FSE 2016)*. Springer-Verlag, Berlin, Heidelberg, 163–184. doi:10.1007/978-3-662-52993-5_9

[2] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. 2016. Verifying constant-time implementations. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 53–70.

[3] Gilad Asharov and Yehuda Lindell. 2017. A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. *J. Cryptol.* 30, 1 (Jan. 2017), 58–151. doi:10.1007/s00145-015-9214-4

[4] José Bacelar Almeida, Denis Firsov, Tiago Oliveira, and Dominique Unruh. 2025. Leakage-Free Probabilistic Jasmin Programs. In *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Denver, CO, USA) *(CPP '25)*. Association for Computing Machinery, New York, NY, USA, 3–16. doi:10.1145/3703595.3705871

[5] Jialu Bao, Emanuele D'Osualdo, and Azadeh Farzan. 2025. Bluebell: An Alliance of Relational Lifting and Independence for Probabilistic Reasoning. *Proc. ACM Program. Lang.* 9, POPL, Article 58 (Jan. 2025), 31 pages. doi:10.1145/3704894

[6] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. *SIGPLAN Not.* 44, 1 (Jan. 2009), 90–101. doi:10.1145/1594834.1480894

[7] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. 2013. McBits: Fast Constant-Time Code-Based Cryptography. In *CHES*. Springer, 250–272. doi:10.1007/978-3-642-40349-1_15

[8] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology* 13, 1 (Jan. 2000), 143–202. doi:10.1007/s001459910006

[9] Marcel Fourné, Daniel De Almeida Braga, Jan Jancar, Mohamed Sabt, Peter Schwabe, Gilles Barthe, Pierre-Alain Fouque, and Yasemin Acar. 2024. "These results must be false": A usability evaluation of constant-time analysis tools. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 6705–6722. https://www.usenix.org/conference/usenixsecurity24/presentation/fourne

[10] Antoine Geimer, Mathéo Vergnolle, Frédéric Recoules, Lesly-Ann Daniel, Sébastien Bardin, and Clémentine Maurice. 2023. A Systematic Evaluation of Automated Tools for Side-Channel Vulnerabilities Detection in Cryptographic Libraries. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1690–1704. doi:10.1145/3576915.3623112

[11] Joost-Pieter Katoen Gilles Barthe and Alexandra Silva (Eds.). 2020. *Foundations of Probabilistic Programming*. Cambridge University Press. https://doi.org/10.1017/9781108770750

[12] Oded Goldreich. 2006. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA.

[13] Xiaoyu Ji, Junru Li, and Yifan Song. 2024. Linear-Communication Asynchronous Complete Secret Sharing withnbsp;Optimal Resilience. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, CRYPTO 2024, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VIII* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 418–453. doi:10.1007/978-3-031-68397-8_13

[14] Yifan Song and Xiaxi Ye. 2025. Honest Majority MPC withnbsp;O (|C|) Communication innbsp;Minicrypt. In *Advances in Cryptology – EUROCRYPT 2025: 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4–8, 2025, Proceedings, Part V* (Madrid, Spain). Springer-Verlag, Berlin, Heidelberg, 453–483. doi:10.1007/978-3-031-91092-0_16