2020 IEEE Symposium on Security and Privacy

Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers

Chun Guo
Université Catholique de Louvain
chun.guo.sc@gmail.com

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

Xiao Wang
Northwestern University
wangxiao@cs.northwestern.edu

Yu Yu
Shanghai Jiao Tong University
yuyu@cs.sjtu.edu.cn

***Abstract*—Many implementations of secure computation use fixed-key AES (modeled as a random permutation); this results in substantial performance benefits due to existing hardware support for AES and the ability to avoid recomputing the AES key schedule. Surveying these implementations, however, we find that most utilize AES in a heuristic fashion; in the best case this leaves a gap in the security proof, but in many cases we show it allows for explicit attacks.**

Motivated by this unsatisfactory state of affairs, we initiate a comprehensive study of how to use fixed-key block ciphers for secure computation—in particular for *OT extension* and *circuit garbling*—efficiently and securely. Specifically:

- **We consider several notions of pseudorandomness for hash functions (e.g., correlation robustness), and show provably secure schemes for OT extension, garbling, and other applications based on hash functions satisfying these notions.**
- **We provide provably secure constructions, in the (non-programmable) random-permutation model, of hash functions satisfying the different notions of pseudorandomness we consider.**

Taken together, our results provide end-to-end security proofs for implementations of secure-computation protocols based on fixed-key block ciphers (modeled as random permutations). Perhaps surprisingly, at the same time our work also results in noticeable performance improvements over the state-of-the-art.

I. Introduction

Over the past few years, secure computation has transitioned from the realm of pure theory to the point where it is implemented in multiple software libraries (see [26] for a recent survey), funded by various government agencies, marketed by many startup companies, and used in several real-world applications [10, 9, 32, 27]. This makes it critical to understand the security provided by *implementations*[1] of secure-computation protocols. Indeed, although published protocols typically come with proofs of security that can be independently verified by the community, published protocol descriptions often omit or ignore important low-level details, and researchers often apply performance optimizations in a haphazard way when implementing those protocols.

In this work we study the use of fixed-key AES (or fixed-key block ciphers more generally) in implementations of secure computation. Using fixed-key AES in this context can be traced back to the work of Bellare et al. [5], who consider fixed-key AES for circuit garbling as part of their JustGarble framework. Prior to that point, most implementations of garbled circuits used a hash function such as SHA-256, modeled as a random oracle. Bellare et al. design several provably secure garbling schemes based on a fixed-key block cipher, and in doing so demonstrated significant performance improvements; in particular, they show that using fixed-key AES can be up to 50× faster than using a cryptographic hash function due to the hardware support for AES provided by modern processors. (While it is also possible to garble circuits using AES with different keys at each gate, the added overhead of key scheduling is substantial.) Prior to their work *CPU time* was the main bottleneck for protocols based on circuit garbling; after the introduction of JustGarble, *network throughput* became the dominant factor. For this reason, fixed-key AES has been almost universally adopted in subsequent implementations of garbled circuits.

Bellare et al. prove security of their constructions when the fixed-key block cipher is modeled as a random permutation. This *random-permutation model (RPM)* is analogous to the random-oracle model, and assumes that all parties have oracle access to a public, random permutation $\pi : \{0,1\}^k \to \{0,1\}^k$ and its inverse. Modeling a fixed-key block cipher as a random permutation is weaker than modeling the block cipher as an ideal cipher (which is also common); in particular, related-key attacks do not apply in the fixed-key setting. Inspired by the work of Bellare et al., many subsequent works on efficient secure computation have relied on fixed-key AES for the purposes of both garbling (e.g., [53]) and *oblivious-transfer (OT) extension* (e.g., [46]), two important building blocks for secure-computation protocols. Unfortunately, however, end-to-end proofs of security are often missing. For example, published OT-extension protocols may be based on a hash function H and are proven secure by modeling H as a random oracle. When the protocols are implemented, however, H is *not* instantiated using a cryptographic hash function but is instead instantiated haphazardly from a fixed-key block cipher. At best this leaves a gap in the security proofs, but at worst—as we show in Section II—it leaves the implemented protocols vulnerable to explicit attacks.

Even the work of Bellare et al. has the drawback that it is not *modular*. That is, they do not prove security of their garbling schemes based on some assumption about the "encryption scheme" used for each gate; instead, they prove security directly in the random-permutation model. This makes it difficult to apply their ideas to new garbling schemes that are developed, as any changes in the scheme require re-doing the entire proof. This was done, for example, in the analysis of the subsequent half-gates garbling scheme [53]. In their paper introducing the scheme, Zahur et al. follow a more modular approach: they construct their garbling scheme based on a hash function H and then prove that their scheme is

[1] We stress that we are *not* referring to general software-security issues such as improper input handling or buffer-overflow attacks (though these are also important). Rather, the focus of this work is on *cryptographic* issues that arise in the course of implementation.

DOI 10.1109/SP40000.2020.00016