Hindawi
Security and Communication Networks
Volume 2019, Article ID 1368905, 28 pages
https://doi.org/10.1155/2019/1368905

Review Article

Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities

Joseph I. Choi and Kevin R. B. Butler

Department of Computer and Information Science and Engineering, University of Florida, E301 CSE Building, P.O. Box 116120, Gainesville, FL 32611, USA

Correspondence should be addressed to Joseph I. Choi; choijoseph007@ufl.edu

Received 30 September 2018; Revised 12 February 2019; Accepted 4 March 2019; Published 2 April 2019

Academic Editor: Bela Genge

When two or more parties need to compute a common result while safeguarding their sensitive inputs, they use secure multiparty computation (SMC) techniques such as garbled circuits. The traditional enabler of SMC is cryptography, but the significant number of cryptographic operations required results in these techniques being impractical for most real-time, online computations. Trusted execution environments (TEEs) provide hardware-enforced isolation of code and data in use, making them promising candidates for making SMC more tractable. This paper revisits the history of improvements to SMC over the years and considers the possibility of coupling trusted hardware with SMC. This paper also addresses three open challenges: (1) defeating malicious adversaries, (2) mobile-friendly TEE-supported SMC, and (3) a more general coupling of trusted hardware and privacy-preserving computation.

1. Introduction

Secure multiparty computation (SMC) allows two or more parties to collectively perform some computation and receive the resulting output without ever exposing any party's sensitive input. Following its beginnings in the 1980s with Yao's work on two-party garbled circuit computation [1, 2], SMC techniques have rapidly improved in the past several decades, with costs lowered by orders of magnitude. For example, recent work [3] demonstrates garbled circuit evaluation at speeds of 1.15 billion gates/second, and secret sharing supported privacy-preserving location services were available at Real World Crypto 2015 [4].

Despite these significant gains, secure multiparty computation cannot yet be considered sufficiently practical for use in a majority of applications where (near) real-time performance is required. (It is certainly possible to support some subset of applications with the current iteration of SMC, but SMC in (near) real-time environmental constraints is not yet a widespread phenomenon.) This is especially true for techniques based on fully homomorphic encryption [5] or secret sharing [6, 7]. (Depending on the application, secret sharing may be preferable to GC, such as for matrix and tensor operations or operations that follow the map-reduce paradigm.) Partial homomorphic encryption, while offering better performance than its fully homomorphic counterpart, is limited in the operations it can support. All these techniques share a common limitation, this being the substantial amount of cryptographic operations that are required of the parties involved, even in the two-party setting, or the requirement for a secure communication channel. (The number of required cryptographic operations does decrease when three or more parties are involved. If secure communication channels were available, this would remove the need for cryptographic primitives.)

The rise of a number of trusted execution environments (TEEs) in recent years, most recently including Intel's Software Guard Extensions (SGX) [8], presents an opportunity to offload some of the costs of secure multiparty computation. Figure 1 demonstrates how TEEs can support SMC and, more generally, privacy-preserving computation. Where before a single physical computing node could only serve as one computational entity, TEEs effectively provide a way to create several computational entities for performing secure computation within a single physical computing node. (Secure computation is impossible if there exists only a single

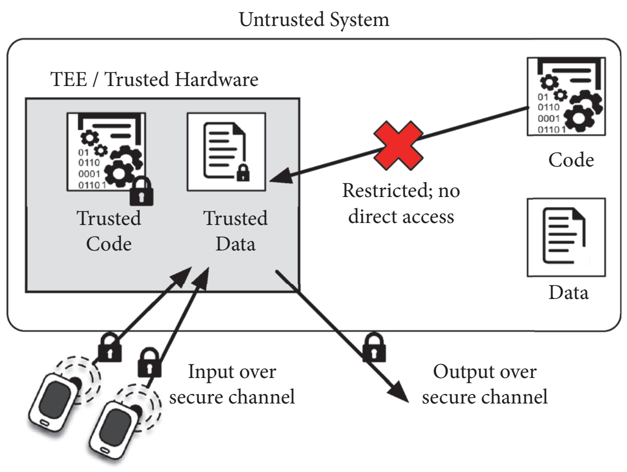
Untrusted System
TEE / Trusted Hardware
Trusted Code
Trusted Data
Restricted; no direct access
Code
Data
Input over secure channel
Output over secure channel

Figure 1: TEEs provide trusted containers for holding trusted code and data. Input is passed into the TEE over a secure channel and computation performed within the TEE, protected from the rest of the untrusted system.

computational entity, as no guarantees can be provided if the single entity is compromised.)

TEEs provide isolation guarantees for code and data during execution, suggesting they could be used interchangeably with cryptographic operations to safeguard secrets. For example, Intel SGX provides secure memory regions (or enclaves); code or data exists in unencrypted form when within the CPU's cache and registers, while it is encrypted when outside the processor boundary. This allows execution within TEEs to be much faster than execution tied to complex cryptography. TEEs also offer the ability to attest to contained code and data (making it possible to reason about their integrity) while also limiting potential attacks. (Ideally, an adversary can only monitor resource consumption.)

However, introducing TEEs to secure multiparty computation problems is not a simple matter of drag-and-drop. The trust model associated with TEEs is substantially different than with cryptographic approaches to SMC; TEE use requires acceptance of extra security assumptions and an altered trusted computing base (TCB), whereas security assumptions in cryptography are often much simpler. The security properties of each TEE should be carefully examined to determine whether they align with the requirements of the computation; certain TEEs will be better suited for specific applications. Care must also be taken to avoid unintentional leakage of secrets over side channels and when communicating between trusted and untrusted components.

In this paper, we survey the current SMC landscape, beginning with the foundations of SMC and tracking its evolution over the years. We follow this with an exploration of the potential for enhancing secure computation using trusted hardware. We make the following contributions:

(i) *SMC techniques/optimizations:* we conduct an extensive literature review of current SMC techniques and provide insight into inherent challenges of SMC solutions through a brief history of optimizations.

(ii) *Trusted execution:* we investigate what it means to ground trust in hardware, the security properties provided by TEEs, and the differences between popular TEE options. We also explore a variety of work that already leverages trusted hardware to get a better understanding of how TEEs are deployed in practice.

(iii) *Open challenges:* guided by our review, we identify several open challenges in integrating TEEs and secure multiparty computation techniques.

The remainder of the paper is organized as follows: Section 2 describes the fundamentals of and explores recent advances in secure multiparty computation; Section 3 considers grounding of trust in hardware and presents a number of security solutions that already make use of trusted hardware; Section 4 addresses several open challenges surrounding hardware-assisted secure computation and proposes steps forward for each; and Section 5 concludes.

2. Secure Multiparty Computation

Secure multiparty computation (SMC), also referred to as secure function evaluation (SFE), is a type of privacy-preserving computation where two or more parties collectively compute a function and receive its output without any party learning the other parties' private inputs. In our investigation, we keep the focus on cryptographic security, that nothing about the input is leaked beyond what can be inferred from the output, but there are stronger notions. (Certain computations (e.g., two-party arithmetic sum or the millionaires' problem) trivially leak information about their inputs.)

In cases where the output itself is sensitive (e.g., a key) or used as input in a continuing computation, *output privacy* [9] is needed. An even stronger notion is that of *differential privacy* [10, 11], which ensures the preservation of uncertainty in any party's input even when an adversary is given the entire transcript of the computation. There is a large corpus of work dealing with differential privacy in the context of SMC [12–17].

Another principal goal of SMC is correctness of the computed output. Other desirable properties of SMC include agreement on abort (honest parties will agree to abort upon detecting dishonest parties), fairness (either all parties receive the output or none do), graceful degradation (security beyond a threshold), and robustness or guaranteed output delivery (outputs not withheld from honest parties). Some of these properties are precursors to others; for example, Cohen et al. [18] show that a fair computation can be transformed into one providing guaranteed output delivery.

2.1. Security Model.

Adversaries attempting to violate the security guarantees of SMC may be classified as *semi-honest*, *malicious*, or *covert*. *Semi-honest* (or *honest-but-curious*) adversaries are interested in faithful execution of the SMC protocol to ensure proper, unmodified evaluation of the function but may otherwise act arbitrarily to reveal the secret input of cooperating parties. *Malicious* adversaries, on the other hand, deviate from the agreed protocol with the aim of manipulating the resulting output or learning parties' secrets. A third type of adversary is proposed by Aumann et al. [19]; these *covert* adversaries are like their *malicious* counterparts in that they are willing to cheat, but only to the extent that they are not caught (thus limiting their deviation).

SMC does not depend on a separate trusted third party. In the *semi-honest* setting, computing parties themselves are trusted to adhere to the protocol. Additional countermeasures are required to catch cheating in the *malicious* and *covert* settings, where computing parties are untrusted.

Adversaries may be either *static* or *adaptive*. *Static* adversaries must choose which parties to corrupt before computation begins, while *adaptive* adversaries are free to corrupt parties at any stage of the computation based on all available information. It is much more difficult to defend against *adaptive* adversaries. Additional factors determining adversarial success are the number of parties an adversary can corrupt and whether or not they collude. Protocols for SMC accordingly vary in the number of corruptions they can handle and in their requirement for a trusted honest majority; the adversarial setting dictates which techniques are better than others.

2.2. Yao's Garbled Circuits. SMC saw its beginnings in the 1980s, with Yao's seminal work [1, 2]. Yao's approach would later be termed "garbled circuits" to reflect the scrambling of circuit and inputs [20]. Yao's approach depends in part on oblivious transfer (OT) [21], which allows parties other than the circuit generator to learn the garbled representations of their inputs. (Only the generator knows the mapping of plaintext bits to their garbled representations.) In 1-out-of-2 oblivious transfer, the circuit generator supplies two alternative garbled representations (corresponding to either a zero or one bit). The nongenerator party receives exclusively one of these, which will be the garbled representation that matches its input bit. The generator does not learn which of the two were selected.

In the case of two parties, *Alice* (the circuit generator) and *Bob* (the circuit evaluator), Yao's garbled circuit protocol proceeds as follows:

(1) Alice generates a Boolean circuit version of some functionality f.

(2) Alice transforms the circuit into a garbled circuit by garbling the truth table for each gate in the circuit. Garbling comprises encrypting each entry of the truth table using randomly generated keys and randomizing the order of rows in the truth table.

(3) Alice sends the garbled circuit, together with her garbled inputs, to Bob.

(4) Bob, using 1-out-of-2 oblivious transfer, learns the garbled representation of his inputs.

(5) Bob evaluates the garbled circuit on both parties' inputs and outputs the result.

Figure 2 shows how each gate of the circuit is garbled or replaced by a table of encrypted values. During evaluation, Bob will decrypt one of the encrypted values at each gate,

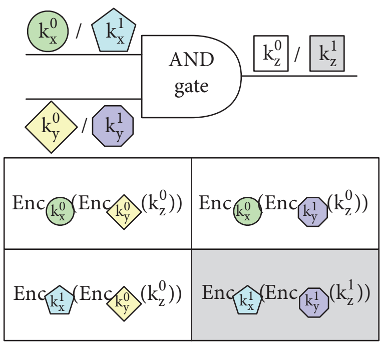
k_x^0 / k_x^1
AND gate
k_z^0 / k_z^1
k_y^0 / k_y^1
$\mathrm{Enc}_{k_x^0}(\mathrm{Enc}_{k_y^0}(k_z^0))$
$\mathrm{Enc}_{k_x^0}(\mathrm{Enc}_{k_y^1}(k_z^0))$
$\mathrm{Enc}_{k_x^1}(\mathrm{Enc}_{k_y^0}(k_z^0))$
$\mathrm{Enc}_{k_x^1}(\mathrm{Enc}_{k_y^1}(k_z^1))$

FIGURE 2: An example of Yao's garbled circuit evaluation of a single AND gate. Each gate's garbled representation is a table of encrypted values, one of which is unlocked by the values of the input wires (or keys). The gate outputs a key that will be used to unlock the output of the next gate, and this process repeats iteratively until arriving at the final output.

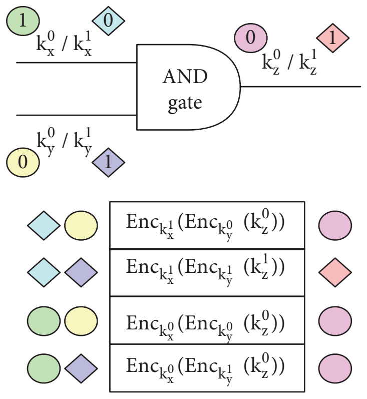
1
0
k_x^0 / k_x^1
AND gate
0
1
k_z^0 / k_z^1
k_y^0 / k_y^1
0
1
$\mathrm{Enc}_{k_x^1}(\mathrm{Enc}_{k_y^0}(k_z^0))$
$\mathrm{Enc}_{k_x^1}(\mathrm{Enc}_{k_y^1}(k_z^1))$
$\mathrm{Enc}_{k_x^0}(\mathrm{Enc}_{k_y^0}(k_z^0))$
$\mathrm{Enc}_{k_x^0}(\mathrm{Enc}_{k_y^1}(k_z^0))$

FIGURE 3: The point-and-permute optimization assigns an independent, special-purpose bit value to each wire label; alternate labels on each wire have opposite bit values. These values determine the order of the garbled gate's table entries. The output wire's labels are assigned, at random, a different special-purpose bit for indexing at the next gate.

depending on the values of the gate's input wires. This decrypted value is passed along as input to the next gate in the circuit, where it will be used to unlock the next value.

The original construction for garbled circuit evaluation is inefficient because the evaluator must try to decrypt all entries of a truth table. This can be alleviated by either padding outgoing keys with trailing zeroes or using the more elegant point-and-permute optimization [20]. Point-and-permute, a technique introduced by Beaver et al. in 1990, works by appending, at random, a special purpose bit to each wire label. As shown in Figure 3, the two labels on each wire have opposite bit values, and truth tables entries are indexed according to these values. During decryption, the evaluator selects the single correct ciphertext to decrypt based on these bit values. Only the generator knows the random association between these bits and the truth values (0 or 1) that the associated wire labels represent. Further, a hash function can replace a chosen plaintext attack (CPA)-secure encryption scheme for producing each gate's ciphertexts. Other important and now ubiquitous optimizations of the original construction are presented in Section 2.5.1.

The formal security properties of garbled circuits have been extensively reasoned about. Bellare et al. [22] establish *garbling schemes* as a first-class cryptographic primitive $\mathcal{G} = (\mathrm{Gb}, \mathrm{En}, \mathrm{De}, \mathrm{Ev}, \mathrm{ev})$, a five-tuple of a probabilistic garbling algorithm, an encoding function, a decoding function, an algorithm for evaluating the garbled function, and an algorithm for evaluating the original (nongarbled) function. Using $\mathcal{G}$, Bellare et al. are able to construct one garbling scheme that achieves privacy and a stronger scheme that simultaneously achieves privacy, obliviousness, and authenticity. These properties can be summarized as follows:

(i) *Privacy*: parties should not learn anything beyond that revealed by knowing just the final output (with respect to a side-information function Φ).

(ii) *Obliviousness*: nothing about the function or its input/output will be leaked to the evaluator when computing the garbled output (with respect to Φ).

(iii) *Authenticity*: a party with knowledge of the garbled function and garbled input should be unable to produce a different but valid garbled output.

The side-information function Φ captures the information expected to be revealed by the computation, which may include circuit size and topology or the original circuit itself.

Some applications require adaptive security, rather than selective security (where the adversary must choose both circuit and input at once). Adaptive security considers the setting where an adversary may choose its inputs to be garbled based on prior knowledge of the garbled circuit being computed. In follow-on work [23], Bellare et al. extend the definitions from their previous work [22] into adaptive ones. Bellare et al. further present both coarse-grained (which treats input as atomic) and fine-grained adaptive security (which allows an adversary to choose successive input bits based on the garbled representations of the preceding bits that it has learned so far). They show that adaptive security of garbled circuits supports one-time programs [24] and secure outsourcing [25]. A one-time program is some function that can only be evaluated on a single input, after which the program will fail to evaluate on any other input. Secure outsourcing refers to the delegation of work to untrusted helpers in a privacy-preserving and failure-evident manner.

Bellare et al. show that a one-time padding technique is sufficient to transform a statically secure garbling scheme into a coarse-grained adaptively secure one; this is further transformed into a fine-grained adaptively secure scheme, supporting one-time programs using their Output Masking and Secret Sharing transform. They also show that their scheme which gives obliviousness and authenticity [22] can be transformed to support outsourcing based on one-way functions. An alternative approach by Hemenway et al. [26] relies only on the existence of one-way functions to realize an adaptively-secure garbling scheme. Their construction encrypts a garbled circuit using a *somewhere equivocal encryption scheme* built from one-way functions. Jafargholi and Wichs [27] prove that Yao's construction itself is already adaptively secure (for NC1 (Nick's Class 1 (named after Nick Pippenger): the set of functions that can be represented by circuits of polynomial size and logarithmic depth) circuits), with security loss captured in the pebbling game (in a graph pebbling game, "pebbles" are placed on vertices of the graph, and "pebbling moves" allow transferring pebbles to other vertices; the goal is to determine whether a pebble can be assigned to each vertex; Hemenway et al. [26] instead assign pebbles to gates, with different colored pebbles representing different modes for creating garbled circuits and a goal of determining whether gates can be converted across different modes) of Hemenway et al. [26].

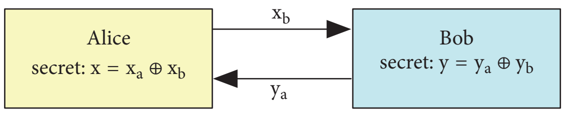

Figure 4: For two parties (Alice and Bob) to initiate GMW computation, they must first split their inputs into shares and each sends one of their shares to the other party. Alice gets all *a* shares; Bob gets all *b* shares.

2.3. The GMW Construction.

The primary alternative to Yao's garbled circuit protocol is given by Goldreich, Micali, and Wigderson [28]. Their GMW construction assumes a honest majority, meaning at least half of the participants are not corrupted. The GMW construction has the added benefit of being extensible to more than two parties. The parties perform coordinated evaluation of each gate, some of which may be evaluated separately on the held shares while others (e.g., AND gates) require OT to establish new shares during evaluation.

Like Yao's protocol, the GMW scheme depends on oblivious transfer and represents the function being computed as a Boolean circuit. However, OT occurs during the entire computation, making GMW particularly costly when communication latency is high or for circuits of greater depth. In the two-party setting, the participants interactively compute a function using shared values (shared by a 2-out-of-2 secret sharing scheme), which correspond to each of the inputs and intermediate wires. A high-level view of the setup is shown in Figure 4. To split each of the i bits of input $x = x_1, x_2, \ldots, x_n$, a random bit a_i is sampled from $\{0, 1\}$ to become the first share. The second share is $b_i = x_i \oplus a_i$.

XOR gates are evaluated by each party simply XORing the two shares it has, since XOR is an associative operation (i.e., $(a \oplus b) \oplus (c \oplus d) = (a \oplus c) \oplus (b \oplus d)$). AND gates are evaluated using either 1-out-of-4 OT alone or oblivious transfer coupled with pregenerated multiplication triples (for reduced communication complexity). One party randomly samples a new bit from $\{0, 1\}$ (its share) and XORs this value with each possibility of the AND operation, resulting in four values. The other party receives one of these four values to be its own share. NOT gates are evaluated by each party negating its share of the value to be negated.

2.4. Alternatives to Garbled Circuits. The principal contenders to garbled circuits are homomorphic encryption (HE) and secret sharing. Homomorphic encryption allows computation on ciphertexts without ever requiring that they be decrypted in the process. These schemes can be grouped according to supported operations:

(i) Partially Homomorphic Encryption (PHE) schemes support a single type of operation, which may be either addition or multiplication.

(ii) Somewhat Homomorphic Encryption (SWHE) schemes support multiple types of operations but only a limited number of them.

(iii) Fully Homomorphic Encryption (FHE) schemes support the evaluation of arbitrary functions with any number of operations.

FHE, originally termed privacy homomorphism by Rivest et al. [29], is the most difficult to realize and was first instantiated by Gentry [5] in 2009. Gentry devises a scheme based on the intuition that an encryption scheme which can evaluate its own decryption circuit (or *bootstrappable* scheme) can also evaluate arbitrary circuits. Gentry builds this scheme using ideal lattices, which provides helpful properties that include low circuit complexity of associated decryption algorithms and additive/multiplicative homomorphisms. To reduce the complexity of the decryption circuit and make the scheme *bootstrappable*, Gentry has the encrypter perform unkeyed preprocessing of the ciphertext, reducing the burden placed on the decrypter. While the practicality of homomorphic encryption has much improved since Gentry's 2009 scheme [5], HE solutions remain many orders of magnitude slower than computation on plaintext.

Secret sharing schemes split and distribute secrets among two or more parties. Together, these shares can be used to reconstruct the original secret. A single share is insufficient for that purpose, although individual shares may expose partial information about the secret. Threshold schemes permit any group of no fewer than k participants (k being the threshold value) to compute the secret. Where threshold schemes are insufficient, secret sharing for general access structures [30, 31] gives added control over what specific subsets (or access structures) of participants are qualified to reconstruct the secret. Secret sharing schemes may be proactive [32], meaning shares are renewed (without the secrets themselves changing) to periodically reset adversarial progress at attacking secrets. Secret sharing schemes may also be characterized as verifiable [33, 34], meaning it is possible to assert that a received share of the secret is valid even without any knowledge of the secret.

In secret sharing schemes, the sharing parties are typically called clients, while the trusted parties that receive shares from clients are called servers. Servers communicate with one another to perform secure computation over the shares, releasing output to the clients once done. A popular secret sharing scheme proposed by Shamir [6] in 1979 uses points on a polynomial as the shares and the constant term as the secret. The secret is recovered by solving a system of linear equations. Shamir's scheme is also an *ideal* secret sharing scheme, since it both prevents nonqualified subsets of participants from learning information about the secret and minimizes the length of shares in relation to the secret's length. Another popular secret sharing scheme proposed by Blakley [7] places the secret at the intersection of hyperplanes. Follow-up work has investigated both fault-tolerant [35] and cheater-tolerant [36] schemes.

Secret sharing may be the stronger contender to GC-based MPC than homomorphic encryption. Shamir secret sharing, in particular, provides a natural way to implement addition and multiplication operations, achieving Turing-completeness [37]. Homomorphic encryption, on the other hand, is best used for creating multiplication triples or for randomized preprocessing in the SPDZ [38] protocol.

Yet another alternative for secure multiparty computation is RAM-based secure computation [39, 40]. This approach builds on Oblivious RAM (ORAM) [41], a technique for concealing access patterns to data on a remote storage by continuous shuffling and reencrypting data as it is accessed.

2.5. A Brief History of Improvements. Secure multiparty computation techniques have much improved in the past several decades, bringing down costs by many orders of magnitude. Improvements have been made in each of the semi-honest, covert, and malicious adversarial models. Key innovations are explored below, with an emphasis on improvements to garbled circuits, as GC-based techniques generally outperform their secret sharing and HE-based counterparts.

2.5.1. Ubiquitous Optimizations. Besides the point-and-permute optimization described earlier in Section 2.2 that allows more efficient evaluation of gates, there are other important optimizations that are now standard in garbled circuit construction. These include free XOR (and flexible-XOR), row reduction, and cut-and-choose.

Kolesnikov and Schneider provide a new GC construction for two-party SFE, a one-round protocol that evaluates XOR gates for free [42]. Free XOR gives a 4x improvement to permutation network and universal circuit (UC) computation and a 2x improvement to integer addition and equality testing. This free XOR optimization is now a staple of the GC approach to SFE. Kolesnikov et al. improve on free XOR with flexible-XOR, or FleXOR [43], which weakens the Random Oracle requirement of free-XOR in favor of the correlation-robustness assumption [44]. FleXOR allows garbling of XOR gates with at most two ciphertexts; the associated savings in GC size are over 30% when compared to free XOR.

Naor et al. [45] introduced garbled row-reduction in 1999. One of the table entries (the first entry, set following point-and-permute) is chosen as the all-zero string, thus reducing the communication complexity by one ciphertext for each gate, as shown in Figure 5. Garbled row-reduction has since become yet another staple of following garbled circuits work. Pinkas et al. [46] took this one step further in 2009, further reducing the number of ciphertexts to two per gate using polynomial interpolation over a quadratic curve. The two values held by the garbled gate are used together with the input labels to construct a quadratic polynomial.

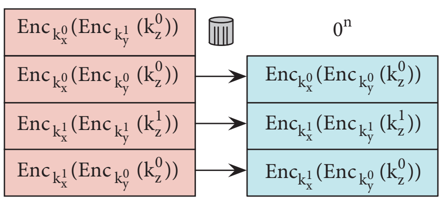
$Enc_{k_x^0}(Enc_{k_y^1}(k_z^0))$
0^n
$Enc_{k_x^0}(Enc_{k_y^0}(k_z^0))$
$Enc_{k_x^0}(Enc_{k_y^0}(k_z^0))$
$Enc_{k_x^1}(Enc_{k_y^1}(k_z^1))$
$Enc_{k_x^1}(Enc_{k_y^1}(k_z^1))$
$Enc_{k_x^1}(Enc_{k_y^0}(k_z^0))$
$Enc_{k_x^1}(Enc_{k_y^0}(k_z^0))$

FIGURE 5: In garbled row-reduction, the number of entries at each garbled gate is reduced by replacing the first entry with the all zero (0^n) string.

The output of the gate is the y-intercept of the resulting quadratic curve. Pinkas et al. analyze this and their other algorithmic improvements to Yao's protocol against various (including malicious) adversaries and show that secure two party computation is practical for reasonably large circuits (including that for AES encryption). (Pinkas et al.'s row-reduction is incompatible with free-XOR.)

The cut-and-choose technique allows Yao's protocol to be lifted into the malicious adversarial setting. When there are two parties, Alice and Bob, Alice first constructs some m copies of the circuit which are structurally identical but have different garblings. Alice also provides commitments (may be simple hashes of the input or more complex functions) of her input for each of these circuit copies. Bob, upon receiving these circuit copies and associated input commitments, selects $m - 1$ copies to verify. Alice de-garbles the chosen copies to assure Bob that all circuits were constructed correctly for performing the expected computation. If verification succeeds, Alice sends Bob her garbled input for the one remaining, unverified (now primary) circuit. Bob checks this against the earlier received commitment to ensure Alice did not alter her input. If this check succeeds, Alice and Bob continue with Yao's protocol.

Some of the earliest works on cut-and-choose date back to 2006-2007 [47–50]. Of these, Lindell and Pinkas [49] were the first to present a rigorous proof of security for a cut-and-choose backed protocol. Lindell and Pinkas [51] proposed a simpler and more efficient cut-and-choose in 2011, based on the decisional Diffie-Hellman (DDH) assumption. Through a concrete analysis of efficiency, Lindell and Pinkas show that their technique is more efficient than prior work for not-very-small circuits (Lindell and Pinkas do not explicitly define "not-very-small"; the overhead of their technique will be more pronounced if the communication cost of sending all the circuit generator's garbled inputs in the check circuits dominates overall circuit evaluation time) while also outperforming alternatives to cut-and-choose, including the committed input [52], LEGO [53], and virtual multiparty [54] methods. Lindell again revisits cut-and-choose to present a revised version [55] that vastly improves on previous protocols in the covert or malicious adversarial settings by permitting cheating only when all (not just a majority) of the evaluated circuits are found to be incorrect. In particular, Lindell demonstrates that, for a cheating probability of at most 2^{-40}, sending 40 circuits will suffice (where the best prior works sent between 125 and 128 circuits). Lindell's technique asymptotically achieves a cheating probability of 2^{-s}, where s denotes the number of garbled circuits.

Huang et al. extend the cut-and-choose optimization with symmetric cut-and-choose [56]. Instead of having one party (Alice in our previous illustration) generate K garbled circuits, symmetric cut-and-choose has both parties (Alice and Bob) generate K circuits to be checked by the other party. In doing so, the same statistical security level can be attained (security level here means the probability that a malicious party can successfully cheat; previous work in cut-and-choose achieved a security level of $2^{-0.32K}$, while Huang et al. [56] achieve $2^{-K+O(\log K)}$) while reducing K by a factor of 3. Because the number of garbled circuits generated for cut-and-choose in malicious settings dominates cost in a GC protocol, symmetric cut-and-choose is expected to give 3x speedup.

Huang, Katz, and Evans [57] propose an alternative means to give stronger security guarantees than semi-honest protocols, building on the dual-execution protocol of Mohassel and Franklin [48]. A dual-execution protocol comprises two independent runs of a semi-honest GC protocol with generator/evaluator roles swapped in the two runs. The outputs are then compared to verify they are identical; in so doing, only a single bit (comparison result) is leaked to a malicious adversary. Huang et al.'s scheme introduces minimal extra cost by relaxing the security properties and allowing leakage of a single bit of extra information.

2.5.2. Compilers for Garbled Circuits. Fairplay [58], in 2004, became the first compiler for generic secure function evaluation and paved the way for practical, deployable SFE. It provides a high-level procedural definition language in Secure Function Definition Language (SFDL) and compiles SFDL into a one-pass Boolean circuit presented in Secure Hardware Definition Language (SHDL). Fairplay also provides Alice/Bob programs that evaluate Yao's garbled circuits. We lay out below the development of other compilers for garbled circuits since the release of Fairplay.

Building on Fairplay, Faerieplay [59] considers the scenario where Alice houses a computation in a strong tamper-protected secure coprocessor at Bob's site. Faerieplay seeks to make this coprocessor as small as possible, with limited RAM and CPU. Faerieplay further augments Yao's blinded circuits with tiny trusted third parties (TTTPs) to reduce overhead by severalfold compared to ORAM and Fairplay.

Huang et al. improve garbled circuit efficiency by (1) not generating/storing the entire garbled circuit at once and (2) providing a programming framework for generating a secure computing protocol from an existing (insecure) implementation of the desired function [60]. They achieve computation rates of $10\,\mu s$ per garbled gate on circuits with over 10^9 gates and show that Yao's garbled-circuit technique can outperform special-purpose protocols for several functions: Hamming distance, Levenshtein distance, Smith-Waterman genome alignment, and AES encryption.

Holzer et al. show how to perform two-party secure computation using ANSI C [61]. They combine model checking techniques with garbled circuits, making nonstandard use of bit-precise model checker CBMC for converting C programs into Boolean circuits. To produce efficient circuits, CBMC attempts to minimize the size of Boolean formulas.

Kamara et al. propose Salus [62] for server-aided, GC-based SFE. Salus provides a new and efficient input-checking technique for cut-and-choose as well as a new pipelining technique that works even in the malicious adversarial setting. Salus is demonstrated to be 4-6x faster than most other preceding, optimized two-party SFE techniques.

Frigate [63], released in 2014, is a principled compiler and fast circuit interpreter for GC-based SMC. The development of Frigate was motivated by the observation that many artifacts for SMC are incomplete, incorrect, or unstable. Frigate is built using the best-practices for compiler design/development and, through extensive validation checks (performed on internal compiler state, operator combinations, and edge cases), is found to build correct circuits. Frigate gives compile time speedups as high as 447x over prior work and comparable gate counts, making it (arguably) the state-of-the-art for garbled circuit compilation/evaluation. In addition to its efficiency, Frigate is also extensible. For example, DUPLO [64] modifies Frigate with dedicated wire pools for each function, allowing the circuit representation for each function to be generated independently from the circuit representation of other functions. The dependency removal allows each distinct function of the input program to easily be garbled separately and soldered back together afterwards.

2.5.3. Platforms/Libraries for Multiparty Computation. While Section 2.5.2 above specifically explores advances to the state of garbled circuit compilers, we also explore the rise of more general platforms and libraries for multiparty computation. These may be more fitting than garbled circuits for SMC on algorithms that use variables and branching instructions which would add pure, unnecessary overhead as a result of unused or dead portions of circuits. The platforms/libraries presented below support homomorphic encryption and secret sharing, with some even supporting mixed-protocol combinations that include garbled circuits.

Sharemind [65], introduced by Bogdanov et al. in 2008, is a virtual machine for privacy-preserving data processing. Sharemind comprises a computation runtime environment and a programming library. Sharemind does additive secret sharing over the ring $\mathbb{Z}_{2^{32}}$, which Bogdanov et al. assert is optimal as it aligns with integer arithmetic on modern computers; share multiplication is also implemented in the same ring, simplifying many share computing protocols in the process. Sharemind protocols are information-theoretically secure in a honest-but-curious setting with three participants.

TASTY [66] is a tool for automating two-party secure computation protocols, allowing generation of protocols based on homomorphic encryption, garbled circuits, and combinations of both. TASTY takes a user-provided high-level description of the desired computation in a domain-specific language and transforms the description into protocol. TASTY can also efficiently evaluate circuits generated by Fairplay. The most recent techniques and optimizations are worked into the resulting low-latency protocols.

Malka et al. provide the VMCrypt library [67], which boasts scalability and modularity, and successfully demonstrate computations on the order of hundreds of millions of gates. The library gives novel algorithms with very small memory requirements and no disk usage. VMCrypt is paired with an API supporting integration into existing projects and customization without modifying of source code.

The Secure Computation API (SCAPI) [68] developed by Ejgenberg et al. emphasizes flexibility, extensibility, efficiency, and ease of use. SCAPI is meant to provide a general and modular library for implementing secure computation that is not limited to particular tasks and can be readily understood or incorporated. The SCAPI library is divided into three layers which provide low-level primitives, noninteractive schemes, and basic protocols for secure computation. Higher-layer functionality may be used directly or constructed from lower-layer building blocks as needed. SCAPI also provides a built-in communication layer.

Bogdanov et al., in 2014, proposed a new programming language SecreC [69], an imperative language which allows programming in a SMC-technique-agnostic manner. Types of data items are annotated with protection domains (which inform the required level of protection and determine the appropriate algorithms and protocols for computing on that data); function declarations may be domain-polymorphic.

ABY [70], developed by Demmler et al., is a mixed-protocol framework that combines arithmetic sharing, Boolean sharing, and garbled circuits. In arithmetic sharing schemes, values are treated as sums of two addends (shares), whereas in Boolean sharing schemes, shares are individual bits that can be XORed to produce the original bit. The goal of a mixed-protocol is to improve overall performance by allowing each part of the computation to be implemented according to its most efficient representation. ABY works by precomputing almost all crypto operations and provides efficient conversions between secure computation schemes based on precomputed OT extensions. Mixed-protocols are demonstrated useful for private set intersection, biometric matching, and modular exponentiation.

ObliVM [71] is a programming framework for building efficient, oblivious representations of programs, the observable execution traces of which do not reveal any information about the value of their secret inputs (a nonoblivious algorithm may leak secrets regardless of the security of the underlying SMC protocol). Similarly to SCAPI, ObliVM emphasizes extensibility, efficiency, and intuitiveness to nonexperts. Programs written in the oblivious programming abstractions of ObliVM's domain-specific language (ObliVM-lang) are translated into efficient oblivious algorithms that outperform generic ORAM when applicable. These are, in turn, converted into a sequence of circuits. ObliVM can be extended beyond garbled circuits (e.g., by being customized to use additively-homomorphic encryption in the backend).

Rmind [72], built atop the Sharemind framework, provides a privacy-preserving environment for statistical analysis of data from multiple sources. Its performance is comparable to that of a standard GNU R environment, which is one of the standard choices for statistics. Rmind uses secret sharing with three noncolluding servers that are provisioned with the same algorithm(s). Servers should be deployed by independent entities without conflicting interests in the computation or the data on which it relies.

HyCC [73] is a tool-chain for fully automated compilation of ANSI C programs into hybrid protocols that can even outperform certain hand-optimized protocols. Decomposed source code is translated into Boolean and arithmetic circuits, which are optimized before being deployed with a combination of Yao-GC, GMW, and arithmetic circuit protocols. Büscher et al. identify homomorphic encryption and integration of TEEs such as SGX as a next step.

Zhu et al. [74] develop NANOPI, a toolchain for semi-automating the development and deployment of extreme-scale actively-secure MPC applications. The underlying protocol which they build is both time- and space-efficient, resolving the space-round dilemma that previously prevented efficient, large-scale computation. They run their protocol on an actively-secure four-party logistical regression (involving 4.7 billion AND and 8.9 billion XOR operations), which completed in less than 28 hours on their testbed comprising four small-memory machines.

The SCALE-MAMBA [75] system builds on SPDZ [38], the semi-homomorphic encryption scheme introduced by Bendlin et al. [76], and an OT-based approach employing OT extensions in the random oracle model referred to as TinyOT [77]. It provides a runtime system SCALE and a Python-like programming language MAMBA. SCALE consists of an offline phase, an online phase, and a compiler, with offline and online phases fully integrated. The compiler takes programs written in the MAMBA language and transforms it into bytecode accepted by SCALE.

For an in-depth investigation of the characteristics/properties of available general purpose compilers for secure multiparty computation (including GC compilers), we refer readers to the work of Hastings et al. [78].

2.5.4. Efficiency Gains. Kruger et al. perform SFE using ordered binary decision diagrams (OBDDs), which are graph-based representations of functions [79]. OBDDs are more succinct than gate-based representations for many Boolean functions. Using OBDDs, Kruger et al. achieve smaller bandwidth compared to Fairplay (e.g., 45% reduction for the millionaire's problem), making their approach useful for constrained environments such as wireless and sensor networks. Despite the advantages of OBDD over gate representation for certain classes of functions (as well as further room for optimization using restriction, negated edges, and specialized OBDD variations), there has been little work on OBBDs since that of Kruger et al. [79].

The Portable Circuit Format (PCF) of Kreuter et al. [80] is intended to make circuit compilation more efficient than in prior work by relying online circuit compression and lazy gate generation [81]. They demonstrate support for a RSA-1024 signature function with over 42 billion gates and identify the underlying crypto primitives as the bottleneck. In addition to scaling beyond previous systems, PCF also decreases resource use requirements.

Whereas much work investigates the efficiency of Yao's construction, optimizations for efficient Goldreich-Micali-Wigderson (GMW)-based computation are given by Schneider et al. [82], who claim that GMW can outperform most implementations of Yao's GC in the semi-honest adversarial setting (given low network latency). They also propose depth-optimized circuit constructions for overcoming latency issues and demonstrate a 100x improvement of facial recognition functionality even when network latency is high. Hazay et al. [83] take protocols in the dishonest majority setting and modify them to use short symmetric keys. Security is based on the concatenation of all honest parties' keys; this technique gains efficiency at the cost of tolerating fewer corruptions, with a 13x reduction in the case of 120 out of 400 parties being honest, compared to the previously fastest protocol given by Dessouky et al. [84].

Shelat and Chen [85] set out to build a Yao-based protocol for secure two-party computation that requires minimal hardness assumptions (an OT protocol secure against malicious adversaries). Given this assumption, their protocol requires constant rounds of communication plus OT rounds, which is preferable to the rounds linear in circuit depth required by alternative approaches such as that of Nielsen et al. [86]. Additionally, Shelat and Chen's protocol has optimal overhead complexity and is highly parallelizable, such that each circuit can be pipelined and all circuits processed in parallel. An auxiliary circuit computes a hash of the generator's input to ensure input consistency; a k-probe-resistant matrix advances Pinkas and Lindell's state-of-the-art approach for handling selective failure attacks. The result is an approach that requires weaker crypto assumptions while giving a several factor improvement in performance.

Zahur et al. [87] devise a scheme that supports both the Free-XOR [42] optimization and the two-ciphertext row reduction of Pinkas [46], where the two were previously incompatible. They achieve this based on the intuition that an AND gate can be broken into two half-gates, each of which can be garbled with a single ciphertext. The resulting circuits are smaller than those produced by prior schemes, though the evaluator must now perform an additional cryptographic operation for each AND gate.

SPDZ [38, 88] is a protocol developed by Damgård et al. which operates in two phases: offline preprocessing in which shared randomness is generated without needing to know the function to be computed or its inputs; and the actual secure computation done in an online phase. Performance of the online phase scales linearly with the number of parties, and operations are as cheap as those used in passively secure Shamir secret sharing-based protocols.

Many recent works push for SMC with minimal rounds or round-optimal SMC. Round complexity, or the number of rounds of communication that occur between computing parties, is a determinant of the efficiency of SMC protocols, with less rounds being preferable. Garg et al. [89] build two-round MPC protocols from the minimal assumption of the existence of two-round OT. More generally, Benhamouda et al. [90] show that k-round MPC can be constructed from k-round OT. Garg et al. [91] show it is possible to build two-round MPC protocols that require a fixed number of OT calls, complemented by a polynomial number of cheaper one-way function calls, thus limiting the number of expensive public key operations. Ananth et al. [92] build a two-round MPC protocol in the honest majority setting that can provide security with abort (an adversary may not prevent honest parties from learning the output of a computation by aborting the protocol). They also build a two-round MPC protocol that provides guaranteed output delivery. Halevi et al. [93] design a four-round actively secure MPC protocol specifically for multiparty coin-tossing.

Ishai et al. [94] instead investigate message complexity of MPC protocols, which represents the number of point-to-point messages communicated between parties. Minimizing message complexity is desirable when the cost of exchanging messages is high. Ishai et al. find that, for functionalities taking inputs from n parties and delivering outputs to k parties, $2n + k - 3$ messages are necessary and sufficient.

2.5.5. Reusable GC. Part of the difficulty of garbled circuits is the inability to save state or to take the encrypted output wires of one computation, save them, and reuse them as input wires to another computation. PartialGC [95] addresses this need as the first implemented technique to allow reuse of encrypted values generated during GC computation. This avoids repeating of expensive processing when similar computations are done again, and PartialGC achieves up to 96% reduction of computing time and 98% reduction of bandwidth. (Computing time reduction occurs only when input checks are the bottleneck. Mood et al. [95] therefore suggest PartialGC works best with a large number of inputs and a very small circuit. For some programs (e.g., edit distance), overall bandwidth requirements actually increase, dominated by the extra bandwidth required for outsourced OT of all circuits.)

Goldwasser et al. [96] take a different approach to reusable garbled circuits, building a reusable scheme that supports any polynomial number of inputs from a new building block: a *succinct single-key functional encryption scheme*. By applying their functional encryption scheme to a garbled circuit, it is possible to generate many different encodings of the same garbled circuit, each taking a different input, without exposing the underlying circuit and any inputs. Agrawal et al. [97] further improve on Goldwasser et al.'s approach by constructing a new functional encryption scheme for circuits that gives stronger security guarantees.

2.5.6. Malicious Adversaries. Kreuter et al. [81] explore the feasibility of two-party SFE in a malicious adversarial setting (previous techniques only supported 35-thousand gate AES). By incorporating best known techniques and parallelizing almost all steps of the protocol, they are able to evaluate even billion-gate circuits in the malicious setting.

Protocols for SMC among n parties traditionally communicate at least a linear (in n) number of bits, but recent work has investigated sublinear (in n) communication complexity. It is possible to construct sublinear SMC protocols that tolerate some number of semi-honest adversaries (provided an honest majority) in the two-client-n-server setting (two parties who wish to compute a function of their inputs have a choice of n servers among which to distribute shares for processing), extensible to a multi-client setting with a constant multiplicative increase in complexity [98]. In the malicious adversarial setting, a sublinear protocol is achievable at the cost of allowing the protocol to abort. Each client will abort during the OT precomputation phase if OT pairs are received from more than a (sublinear) number of parties, as this would only occur if an adversary employs more corrupted parties than honest volunteers [98]. OT pairs received from the various servers can be fed into an OT-combiner to achieve secure OT [99].

Other recent work breaks the 1-billion gates/second barrier in the presence of malicious adversaries (specifically considering the case of three parties and an honest majority), making it one of the most efficient to-date [3]. Araki et al. do this by improving the efficiency of cut-and-choose on multiplication triples, and they build on work by Furukawa et al. [100], reducing 10-bit bandwidth per AND gate to 7 bits and designing cache-efficient shuffling techniques to cut-and-choose without needing to randomly permute large arrays. Despite surpassing competing work [101] by at least two orders of magnitude, they incur higher latency than techniques that follow the garbled circuit approach.

Wang et al. [102] present a new, constant round Boolean-circuit protocol that is secure against an arbitrary number of malicious corruptions. A preprocessing phase outputs a single, authenticated garbled circuit to be evaluated by one of the parties. Wang et al.'s protocol allows 128 parties across 5 continents to perform an AES computation in under 3 minutes and is the first to examine garbled circuits at such a large scale. While certainly an impressive result, more analysis is required to determine the effects on SMC problems besides the often used AES for benchmarking. Katz et al. [103] further optimize Wang et al.'s protocol by pairing authenticated garbling with the half-gate optimization of Zahur et al. [87]. Communication cost is further reduced by no longer sending an information-theoretic message authentication code (MAC) for each garbled row.

Garg et al. [89] demonstrate that maliciously secure round-optimal MPC protocols can be built from maliciously secure two-round OT. Before their contribution, maliciously secure round-optimal MPC relied on additional noninteractive zero-knowledge proofs [104]. Benhamouda et al. [90] demonstrate that, for any $k \geq 5$, k-round malicious OT is necessary and complete for k-round malicious MPC.

2.5.7. Proactive SMC. The proactive security model proposed by Ostrovsky et al. [105] considers an adversary's ability to eventually corrupt all parties. Eldefrawy et al. [106] show that it is possible to design MPC in this model, and they provide the first proactive secure multiparty computation (PMPC) protocol that is secure in the presence of a dishonest majority. To do this, they require new techniques for refreshing, recovering, and performing operations on secret-shared data that use a combination of information-theoretic secret-sharing and cryptographic protection against active adversaries. The protocol itself is based on any one-way function and oblivious transfer (like the GMW protocol).

2.5.8. SMC on the Web. Halevi et al. [107] explore the feasibility of SMC on the web and assert the model of web computation is generally not suitable for secure computation exchanges. On the web, clients independently connect to servers, interact with them, and leave when done; this rules out protocols that require simultaneous interaction of all parties. Instead, Halevi et al. propose a new client-server protocol, secure in both the semi-honest and malicious adversarial models. In it, each client independently connects to a server once and interacts with it; no other clients need to be connected at the same time.

Harnik et al. [108] attempt to minimize online complexity by incorporating parties' inputs one at a time to minimize the number of OTs needed for each new input. Choi et al. [109] alternatively show how to minimize online rounds using a preprocessing phase that occurs before parties receive their inputs. One flavor of their protocol generates some constant number of garbled gates during preprocessing and requires only two online rounds. The other flavor of their protocol generates a garbled universal circuit during preprocessing and requires just one online round.

2.6. Mobile and Resource-Constrained Platforms. High communication and computation costs make it especially hard to perform SMC on mobile and resource-constrained platforms. While highly provisioned smartphones, such as the newest generation of the iPhone and flagship Android devices, may now be able to handle the stress of certain SMC functions, SMC may just as likely be needed between low-cost authentication tokens/tablets issued by banks or state entities. Protocols need to be revisited and specifically designed to overcome the limitations of these platforms. Key innovations include offloading the majority of computation to the cloud via secure outsourcing.

Mood et al. introduce a memory-efficient technique for generating garbled circuits using a standard SFDL language for describing secure functions as input. They introduce a new pseudo-assembly language (PAL) [110], an intermediate language between SFDL and SHDL. Together with PAL, they provide a template-driven compiler that generates circuits which can be evaluated with Fairplay. By deploying this compiler on Android devices, a large new set of circuits can be generated on smartphones. Circuit generation using PAL can be used together with other SFE optimizations and can reduce memory overhead by 95.6% compared to Fairplay when calculating the intersection of two sets.

Carter et al. give a set of SFE protocols customized for mobile devices: Efficient Mobile Oblivious Computation [111]. Partially homomorphic cryptosystems are used to develop protocols for location-based and social networking apps, and the resulting protocols show execution time improvements of 99% and networking overhead improvements of 96% over the most optimized GC techniques. Their work provides, for the first time, mobile app developers with a practical and equally secure alternative to garbled circuits based on partially homomorphic cryptographic primitives.

Follow-on works investigate the outsourcing of computation from the mobile device to a public cloud [112–114]. Carter et al. [114] give a generic technique for lifting any secure two-party protocol into outsourced two-party SMC. The function being evaluated is augmented with auxiliary consistency checks and input values to create an outsourced protocol with low overhead cost. This work shows that efficient outsourcing is possible, regardless of the underlying SMC scheme. Carter et al. [112] propose a new outsourced OT primitive that requires significantly less bandwidth and computation than standard OT. They also provide outsourced input validation techniques that force the cloud to prove the correctness of its protocol execution. Their extensions are secure in the malicious adversarial model and reduces execution time by 98.92% and bandwidth by 99.95% for edit distance of size 128 compared to nonoutsourced evaluation. In so doing, they show that even limited devices can evaluate some of the largest garbled circuits. Whitewash [113] specifically tackles the circuit-generation phase of GC computation, providing a new technique for securely outsourcing this phase to a cloud provider. On average, the Whitewash protocol reduces execution time by 75% and network costs by 60% compared to prior outsourcing protocols. Through Whitewash, Carter et al. show that garbled-circuit generation can be nearly as practical on mobile devices as on server-class machines.

2.7. Real-World Deployment. Given the vast corpus of work on secure multiparty computation in recent decades, how do attempts to use it in practice fare? The first large-scale, practical application of multiparty computation occurred on January 14, 2008, when 1229 bidders participated in a nationwide auction for Danish sugar-beet contracts with the goal of determining a market clearing price (price per unit of commodity being traded) based on total supply and demand [115]. The entire computation completed in about 30 minutes and led to the change in ownership of about 25,000 tons of production rights. Based on this, Bogetoft et al. [115] write, "we expect that multiparty computation will turn out to be useful in many practical scenarios in the future."

One such scenario demonstrated by Nipane et al. [116] is anonymous message exchange. Their proposed Mix-in-Place Anonymous Networks (MIPNets) replace the multiple intermediaries of mix networks with a single, central oblivious proxy. (Mix networks provide sender/receiver anonymity and unlinkability by pooling messages at intermediate proxy nodes and shuffling them before sending them to their next destination, which may be another proxy.) Clients send to and receive from the proxy using SFE, and each operation changes the appearance of all messages kept by the proxy. Using MIPNets, Nipane et al. realized instant messaging for up to 100 clients.

Bogdanov et al. [117] used Sharemind [65] to implement a secret-sharing-based system that would aid the Estonian Tax and Customs Board (MTA) in detecting value-added tax (VAT) fraud. Though the system was ultimately not adopted, it promised the ability to process one month of Estonian economic transactions (approximately 50 million transactions from 80 thousand companies) in ten days using 20,000 euros of hardware. Secure multiparty computation also powered a 2015 study [118] by the Boston Women's Workforce Council (BWWC) of the gender wage gap in the Greater Boston Area, which computed the sum of compensation data contributed by 40 employers. (The study used a modified variant of the privacy-preserving distributed data mining technique of Clifton et al. [119].) Another area where secure multiparty computation has made an impact is conjunction analysis, specifically computing satellite collision probabilities. Hemenway et al. [120] propose a combination of GMW and garbled circuits that allows satellite operators to compute collision probability without sharing private orbital information, resulting in a protocol that takes only a few minutes to run on a commodity laptop.

Archer et al. [121] survey some of the more current uses of SMC in the real world, including the Private Data as a Service (PDaaS) system of Jana (https://galois.com/project/jana-private-data-as-a-service/), ongoing commercial efforts by Partisia (https://partisia.com/mpc-goes-live/) (which powered the Danish sugar-beet auction mentioned before), and protection of corporate secrets by Unbound Tech (https://www.unboundtech.com/). SMC has also garnered the interest of legislative bodies. Bipartisan Senate Bill 2169/House Bill 4479 (Student Right to Know Before You Go Act), introduced in 2017 to the United States Congress, would require SMC for the assessment of student records [122]. The FORWARD Act introduced in 2018 to the United State House of Representatives would create a pilot program for the National Institutes of Health (NIH) to use SMC for conducting research on a fungal infection.

2.8. Limitations. Despite the numerous advances made and the beginnings of real-world deployment of secure multiparty computation (Section 2.7), SMC still suffers from the high overhead of cryptographic operations. Even two-party secure computation remains impractical for most real-time (on the order of seconds or even milliseconds), online computations. Defenses against malicious or adaptive adversaries and resource constraints further degrade performance.

Besides performance limitations, several other barriers to implementation of SMC are identified by the Usable and Efficient Secure Multiparty Computation (UaESMC) [123] project. Their findings are summarized below:

(i) *Legality*: current laws do not have adequate language to describe SMC; cartel/competition laws and data jurisdiction restrictions discourage cooperation.

(ii) *Receptiveness*: SMC techniques are simply too complex for nonexperts to understand; SMC needs to be accompanied by proper monitoring/auditing for establishing trustworthiness.

(iii) *User risks*: intentional misuse by users can break security, esp., if functions are not carefully crafted to leak no additional information in the output.

(iv) *Technology*: data handling and scalability concerns.

(v) *Data visibility*: the inability to see the input data fosters suspicion in the computation result.

Evans et al. [124] specifically suggest that more widespread adoption of SMC will depend on building increased confidence in SMC techniques and better catering SMC to parties without a strong technical cryptographic background.

3. Hardware-Assisted Computation

Cryptography alone may not be enough to pose practical approaches for secure multiparty computation, even as the computing environment continues to change. For instance, many complex computations are now offloaded onto cloud service providers for better performance. However, this removes cloud subscribers' direct control over their own data, and cloud providers are trusted with this (possibly sensitive) data. While advances in SMC have been largely based in cryptography through this point, van Dijk and Juels [125] assert that "general multi-client private computing based solely on cryptographical assumptions is impossible."

The new enabler may be existing or emerging hardware technologies, which give us the means to tackle some of the limitations of solutions that are purely cryptographic in nature. (Limitations such as inadequate legal language and data visibility concerns also apply to hardware and are symptomatic of SMC in general.) For example, trusted hardware allows sidestepping the problem of using two or more legal entities (potentially falling under different jurisdictions) to process data, which creates legal and organizational deployment obstacles. Trusted hardware in terms of hardware security modules (HSMs) and air-gapped computer systems has been around for decades and is well understood, so relying on trusted hardware to do SMC is a sensible direction for exploration.

3.1. Grounding Trust in Hardware. Behind any secure computing system is a root of trust. A root of trust is the lowest level of a system from which trust originates. It comprises highly reliable hardware, firmware, or software components that are the product of a secure design process and is typically small to support verification. A root of trust is commonly implemented in hardware to enforce tamper-resistance. The root of trust is charged with security-critical functions such as platform measurement and cryptographic key management; this functionality makes it possible to reason about the security of a larger system.

Approaches like AEGIS [126] build a root of trust from a minimal set of trusted software, performing integrity checks of each successive level by comparing a computed cryptographic hash with a stored (signed) hash value. Building on top of software requires an assumption that the underlying hardware and system firmware are trusted. For example, AEGIS assumes an uncompromised motherboard, processor, and BIOS. However, blind trust in underlying hardware is no longer an appropriate attitude in the face of a number of possible hardware vulnerabilities, which include physical attacks, hardware Trojan and backdoor insertion, counterfeiting, and tampering. This reality suggests the root-of-trust needs to occur deeper within the system (i.e., below the software layer), with the help of trusted hardware.

Superdistribution [127, 128], originally termed software service system by Ryoichi Mori in 1983, refers to unrestrained distribution of digital content in encrypted form (for purposes of Digital Rights Management or DRM). Superdistribution relies on and requires the installation of a special tamperproof hardware black-box or Superdistribution Label Reader (SDLR) (originally Software Usage Monitor (SUM), connected by serial port (Prototype I); later iterations included a Hyper Security Integrated Circuit and a more efficient coprocessor-based variant (Prototype II)), which can decrypt received content and facilitate payments to content owners. Each piece of content has an associated Superdistribution Label (SDL) which contains identifying information and terms/conditions of use. Superdistribution, which now forms the basis of most of the work on DRM, provides one of the first illustrative examples of secure operations grounded in trusted hardware.

There are a number of hardware-based technologies for security, starting with external hardware security modules which are carefully designed to give resistance to tampering and physical attacks (e.g., IBM 4758 [129] and its descendants). Hardware security modules may alternatively be embedded into the motherboard to reduce costs; one such example is the Trusted Platform Module (TPM) [130] designed by the Trusted Computing Group (TCG). While external hardware security modules offer a high level of physical security, they come at a high monetary cost and cannot work on an SoC. In contrast, on-board TPMs are already widely deployed in commodity systems, making them a convenient building block for secure systems.

The TPM is a tamper-resistant, secure microcontroller that manages cryptographic keys and performs measurement of code loaded by the system in Platform Configuration Registers (PCRs). The TPM provides a hardware-based root of trust; it is possible to check whether a system is in a good state by using the TPM to measure underlying hardware, BIOS, boot loader, and operating system. These measurements are used during trusted boot and attestation of the identity of a system. Among other things, the TPM can be used to build an integrity measurement architecture by holding an aggregate value corresponding to an in-kernel list of measurements of loaded executable content [131]. The TPM may be virtualized to support large numbers of virtual machines on a single hardware platform [132]. The TPM is also a key component of several more comprehensive platform security solutions, such as Intel Trusted Execution Technology. A TPM may be implemented as a standalone hardware chip embedded into the motherboard, in which case it is referred to as a discrete TPM (dTPM). dTPMs have dedicated compute resources and storage and are hardened against physical attacks in accordance with Trusted Computing Group (TCG) specifications. Alternatively, ARM TrustZone may be leveraged to build a firmware-based TPM (fTPM) [133]. fTPMs offer better performance than dTPMs while also sidestepping bus attacks. However, fTPMs, unlike dTPMs, are vulnerable to physical attacks on memory (due to reliance on system DRAM) and side-channel attacks (due to sharing of resources with an untrusted OS).

Intel Trusted Execution Technology (Intel TXT) [134] provides a Measured Launch Environment (MLE). Critical elements of the launch environment are compared with known good values, and mismatching code is prevented from loading by the launch policy. Intel TXT also provides protection of secrets following improper MLE shutdown and platform attestation. Intel TXT depends on the PCRs and nonvolatile user-defined indices (NV indices) that are provided by TPMs. Besides the TPM, Intel TXT is enabled by trusted extensions integrated directly into the silicon, authenticated code modules (ACMs) that serve as the first measurement targets, and Launch Control Policy tools.

ARM TrustZone [135] partitions all of a system's resources into one of two worlds: secure or normal. Hardware logic built into TrustZone-enabled bus fabric ensures secure world assets are inaccessible by normal world components, while ARM processor core extensions enable processor cores to safely/efficiently execute code from both worlds in a time-sliced fashion. The secure world can support anything from a separate operating system to a code library managed by the normal world. ARM TrustZone differs from Intel TXT in that TrustZone itself implements TPM functionality in firmware without relying on dedicated hardware. This is in following with the TCG Mobile Trusted Module (MTM) specification [136] (TrustZone is capable of supporting software-only MTM implementations, as demonstrated by Winter [137]). ARM TrustZone technology is widely deployed on Android devices and forms the basis of secure platform solutions such as Samsung KNOX [138] and AMD Secure Technology (Platform Security Processor) [139].

Intel Software Guard Extensions (Intel SGX) [8] is a more recent technology that offers a different take on trusted computing. SGX is implemented in microcode and only requires trust in the processor package and several privileged enclaves supplied by Intel. In other words, the processor takes on the root-of-trust role; any privileged system software is assumed untrusted. Enclaves are instantiated within the enclave page cache in processor-reserved memory. Code and data are partitioned, with trusted/security-sensitive portions loaded into enclaves and protected from unauthorized access or memory snooping by processor-enforced checks.

AMD Virtualization (AMD-V) [140], formerly known as Secure Virtual Machine (SVM), provides hardware extensions for secure, resource-guaranteed isolation and Virtual Machine Monitor (VMM) functionality. Additional features provided by AMD-V include attestation and clearing of system memory on reset. AMD recently proposed augmenting AMD-V with AMD Secure Encrypted Virtualization (SEV) [141, 142] technology, which will protect virtual machines from malicious hypervisors or virtual machine monitors. Rather than providing CPU-enforced isolation like Intel SGX or ARM TrustZone, SEV works by transparently encrypting VMs with a secure processor implemented in hardware. Data is encrypted but not integrity-protected. Similarly to SGX, SEV only trusts the CPU, but SEV exposes a substantially larger attack surface due to a large code base and seemingly incomplete mediation of VMM functionality.

3.2. Formal Basis. Collectively, we may refer to technologies such as TPM, TrustZone, and SGX as *attested execution secure processors*. Formal abstractions and a rigorous exploration of their expressive power are given by Pass, Shi, and Tramèr [143]. In their work, SGX-like secure processors are formally abstracted as an ideal functionality $\mathscr{G}_{att}$, globally shared by all users, applications, and protocols. $\mathscr{G}_{att}$ is parametrized with a registry `reg`. Enclave operations (e.g., install new and stateful resume) may be invoked by a platform $\mathscr{P}$ in the registry, and anonymous attestation is possible using group signatures and other anonymous credential techniques.

Pass et al. confirm, through their investigation, that attested execution processors are "indeed extremely powerful" and allow realization of primitives such as stateful obfuscation, which are otherwise impossible "even when assuming stateless hardware tokens or virtual blackbox secure cryptographic obfuscation" [143]. They also present a number of findings concerning the feasibility of attested execution processors for doing secure multiparty computation:

(i) Universal Composability (UC) secure multiparty computation cannot be realized if at least one party is not equipped with an attested execution processor, which goes against the intuition that an attested execution processor could fulfill the role of a true trusted third party. However, the addition of a global Augmented Common Reference String (ACRS) [144] allows extraction of corrupt clients' inputs by the simulator, thus making UC-secure MPC realizable.

(ii) Assuming secure key exchange protocols exist, composable 2-party computation can be achieved (with all program-dependent evaluation performed in the enclave (not cryptographically)) when both parties have an attested execution processor.

(iii) Assuming secure key exchange protocols exist, if both parties have an attested execution processor with trusted clocks, fairness is achievable.

(iv) Fairness is impossible for general functionalities when only one party's processor is not clock-aware, but certain functions can be computed fairly with the help of ACRS.

These results are promising and confirm the hypothesis that attested execution processors can be applied to secure multiparty computation problems.

Barbosa et al. [145] also take a provable security approach and define a similar notion. They define the notion of attested computation, which demands that the user's local view of the execution meets expectations, and code is actually executed in isolation within a prescribed remote machine. The attested program should also ensure minimal leakage. Another supporting notion is key exchange for attested computation, which differs from key exchange in a nonattested setting and must be built from a passively secure key exchange protocol and an additional existentially unforgeable signature scheme (which provides freshness). Barbosa et al. prove, given a correct and secure attested computation protocol and a correct and secure attested key exchange that ensures minimal leakage (in accordance to their definitions), the probability that an adversary violates two-sided entity authentication is negligible, and so is the key secrecy advantage. Barbosa et al. demonstrate that key exchange for attested computation may be combined with an authenticated symmetric encryption scheme and replay protection to achieve secure outsourced computation that offers both verifiability and privacy. Further, they identify secure multiparty computation as an interesting next target for applying their attested computation building blocks.

3.3. Adoption Considerations. Although the hardware-based technologies described in Section 3.1 are attractive, they should not be carelessly introduced to privacy-preserving applications. Naïve use of these technologies may give way to unintended vulnerabilities.

For example, while true that a TPM can provide a root-of-trust, Parno reveals the need to bootstrap trust in the TPM itself [146]. In a cuckoo attack, malware may redirect messages intended for a local TPM to a different adversary-controlled TPM on a different machine. The adversary can then, unknown to the victim, take control of all communication between victim and local TPM. To prevent this, a secure channel to the local TPM should be established, possibly using an alphanumeric hash (Parno's preferred solution with plausible deployability: the manufacturer affixes an encoding of the hash of a platform's identity on the platform's case; to lessen the burden on the user, who would manually enter this string into a smartphone or dedicated fob, Parno specifically suggests an alphanumeric string encoding of the hash) of the TPM's public key or (for strongest security) a special-purpose hardware interface that opens up direct communication with the TPM [146].

In the case of Intel SGX, an improper partitioning of code into trusted (enclave) and untrusted components can cause leakage of sensitive information. Automated partitioning tools such as Glamdring [147] may help to perform proper, minimal-leakage partitioning. The partitioning can be further inspected using verifiers such as Moat [148], which uses automated theorem proving and information flow analysis to formally verify the confidentiality properties of SGX applications. Besides partitioning errors, Intel SGX is vulnerable to a number of attacks. Side channels have been a particularly large issue with SGX, and notably, Intel left these attacks out of the threat model, considering their mitigation to be the developer's responsibility.

Even given this caveat, the vulnerable surfaces found within SGX represent an area of serious concern. Controlled-channel attacks [149] use memory access patterns to exfiltrate information from enclaves, representing a cache-based attack. Schwarz et al. demonstrate that it is possible to use SGX itself as a means of concealing cache attacks that originate within enclaves that are malicious [150]. It is not just the cache that is of concern, as subsequent work [151] shows how side channels can be found throughout the memory hierarchy when SGX is employed, from DRAM to transaction lookaside buffer (TLB). Attacks against the microarchitecture have been an area of intense recent interest, with Meltdown [152] and Spectre [153] attacks that target speculative execution units by allowing reads to privileged kernel memory. While SGX has proven largely resilient to these attacks, more recent Foreshadow attacks [154] also target speculative execution and SGX specifically and demonstrate methods to expose attestation keys within secure enclaves. Other recent attacks demonstrate that SGX can be susceptible to code-reuse attacks without requiring privileges from the kernel [155]. While there is substantial research into developing defenses against these attacks, it is clear that assuring the security of SGX against memory vulnerabilities will be an active area of investigation for some time to come.

Several attacks have been demonstrated against AMD SEV. The SEVered [156] attack leverages the lack of integrity protection and allows the hypervisor to change the memory layout of VMs and trick services into returning memory contents in plaintext. Du et al. [157] show that a malicious/compromised hypervisor could arbitrarily modify a VM's ciphertext content using AMD's physical address-based tweak algorithm; these modifications go undetected due to the lack of integrity checks of encrypted memory.

Intel TXT and ARM TrustZone invite attacks of their own as well. Each technology has its individual strengths and weaknesses, making it important to choose the right technology (or technologies) for each particular threat-model, computing environment, or application.

More generally, a primary drawback of trusted computing primitives is the possible leakage that occurs during communication between trusted and untrusted components. Dang et al. propose Scramble-then-Compute [158], an approach that scrambles input before it is processed in order to eliminate such leakage and is designed to achieve the same goals as Oblivious RAM (ORAM). Proper use of hardware-based technologies will require a careful assessment of whether the chosen primitive meets all of the required security properties of a particular domain; in cases where it cannot, techniques such as ORAM need be applied.

3.4. Alternative Protection Systems. Certain computation settings have additional requirements that cannot be met by general solutions. To meet these demands, entirely new protection systems are proposed, many of which are built atop of TrustZone, SGX, etc. and provide additional guarantees.

Flicker [159] executes security-sensitive code (Piece of Application Logic or PAL) in complete isolation while also providing meaningful, fine-grained attestation of executing code to a remote party. Developers provide the PAL and define its interface with the rest of the application (this process can be automated). Flicker is built atop AMD's Secure Virtual Machine (SVM) extensions to leverage its late launch capabilities: Flicker pauses the current execution environment, executes a PAL using the SKINIT instruction, and then resumes the paused environment. Flicker uses the TPM for sealed storage, allowing state to be maintained across sessions; replay protection can be achieved using the TPM's nonvolatile storage and PCR-based access control. TrustVisor [160] improves on Flicker, which incurs high performance overhead from its frequent use of hardware support for a dynamic root of trust for measurement (DRTM). TrustVisor is a hypervisor-based approach for fine-grained code integrity and data integrity/secrecy that also supports attestation of isolated execution. A measured, isolated execution environment is initialized via a DRTM-like process, TrustVisor Root of Trust for Measurement (TRTM), which interacts with a software-based micro-TPM (μTPM). (This is different from an fTPM and is specific to TrustVisor. The μTPM executes on the platform's primary CPU and provides only basic randomness, measurement, attestation, and sealing. Other features require direct interaction with a hardware TPM.)

Virtual Ghost [161] protects applications from a potentially malicious OS by combining compiler instrumentation and run-time checks on OS code, creating "ghost memory" inaccessible by the OS (not just tamperproof). Virtual Ghost introduces a thin hardware abstraction layer between the kernel and hardware, but it is different from other hypervisor-based solutions because no software runs at a higher privilege level than the kernel, and compiler techniques are used instead of hardware page protection. Haven [162] has similar goals: protecting the confidentiality and integrity of code/data from the executing platform. Haven realizes shielded execution of unmodified legacy applications by leveraging Intel SGX. Haven defends against privileged code and physical attacks.

SeCReT [163] seeks to fill the gap between rich execution environment (REE) and trusted execution environment (TEE). SeCReT implements an access control list that restricts access to resources in TrustZone to certain REE processes. Additionally, session keys are used to sign messages transferred between REE and TEE for authentication, only allowing requests from the REE that originated from authenticated processes while blocking malicious messages crafted by attackers. Session keys are supplied only upon verification of requestors' code and control-flow integrity, and keys are flushed from memory at every processor switch into kernel mode. In particular, SeCReT is implemented for ARM TrustZone, leveraging existing active monitoring (e.g., TIMA) that protects the kernel's static region in the REE.

Ryoan [164] tackles the problem of protecting secret data while being processed by untrusted services. Ryoan is a distributed sandbox that uses hardware enclaves (e.g., Intel SGX) to protect sandbox instances. Applications are confined by a trusted sandbox, Google Native Client (NaCl) (https://developer.chrome.com/native-client), and each SGX enclave contains a NaCl sandbox instance for loading and executing untrusted modules; these instances can communicate with each other for distributed processing. Ryoan follows a request-oriented data model in which confined modules process input once and have no persisting state tied to the input. Use of SGX has also been explored in containerized cloud environments. SCONE [165] is a secure container mechanism for Docker (https://www.docker.com) that uses SGX to protect container processes from outside attacks. SCONE provides a secure C standard library interface for encrypting and decrypting I/O data. SCONE also supports user-level threading and asynchronous system calls to reduce performance overhead. Unmodified applications can run with 0.6x-1.2x native throughput with the help of SCONE.

Unlike SCONE, which requires placing the entire application within the SGX enclave where memory in the enclave page cache (ePC) is limited, *lxcsgx* is a solution that specifically develops a mechanism for using containers which is built upon the Intel SGX SDK, allowing for local and remote SGX-based CPU attestation and enforcing limits on EPC memory usage per container [166]. Alternative approaches to offering SGX in a cloud environment are given by Graphene-SGX [167] and Panoply [168] .

Rather than using SGX outright, Sanctum [169] takes a principled approach to isolation and a system developed in response to SGX, which provably defends against known software side channels (SGX leaves side-channel defense to application developers). For example, to defend against attacks on memory access patterns, Sanctum uses a page-coloring-based cache partitioning scheme that outperforms ORAM. Designed for the Rocket RISC-V chip, most of its logic is implemented in trusted software (adopting many elements of SGX design, including enclaves and attestation).

3.5. Existing Hardware-Assisted Applications. In addition to enabling new computational settings, as demonstrated by the efforts described in the previous subsection, hardware has also been integrated into a variety of applications to overcome their specific challenges. These applications range from set operations and trusted databases to program obfuscation and functional encryption. This section analyzes the innovative decisions made when extending hardware support to these various applications.

3.5.1. Using ARM TrustZone. TrustShadow [170] shields legacy applications from an untrusted OS using TrustZone. Each application has two parts: a "zombie" part that runs in the normal world and a "shadow" counterpart that runs in the secure world. A lightweight runtime system in the secure world maintains a TEE for applications by intercepting exceptions and forwarding them to the normal world. System services are provided by the normal world, reducing the required TCB, and the runtime system verifies the correctness of responses from the OS while introducing negligible overhead. The normal world OS is prevented from directly accessing the "shadow" application's virtual memory.

Cho et al. [171] investigate issues surrounding deployment of TEEs on mobile devices and assert that TrustZone-based approaches bloat the trusted computing base of a system, while hypervisor-based approaches incur sizable performance overhead. To defeat the limitations of either, they propose a hybrid approach that combines the memory protection of TrustZone with a hypervisor. The resulting system, which they term On-Demand Software Protection, activates the hypervisor only when a TEE is demanded by security-critical code to avoid virtualization overhead.

Private membership tests may also benefit from TrustZone, as demonstrated by Tamrakar et al. [172]. They develop a carousel approach in which the entire dictionary is circled through trusted hardware on the cloud server, with Kinibi OS running in the trusted world. The carousel approach has the advantage of efficient batch processing over ORAM, which incurs 10x more latency when processing 2,000 queries. The four nonoverlapping Cuckoo hashes on a carousel variant are found most efficient, able to support a larger number of queries than either ORAM or Bloom filters, and stable when handling 1025 queries/second.

3.5.2. Using Intel SGX. Gupta et al. [173] propose that Intel SGX is an enabler for efficient secure multiparty computation. Rather than relying on SGX enclaves to process the entire computation, they call for the splitting of computation into separate garbled circuit and SGX components. Garbled circuits could be used to handle especially sensitive portions of the computation, so secrets are safe in the event of enclave compromise. They consider the semi-honest adversarial model when one or both parties have access to SGX hardware and suggest similar protocols could support the malicious adversarial setting. Portela et al. [174] confirm the potential of SGX to support secure multiparty computation and propose a novel notion of *labelled attested computation* to capture this setting. The intuition behind labelled attested computation is as follows: code loaded into an isolated execution environment such as SGX is marked with labels pertaining to users, and individual users can get attestations of parts of code corresponding to specific labels. This allows users to be oblivious of other users' interactions while making it possible to get an indirect attestation of the overall execution of the system. Portela et al. build a new SMC protocol from SGX based on labelled attested computation by composing attested key exchange procedures for each participant in parallel. Once a secure channel is established with each participant, the function is evaluated in the enclave (taking inputs and releasing outputs to each party as needed). Portela et al. essentially treat code within the enclave environment of SGX as a trusted third party, relying on the attestation capability of SGX to bootstrap secure communication between enclave and participants.

Besides supporting secure multiparty computation, SGX may also give added security and privacy to network applications (including software-defined inter-domain routing, peer-to-peer anonymity networks, and middleboxes) [175]. Kim et al. claim the difficulty of transforming arbitrary computations into secure multiparty versions when compared to adopting SGX. For example, they suggest running Tor's directory authorities within SGX enclaves to safeguard authority keys and lists of Tor nodes; attestation further prevents attackers from altering directory behavior.

SGX has also been applied to distributed operations in cloud environments. VC3 [176] supports distributed MapReduce in the cloud while keeping code/data secret and ensuring both the correctness and completeness of results. By relying on SGX, Hadoop (http://hadoop.apache.org/), the OS, and the hypervisor can all be pushed outside the TCB. SGX is used to isolate memory regions on individual computers, and region self-integrity invariants can optionally be enforced for all MapReduce code running in these isolated regions to prevent unsafe memory operations. VC3 introduces an additional quoting enclave to attest that an application enclave is running on hardware owned and certified by the cloud provider, in a certain data center.

Iron [177] performs functional encryption (FE) and multi-input FE using SGX. Functional encryption allows authorized entities to perform selective computation on encrypted data and learn results in the clear or gain partial access to encrypted data (whereas traditional encryption schemes strictly allow decryption of either all or none of the ciphertext). FE schemes support many possible keys, each with different decryption capabilities. In Iron, a key manager enclave (KME) takes the role of the trusted authority who generates (using a master key) secret keys associated with a function, and these keys are used to decrypt ciphertexts. A client, once authorized by the KME, receives a functional secret key and performs decryption in its decryption and function enclaves. By relying on SGX, Iron is able to outperform cryptographic implementations of FE for complex functionalities.

OblivML [178] relies on SGX to perform privacy-preserving multiparty machine learning. Algorithms are carefully selected, adapted, and implemented to prevent the exploitation of side channels (e.g., memory access patterns). As part of OblivML, Ohrimenko et al. provide data-oblivious machine learning algorithms for support vector machines, matrix factorization, neural networks, decision trees, and k-means clustering. OblivML scales to large, realistic datasets and lowers overhead by orders-of-magnitude by relying on high-performance SGX-assisted execution.

Tamrakar et al. [172] provide a carousel implementation for private membership tests using SGX. The SGX implementation can outperform its TrustZone-based counterpart, because enclave entry/exit adds little overhead, and supports a query arrival rate of 3720 queries/second. Unlike TrustZone, SGX does not provide private memory, meaning an adversary could observe memory access patterns at page-level granularity. To overcome this challenge, Tamrakar et al. design trusted SGX applications in such a way that memory access patterns are data-independent. When private data structures span multiple oblivious pages, a memory access is made to each of these pages to hide data dependencies.

FastBFT [179] is a byzantine fault-tolerant (BFT) protocol that reduces message complexity by combining TEEs with lightweight secret sharing. Further optimized by optimistic execution, tree topology, and failure detection, FastBFT achieves (near-optimal) low latency and high throughput even in large networks; previous BFT protocols scaled poorly due to $O(n^2)$ message complexity, where n denotes the number of replicas. The TEE part of the FastBFT replica can be implemented as an SGX enclave, and a virtual monotonic counter is used to provide rollback-resistant memory.

Kurnikov et al. propose SafeKeeper [180], a remote password manager that uses SGX. An add-on installed on the client's web browser first performs remote attestation of the server's password protection service. Passwords and other sensitive input fields are sent encrypted to the server's enclave and are only processed within the enclave. Password and salt are inputs to a keyed one-way function, with the result stored in the server's password database and the key available exclusively in the enclave. This successful integration of password manager and SGX affirms the ability of SGX to support a wide variety of applications. A more general database implementation using SGX is given in EnclaveDB [181]. EnclaveDB works like a conventional relational database, but all sensitive data (tables, indexes, queries, and intermediate state) is hosted in enclave memory. Additionally, there is a decoupling of query compilation from execution, allowing precompiled queries to be deployed within the enclave while query-parser/compiler/optimizer are hosted elsewhere in a trusted environment. By using SGX, the TCB is 100x smaller than that in a conventional database server.

SGX is also used by Sasy et al. to build ZeroTrace [182], a library of oblivious memory primitives. ZeroTrace combines state-of-the-art oblivious RAM techniques with SGX, achieving improved performance over a pure cryptographic implementation and resistance to software side-channels. Sasy et al. also build the first oblivious memory controller from SGX that is able to protect against active software adversaries while also being able to handle the asynchronous terminations of SGX enclaves.

3.5.3. Using Specialized Hardware. Fischlin et al. [183] demonstrate that hardware can support secure set operations (e.g., set intersection) even when tokens are not necessarily trusted by both participants. They achieve the same level of security as Hazay and Lindell [184], giving privacy guarantees in a malicious adversarial setting and correctness guarantees in a covert adversarial setting. Whereas preceding work assumed tokens were tamperproof and provided by trusted manufacturers (e.g., certified smartcards), Fischlin et al. claim that even well-tested tokens may contain errors/backdoors. The protocols they build require only the issuer's trust in hardware tokens to ensure privacy in the malicious and correctness in the covert adversarial models.

TrustedDB [185] is motivated by the inherit limitations on SQL query expressiveness imposed by software-based, cryptographic constructs. TrustedDB instead leverages server-hosted tamperproof trusted hardware for cost-efficient, critical query-processing. To get around the computational and memory-capacity constraints of secure coprocessors (e.g., IBM 4764), Bajaj and Sion propose max utilization of common unsecured server resources to reduce cost by orders-of-magnitude. They also propose installing multiple secure coprocessors on a TrustedDB-hosting server for simultaneous servicing of multiple clients. Cipherbase [186], a full-fledged SQL database system that extends Microsoft's SQL Server (https://www.microsoft.com/en-us/sql-server/), is another database solution that relies on trusted hardware and builds on TrustedDB. Cipherbase aims to simulate fully homomorphic encryption on top of nonhomomorphic encryption schemes by shipping encrypted data (encrypted using an application-specific secret key) to a trusted machine, within which it is decrypted, processed, and reencrypted before being sent back. Arasu et al. implement their trusted machine for Cipherbase using Field Programmable Gate Array (FPGA), which provides the flexibility of processor-based systems alongside the security and performance of dedicated hardware.

Canim et al. [187] argue that simple automation could be used to reidentify biomedical data. Further, they point out that practical cryptographic protocols for sharing/managing/analyzing biomedical data require the addition of multiple third parties. To remove this dependency, Canim et al. propose colocating services to store and process sensitive data using secure coprocessors. All data processing is performed within the coprocessor, which also provides a secure Ethernet channel for communication between hospitals and the data processing/storage server. The secure coprocessor can also facilitate secure auditing, which could be useful to show that data handling satisfies regulations.

A special hardware recryption box [188] can be used to speed up homomorphic function evaluation and encrypted search. The recryption box does this by refreshing ciphertexts and lowering noise, a limiting factor for SWHE operations; once noise reaches a critical bound, no further operations can be supported. FHE schemes perform an additional bootstrapping operation to refresh noisy ciphertexts and enable evaluations of arbitrary depth, but the evaluation of decryption circuits during refreshing is slow. (All SWHE schemes (and FHE schemes built atop them) are noisy. PHE schemes vary in noisiness: multiplicatively-homomorphic textbook RSA, although not affected by noise, is not IND-CPA/semantically secure; Paillier encryption [189], an additively homomorphic PHE scheme, is IND-CPA/semantically secure but uses a noise factor to mask messages.) Roy et al. implement the recryption box on FPGA and demonstrate a 20x performance improvement for encrypted search.

Järvinen et al. [190] give a generic architecture for using garbled circuits and one-time programs modularly to achieve leakage resilience. Two FPGA-based prototypes are provided, a system-on-a-programmable-chip paired with a SHA-256 hardware accelerator (representative of next generation smartcards/smartphones) and a standalone hardware implementation. They demonstrate that one-time programs relying on hardware can provide leakage-resilient evaluation of arbitrary functions in untrusted environments.

3.5.4. Using Existing Parts in New Ways. Husted et al. [191] show how to use graphical processor units (GPUs) to optimize two-party garbled circuit computation in both the semi-honest and malicious (with 1 bit leaked) models. With the help of a GPU, they are able to completely parallelize the generation of garbled circuits, even with the Free-XOR [42] optimization enabled. This generation using GPUs can further be concurrently paired with evaluation on CPUs. To remove the dependencies introduced by Free-XOR, Husted et al. virtually generate labels for all a circuit's wires and calculate offsets unique to each XOR gate with which it becomes possible to serially compute each gate in every XOR gate chain of a circuit. Their approach allows generation of 75 million gates per second, outperforming other CPU- and GPU-based systems alike.

PixelVault [192] relies exclusively on GPUs for protecting keys and performing cryptographic operations. Keys are exposed only in GPU registers (never in observable memory), and critical code only appears in the GPU instruction cache; access to either from the host is prevented, even upon full compromise of the host. The nonpreemptive execution mode of the GPU prevents adversarial tampering of PixelVault's GPU code. Relying on GPUs for secure key storage comes with added support for higher processing throughput of cryptographic operations for server applications.

GhostRider [193] is a co-designed compiler and architecture for privacy-preserving cloud computation that employs nonoblivious encrypted RAM and scratchpad memory when doing so would not compromise memory-trace obliviousness (MTO). Otherwise, oblivious RAM (ORAM) banks are used to contain sensitive access patterns. GhostRider outperforms approaches to MTO that rely solely on ORAM. The GhostRider compiler can also allocate to multiple ORAM banks for further reduction of access times.

HOP [194] provides simulation-secure obfuscation for RAM programs with the help of secure hardware, following impossibility results of software-only virtual black box obfuscation of general programs. HOP puts trust only in a hardware single-chip processor. HOP optimizations include use of hardware ORAM, hardware scratchpad memories, and instruction-scheduling/context-switching improvements. HOP improves on prior work by over three orders of magnitude, bringing deployment of obfuscation closer to practice.

OASIS [195] is a CPU instruction set extension for externally verifiable initialization, execution, and termination of isolated execution environments (IEEs). OASIS leverages the hardware components available on commodity CPUs and, as a result, only includes the CPU in the TCB. Specifically, OASIS assumes that the CPU contains a physically unclonable function (PUF) and a true random number generator (TRNG). OASIS instantiates an IEE using several Mb of on-die memory to create a Cache-as-RAM (CAR) execution environment, typically used for system boot-up tasks.

TRESOR [196] is a Linux kernel patch that implements AES (AES-NI) and key management solely on the microprocessor. Under TRESOR, the secret key and all encryption states are only stored in processor registers. x86 debug registers are repurposed for cryptographic key storage, and performance is comparable with standard AES. Despite handling keys only within the CPU boundary, TRESOR was demonstrated to be vulnerable to the TRESOR-HUNT [197] direct memory access (DMA) attack, in which an adversary injects code into the kernel that will transfer keys from the CPU into RAM; this attack can be applied to any CPU-bound encryption technique. The success of TRESOR-HUNT suggests that "secure" systems may still be vulnerable due to overlooked attacker capabilities and confirms that the secure integration of hardware primitives in new ways is an attractive but nontrivial endeavor.

3.5.5. Summary. Table 1 compares several properties of the major SMC techniques presented in Section 2 with those of TrustZone, SGX, and specialized hardware. In summary, the protections offered by hardware-assisted trusted execution environments (such as TrustZone and SGX) allow simplified application and protocol design. Specialized hardware, or

Table 1: Comparison of the properties provided by classical SMC techniques with those provided by trusted hardware.

Technique	Required Trust	Code Integrity	Data Integrity	Secure Channel	Enforcement Mechanism	Overhead
Garbled Circuits	Minimal	Cut-and-choose	Cut-and-choose	Oblivious Transfer; garbled exchange	Strong cryptography	High
Homomorphic Encryption	Minimal	Restricted ops: PHE/SWHE	Not built-in	Operations on encrypted data	Strong cryptography	High
Secret Sharing	Minimal, with exceptions (threshold)	Limited to well-defined splitting and reconstruction	Verifiable schemes	Secrets not exposed to shareholders	Strong cryptography	Scheme-dependent; generally high
Intel SGX	CPU, Intel	Local/Remote attestation of enclave contents; enclave only accessible through ECALLs	Local/Remote attestation of enclave contents; enclave only accessible through ECALLs	Shared secret set during attestation	CPU	Low
ARM TrustZone	Secure world OS and trustlets	Code and data in secure world not directly accessible by normal world components	Code and data in secure world not directly accessible by normal world components	Secure Monitor Call	CPU and bus logic	Low
Specialized Hardware (e.g., coprocessors)	Hardware elements	Sensitive data and critical code only appear within the (often tamperproof) hardware elements	Sensitive data and critical code only appear within the (often tamperproof) hardware elements	Varies: encrypt if not hardware-confined	Hardware	Low

hardware repurposed for new uses (e.g., GPUs), can also significantly improve performance in a variety of domains. Thus, TEEs and specialized hardware are both promising directions for realizing more practical secure multiparty computation, a stance that is reinforced by several preliminary works [172–174].

4. Open Challenges

After investigating the state of secure multiparty computation, we see that SMC has improved by leaps and bounds in the past several decades and even saw real-world use (Section 2.7). Given that SMC has already succeeded in practice, perhaps the missing piece is a set of compelling applications to make full use of the much-improved techniques, rather than a fixation on improving the techniques further. We have plans to revisit our previous work [198] on transmitter localization in sensor networks as one such use case for which SMC techniques may be mature enough.

We see as well the potential of trusted hardware to contribute to the SMC space, and we are convinced of the value in continued exploration of their combination. This section identifies several remaining challenges that were inspired by our investigation. These challenges are as follows:

(1) Do TEEs offer a shortcut for defending against malicious adversaries?

(2) Are TEE-based solutions just as viable for mobile and similarly resource-constrained platforms?

(3) Can we apply trusted hardware to other types of privacy-preserving computation?

Each of these challenges is unpacked in more detail below.

4.1. Challenge 1: Defeating Malicious Adversaries.

Defending against malicious adversaries is significantly harder than defending against semi-honest adversaries. When using garbled circuits, a cut-and-choose [51] procedure is needed to assure parties of the correctness of a circuit being evaluated; this cut-and-choose substantially increases the computation/communication complexity, because many copies of the same circuit need to be generated.

Symmetric cut-and-choose [56] and related optimizations (detailed in Section 2.5.1) have brought down the cost of defense, but the attestation component of TEEs may allow us to sidestep the cut-and-choose procedure altogether. This is especially true in the context of Intel SGX. The SGX-provided remote attestation capability allows computing parties to be assured that the function code loaded into the evaluating party's enclave is as expected (prior to passing their sensitive inputs into the enclave).

A downside of this reliance on TEE-provided attestation is the shift in trust model from that of traditional cryptographic approaches. For example, parties may be uncomfortable with Intel's control over the attestation process in SGX (with Intel potentially able to perform man-in-the-middle attacks if they so choose). (A future version of Intel SGX is expected to provide attestation capabilities that do not require the direct involvement of Intel.) Using TrustZone implies the (possibly large) OS running in the secure world needs to be trusted. Moreover, though the normal and secure worlds are isolated, they still share the same system resources. AMD's SEV, with its lack of integrity protections, means possibly misplaced trust in arbitrarily modifiable code/data while also requiring trust in an additional secure processor.

Besides trust concerns, TEE-provided attestation itself may be insufficient. Considering again in the context of SGX, attestation can only be used to confirm the contents of the enclave-portion of an application, not of any untrusted components outside of the enclave. Mechanisms provided by other TEEs may be similarly incomplete. The modular design of TEEs poses issues when any of their components are found vulnerable. That is, the compromise of any single component may lead to compromise of the entire TEE.

TEEs certainly have their own problems, but they may provide a starting point for an alternative means of "lifting" security in the semi-honest adversarial setting to the malicious or covert settings. Despite the many recent advancements in SMC performance in the presence of malicious adversaries (as described in Section 2.5.6), a pure TEE-based approach appears to be a worthwhile line of future work to take advantage of their wide availability.

4.2. Challenge 2: Mobile-Friendly SMC.

Conducting business on mobile devices is on the rise, and mobile devices even outnumber desktop PCs and laptops in some parts of the world. Although mobile devices continue to become better-provisioned, they are not as computationally powerful as desktop- or server-class machines in general. Even as the gap between desktop computer and smartphone diminishes (e.g., modern smartphones are equipped with decent GPUs to support their dense displays), mobile devices are still held back by power constraints and limited space available for functions such as efficient cooling. With this in mind, can trusted hardware be even more of a boon for secure computation on mobile devices?

Consider Samsung KNOX [138], which is Samsung's proprietary flavor of TEE for mobile devices that is built atop ARM TrustZone [135]. KNOX is advertised as the pinnacle of security on mobile devices, but Atamli-Reineh et al. [199] show that KNOX is not without its faults. (The work of Atamli-Reineh et al. [199] was published in 2016, and the vulnerabilities stated therein have probably since been patched. However, it serves as a reminder that the security of a TEE-based system depends on proper usage of the TEE. This point is echoed by Kanonov and Wool [200], who assert "TrustZone's mere existence is not enough.") In particular, Atamli-Reineh et al. point out several vulnerabilities of KNOX resulting from improper use of TEE, including:

(i) Exposure of KNOX clipboard data.

(ii) Exposure of audio channels used by applications running in KNOX space.

(iii) Exposure of KNOX-space SSL certificates.

Even when the TEE was used properly (e.g., processing of user input in the secure world), a bug in KNOX's eCryptFS allowed improper password verification for purposes of authentication. As a takeaway, Atamli-Reineh et al.'s study shows that it is difficult to get TEE usage right.

Indeed, both Qualcomm's QSEE (http://bits-please.blogspot.com/2016/04/exploring-qualcomms-secure-execution.html) and Trustonic's Kinibi (https://trustonic.com/solutions/trustonic-secured-platforms-tsp/), two major TEE implementations for Android, are at risk due to the large number of trustlets they load [201]. Any vulnerability found within a trustlet can be easily exploited due to limited or absent address space layout randomization (ASLR) for trustlets; lack of guard pages between global variables, heap, and stack; and lack of stack cookies in Kinibi. Compromising the TEE, in turn, allows an attacker to compromise the security of the entire device.

These identified vulnerabilities of Samsung KNOX (and other mobile TEE implementations) and the underlying TrustZone pose an obstacle to adoption of TEE-based SMC on mobile devices, as TrustZone is currently the primary mobile TEE offering. AMD Secure Technology [139], like KNOX, is also dependent on TrustZone, while technologies such as Intel TXT are only available on desktop- and server-class devices. Atamli-Reineh et al. propose SGX as an appropriate countermeasure for several of the KNOX vulnerabilities revealed by their study. This suggests an SGX parallel is desirable for mobile devices; however, it remains to be seen whether SGX-like solutions for mobile will be equally ineffective or introduce a slew of new problems. (Intel SGX is not currently available on mobile; its availability is limited to desktop- and server-class machines.)

Assuming it is possible to overcome the issues surrounding TEE use in the mobile environment, a TEE-based approach is certainly a promising direction for SMC on mobile. Reduced computational resource demand will allow mobile devices to handle more complicated SMC problems on-device and will likely pair well with existing techniques that leverage the cloud beyond the device.

The solutions presented in Section 2.6 required rewriting of cryptography-backed SMC protocols to cater to mobile devices. Regardless of the choice of mobile TEE instantiation, similar efforts will be needed to port TEE-based SMC solutions to mobile devices.

4.3. Challenge 3: Privacy-Preserving Computation. Secure multiparty computation is just one specific example drawn from a larger domain of privacy-preserving computation, which spans a wide range of applications (from genomics to data mining and set operations). Traditionally, applications in this space have relied on cryptography- or obfuscation-based techniques to provide the necessary privacy guarantees, but these other types of privacy-preserving computation may also benefit from alternative, hardware-based solutions. This hypothesis is supported by existing hardware-based approaches for privacy-preserving machine learning [178], membership tests [172], medical data management [187], set operations [183], and more.

Nevertheless, the coupling of hardware and privacy-preserving computation is still a relatively new direction of research that warrants further exploration. The specific research challenge here is twofold:

(1) Can TEEs offer a general means of converting functions into their privacy-preserving counterparts?

(2) Can these TEE-backed solutions outperform custom protocols without weakening security?

In the following, we examine select subdomains of privacy-preserving computation to determine how receptive they may be to integration of trusted hardware. Overall, this coupling appears to be a promising direction for a more principled building of privacy-preserving applications.

4.3.1. Genomic Privacy. Genomic data contains a wealth of information and is now more available than ever before due to decreasing costs and commoditization of genomic analysis. Genomic data supports domains ranging from personalized medicine to forensics. Some of the things that can be revealed by the genome include phenotypic traits, risk of contracting certain diseases, identity, and family relationships. Genomic data comes with significant privacy concerns, and there is no telling what additional information will be extractable from the genome in the future. Naveed et al. [202] identify several challenges specific to genomics:

(i) Consumer-driven genomics makes genomic data available outside controlled/well-regulated healthcare systems (i.e., freely available on the Internet).

(ii) Privacy is in contention with utility.

(iii) Privacy requirements vary depending on context. For example, quick access to accurate genomic data is needed in life-threatening situations; genetic testing is far more delay-tolerant.

(iv) Privacy mechanisms may fail to prevent association with identity information found on other platforms.

Privacy-preserving solutions for genomic computation should tackle these challenges while minimizing negative effects on performance or accuracy of computed results.

Prior work includes homomorphic encryption for secure storage/processing of genomic, clinical, and environmental data [203]; splitting raw genomic data into millions of short reads (each short read contains 100-400 nucleotides; requests for subsets of short reads do not reveal the nature of the genetic test they support) for finer-grained access [204]; and symbolic execution on sensitive user data [205]. Another direction of work builds privacy-preserving versions of the underlying techniques, some examples being: private edit distance [206, 207], Smith-Waterman similarity [206], and set-operation inspired techniques [208].

The addition of hardware may help further bolster these existing techniques or provide an alternative means for privacy to medical professionals not well-versed in security. TEEs may be especially desirable in this subdomain, given that despite the availability of SMC techniques for performing potentially interesting studies on genomic data, the application of SMC is largely hindered by current practices and legal restrictions. Secure processing of genomic data at trusted

TEE-supported servers with restricted output may be an approach more amenable to tight regulation.

4.3.2. Data Mining and Machine Learning. The availability of data is increasing, but this data is owned/managed by different principals who do not necessarily trust one another. Privacy-preserving techniques are needed to extract information from these large, heterogeneous collections of data. Mohassel et al. [209] propose privacy-preserving implementations of linear regression, logistic regression, and neural networks, while Lindell and Pinkas [210] demonstrate privacy-preserving decision tree-based data mining. In particular, Miyaji et al. [211] explore using the homomorphic property of Paillier encryption [189] together with the two-party secure computation scheme of Aumann et al. [19]; Chabanne et al. [212] present CryptoNets, which combines fully homomorphic encryption with neural networks.

Bogdanov et al. [72] design Rmind, a collection of algorithms supporting the complete statistical analysis of data from various sources in a privacy-preserving manner. The Rmind algorithms are claimed compatible with hardware isolation platforms such as SGX. Not just Rmind, but privacy-preserving machine learning on the whole, with its many parallels with general secure computation, has potential for improvement with the help of trusted hardware.

4.3.3. Miscellaneous. A number of privacy-preserving problems have been explored that do not fall under the major areas identified above. These include (this is not and does not attempt to be an exhaustive list): anonymous messaging [213], joint graph computation and distance metrics [214, 215], street navigation [216], data outsourcing and compliance checking [217], computation outsourcing [25], one-time programs [24], controlled functional encryption [218], and private cloud payments [219]. Private set operations may also benefit from coupling with hardware; these include intersection (PSI) [220, 221], union [221], pattern matching [220], and element reduction [221].

It would be interesting to see how many of these problems, if any, can benefit from the introduction of trusted hardware. One example of this coupling is given by Zhao et al. [222]; they investigate how TEEs may be used to realize one-time programs, ultimately building a system that uses Intel TXT in conjunction with the TPM. A unique and tailored approach to hardware-assisted computation may be necessary for each of the above problems, or it may be the case that several problems share characteristics that allow them to be tackled in a similar manner; such exploration would be an interesting direction for future work.

5. Conclusion

Secure multiparty computation refers to a powerful set of techniques that enable mutually distrusting parties to compute a common result without exposing sensitive inputs to one another. Although powerful, even the best circuit-based techniques remain impractical for most real-time online computations. This paper surveyed recent improvements in SMC techniques and explored trusted hardware (in the form of TEEs) as an enabler for further improvement; this direction is promising overall. Several challenges specific to the joining of SMC and TEEs were also addressed, these being (1) defeating malicious adversaries, (2) mobile friendly TEE-supported SMC, and (3) a more general coupling of trusted hardware and privacy-preserving computation.

Disclosure

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

Special thanks are due to Patrick Traynor and Thomas Shrimpton for their interest in and constructive criticisms of this work. This work is supported in part by the US National Science Foundation under grant CNS-1540217.

References

[1] A. C. Yao, “Protocols for secure computations,” in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS ’82)*, pp. 160–164, 1982.

[2] A. C. Yao, “How to generate and exchange secrets,” in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pp. 162–167, Toronto, Canada, October 1986.

[3] T. Araki, A. Barak, J. Furukawa et al., “Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier,” in *Proceedings of the 2017 IEEE Symposium on Security and Privacy, SP ’17*, IEEE, May 2017.

[4] M. Jõemets, “Use Sharemind to Build Location Services Without Breaking Privacy Laws,” 2015, https://sharemind.cyber.ee/location-services/.

[5] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the 41st annual ACM symposium on Theory of Computing (STOC ’09)*, pp. 169–178, ACM, Bethesda, Md, USA, 2009.

[6] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979.

[7] G. R. Blakley, “Safeguarding cryptographic keys,” in *Proceedings of the AFIPS National Computer Conference*, pp. 313–317, 1979.

[8] V. Costan and S. Devadas, “Intel SGX Explained,” Tech. Rep., 2016, Cryptology ePrint Archive, Report 2016/086, http://eprint.iacr.org, 2016.

[9] E. Bresson, D. Catalano, N. Fazio, A. Nicolosi, and M. Yung, “Output privacy in secure multiparty computation,” in *Proceedings of the 3rd Yet Another Conference on Cryptography (YACC ’06)*, 2006.

[10] C. Dwork, “Differential privacy: a survey of results,” in *Proceedings of the Annual Conference on Theory and Applications of Models of Computations (TAMC ’08)*, 2008.

[11] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan, “Computational Differential Privacy,” in *Proceedings of the 29th Annual International Cryptology Conference (CRYPTO ’09)*, 2009.

[12] P. Kairouz, S. Oh, and P. Viswanath, “Secure multi-party differential privacy,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS ’15)*, 2015.

[13] V. Bindschaedler, S. Rane, A. Brito, V. Rao, and E. Uzun, “Achieving differential privacy in secure multiparty data aggregation protocols on star networks,” in *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy (CODASPY ’17)*, 2017.

[14] X. He, A. MacHanavajjhala, C. Flynn, and D. Srivastava, “Composing differential privacy and secure computation: a case study on scaling private record linkage,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS ’17)*, 2017.

[15] A. Acar, Z. B. Celik, H. Aksu, A. S. Uluagac, and P. McDaniel, “Achieving secure and differentially private computations in multiparty settings,” in *Proceedings of the IEEE Symposium on Privacy-Aware Computing (PAC ’17)*, 2017.

[16] S. Mazloom and S. D. Gordon, “Secure computation with differentially private access patterns,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS ’18)*, 2018.

[17] M. Pettai and P. Laud, “Combining differential privacy and secure multiparty computation,” in *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC ’15)*, 2015.

[18] R. Cohen, I. Haitner, E. Omri, and L. Rotem, “From fairness to full security in multiparty computation,” in *Proceedings of the International Conference on Security and Cryptography for Networks (SCN ’18)*, 2018.

[19] Y. Aumann and Y. Lindell, “Security against covert adversaries: efficient protocols for realistic adversaries,” in *Proceedings of the Theory of Cryptography Conference (TCC ’07)*, 2007.

[20] D. Beaver, S. Micali, and P. Rogaway, “The round complexity of secure protocols,” in *Proceedings of the ACM Symposium on Theory of Computing (STOC ’90)*, 1990.

[21] M. O. Rabin, How to Exchange Secrets with Oblivious Transfer TR-81, Aiken Computation Lab, Harvard University, 1981.

[22] M. Bellare, V. T. Hoang, and P. Rogaway, “Foundations of garbled circuits,” in *Proceedings of the 2012 Conference on Computer and Communications Security (CCS ’12)*, ACM, 2012.

[23] M. Bellare, V. T. Hoang, and P. Rogaway, “Adaptively secure garbling with applications to one-time programs and secure outsourcing,” in *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT ’12)*, vol. 7658, 2012.

[24] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, “One-time programs,” in *Proceedings of the Annual International Cryptology Conference (CRYPTO ’08)*, 2008.

[25] S. Hohenberger and A. Lysyanskaya, “How to securely outsource cryptographic computations,” in *Proceedings of the 2nd International Conference on Theory of Cryptography (TCC ’05)*, 2005.

[26] B. Hemenway, Z. Jafargholi, R. Ostrovsky, A. Scafuro, and D. Wichs, “Adaptively secure garbled circuits from one-way functions,” in *Proceedings of the Annual International Cryptology Conference (CRYPTO ’16)*, 2016.

[27] Z. Jafargholi and D. Wichs, “Adaptive security of Yao’s garbled circuits,” in *Proceedings of the Theory of Cryptography Conference (TCC ’16)*, 2016.

[28] O. Goldreich, S. Micali, and A. Wigderson, “How to play ANY mental game or a completeness theorem for protocols with honest majority,” in *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC ’87)*, 1987.

[29] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” in *Foundations of Secure Computation*, pp. 169–180, 1978.

[30] M. Ito, A. Saito, and T. Nishizeki, “Secret sharing scheme realizing general access structure,” in *Proceedings of the IEEE Global Communications Conference (GLOBECOM ’87)*, 1987.

[31] J. Benaloh and J. Leichter, “Generalized secret sharing and monotone functions,” in *Proceedings of the Conference on the Theory and Application of Cryptography (CRYPTO)*, 1988.

[32] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: how to cope with perpetual leakage,” in *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, 1995.

[33] B. Choc, S. Goldwasser, S. Micali, and B. Awerbuch, “Verifiable secret sharing and achieving simultaneity in the presence of faults,” in *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (SFCS)*, 1985.

[34] P. Feldman, “A practical scheme for non-interactive verifiable secret sharing,” in *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pp. 427–437, IEEE, Los Angeles, Calif, USA, 1987.

[35] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC ’88)*, pp. 1–10, USA, May 1988.

[36] D. Chaum, C. Crépeau, and I. Damgård, “Multiparty unconditionally secure protocols,” in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC 1988*, pp. 11–19, USA, May 1988.

[37] R. Cramer, I. B. Damgard, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*, Cambridge University Press, Cambridge, Mass, USA, 2015.

[38] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, “Practical covertly secure MPC for dishonest majority –or: breaking the SPDZ limits,” in *Proceedings of the Annual International Cryptology Conference (CRYPTO ’12)*, 2012.

[39] N. Buescher, A. Weber, and S. Katzenbeisser, “Towards practical RAM based secure computation,” in *Proceedings of the European Symposium on Research in Computer Security (ESORICS ’18)*, 2018.

[40] M. Keller and A. Yanai, “Efficient maliciously secure multiparty computation for RAM,” in *Proceedings of the Annual International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT ’18)*, 2018.

[41] O. Goldreich and R. Ostrovsky, “Software protection and simulation on oblivious RAMs,” *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[42] V. Kolesnikov and T. Schneider, “Improved garbled circuit: Free XOR gates and applications,” in *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP 08)*, vol. 5126 of *Lecture Notes in Computer Science*, pp. 486–498, Springer, Berlin, Germany, 2008.

[43] V. Kolesnikov, P. Mohassel, and M. Rosulek, “FleXOR: Flexible garbling for XOR gates that beats free-XOR,” in *Proceedings of*

the Annual International Cryptology Conference (CRYPTO '14), 2014.

[44] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proceedings of the Annual International Cryptology Conference: (CRYPTO '03)*, vol. 2729 of *Lecture Notes in Computer Science*, pp. 145–161, Springer, 2003.

[45] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *Proceedings of the 1st ACM Conference on Electronic Commerce, EC 1999*, pp. 129–139, USA, November 1999.

[46] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT '09)*, vol. 5912 of *Lecture Notes in Computer Science*, pp. 250–267, Springe, Berlin. Germany, 2009.

[47] M. Kiraz and B. Schoenmakers, "A protocol issue for the malicious Case of Yao's garbled circuit construction," in *Proceedings of the 27th Symposium on Information Theory in the Benelux*, 2006.

[48] P. Mohassel and M. Franklin, "Efficiency tradeoffs for malicious two-party computation," in *Proceedings of the International Conference on Practice and Theory of Public Key Cryptography (PKC '06)*, vol. 3958 of *Lecture Notes in Computer Science*, pp. 458–473, Springer, Berlin, Germany, 2006.

[49] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Proceedings of the 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '07)*, vol. 4515 of *Lecture Notes in Computer Science*, pp. 52–78, Springer, Berlin, Germany, 2007.

[50] D. P. Woodruff, "Revisiting the efficiency of malicious two-party computation," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '07)*, vol. 4515 of *Lecture Notes in Computer Science*, pp. 79–96, Springer, Berlin, Germany, 2007.

[51] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," in *Proceedings of the Theory of Cryptography Conference (TCC '11)*, vol. 6597 of *Lecture Notes in Computer Science*, pp. 329–346, Springer, Berlin, Germany, 2011.

[52] S. a. Jarecki and V. Shmatikov, "Efficient two-party secure computation on committed inputs," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '07)*, vol. 4515 of *Lecture Notes in Computer Science*, pp. 97–114, Springer, Berlin, Germany, 2007.

[53] J. B. Nielsen and C. Orlandi, "LEGO for two party secure computation," in *Proceedings of the Theory of Cryptography Conference (TCC 09)*, vol. 5444 of *Lecture Notes in Computer Science book*, pp. 368–386, Springer, Berlin, Germany, 2009.

[54] Y. Ishai, M. Prabhakaran, and A. Sahai, "Founding cryptography on oblivious transfer – efficiently," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '08)*, vol. 5157 of *Lecture Notes in Computer Science*, pp. 572–591, Springer, Berlin, Germany, 2008.

[55] Y. Lindell, "Fast cut-and-choose based protocols for malicious and covert adversaries," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '13)*, vol. 8043 of *Lecture Notes in Computer Science*, pp. 1–17, Springer, 2013.

[56] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '13)*, vol. 8043 of *Lecture Notes in Computer Science*, pp. 18–35, Springer, 2013.

[57] Y. Huang, J. Katz, and D. Evans, "Quid Pro Quo-tocols: strengthening semi-honest protocols with dual execution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12)*, IEEE, 2012.

[58] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay–a secure two-party computation system," in *Proceedings of the USENIX Security Symposium*, 2004.

[59] A. Iliev and S. W. Smith, "Small, stupid, and scalable: secure computing with faerieplay," in *Proceedings of the ACM Workshop on Scalable Trusted Computing (STC '10)*, pp. 41–51, USA, October 2010.

[60] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proceedings of the USENIX Security Symposium*, 2011.

[61] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith, "Secure two-party computations in ANSI C," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, pp. 772–783, October 2012.

[62] S. Kamara, P. Mohassel, and B. Riva, "Salus: a system for server-aided secure function evaluation," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, pp. 797–808, October 2012.

[63] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, "Frigate: a validated, extensible, and efficient compiler and interpreter for secure computation," in *Proceedings of the IEEE European Symposium on Security and Privacy (Euro SP)*, pp. 112–127, IEEE, Saarbrucken, Germany, March 2016.

[64] V. Kolesnikov, J. B. Nielsen, M. Rosulek, N. Trieu, and R. Trifiletti, "DUPLO: unifying cut-and-choose for garbled circuits," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pp. 3–20, November 2017.

[65] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: a framework for fast privacy-preserving computations," in *Proceedings of the European Symposium on Research in Computer Security (ESORICS '08)*, 2008.

[66] W. Henecka, A.-R. Sadeghi, T. Schneider, I. Wehrenberg et al., "TASTY: tool for automating secure two-party computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, pp. 451–462, USA, October 2010.

[67] L. Malka, "VMCrypt: modular software architecture for scalable secure computation," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 715–724, USA, October 2011.

[68] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, "SCAPI: the secure computation application programming interface," Cryptology ePrint Archive, Report 2012/629, 2013.

[69] D. Bogdanov, P. Laud, and J. Randmets, "Domain-polymorphic programming of privacy-preserving applications," in *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security (PLAS '14)*, 2014.

[70] D. Demmler, T. Schneider, and M. Zohner, "ABY-a framework for efficient mixed-protocol secure two-party computation," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '15)*, 2015.

[71] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "ObliVM: a programming framework for secure computation," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '15)*, IEEE, 2015.

[72] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk, "Rmind: a tool for cryptographically secure statistical analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 3, 2018.

[73] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider, "HyCC: compilation of hybrid protocols for practical secure computation," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, 2018.

[74] R. Zhu, D. Cassel, A. Sabry, and Y. Huang, "NANOPI: extreme-scale actively-secure multi-party computation," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, pp. 862–879, ACM, Toronto, Canada, October 2018.

[75] A. Aly, M. Kelly, D. Rotaru et al., "SCALE and MAMBA," https://github.com/KULeuven-COSIC/SCALE-MAMBA, 2018.

[76] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '11)*, pp. 169–188, 2011.

[77] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '12)*, 2012.

[78] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: general-purpose compilers for secure multi-party computation," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '19)*, IEEE, 2019.

[79] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh, "Secure function evaluation with ordered binary decision diagrams," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '06)*, 2006.

[80] B. Kreuter, B. Mood, A. Shelat, and K. Butler, "PCF: a portable circuit format for scalable two-party secure computation," in *Proceedings of the USENIX Security Symposium*, 2013.

[81] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *Proceedings of the USENIX Security Symposium*, 2012.

[82] T. Schneider and M. Zohner, "GMW vs. Yao? efficient secure two-party computation with low depth circuits," in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC '13)*, 2013.

[83] C. Hazay, E. Orsini, P. Scholl, and E. Soria-Vazquez, "TinyKeys: a new approach to efficient multi-party computation," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '18)*, 2018.

[84] G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '17)*, 2017.

[85] A. Shelat and C.-H. Shen, "Fast two-party secure computation with minimal assumptions," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '13)*, 2013.

[86] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proceedings of the 32nd Annual International Cryptology Conference (CRYPTO '12)*, 2012.

[87] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole: reducing data transfer in garbled circuits using half gates," in *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '15)*, vol. 9057, 2015.

[88] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '12)*, 2012.

[89] S. Garg and A. Srinivasan, "Two-round multiparty secure computation from minimal assumptions," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '18)*, 2018.

[90] F. Benhamouda and H. Lin, "k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '18)*, 2018.

[91] S. Garg, P. Miao, and A. Srinivasan, "Two-round multiparty secure computation minimizing public key operations," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '18)*, 2018.

[92] P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain, "Round-optimal secure multiparty computation with honest majority," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '18)*, 2018.

[93] S. Halevi, C. Hazay, A. Polychroniadou, and M. Venkitasubramaniam, "Round-optimal secure multi-party computation," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '18)*, 2018.

[94] Y. Ishai, M. Mittal, and R. Ostrovsky, "On the message complexity of secure multiparty computation," in *Proceedings of the IACR International Workshop on Public Key Cryptography (PKC '18)*, 2018.

[95] B. Mood, D. Gupta, K. R. B. Butler, and J. Feigenbaum, "Reuse it or lose it: more efficient secure computation through reuse of encrypted values," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '14)*, 2014.

[96] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable Garbled Circuits and Succinct Functional Encryption," in *Proceedings of the 45th Annual Symposium on Theory of Computing (STOC '13)*, 2013.

[97] S. Agrawal, "Stronger security for reusable garbled circuits, general definitions and attacks," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '17)*, 2017.

[98] J. Garay, Y. Ishai, R. Ostrovsky, and V. Xikas, "The price of low communication in secure multi-party computation," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '17)*, 2017.

[99] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen, "On robust combiners for oblivious transfer and other primitives," in *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '05)*, 2005.

[100] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein, "High-throughput secure three-party computation for malicious adversaries and an honest majority," in *Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '17)*, 2017.

[101] P. Mohassel, M. Rosulek, and Y. Zhang, "Fast and secure three-party computation: the garbled circuit approach," in *Proceedings*

of the 22nd ACM Conference on Computer and Communications Security (CCS '15), 2015.

[102] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '17)*, 2017.

[103] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang, "Optimizing authenticated garbling for faster secure two-party computation," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '18)*, 2018.

[104] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC '85)*, 1985.

[105] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC '91)*, 1991.

[106] K. Eldefrawy, R. Ostrovsky, S. Park, and M. Yung, "Proactive secure multiparty computation with a dishonest majority," in *Proceedings of the International Conference on Security and Cryptography for Networks (SCN '18)*, 2018.

[107] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: computing without simultaneous interaction," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '11)*, 2011.

[108] D. Harnik, Y. Ishai, and E. Kushilevitz, "How many oblivious transfers are needed for secure multiparty computation?" in *Proceedings of the Annual International Cryptology Conference (CRYPTO '07)*, 2007.

[109] S. G. Choi, A. Elbaz, T. Malkin, and M. Yung, "Secure multi-party computation minimizing online rounds," in *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT '09)*, 2009.

[110] B. Mood, L. Letaw, and K. Butler, "Memory-efficient garbled circuit generation for mobile devices," in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC '12)*, 2012.

[111] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor, "For your phone only: Custom protocols for efficient secure function evaluation on mobile devices," *Security and Communication Networks*, vol. 7, no. 7, pp. 1165–1176, 2014.

[112] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," in *Proceedings of the USENIX Security Symposium*, 2013.

[113] H. Carter, C. Lever, and P. Traynor, "Whitewash: outsourcing garbled circuit generation for mobile devices," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC '14)*, 2014.

[114] H. Carter, B. Mood, P. Traynor, and K. Butler, "Outsourcing secure two-party computation as a black box," in *Proceedings of the International Conference on Cryptology and Network Security (CANS '15)*, 2015.

[115] P. Bogetoft, D. L. Christensen, I. Damgård et al., "Secure multi-party computation goes live," in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC '09)*, 2009.

[116] N. Nipane, I. Dacosta, and P. Traynor, "'Mix-in-Place' anonymous networking using secure function evaluation," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC '11)*, 2011.

[117] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, "How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation," in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC '15)*, 2015.

[118] A. Lapets, E. Dunton, K. Holzinger, F. Jansen, and A. Bestavros, Web-Based Multi-Party Computation with Application to Anonymous Aggregate Compensation Analytics, Computer Science Department, Boston University, 2015.

[119] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *ACM SIGKDD Explorations Newsletter*, vol. 4, no. 2, pp. 28–34, 2002.

[120] B. Hemenway, S. Lu, R. Ostrovsky, and W. Welser, "High-precision secure computation of satellite collision probabilities," in *Proceedings of the International Conference on Security and Cryptography for Networks (SCN '16)*, 2016.

[121] D. W. Archer, D. Bogdanov, Y. Lindell et al., "From keys to databases – real-world applications of secure multi-party computation," *The Computer Journal*, vol. 6, no. 12, pp. 1749–1771, 2018.

[122] R. Wyden, M. Rubio, and M. Warner, "Student right to know before you go act of 2017," United States Congress S.B. 2169/H.B. 4479, 2017.

[123] K. Tõldsepp, P. Pruulmann-Vengerfeldt, and P. Laud, "Deliverable d1.2: requirements specification based on the interviews," UaESMC project, 2012, http://usable-security.eu/files/d12final.pdf.

[124] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[125] M. van Dijk and A. Juels, "On the impossibility of cryptography alone for privacy-preserving cloud computing," in *Proceedings of the USENIX Summit on Hot Topics in Security (HotSec '10)*, 2010.

[126] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A secure and reliable bootstrap architecture," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '97)*, IEEE, 1997.

[127] R. Mori and M. Kawahara, "Superdistribution: the concept and the architecture," *Transactions of the Institute of Electronics, Information, and Communication Engineers*, vol. E73, no. 7, pp. 1133–1146, 1990.

[128] R. Mori and M. Kawahara, "Superdistribution: an electronic infrastructure for the economy of the future," *Transactions of Information Processing Society of Japan*, vol. 38, no. 7, pp. 1465–1472, 1997.

[129] J. G. Dyer, M. Lindemann, R. Perez et al., "Building the IBM 4758 secure coprocessor," *The Computer Journal*, vol. 34, no. 10, pp. 57–66, 2001.

[130] Trusted Computing Group, "TPM main specification version 1.2: part 1 design principles," 2011, https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf.

[131] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proceedings of the USENIX Security Symposium*, 2004.

[132] S. Berger, R. Cáceres, K. A. Goldman et al., "vTPM: virtualizing the trusted platform module," in *Proceedings of the 15th USENIX Security Symposium*, 2006.

[133] H. Raj, S. Saroiu, A. Wolman et al., "fTPM: a software-only implementation of a TPM chip," in *Proceedings of the 25th USENIX Security Symposium*, 2016.

[134] J. Greene, "Intel Trusted Execution Technology," Intel Technology White Paper, 2012.

[135] ARM Ltd, Building a Secure System using TrustZone Technology, 2009.

[136] Trusted Computing Group, "TCG Mobile Trusted Module Specification Version 1.0 Revision 6," 2008, https://trustedcomputinggroup.org/wp-content/uploads/TCG-Mobile-Trusted-Module-Specification-V1-R6-26-June-2008.pdf.

[137] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing (STC '08)*, 2008.

[138] Samsung Research America, Samsung Knox Security Solution, version 2.2, 2017.

[139] Advanced Micro Devices, "AMD secure technology," 2018, https://www.amd.com/en/technologies/security.

[140] Advanced Micro Devices, "AMD64 virtualization technology: secure virtual machine architecture reference manual rev. 3.02," 2005.

[141] Advanced Micro Devices, Secure encrypted virtualization API Version 0.16: technical preview, Advanced Micro Devices, 2018.

[142] D. Kaplan, J. Powell, and T. Woller, AMD Memory Encryption, Advanced Micro Devices, 2016.

[143] R. Pass, E. Shi, and F. Tramèr, "Formal abstractions for attested execution secure processors," in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '17)*, 2017.

[144] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Proceedings of the IACR Theory of Cryptography Conference (TCC '07)*, 2007.

[145] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi, "Foundations of hardware-based attested computation and application to SGX," in *Proceedings of the IEEE European Symposium on Security and Privacy (Euro SP)*, 2016.

[146] B. Parno, "Bootstrapping Trust in a 'Trusted' Platform," in *Proceedings of the USENIX Summit on Hot Topics in Security (HotSec '08)*, 2008.

[147] J. Lind, C. Priebe, D. Muthukumaran et al., "Glamdring: automatic application partitioning for intel SGX," in *Proceedings of the USENIX Annual Technical Conference (ATC '17)*, 2017.

[148] R. Sinha, S. Rajamani, S. A. Seshia, and K. Vaswani, "Moat: verifying confidentiality of enclave programs," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, 2015.

[149] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: deterministic side channels for untrusted operating systems," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '15)*, 2015.

[150] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: using SGX to conceal cache attacks," in *Proceedings of the 14th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '17)*, 2017.

[151] W. Wang, G. Chen, X. Pan et al., "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pp. 2421–2434, USA, November 2017.

[152] M. Schwarz, M. Lipp, and D. Gruss, "Meltdown: reading kernel memory from user space," in *Proceedings of the USENIX Security Symposium*, 2018.

[153] P. Kocher, J. Horn, A. Fogh et al., "Spectre attacks: exploiting speculative execution," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '19)*, IEEE, 2019.

[154] J. Van Bulck, M. Minkin, O. Weisse et al., "Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[155] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi, "The guard's dilemma: efficient code-reuse attacks against intel SGX," in *Proceedings of the 27th USENIX Security Symposium*, 2018.

[156] M. Morbitzer, M. Huber, J. Horsch, and S. Wessel, "SEVered: subverting AMD's virtual machine encryption," in *Proceedings of the 11th European Workshop on Systems Security (EuroSec '18)*, 2018.

[157] Z.-H. Du, Z. Ying, Z. Ma et al., "Secure Encrypted Virtualization is Unsecure," ArXiv e-prints, 2017.

[158] H. Dang, T. T. Dinh, E. Chang, and B. C. Ooi, "Privacy-preserving computation with trusted computing via scramble-then-compute," in *Proceedings of the Privacy Enhancing Technologies Symposium (PETS '17)*, 2017.

[159] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: an execution infrastructure for TCB minimization," in *Proceedings of the 3rd ACM European Conference on Computer Systems (EuroSys '08)*, 2008.

[160] J. M. McCune, Y. Li, N. Qu et al., "TrustVisor: efficient TCB reduction and attestation," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '10)*, IEEE, 2010.

[161] J. Criswell, N. Dautenhahn, and V. Adve, "Virtual ghost: protecting applications from hostile operating systems," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14)*, 2014.

[162] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, 2014.

[163] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "SeCReT: secure channel between rich execution environment and trusted execution environment," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '15)*, 2015.

[164] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: a distributed sandbox for untrusted computation on secret data," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016.

[165] S. Arnautov, B. Trach, F. Gregor et al., "SCONE: secure linux containers with intel SGX," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016.

[166] D. J. Tian, J. I. Choi, G. Hernandez, P. Traynor, and K. R. B. Butler, "A practical intel SGX setting for linux containers in the cloud," in *Proceedings of the 9th ACM Conference on Data and Application Security and Privacy (CODASPY '19)*, 2019.

[167] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-SGX: a practical library OS for unmodified applications on SGX," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017.

[168] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: low-TCB Linux applications with SGX enclaves," in *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[169] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: minimal hardware extensions for strong software isolation," in *Proceedings of the 25th USENIX Security Symposium*, 2016.

[170] L. Guen, P. Liu, X. Xing et al., “TrustShadow: secure execution of unmodified applications with ARM TrustZone,” in *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys ’17)*, 2017.

[171] Y. Cho, J. Shin, D. Kwon et al., “Hardware-assisted on-demand hypervisor activation for efficient security critical code execution on mobile devices,” in *Proceedings of the USENIX Annual Technical Conference (ATC ’16)*, 2016.

[172] S. Tamrakar, J. Liu, A. Paverd, J.-E. Ekberg, B. Pinkas, and N. Asokan, “The circle game: scalable private membership test using trusted hardware,” in *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS ’17)*, 2017.

[173] D. Gupta, B. Mood, J. Feigenbaum, K. Butler, and P. Traynor, “Using intel software guard extensions for efficient two-party secure function evaluation,” in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC ’16)*, 2016.

[174] R. Bahmani, M. Barbosa, F. Brasser et al., “Secure multiparty computation from SGX,” in *Proceedings of the 21st International Conference on Financial Cryptography and Data Security (FC ’17)*, 2017.

[175] S. Kim, Y. Shin, J. Ha, T. Kim, and D. Han, “A first step towards leveraging commodity trusted execution environments for network applications,” in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets ’15)*, 2015.

[176] F. Schuster, M. Costa, C. Fournet et al., “VC3: trustworthy data analytics in the cloud using SGX,” in *Proceedings of the IEEE Symposium on Security and Privacy (SP ’15)*, IEEE, 2015.

[177] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, “Iron: Functional Encryption using Intel SGX,” IACR eprint, 2016.

[178] O. Ohrimenko, F. Schuster, C. Fournet et al., “Oblivious multi-party machine learning on trusted processors,” in *Proceedings of the 25th USENIX Security Symposium*, 2016.

[179] J. Liu, W. Li, G. O. Karame, and N. Asokan, “Scalable Byzantine Consensus via Hardware-assisted Secret Sharing,” arXiv.org, 2017.

[180] A. Kurnikov, K. Krawiecka, A. Paverd, M. Mannan, and N. Asokan, “Using safekeeper to protect web passwords,” in *Proceedings of The Web Conference (WWW ’18)*, 2018.

[181] C. Priebe, K. Vaswani, and M. Costa, “EnclaveDB: a secure database using SGX,” in *Proceedings of the IEEE Symposium on Security and Privacy (SP ’18)*, IEEE, 2018.

[182] S. Sasy, S. Gorbunov, and C. W. Fletcher, “ZeroTrace: oblivious memory primitives from intel SGX,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS ’18)*, 2018.

[183] M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, and T. Visconti, “Secure set intersection with untrusted hardware tokens,” in *Proceedings of the Cryptographers’ Track at the RSA Conference (CT-RSA ’11)*, 2011.

[184] C. Hazay and Y. Lindell, “Constructions of truly practical secure protocols using standard smartcards,” in *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS ’08)*, 2008.

[185] S. Bajaj and R. Sion, “TrustedDB: a trusted hardware-based database with privacy and data confidentiality,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.

[186] A. Arasu, S. Blanas, K. Eguro et al., “Orthogonal security with cipherbase,” in *Proceedings of the Conference on Innovative Data Systems Research (CIDR ’13)*, 2013.

[187] M. Canim, M. Kantarcioglu, and B. Malin, “Secure management of biomedical data with cryptographic hardware,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 1, pp. 166–175, 2012.

[188] S. S. Roy, F. Vercauteren, J. Vliegen, and I. Verbauwhede, “Hardware assisted fully homomorphic function evaluation and encrypted search,” *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1562–1572, 2017.

[189] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT ’99)*, 1999.

[190] K. Järvinen, V. Kolesnikov, A. Sadeghi, and T. Schneider, “Garbled circuits for leakage-resilience: hardware implementation and evaluation of one-time programs,” in *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES ’10)*, 2010.

[191] N. Husted, S. Myers, A. Shelat, and P. Grubbs, “GPU and CPU parallelization of honest-but-curious secure two-party computation,” in *Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC ’13)*, 2013.

[192] G. Vasiliadis, E. Athanasopoulos, M. Polychronakis, and S. Ioannidis, “PixelVault: using GPUs for securing cryptographic operations,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS ’14)*, 2014.

[193] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, “GhostRider: a hardware-software system for memory trace oblivious computation,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’15)*, 2015.

[194] K. Nayak, C. W. Fletcher, L. Ren et al., “HOP: hardware makes obfuscation practical,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS ’17)*, 2017.

[195] E. Owusu, J. Guajardo, J. McCune, J. Newsome, A. Perrig, and A. Vasudevan, “OASIS: on achieving a sanctuary for integrity and secrecy on untrusted platforms,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS ’13)*, 2013.

[196] M. Green, S. Hohenberger, and B. Waters, “TRESOR runs encryption securely outside RAM,” in *Proceedings of the USENIX Security Symposium*, 2011.

[197] E. Blass and W. Robertson, “TRESOR-HUNT: attacking CPU-bound encryption,” in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC ’12)*, 2012.

[198] T. Ward, J. I. Choi, K. Butler, J. M. Shea, P. Traynor, and T. F. Wong, “Privacy preserving localization using a distributed particle filtering protocol,” in *Proceedings of the 2017 IEEE Military Communications Conference, MILCOM ’17*, pp. 835–840, USA, October 2017.

[199] A. Atamli-Reineh, R. Borgaonkar, R. A. Balisane, G. Petracca, and A. Martin, “Analysis of trusted execution environment usage in samsung KNOX,” in *Proceedings of the 1st Workshop on System Software for Trusted Execution (SysTEX ’16)*, 2016.

[200] U. Kanonov and A. Wool, “Secure Containers in Android: the Samsung KNOX Case Study,” ArXiv e-prints, 2016.

[201] G. Beniamini, “Trust Issues: Exploiting TrustZone TEEs,” Google Project Zero Blog, 2017.

[202] M. Naveed, E. Ayday, E. W. Clayton et al., “Privacy in the genomic era,” *ACM Computing Surveys*, vol. 48, no. 1, 2015.

[203] E. Ayday, J. L. Raisaro, U. Hengartner, A. Molyneaux, and J. Hubaux, “Privacy-preserving processing of raw genomic data,”

in *Proceedings of the International Workshop on Data Privacy Management (DPM '14)*, 2014.

[204] E. Ayday, J. L. Raisaro, P. J. McLaren, J. Fellay, and J.-P. Hubaux, "Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data," in *Proceedings of the USENIX Security Workshop on Health Information Technologies (HealthTech '13)*, 2013.

[205] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong, "Privacy-preserving genomic computation through program specialization," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '09)*, 2009.

[206] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '08)*, IEEE, 2008.

[207] X. S. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu, "Efficient genome-wide, privacy-preserving similar patient query based on private edit distance," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '15)*, 2015.

[208] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering GATTACA: efficient and secure testing of fully-sequenced human genomes," in *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*, 2011.

[209] P. Mohassel and Y. Zhang, "SecureML: a system for scalable privacy-preserving machine learning," in *Proceedings of the IEEE Symposium on Security and Privacy (SP '17)*, IEEE, 2017.

[210] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '00)*, 2000.

[211] A. Miyaji and M. S. Rahman, "Privacy-preserving data mining in presence of covert adversaries," in *Proceedings of the International Conference on Advanced Data Mining and Applications (ADMA '10)*, vol. 6440, 2010.

[212] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," in *Proceedings of the Real World Crypto Symposium (RWC '17)*, 2017.

[213] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "MCMix: anonymous messaging via secure multiparty computation," in *Proceedings of the USENIX Security Symposium*, 2017.

[214] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *Proceedings of the International Conference on Theory and Applications of Cryptology and Information Security (ASIACRYPT '05)*, 2005.

[215] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner, "GSHADE: Faster Privacy-Preserving Distance Computation and Biometric Identification," in *Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security*, 2014.

[216] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, "Privacy-preserving shortest path computation," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '16)*, 2016.

[217] G. Di Crescenzo, J. Feigenbaum, D. Gupta, E. Panagos, J. Perry, and R. N. Wright, "Practical and privacy-preserving policy compliance for outsourced data," in *Proceedings of the Workshop on Applied Homomorphic Cryptography (WAHC '14)*, 2014.

[218] M. Naveed, S. Agrawal, M. Prabhakaran et al., "Controlled functional encryption," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '14)*, 2014.

[219] M. Pirker, D. Slamanig, and J. Winter, "Practical privacy preserving cloud resource-payment for constrained clients," in *Proceedings of the International Symposium on Privacy Enhancing Technologies (PETS '12)*, 2012.

[220] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *Proceedings of the International Conference on Theory of Cryptography (TCC '08)*, 2008.

[221] L. Kissner and D. Song, "Privacy-preserving set operations," in *Proceedings of the Annual International Cryptology Conference (CRYPTO '05)*, 2005.

[222] L. Zhao, J. I. Choi, D. Demirag et al., "One-time programs made practical," in *Proceedings of the International Conference on Financial Cryptography and Data Security (FC '19)*, 2019.