MEG: Memory and Energy Efficient Garbled Circuit Evaluation on Smartphones

Qing Yang, Ge Peng, Paolo Gasti, *Member, IEEE,* Kiran S. Balagani, *Member, IEEE,* Yantao Li, *Member, IEEE,* and Gang Zhou, *Senior Member, IEEE*

Abstract—**Garbled circuits are a general tool that allows two parties to compute any function without disclosing their respective inputs. Applications of this technique vary from distributed privacy-preserving machine learning tasks, to secure outsourced authentication. Unfortunately, the energy cost of garbled circuit evaluation protocols is substantial. This limits the applicability of garbled circuits in scenarios that involve battery-operated devices, such as Internet of things (IoT) devices and smartphones.**

In this paper, we propose MEG, a Memory- and Energy-efficient Garbled circuit evaluation mechanism. MEG utilizes batch data transmission and multi-threading to reduce memory and energy consumption. We implement MEG on an Android smartphone, and compare its performance and energy consumption to state-of-the-art techniques using two garbled circuits of widely different sizes (AES-128, and 256-bit edit distance). Our results show that, compared to "plain" garbled circuit evaluation, MEG decreases memory consumption by more than 90%. When compared with current pipelined garbled circuit evaluation techniques, MEG's energy usage was 42% lower for AES-128, and 23% lower for EDT-256. Further, our multi-thread implementation of MEG decreased circuit evaluation time by up to 56.7% for AES-128, and by up to 13.5% for EDT-256, compared to state-of-the-art pipelining techniques.

I. Introduction

Secure computation techniques allow two or more parties to compute a joint function of their (private) inputs without disclosing any information besides the output of the function. This model of computation enables secure realization of countless functionalities, including electronic voting [1], privacy-preserving data mining [2], and outsourced biometric-based authentication [3], [4].

As Internet of things (IoT) devices become ubiquitous, they increasingly collect, generate, and process sensitive information. Therefore, the ability to run secure computation protocols on these low-power, often battery-operated devices has become an important factor towards their adoption. For example, when an IoT device is part of a crowdsourcing system which involves exchanging sensitive information among mutually untrusted parties, secure computation protocols can be used to achieve the system goal without disclosing any user-specific private information [5], [6].

(Qing Yang and Ge Peng contributed equally to this work.)

Q. Yang, G. Peng and G. Zhou are with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23185, USA (e-mail: qyang@cs.wm.edu; gpeng@cs.wm.edu; gzhou@cs.wm.edu).

P. Gasti and K. S. Balagani are with the School of Engineering and Computing Sciences, New York Institute of Technology, New York, NY 10023, USA (e-mail: pgasti@nyit.edu; kbalagan@nyit.edu).

Y. Li is with the College of Computer & Information Science, Southwest University, Chong Qing 400715, China (email: yantaoli@swu.edu.cn).

Garbled circuit evaluation [7] is a popular technique in the case of secure two-party computation. With garbled circuits, one party (the *generator*) transforms a function into a Boolean circuit composed of wired binary gates. The generator then "garbles" this circuit by assigning encryption keys to represent 0 or 1 values to each wire. The circuit generator sends the garbled circuit, as well as its garbled input, to the other party (the *evaluator*). The evaluator computes the output keys and then decodes the function output.

For most non-trivial functions, garbled circuits are composed of a large number of gates. For example, a typical AES circuit implementation consists of more than 30,000 gates [8], while 256-bit Levenshtein distance (edit distance) implementations typically require more than 9 million gates. Because of the sheer number of gates in a circuit, the evaluation process requires substantial computation and data transmission. This causes significant memory and energy costs. On systems with limited RAM, battery capacity, and computing power, such as IoT devices and smartphones, garbled circuit evaluation has only become practical—at least in limited scenarios—in recent years [9]. However, current techniques tend to impose severe tradeoffs between memory usage, computation time, and energy consumption. To make garbled circuits practical for resource-constrained devices, it is crucial to use techniques that are simultaneously efficient in all these aspects.

To address this issue, in this paper we propose Memory- and Energy-efficient Garbled (MEG), a novel garbled circuit evaluation method that provides energy- and memory-efficient fast evaluation of garbled circuits. MEG combines the advantages of two popular garbled circuit evaluation techniques: pipelined, and non-pipelined circuit evaluation. With non-pipelined techniques, the evaluator stores the entire garbled circuit in memory before starting evaluation. Because the size of garbled circuits can be significant even for fairly simple functionalities, non-pipelined techniques use a substantial amount of memory during circuit evaluation. To reduce memory usage, pipelined techniques (e.g., [10], [11], [12], [13]) rely on the generator to send garbled gate one at a time, as soon as they have been garbled. The evaluator only needs to keep in memory one gate at a time. As a result, the pipelined evaluation of garbled circuits requires a very small amount of memory. Unfortunately, pipelined techniques tend to consume substantially more energy than non-pipelined constructions, as shown by our evaluation (see Section V). This is because data is continuously sent from the generator to the evaluator, which prevents the processor and network interface (i.e., the WiFi module) of the evaluator from entering low power states.

To reduce energy and memory consumption, while decreasing circuit evaluation time, MEG utilizes batch transmission and evaluation of circuit gates, and multi-threaded execution. MEG is designed to carefully arrange computation and communication to enable the smartphone's network interface to enter power saving mode between consecutive batches, and to maximize data rate while the network interface is in use. To our knowledge, MEG is the first technique that applies variable-window batching to garbled circuit evaluation.

To evaluate MEG, we implemented a garble circuit evaluation app on an Android smartphone, and circuit generation on a laptop. We report our results on various batch sizes, and compared the resulting memory and energy consumption with current circuit evaluation methods. Further, we compared single-threaded and multi-threaded implementations of MEG. Our evaluation shows that MEG uses up to 42% less energy than state-of-the-art pipelined implementations. Compared to non-pipelined implementations, MEG uses between 8% and 28% more energy. However, while non-pipelined circuit evaluation requires an amount of memory proportional to the size of the circuit, MEG uses the same amount of memory regardless of the functionality being evaluated, making it suitable for devices with limited resources. Further, MEG reduces the protocol running time compared to non-pipelined execution by up to 57%. These factors make MEG better suited for low-memory IoT devices with slow CPUs compared to current implementations.

Without loss of generality, in this paper, we assume that the output of the circuit must be disclosed to the battery-operated device. For this reason, we focus on reducing memory usage, energy cost, and computation for the circuit evaluator. When the output of the computation must be disclosed only to the circuit constructor, techniques such as [14] can be used with no changes to MEG.

Organization. The rest of this paper is structured as follows. In Section II we briefly summarize garbled circuit construction and evaluation, and review the main problems of current implementations. In Section III, we review related work. Section IV introduces the system design of MEG. In Section V we present the evaluation of MEG on smartphones. We conclude the paper in Section VI.

II. Background and Motivation

In this section, we summarize garbled circuit construction and evaluation. We then discuss pipelined and non-pipelined garbled circuit protocols. We assume that the circuit generator and evaluator are malicious, i.e., that they can both arbitrarily deviate from the correct execution of the protocol [15]. The techniques discussed in the rest of the paper are secure against malicious adversaries.

A. Garbled Circuits Construction and Evaluation

Garbled circuit protocols involve two mutually untrusted parties: the circuit generator (Alice), with private input x, and the circuit evaluator (Bob), with private input y. Bob wants to compute $z = f(x, y)$, where f is a mutually agreed upon function. However, neither Bob nor Alice are willing to reveal information on their inputs, beyond input size and what can be leant from z. The basic garbled circuit evaluation protocol calculates $f(x, y)$ using the steps illustrated in Figure 1. A simplified description of each block is presented next.

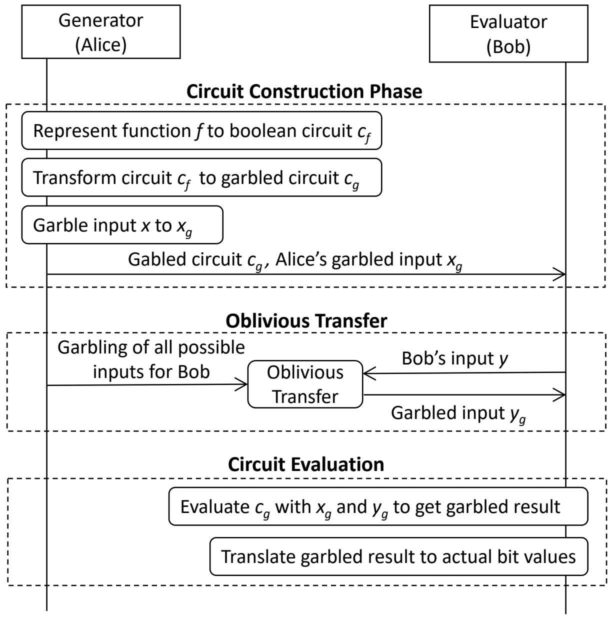

Generator (Alice)
Evaluator (Bob)
Circuit Construction Phase
Represent function f to boolean circuit c_f
Transform circuit c_f to garbled circuit c_g
Garble input x to x_g
Gabled circuit c_g, Alice's garbled input x_g
Oblivious Transfer
Garbling of all possible inputs for Bob
Oblivious Transfer
Bob's input y
Garbled input y_g
Circuit Evaluation
Evaluate c_g with x_g and y_g to get garbled result
Translate garbled result to actual bit values

Fig. 1: Basic garbled circuit evaluation process.

Circuit Construction Phase. In this phase, Alice performs the following steps to construct a garbled representation of the function that she wants to evaluate on her input, as well as Bob's:

1) Alice represents f as a Boolean circuit c_f. Each circuit gate is represented by a truth table consisting of four entries. Every entry describes the Boolean values for the gate's two input wires (*input columns*) and one output wire (*output column*).
2) Alice garbles circuit c_f to obtain garbled circuit c_g. For each gate of c_f, Alice generates an encryption key for the two values that each input or output wire can assume. She then encrypts the output column of the truth table using the corresponding input wire keys. Alice then shuffles the entries in the garbled truth table.
3) Alice garbles each bit of her input by representing it using the encryption key of the corresponding input wire, as defined by c_g. Let x_g be the result of this process.
4) Alice sends c_g and x_g to Bob

Intuitively, because c_g is computed independently from Alice's input, it does not disclose any private information to Bob.

Oblivious Transfer Phase. Bob receives garbled version (y_g) of his input by running an instance of the oblivious transfer protocol [16] for each of his input bit. The input of each oblivious transfer protocol instance is a pair of keys (k_0, k_1) from Alice, and a selector bit b from Bob, while the output is k_b. Disclosing k_b via oblivious transfer guarantees that k_{1-b} is not disclosed to Bob (and therefore Bob cannot compute

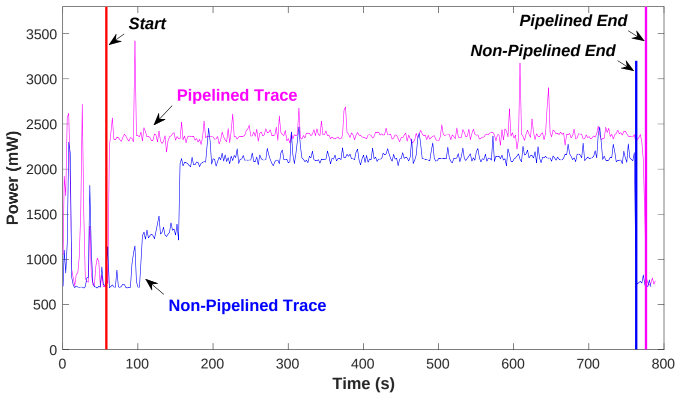


Fig. 2: Power traces of non-pipelined vs. pipelined garbled circuit evaluation of 256-bit edit distance on a smartphone. Values between *Start* and *End* refer to circuit evaluation.

the output of f for any input other than y), and that b is not disclosed to Alice.

Circuit Evaluation Phase. Bob evaluates each gate in c_g using the elements of x_g and y_g as decryption keys. He then decodes the final output keys into the corresponding bit value. Because Bob is unable to evaluate the circuit on any input other than x and y, and because he cannot "decode" any intermediate result, the protocol discloses only $z = f(x, y)$ to Bob.

Security Model. In this paper, we consider garbled circuit protocol constructions secure against malicious adversaries. There are many attacks that a malicious circuit generator might try to implement in order to alter the result of the computation, or to learn information about the evaluator's input. These attacks include: making arbitrary undetectable changes to the circuit [17], generator's input inconsistency attack [20], selective failure attack [20], and the circuit generator's output authenticity attack [20].

Security against malicious circuit generators guarantees that these and other attacks cannot be performed. It can be achieved using cut-and-choose [18]. With this technique, Alice constructs multiple garbled versions of the circuit. Bob randomly selects a subset of these circuits, and challenges Alice to prove that these circuits are garbled correctly. If Alice is able to prove that all circuits are correct, Bob evaluates the remaining circuits, and uses the majority of the evaluation results as the circuit output.

B. Pipelined and Non-pipelined Evaluation

In non-pipelined implementations (e.g., [19]), circuit evaluation does not start until the whole circuit is loaded into the evaluator's memory. The problem with this approach is that even relatively simple functions can lead to very large circuits, and therefore devices with limited memory cannot evaluate most functionalities.

To address this issue, pipelined techniques can be used to evaluate garbled circuits while loading in memory a single gate at a time. Generation and evaluation of the garbled gates are interleaved in time: when the evaluator receives one garbled gate, it finds the corresponding gate of the circuit and begins to evaluate it as soon as the necessary inputs are available. Gates are deleted from memory immediately after they are evaluated. As a result, the amount of memory required to evaluate a garbled circuit using pipelining is small, and does not depend on the size of the circuit.

Unfortunately, on modern embedded systems, pipelined circuit evaluation requires substantially more energy than the non-pipelined method. This is because continuously receiving data at relatively low rates (due to the cost of performing garbling and evaluation) keeps both the CPU and the network interface in high power states for most of the computation. (This is confirmed in our evaluation, shown in Section V-B.) In contrast, without pipelining, transmission and evaluation of the circuit are performed in distinct phases. This allows network and CPU to be either fully active, and thus operating at maximum efficiency, or to be idling in a low-power state. Figure 2 shows this phenomenon during the evaluation of a garbled circuit corresponding to the 256-bit Levenshtein distance using non-pipelined and pipelined methods. In our experiments, the resulting average power consumption was 30.6% higher when using the pipelining. Further, the evaluation of a circuit using pipelining took longer than the evaluation of the same circuit without using pipelining. This is because with pipelining each gate of the garbled circuit is requested separately once the evaluator is ready to process it. Therefore, the evaluator must wait a full round-trip time for each gate. Even with low-latency LANs, this delay has a meaningful impact on the evaluation time of non-trivial circuits.

We provide further details on three representative garbled circuit implementations [13], [20], [21] in Section III.

III. Related Work

Previous research on energy optimizations investigated how bursting affects energy consumption in WiFi and LTE (see, for instance, [22], [23]). There is a substantial amount of prior work on garbled circuits and their optimizations. In what follows, we present three representative implementations of garbled circuit evaluation protocols. We then summarize the main results in the areas of *general* garbled circuit optimization and *smartphone-specific* optimization.

Representative Garbled Circuit Implementations. To our knowledge, there are three representative garbled circuit implementations: Huang et al. [13], Kreuter et al. [20], and Obliv-C [21]. In contrast with MEG, none of these techniques has been optimized for energy efficiency. Next, we provide a brief comparison of these techniques.

Huang et al. [13] introduced a Java-based garbled circuit implementation that leverages several known and new optimizations for improving the running time and memory requirements of garbled circuits evaluation. These optimizations include fast m-to-n garbled gates to speed up table lookups and pipelining. Their results show that these optimizations lead to significantly faster garbled circuit implementations (between 16 and 4,176 times faster than previous implementations), which are able to scale to very large circuits. [13] also provides an extensive circuit library, which allows developers to build a new circuit by composing and extending highly-optimized sub-circuits. Finally, an important feature of [13] is its small and clean codebase, which consists of about 1,500 lines of code for the main framework, and about 700 lines of code implementing the circuit library.

Kreuter et al. [20] implemented a highly optimized garbled circuit evaluation framework, suitable for the evaluation of billion-gate circuits. This result relies on several key optimizations, including: (1) algorithmic optimizations for circuit evaluation and for cut-and-choose; (2) heavy parallelization of circuit evaluation, achieved using Message Passing Interface (MPI). This interface enables efficient use of massively parallel infrastructures, including cluster computers; (3) pipelining; and (4) the use of hardware-accelerated cryptographic instructions, such as AES-NI, in contrast with [13] which relies on SHA-1 for the encryption of gates. The framework of Kreuter et al. [20] relies on a circuit compiler, which allows developers to implement the circuit in a higher-level programming language rather than by combining smaller circuits as with [13].

Obliv-C [21] is a clever extension to the C programming language targeted at implementing data-oblivious protocols. It supports all C features, and adds control structures that enable data-oblivious computation. For instance, the oblivious conditional statement (obliv if) implements branching on conditions that are not known even at runtime because they are based on private inputs. As a result, Obliv-C enables programmers that do not have a deep background in cryptographic protocols and circuit design to implement secure protocols for complex functions. Performance-wise, Obliv-C is competitive with [20], despite the lack of optimizations such as the use of AES-NI instructions [21].

General Optimization of Garbled Circuit Protocol. Kolesnikov et al. [24] introduced a garbled circuit construction method in which garbled XOR gates are replaced with plain XOR operations. Therefore, XOR gates in a circuit are evaluated "for free", i.e., without requiring cryptographic operations. This technique also reduces communication cost of circuit evaluation, because no information needs to be sent to the evaluator with respect to XOR gates.

Goyal et al. [25] proposed a method where the generator uses a random seed to construct the circuits used for cut-and-choose, and later only sends the seeds corresponding to some of the circuits instead of the entire circuits to the evaluator. This technique substantially reduces communication cost.

Pinkas et al. [8] analyzed several algorithmic improvements to the original garbled circuits protocol, including garbled row reduction and free XOR under non-correlation robust KDF. They tested these optimizations using the AES-128 circuit, and showed that evaluation of large circuits is indeed feasible.

Lindell et al. [18] presented an optimized garbled circuit evaluation protocol based on cut-and-choose to achieve security against malicious adversaries. Their protocol is more efficient than prior protocols based on cut-and-choose, and requires a smaller number of circuits to be evaluated to achieve the same level of security, compared to prior work.

Garbled Circuit Optimization for Smartphones. Huang et al. [9] introduced a Java-based secure computation framework for Android. Their results show that processing power, rather than network bandwidth, represents the biggest performance bottleneck for secure computation on smartphones.

Mood et al. [26] developed a new memory-efficient technique for generating garbled circuits. They used the standard SFDL language for describing secure functions as input, and designed a new pseudo-assembly language and a template-driven compiler to generate circuit. The evaluation results on Android devices showed up to 95.6% memory overhead reduction for circuits implementing 2-party set intersection protocols.

Šeděnka et al. [14] designed privacy-preserving protocols for scaled Manhattan and scaled Euclidean verifiers, which are secure against malicious clients and honest-but-curious servers, and do not require cut-and-choose. In their protocol, the circuit constructor learns the output of the protocol.

Carter et al. [27] created a new SFE protocol that allows mobile devices to securely outsource the majority of computation required to evaluate a garbled circuit. Their protocol contained a new outsourced oblivious transfer primitive that requires significantly less bandwidth and computation than standard OT primitives, as well as an outsourced input validation technique that guarantees that the cloud is executing the protocol correctly.

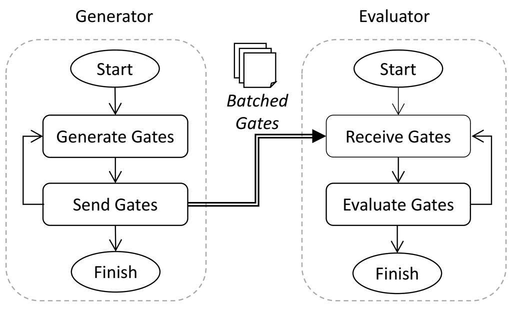
Gasti et al. [3] proposed an outsourced privacy-preserving protocol for continuous authentication. The goal of their protocol is to minimize energy consumption during circuit evaluation by offloading most of the computation from the circuit evaluator (typically a smartphone) to an untrusted cloud infrastructure. As a result, their protocol requires about 1,000 times less energy compared to evaluating the garbled circuit entirely on the smartphone. While these improvements are impressive, [3] achieves them by requiring the use of a third party (the cloud), which must not collude with the circuit constructor. In contrast, MEG reduces the cost of circuit evaluation phase *on the smartphone*, and as such does not rely on any third party. Further, MEG and [3] are *composable*, i.e., MEG can be used in the circuit execution phase of [3] in order to reduce circuit evaluation time and energy for the cloud.

IV. Design of MEG

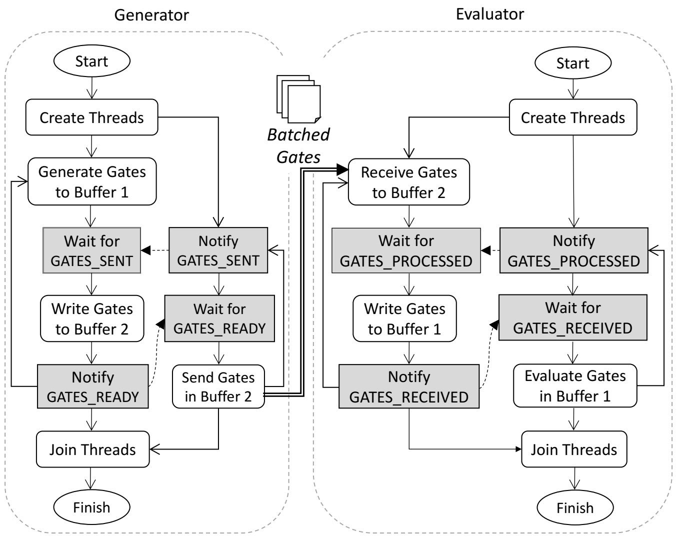
We use three techniques to reduce energy consumption and circuit evaluation time in MEG: gates batching, multi-threading, and slow start.

A. Gates Batching

With gates batching, the circuit generator garbles several gates at once, and stores them in a buffer. When the buffer is full, its entire content is sent to the evaluator. The evaluator maintains a buffer of the same size, and uses it to store and process a batch of incoming gates at once. When all gates in a batch have been evaluated, the evaluator empties the buffer and requests a new batch. The benefits of gates batching are two-fold. First, because the size of the buffer is fixed (i.e., it does not depend on the size of the circuit), the amount of memory required for circuit evaluation can be set arbitrarily, based on the evaluator's resources. Second, batch transmission reduces energy consumption, because CPU and network operate on



(a) Single-thread MEG



(b) Multi-thread MEG

Fig. 3: Workflows of single-thread and multi-thread MEG.

larger chunks of computation and communication compared to standard pipelining. This allows the evaluator to operate more efficiently, because neither the network nor the CPU are throttled in a high-power state while they wait to receive or process data.

B. Multi-Threading

With single-threaded implementations of garbled circuits (such as [13]), the generator performs garbling and communication in the same thread. Similarly, the evaluator receives and decrypts each gate using one thread. Figure 3a presents the single-thread version of MEG. The problem with this approach is that, although data transmission and circuit garbling/evaluation could run in parallel, they are instead forced to run sequentially. As a result, circuit evaluation takes longer than necessary—especially on devices with more than one CPU core—while saving virtually no additional energy.

Our multi-threaded implementation of MEG addresses this issue as follows. The circuit generator and the evaluator create two threads. On the generator, one thread is used for gates generation, and the other thread is for gates transmission. These two threads use two signals (GATES_READY and GATES_SENT) to notify each other about current running status and synchronize. On the evaluator, one thread is used for network communication, and the other for gates evaluation. These threads use two signals (GATES_RECEIVED and GATES_PROCESSED) to synchronize. This allows the two threads to operate simultaneously, thus keeping the network and the CPU consistently operating at consistent loads.

C. Slow Start

One issue with processing gates in batches is that if the batch is very large (say, greater than 16MB), the evaluator must pause for a relatively long amount of time before it can start processing the first batch. For instance, the evaluator might have to wait for about five seconds while it receives the first 16MB batch at 22.7MBps (the average LTE connection speed in the United States [28] in 2017). To address this issue, we designed MEG to leverage the slow-start technique used by the TCP protocol. With slow start, we set the initial batch size to 0.25MB, and double it for subsequent batches until it reaches a pre-determined value. For instance, for batch sizes of 16MB, the size of the first few batches is 0.25MB, 0.5MB, 1MB, etc. With this technique, the evaluator only needs to wait for about 0.1 second on the same LTE connection before starting the evaluation of the first batch of gates.

V. Evaluation

In this section, we present the results of our evaluation of MEG. Our experiments focused on energy, memory consumption, and execution time of MEG with respect to the circuit evaluator. This is because the circuit generation is usually performed on a system that is not energy- or memory constrained, while the evaluation is run on a smartphone in several common scenarios [3], [14]. [1]

Evaluation Setup. All experiments were performed using a Samsung Galaxy S4 smartphone as circuit evaluator, and a ThinkPad T440p laptop as the circuit generator. The Samsung Galaxy S4 has a 1.9GHz quad-core Qualcomm Snapdragon CPU, and 2 GB of RAM. The ThinkPad T440p has a 2.4GHz Intel Core i7 CPU, 12 GB of RAM, and runs Ubuntu 16.04. For our evaluation, we created a controlled 802.11n WiFi network using a Netgear WNDR3700v2 router as the access point. Both the smartphone and the laptop were connected to the access point via WiFi.

Implementation Details. We based our implementation of MEG on the code of Kreuter et al. [20]. This code provides a C++ implementation of pipelined and non-pipelined garbled circuits. However, because the code is highly optimized for the Intel x86 architecture, we ported it to ARM/Android systems as a native application using the Android NDK framework. The porting process included the following steps. (1) We cross-compiled the dependent libraries (PBC, GMP, OpenSSL etc.) to the ARM processor used by Android phones. (2) The original C++ code relies on the Intel SSE instruction set to support Single Instruction, Multiple Data (SIMD) parallel computation. Because SSE instructions are not available on ARM CPUs, we re-wrote all the SSE code using ARM NEON

[1]Although the security model of [3] assumes non-collusion between some of the parties involved in the computation (namely, the server and the Cloud), the technique presented in our paper directly applies to the protocol in [3].

TABLE I: Information on the circuits used in our evaluation.

	Circuit	
	AES-128	EDT-256
Number of binary gates	34,136	9,959,904
XOR gates (% of all gates)	22,546 (66%)	7,228,604 (73%)
Non-XOR gates (% of all gates)	11,590 (34%)	2,731,300 (27%)
Circuit size	343.83kB	80.27MB

instructions for SIMD. (3) Because Android does not support Message Passing Interface (MPI) for parallel computing, we modified the code from [20] that uses MPI by re-implementing all the required functionalities on the smartphone. (4) Finally, we used Android NDK to build the C++ code into an Android shared object, and created an Android Java application that interfaced to the shared object implementing [20] using Android JNI.

To implement MEG, we made the following modifications to the code in [20]. To implement gates batching, we modified the circuit constructor to split the whole circuit into batches. We modified the evaluator accordingly, in order to process each batch at once, and added signaling code that enabled the evaluator to notify the constructor when it consumed a batch.

We then implemented the sender and receiver using multiple threads: one thread for constructing/evaluating the circuit, and one for sending/receiving garbled gates. (See Figure 3 for a visual representation of the differences between single-threaded and multi-thread MEG.) To achieve this, we used the C++ classes `unique_lock` and `mutex` for shared data locking, `lock_guard` for lock ownership, and `notify_one` for communication between threads.

In order to minimize waiting time at the beginning of the circuit evaluation process, we also implemented a slow start mechanism. This mechanism resulted in variable batch sizes. As a result, we included batch size as part of the metadata corresponding to each batch. A further benefit of this modification is that it enabled us to implement a more efficient gate encoding. In [20], the encoding of each garbled gate includes the size of the gate. This is redundant, because both parties know what each gate in the circuit is, and therefore only need to know either how many gates are sent, or the total size of the gates being sent in a batch. (We employed the latter in our implementation because it substantially simplified our code.) We indicate the implementation which removes individual gate sizes from the gate encoding as "pipelined+".

To summarize, our comparison included the following techniques: (1) non-pipelined (the evaluator receives the entire circuit before the evaluation begins); (2) pipelined (the evaluator receives one gate at a time, and evaluates it immediately); (3) pipelined+ (same as pipelined, with the exception that gates are sent using an efficient encoding); (4) single-thread MEG; and (5) multi-thread MEG. For both single-thread MEG and multi-thread MEG, we considered six batch sizes: 0.5MB, 1MB, 2MB, 4MB, 8MB, and 16MB.

To measure the energy consumption of the smartphone, we used a Monsoon Power Monitor [29]. In our experiments, the smartphone's battery was removed, and the smartphone was powered solely by the Power Monitor. We recorded power consumption data at a rate of 5kHz.

Circuits. We evaluated MEG on two circuits: (1) AES-128, which is a circuit that implements the AES block cipher with a 128-bit key. The generator's input is a 128-bit string representing an AES key, and the evaluator's input is a 128-bit block of data; and (2) EDT-256, which is a circuit that implements the Levenshtein distance (edit distance). The generator's and the evaluator's inputs are 256-bit strings. Table I reports the number of gates and the size of the two garbled circuits. In addition to be common choices for evaluating secure two-party protocols, these circuits are characterized by very different ratios between their size and the number of their inputs. Specifically, the non-XOR-gates-to-input-size ratio is 45:1 for the AES circuit, and 5,334:1 for the EDT-256 circuit.

Because MEG affects only the circuit evaluation phase of the garbled circuit protocol, our analysis focused exclusively on this phase. We reported the energy consumption and elapsed time on the smartphone using different evaluation methods. To measure the *net* energy consumption for circuit evaluation, we subtracted the average power consumption of the smartphone when idle from the energy used during the evaluation of the garbled circuit. Due to the low variance between experiments, five protocol runs for each circuit/technique pair were sufficient to obtain reliable measurements.

Cut-and-choose requires the evaluation of multiple circuits to achieve security against malicious adversaries. The number of circuits to be evaluated depends on the cut-and-choose technique used, and on the security parameter. For instance, with a circuit constructor's cheating probability of 2^{-40} (the security parameter) the total number of circuits is 125 using the technique in [30], and 40 using the technique in [31]. For this reason, and without loss of generality, we report the average cost of the circuit evaluation phase for a single evaluation instance. Because the impact of MEG applies to each circuit evaluation *independently*, the relative gains presented in our analysis apply when using any cut-and-choose technique.

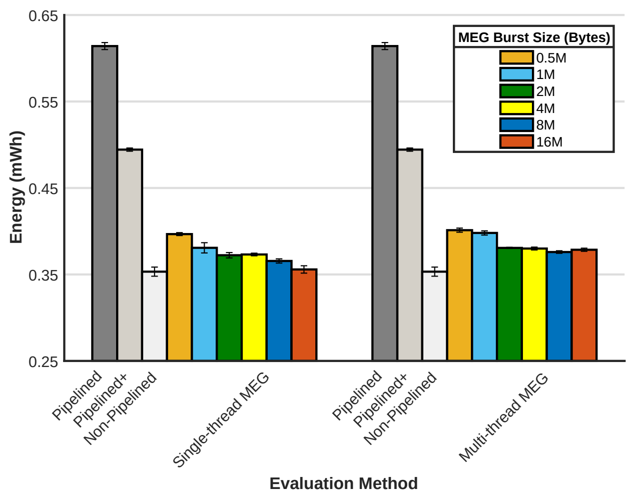
A. Results

Energy consumption. The energy consumption of all five methods is shown in Figure 4. Tables II and III list the relative energy savings of MEG with respect to the other circuit evaluation techniques. Our results show that MEG is able to offer a substantial reduction in energy consumption compared to state-of-the-art techniques. Compared to pipelined evaluation, MEG reduces energy consumption by 35% to 42% for AES-128, and by 18% to 23% for EDT-256. Compared to pipelined+, MEG decreases energy consumption by 19% to 28% for AES-128, and by 8% to 14% for EDT-256.

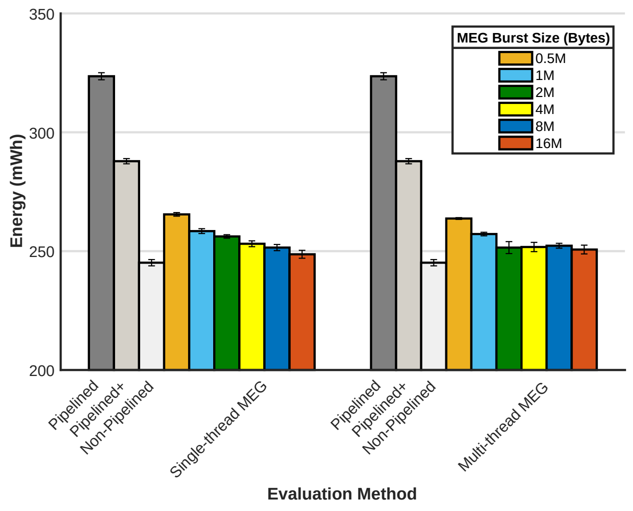
With batch sizes larger than 2MB, the energy consumption of MEG is very close to that of non-pipelined evaluation—with only 1% to 8% additional overhead for AES-128, and 1% to 5% additional overhead for EDT-256.

Single-thread MEG and multi-thread MEG have similar energy consumption. With both, an increase in batch size leads to a decrease in energy consumption.

Execution Time. The execution time of all circuit evaluation methods is shown in Figure 5. Execution time for multi-thread

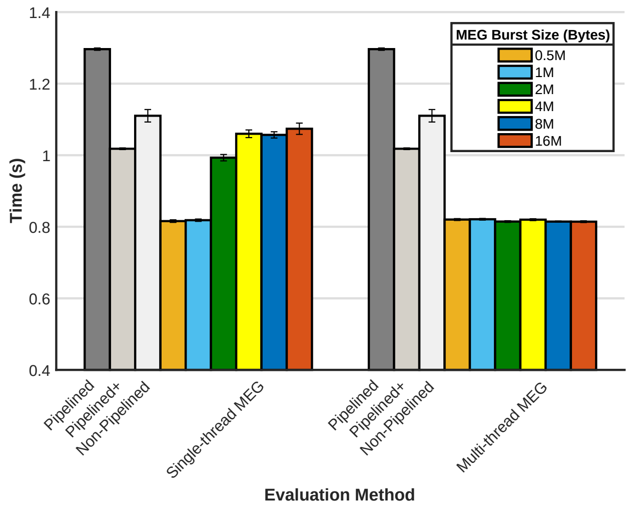


(a) AES-128

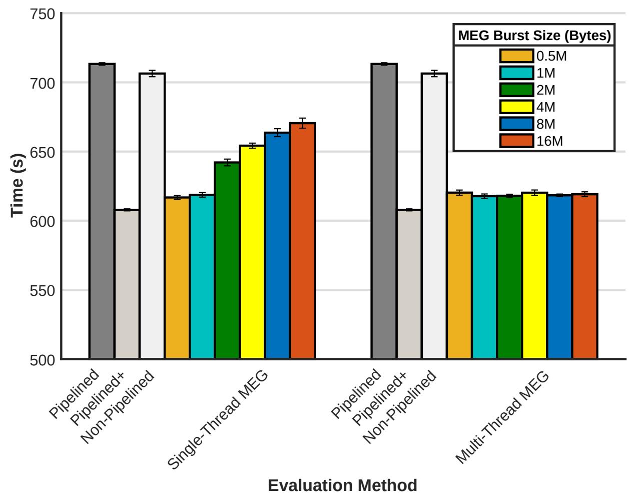


(b) EDT-256

Fig. 4: Evaluator's energy consumption for the evaluation phase of a single garbled circuit instance using MEG with different batch sizes, in comparison with other methods.



(a) AES-128



(b) EDT-256

Fig. 5: Evaluator's execution time for the evaluation phase of a single garble circuit instance using MEG with different batch sizes, in comparison with other methods.

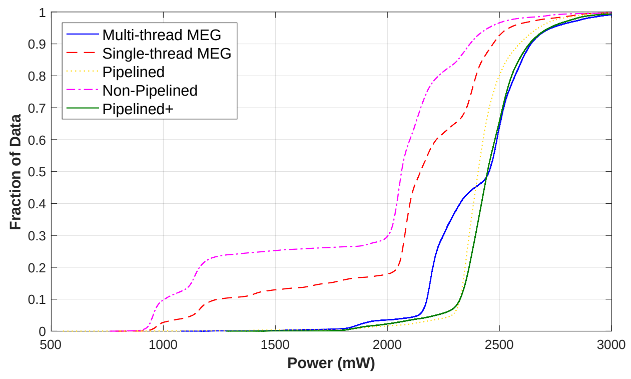
MEG is consistent across all the different batch sizes. Multi-thread MEG is significantly faster than all other evaluation methods for AES-128. In particular, it is 56.7% faster than pipelined evaluation, 25.8% faster than non-pipelined evaluation, and 19.8% faster than pipelined+ evaluation. With EDT-256, multi-thread MEG is 13.5% faster than pipelined evaluation, and 12.6% faster than non-pipelined evaluation. However, it is 1.5% slower than pipelined+ evaluation. This is primarily due to the relatively higher number of XOR gates in the EDT-256 circuit (73% of the circuit gates are XOR gates, as listed in Table I) compared to the AES circuit (66% of the circuit gates).

With single-thread MEG, circuit evaluation time increases significantly as the batch size increases. When the batch size increases above 2MB, the execution time increases by 21.6% to 31.5% for AES-128, and by 3.8% to 8.4% for EDT-256. Circuit evaluation with single-thread MEG is slower than with multi-thread MEG in all settings.

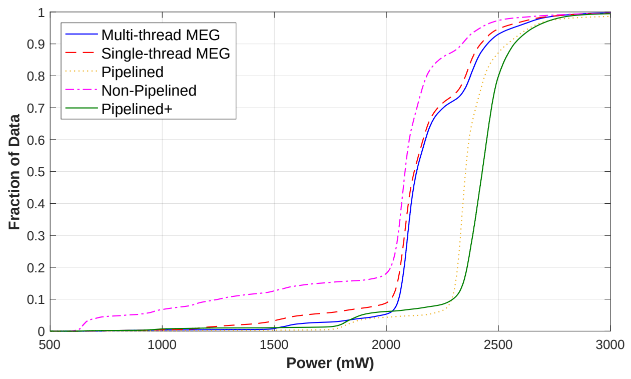
Memory consumption. The memory consumption of MEG is determined by the batch size, which ranges from 0.5MB to 16MB in our evaluation. Compared with the non-pipelined implementation, MEG decreases the memory consumption by 97.5% (from 80.27MB to 2MB) for EDT-256 without incurring in a meaningful increase in energy consumption.

B. *Results Discussion*

MEG's energy efficiency is explained by the efficient utilization of CPU and networking resources. Specifically, these resources are either used at their most efficient setting, or they are completely idle for a relatively long period of time. The latter enables both the CPU and the network interface to enter



(a) AES-128



(b) EDT-256

Fig. 6: Empirical cumulative distribution function (CDF) of power consumption during circuit evaluation (not including oblivious transfer). The batch size for MEG is 2MB.

TABLE II: Energy cost of MEG compared to other techniques for the evaluation phase of the AES-128 circuit. Negative values represent energy savings of MEG compared to Pipelined and Pipelined+. Positive values represent energy overhead compared to Non-pipelined.

(a) Single-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-35%	-38%	-39%	-39%	-40%	-42%
Pipelined+	-20%	-23%	-25%	-25%	-26%	-28%
Non-Pipelined	+12%	+8%	+5%	+6%	+4%	+1%

(b) Multi-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-35%	-35%	-38%	-38%	-39%	-38%
Pipelined+	-19%	-19%	-23%	-23%	-24%	-23%
Non-Pipelined	+14%	+13%	+8%	+8%	+6%	+7%

TABLE III: Energy cost of MEG compared to other techniques for the evaluation phase of the EDT-256 circuit. Negative values represent energy savings of MEG compared to Pipelined and Pipelined+. Positive values represent energy overhead compared to Non-Pipelined.

(a) Single-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-18%	-20%	-21%	-22%	-22%	-23%
Pipelined+	-8%	-10%	-11%	-12%	-13%	-14%
Non-Pipelined	+8%	+5%	+5%	+3%	+3%	+1%

(b) Multi-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-18%	-21%	-22%	-22%	-22%	-23%
Pipelined+	-8%	-11%	-13%	-13%	-12%	-13%
Non-Pipelined	+8%	+5%	+3%	+3%	+3%	+2%

power-saving mode for a substantial amount of the circuit evaluation time, without affecting the duration of the protocol execution.

Figure 6 shows the cumulative distribution function (CDF) of power consumption values during circuit evaluation for AES-128 and EDT-256. Compared to pipelined and pipelined+ circuit evaluation, MEG consistently reduces the amount of time spent in high power state (i.e., above 2000 mW), with the exception of the range above 2400 mW for multi-thread MEG and AES-128. This is due to short bursts of high-energy networking and computation performed simultaneously on MEG's two threads.

Multi-thread MEG has a lower execution time compared to single-thread MEG with batch sizes above 2MB. This is explained by Figure 7, which shows the power traces of single-thread and multi-thread MEG, recorded while evaluating EDT-256 using 16MB batch size. Throughout the evaluation of the circuit, single-thread MEG has multiple drops in energy consumption, which occur at the end of the evaluation of a batch of gates, i.e., when the evaluator receives the next gates batch. While this behavior helps to reduce energy consumption, because the CPU can stay at a low power state for a prolonged amount of time, it also increases circuit evaluation time. In contrast, with multi-thread MEG the CPU is continuously decrypting gates, thus reducing overall evaluation time.

We also evaluated the time and energy consumption of MEG when considering the cost of oblivious transfer (see Table IV). The resulting energy savings were almost identical to the energy savings obtained in the circuit evaluation stage alone (presented in Table III), primarily due to the relative cost of circuit evaluation and oblivious transfer.

Because our optimizations are independent of the specific garbled circuit implementation, our evaluation is indicative of the benefits of MEG on other pipelined garbled circuit implementations, e.g., Obliv-C [21] and the implementation of Huang et al. [9].

VI. Conclusion

In this paper, we present MEG, a memory- and energy-efficient garbled circuit evaluation technique specifically designed for embedded devices, such as IoT devices and smartphones. To reduce energy use during circuit evaluation, MEG relies on batch data transmission, multi-threading execution, and slow-start. Our experiments show that MEG is able to

TABLE IV: Energy cost of MEG compared to other techniques for both the oblivious transfer and the circuit evaluation phases of the EDT-256 circuit. Negative values represent energy savings of MEG compared to Pipelined and Pipelined+. Positive values represent energy overhead compared to Non-Pipelined.

(a) Single-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-17%	-19%	-20%	-21%	-22%	-23%
Pipelined+	-8%	-10%	-11%	-12%	-13%	-14%
Non-Pipelined	+8%	+6%	+5%	+4%	+3%	+1%

(b) Multi-thread MEG

	Batch Size (MB)					
	0.5	1	2	4	8	16
Pipelined	-18%	-20%	-22%	-22%	-22%	-22%
Pipelined+	-9%	-10%	-13%	-13%	-13%	-13%
Non-Pipelined	+7%	+5%	+3%	+3%	+3%	+2%

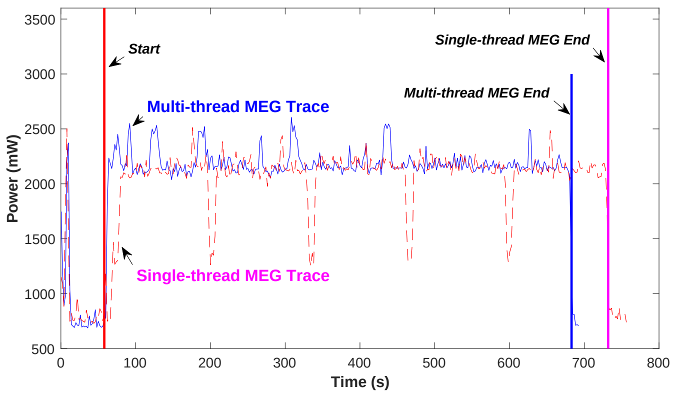


Fig. 7: Power traces of single-thread vs. multi-thread MEG. The circuit used in this evaluation is EDT-256, and the batch size is 16MB. Values between *Start* and *End* correspond to circuit evaluation.

simultaneously provide the advantages of both non-pipelined circuit evaluation (low energy consumption and low evaluation time), and pipelined circuit evaluation (scalability, as the amount of memory required to evaluate a circuit does not depend on the circuit size).

Compared to non-pipelined circuit evaluation, MEG reduces memory consumption by 97.5% for EDT-256 when the batch size is 2MB. With respect to energy consumption, MEG requires 42% less energy for AES-128 and up to 23% less energy for EDT-256 than state-of-the-art pipelined evaluation techniques. Further, compared to non-pipelined circuit evaluation, MEG shows a very small additional energy overhead (between 1% and 8%). Additionally, the multi-threading implementation of MEG significantly reduces the circuit evaluation time on smartphones compared to previous techniques.

References

[1] A. C. Yao, "Protocols for secure computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, ser. SFCS '82. Washington, DC, USA: IEEE Computer Society, 1982, pp. 160–164. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1982.88

[2] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining." *IACR Cryptology ePrint Archive*, vol. 2008, p. 197, 2008. [Online]. Available: http://dblp.uni-trier.de/db/journals/iacr/iacr2008.html#LindellP08a

[3] P. Gasti, J. Šeděnka, Q. Yang, G. Zhou, and K. S. Balagani, "Secure, fast, and energy-efficient outsourced authentication for smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2556–2571, 2016.

[4] M. Blanton and P. Gasti, *Secure and Efficient Protocols for Iris and Fingerprint Identification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 190–209. [Online]. Available: https://doi.org/10.1007/978-3-642-23822-2_11

[5] S. Nandha Premnath and Z. J. Haas, "Supporting privacy of computations in mobile big data systems," *Future Internet*, vol. 8, no. 2, 2016. [Online]. Available: http://www.mdpi.com/1999-5903/8/2/17

[6] G. Zhuo, Q. Jia, L. Guo, M. Li, and P. Li, "Privacy-preserving verifiable set operation in big data for cloud-assisted mobile crowdsourcing," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 572–582, April 2017.

[7] A. C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, ser. SFCS '86. Washington, DC, USA: IEEE Computer Society, 1986, pp. 162–167. [Online]. Available: https://doi.org/10.1109/SFCS.1986.25

[8] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 250–267.

[9] Y. Huang, P. Chapman, and D. Evans, "Privacy-preserving applications on smartphones." in *HotSec*, 2011.

[10] W. Henecka, S. K ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Tasty: Tool for automating secure two-party computations," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 451–462. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866358

[11] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, *Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 383–397. [Online]. Available: https://doi.org/10.1007/978-3-642-15031-9_26

[12] L. Malka, "Vmcrypt: Modular software architecture for scalable secure computation," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 715–724. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046787

[13] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security Symposium*, vol. 201, no. 1, 2011.

[14] J. Šeděnka, S. Govindarajan, P. Gasti, and K. S. Balagani, "Secure outsourced biometric authentication with performance evaluation on smartphones," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 2, pp. 384–396, 2015.

[15] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[16] M. O. Rabin, "How to exchange secrets with oblivious transfer," 2005, harvard University Technical Report 81 talr@watson.ibm.com 12955 received 21 Jun 2005. [Online]. Available: http://eprint.iacr.org/2005/187

[17] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 52–78.

[18] ——, "Secure two-party computation via cut-and-choose oblivious transfer," *Journal of cryptology*, vol. 25, no. 4, pp. 680–722, 2012.

[19] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay—a secure two-party computation system," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 20–20. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251375.1251395

[20] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries." in *USENIX Security Symposium*, vol. 12, 2012, pp. 285–300.

[21] S. Zahur and D. Evans, "Obliv-c: A language for extensible data-oblivious computation." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1153, 2015.

[22] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New

York, NY, USA: ACM, 2010, pp. 107–122. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814446

[23] S. Deng and H. Balakrishnan, "Traffic-aware techniques to reduce 3g/lte wireless energy consumption," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 181–192. [Online]. Available: http://doi.acm.org/10.1145/2413176.2413198

[24] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," *Automata, Languages and Programming*, pp. 486–498, 2008.

[25] V. Goyal, P. Mohassel, and A. Smith, "Efficient two party and multi party computation against covert adversaries," in *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, ser. EUROCRYPT'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 289–306. [Online]. Available: http://dl.acm.org/citation.cfm?id=1788414.1788431

[26] B. Mood, L. Letaw, and K. Butler, "Memory-efficient garbled circuit generation for mobile devices," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 254–268.

[27] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, vol. 24, no. 2, pp. 137–180, 2016.

[28] "2017 united states speedtest market report," http://www.speedtest.net/reports/united-states/, accessed: 2017-10-29.

[29] "Monsoon power monitor," http://www.msoon.com/LabEquipment/PowerMonitor/, accessed: 2017-10-9.

[30] A. Shelat and C. Shen, "Two-output secure computation with malicious adversaries," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 386–405.

[31] Y. Lindell, "Fast cut-and-choose-based protocols for malicious and covert adversaries," *Journal of Cryptology*, vol. 29, no. 2, pp. 456–490, 2016.