Maturity and Performance of Programmable Secure Computation*

David W. Archer
dwa@galois.com

Dan Bogdanov
dan.bogdanov@cyber.ee

Benny Pinkas
benny@pinkas.net

Pille Pullonen
pille.pullonen@cyber.ee

October 27, 2016

1 Introduction

Secure computation (SC) stands for a group of technologies for computing functions of private inputs, while keeping the inputs themselves hidden. The canonical example of secure computation is the millionaires' problem, where two millionaires, Alice and Bob, who own $X and $Y, respectively, wish to run a computation that tells them which one of them is richer, but reveals no other information. Obviously, if both parties trust some third party they could reveal X and Y to that party, who could then tell them whether X>Y. Their goal is, however, to do the same computation without the help of any third party and while revealing nothing more than the final output of the computation.

Secure computation is essentially based on processing data that is protected by encryption or a similar method. There are SC solutions that are targeted for computing specific functions. We call a particular secure computation technology programmable if it is Turing-complete and can efficiently run at least a certain class of algorithms. Most SC solutions are designed in order to protect data during sharing or outsourced processing, for example in the context of cloud computing. Technologies for programmable SC include (but are not limited to) secure multi-party computation (MPC) and fully homomorphic encryption (FHE).

Secure computation research has gained traction internationally in the last five years. In the United States, the DARPA PROCEED program (2011-2015) focused on development of multiple SC paradigms and improving their performance. In the European Union, the PRACTICE program (2013-2016) focuses on its use to secure cloud computing. Both programs have demonstrated exceptional prototypes and performance improvements.

*This is an extended version of [ABPP16]

In PROCEED, Archer and Rohloff [AR15] demonstrated VoIP streaming where an untrusted server decompresses, mixes, adds, clips, and recompresses audio data while it remains encrypted. Carter et al. demonstrated the capability to compute route maps while map, source, and destination remain secret [CLT14a]. In PRACTICE, Bogdanov et al. evaluated a tax fraud detection system together with the Estonian Tax and Customs Board [BJSV15] and showed significant speedups of MPC using cloud computing. In addition, both programs contributed to speeding up the basic technologies of SC.

In this paper, we collect the results from both programs and other published literature to present the state of the art in what can be achieved with today's secure computing technology. In the following, Sec. 2 describes three approaches of programmable secure computation that we analyse in this paper. These are homomorphic encryption, garbled circuits and linear secret sharing. This introduction is followed by a set of interesting properties in Sec. 3 that are used to characterize and differentiate between the approaches. In Sec. 4, we present a taxonomy based on implementation maturity and runtime performance of each technique showcasing the readiness for real-world use. The taxonomy has five components—usage model, programming paradigm, implementation maturity, developer tool maturity and performance. Sec. 5 gives concrete performance evaluation of different secure computation artifacts based on common benchmark applications like AES evaluation. Finally, Sec. 6 summarises collected information using the proposed maturity taxonomy.

2 Secure Computation Paradigms

Secure computation is a multi-party processing of private data where different parties play different roles. The *computing parties* $\mathcal{C}$ are the ones actually carrying out the computations. The *input parties* $\mathcal{I}$ give their private data for the computations, and it is important to ensure that the data remains private except for the desired computation outcomes. Finally, the outcomes are obtained by the *result parties* $\mathcal{R}$. One participant may carry several of these roles, for example a party that gives inputs and receives outputs is denoted as $\mathcal{IR}$. The theory of secure computation is mostly centred around the computing parties and often expects the result and input parties to be the same as the computing parties (depicted as $\mathcal{ICR}$ on Fig. 1c). However, many practical deployments, such as surveys, separate these roles. A longer discussion about roles and deployment scenarios can be found in [BKLPV13] and in Sec. 4.

Secure computation can be done in many ways depending on the needed functionality and existing resources. In general, SC is required if it is necessary to avoid leaking any information except for the final output of the computations that is given to the result parties. A secure computation protocol can be designed to have *passive security*, also known as security against semi-honest adversaries, meaning that it is secure if the computing parties follow the protocol but might try to infer extra information from what they see during the protocol. A protocol secure against *actively malicious* participants is secure even if the computing parties try to cheat and do not follow the protocol. The intermediate case, security against *covert adversaries*, guarantees that cheating participants are caught with a reasonable probability, say 25%. This security guarantee is effective if the participants have a strong incentive not to be caught cheating, but might try to deceive if it is likely to avoid detection. Security against actively malicious adversaries is stronger than security against covert adversaries, which in turn offers more guarantees than passive security.

Most of the secure computation literature handles the case of corrupted computation parties, and gives absolute freedom for the input players to choose their inputs, and for the output players to choose the functionality that is computed. However, a corrupt behaviour of these parties can easily render the computation useless or insecure. For example, if the result parties are allowed to propose queries or algorithms for the computation then such corrupted parties might try to learn more outputs than originally intended. In this case, the computing parties should verify that it is safe to run each piece of the computation. On the other hand, input parties might try to corrupt the computation by giving invalid inputs. To that end, the computing parties should obliviously verify that the inputs fall to the desired bounds or have the desired format and discard invalid inputs and outliers from the computation. Furthermore, the input and result parties should have means to check that the outputs are correct and the computation really followed the safe procedure.

There are several major technologies that are used for secure computation, on which we elaborate in the next sections. In case there is one well equipped server, fully homomorphic encryption (FHE) can be deployed. Two-party computation is well supported by garbled circuits (GC) as well as linear secret sharing (LSS). However, the latter also allows for secure multi-party computation, involving more than two parties (GC can also be applied in the multi-party setting [BMR90, BNP08], but most of the research on GC focuses on the two-party setting). We note that there are additional technologies for secure computation, but they have received less attention both in the theoretical literature and with developers. For example, the Random Access Machine (RAM) model using oblivious RAM constructions, e.g. ObliVM [LWN$^+$15].

2.1 Homomorphic Encryption

Homomorphic encryption (HE) is an encryption scheme that enables computations on encrypted values. Figuratively, HE is an opaque locked glovebox [Gen09a]. A party can input its valuables to the box and lock it. Anyone with the box can use the gloves to manipulate the items inside, but only the box owner has the key to open the box and take the contents out. Hence, HE is an encryption with means to combine ciphertexts so that the result is a meaningful operation like addition or multiplication on plaintexts. These operations can be applied to the ciphertexts even without knowing the decryption key. Commonly HE is considered in a two-party setting where the client (input and result party) has the keys and outsources some computation to the server (the computing party) by providing it with the encrypted inputs. Such division of the roles is represented by Fig. 1a. At its best, HE requires interaction only for sending the inputs and retrieving the outputs, making it very communication efficient.

Many schemes allow to compute one kind of operation over encrypted data, for example addition in the Paillier encryption scheme [Pai99] or multiplication in the Elgamal [Elg85] scheme. Somewhat homomorphic encryption (SWHE) allows for computing both operations, but only a limited amount of one of them. Usually operations introduce noise to the ciphertext and after some operations the level of noise is too high for successful decryption. However, starting with seminal work of Gentry [Gen09a, Gen09b], fully homomorphic encryption (FHE) schemes that allow for unlimited number of both operations have become feasible. Remarkably, these two operations enable to compute any arithmetic functionality. In general, FHE is achieved from SWHE by introducing bootstrapping phase that resets the noise to a low level. FHE schemes used in secure computation include DGHV [VDGHV10], NTRU [HPS98] and BGV [BGV12]. As a practical example, NTRU has been used for secure teleconferencing and e-mail filtering [AR15].

2.2 Garbled Circuits

The first secure computation method was Yao's GC [Yao82, LP09, BHR12] proposed in 1982. Garbled circuits are like integrated digital circuits where it is hard to observe the values carried between single gates and only the output of the total circuit is revealed. Moreover, the materials used in the construction are fragile and dissolve after one use. Hence, although the circuit diagram remains the same a new circuit needs to be built for every evaluation.

In more detail, the idea is to evaluate boolean circuits by encoding wire values as random strings and encrypting the truth tables of each gate. The encodings of the input wires of a gate can be used to decrypt the encoding corresponding to the gate output. The first party, the garbler, chooses the encodings, generates the encrypted truth tables and forwards the circuit to the evaluator. The other party, the evaluator, obtains the encodings corresponding to the secret inputs and uses them to decode the encrypted truth tables and obtain the output. Both parties have the role of the computing party and usually both also provide inputs and obtain the outputs as on Fig. 1c, however this may vary. Furthermore the process can be modified to allow external input and result parties. In recent years there has been tremendous improvement in the performance of GC protocols, resulting in many efficient prototypes such as justgarble [BHKR13], SCAPI [EFLL12], and tinygarble [SHS$^+$15]. The basic method is secure against passive adversaries, however, recent research mostly considers active security [LP07, NO09, LP11, Lin13, MR13].

The GC approach excels in high latency networks as it requires a small number of rounds of interaction. It is straightforward to securely compute any functionality as there exist several compilers (e.g. [HFKV12, KMsB13]) that produce optimized circuits and software libraries implementing GC computation. On the downside, many interesting functionalities have huge boolean circuits and thus require a lot of bandwidth to transfer the garbled circuit. Actively secure GC is deployed, for example, by Dyadic Security for safeguarding cryptographic keys from corrupt administrators by splitting them between several servers and computing encryption without storing the keys in a single location [Sec14]. Another example of GC deployment is finding common contacts between Android users in application CommonContacts[1].

2.3 Linear Secret Sharing

Secret sharing was proposed in 1979 [Sha79, Bla79] and is the basis for a prolific branch in secure computation with seminal works [GMW87, BGW88, CCD88]. To illustrate LSS-based computation, consider a set of interconnected gloveboxes with input hatches. Any party can distribute their valuables between the boxes through the hatches. Afterwards the box operators can exchange pieces and manipulate the inputs using the gloves. However, the final product is obtained only when all boxes are opened and their results are combined.

More concretely, LSS enables parties to divide secrets to multiple shares where any unqualified set of shares does not reveal information about the secret. Each share is given to a different party. Homomorphic properties of the sharing are used to apply arithmetic operations without revealing the shared values. Each arithmetic operation is computed collaboratively by a dedicated interactive protocol that is run between the computing parties and large functionalities can be combined from basic operations or have a new specialized protocols. The strength of LSS is allowing reactive protocols where new inputs depend on the previous outputs, as well as securely storing intermediate results. The computing parties carry out the interactive protocols, however especially in the case of passive security it is straightforward to incorporate external input and result parties as on Fig. 1b.

A significant advancement of SC was LSS-based computation deployment for the Danish sugar beet auction [BCD$^+$09] in 2008. Current practical implementations of LSS-based secure computation include Sharemind [BLW08, Bog13] and ShareMonad [LDDAM12] for passive security and SPDZ [DPSZ12, DKL$^+$13] and TinyOT [NNOB12, LOS14] for active security. ShareMonad is used for spam filtering and secure teleconference [LADM14, AR15]. Actively secure LSS is deployed to secret share cryptographic keys and compute cryptographic operations using secure computation thereby mitigating threats from corrupted servers [Sec14]. Sharemind has been deployed to analyse ICT companies economic indicators [BTW12], perform genome-wide association studies [KBLV13], run government statistics [Kam15] and detect tax fraud [BJSV15].

[1] CommonContacts: http://mightbeevil.com/contacts/

3 Security Properties and Comparison Criteria

The main goal of SC is to enable useful and potentially collaborative computations while hiding the private data of the input parties. A protocol is considered secure if the only thing revealed in the computation is the output (and, of course, whatever information that can be deduced from the output). Although all SC protocols follow this general definition, the settings in which they are proposed differ significantly. This section mentions important theoretical criteria that can be used to label SC protocols as well as points out common properties of different paradigms where possible. We base this section on a recent classification by Perry et al [PGFW14]. In the following, Sec. 4 considers a classification from a more practical perspective.

As introduced in Sec. 2, two important criteria for characterizing secure computation are the *computation technology*, where we consider FHE, GC and LSS, and the *adversarial model* which is either active, passive or covert. Characteristics that are tightly coupled with the computation paradigm are the *model of computation* and the number of *communication rounds*. Common computation models include boolean and arithmetic circuits, although other models such as random access machines and Turing machines are also occasionally used. The garbled circuits method is mostly described for boolean circuits whereas LSS is applied to arithmetic circuits (although both methods can also be applied to the other kind of circuits). FHE schemes support either one of the circuit models depending of the concrete scheme. GC and FHE have a constant number of communication rounds while the number of communication rounds of LSS schemes is linear in the depth of the circuit that is computed.

Different schemes can also be compared based on the desired deployment scenario. A central property is the *number of computing parties* required for the computation coupled with the *fraction of tolerated corrupted parties* for which the protocol is still secure. FHE and GC focus on two party computation where one party is allowed to be corrupted. LSS works for any number of parties but a common model used in practice is two or three computing parties with one corrupted party. A significant exception is the SPDZ model that allows to corrupt all but one of the participants.

The *communication model* can assume that all parties have point-to-point connections or that there exists a broadcast channel (a broadcast channel is relevant in the case that there are more than two parties). All considered schemes except SPDZ work in a point-to-point setting. SPDZ requires a broadcast channel but also discusses the possibility of obtaining broadcast via a specific protocol in a point-to-point setting. In addition, it is often meaningful to divide computation to a *preprocessing* phase that does not require any knowledge of the actual inputs and an *online* phase that uses preprocessing results and the actual inputs to efficiently compute the result. This setting is applicable when it is known beforehand that some computations are coming up. Out of the aforementioned schemes, TinyOT and SPDZ use preprocessing.

From a programming perspective it is important to consider handling *conditional statements*. Conditional statements with secret conditions are commonly processed by evaluating all branches and obliviously choosing the right outcome.

Many properties describe the behaviour of the adversary or the possible outcomes for the corrupted parties.

- Commonly the model of corruption is *static* meaning that corrupted parties are fixed ahead of the protocol, but it is also possible to consider *adaptive* adversaries that decide during the protocol execution which parties to corrupt.

- A protocol is *fair* if whenever an output is obtained all parties are guaranteed to receive it (rather than the adversary being able to obtain the output while keeping other participants from learning it).

- An *abort capability* means that a protocol run can be interrupted without leaking information about the input. The *reconstruction capability* means that it is possible for honest parties to restore the output even if corrupted parties stop participating in the protocol. Schemes considered in this paper are capable of abort but not of reconstruction.

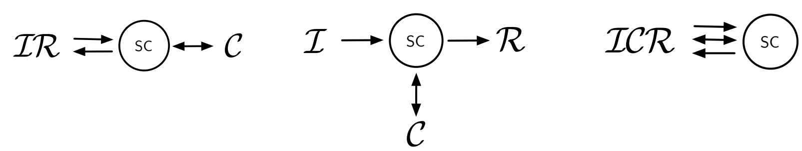
- Interesting but more theoretical properties include the *security assumptions* and whether security is preserved in a *concurrent execution*. For example, security can be information theoretic meaning that it is infeasible for any adversary (even with infinite computation powers) to break the scheme. Alternatively, security can be based on computational assumptions, meaning that breaking security in reasonable time is equivalent to breaking some well known hardness assumption (such as the hardness of factoring large numbers). Information theoretic security can only be achieved if a majority of the parties are honest. Passively secure LSS-based multi-party computation usually has information theoretic security whereas all two-party computation schemes offer only computational security.

- The *security level* estimates the expected number of operations required to break the scheme (say, by doing a brute-force search over all possible keys). The level of security is measured in bits where b-bit security means that the attack is expected to take 2^b operations. It is customary to support at least 80 bit or 128 bit of security.

Most of the research on SC focuses on the computing parties. The input or result parties could also be computing parties or could alternatively *outsource* the computation. In particular, it is often reasonable to consider settings where the computing parties are some fixed entities to whom input parties can send private data and from whom result parties can request queries. In this context, it is also possible to ask which guarantees regarding the correctness or privacy of the computation can be given to the input and result parties that do not take part in the computation. For example, these parties may be able to *verify* the correctness of the result or *audit* the computation process. In theory public verification is possible for all SC protocols, however doing so efficiently is currently an open question. Auditing the computation process has been studied for Sharemind [Pik14] and verification of outputs has been studied for SPDZ [BDO14]. FHE-based SC is also well suited for outsourcing and achieving verifiability although no practical auditing solutions have currently been implemented.

4 Maturity Taxonomy

This section provides a taxonomy for secure computation techniques that aims to summarize various aspects of real world use of SC. We consider five different features — the usage model, programming paradigm, implementation maturity, developer tool maturity and performance. This section focuses on aspects of practical usability complementing the formal characterization criteria from Sec. 3.



(a) Outsourced processing (b) Outsourced services (c) Joint processing

Figure 1: Party roles and communication in abstract usage models of SC

Not all secure computation techniques are well-suited for all kinds of applications. We define three general *usage models* that describe how data is obtained and used by the application. Each of these is illustrated in Table 1 by well-known services that could be replaced with analogous privacy preserving tools. The separation of the expected roles of the parties together with the direction of communication is illustrated by Fig. 1. Out of the considered secure computation techniques, HE is well suited for outsourced processing whereas LSS and GC are better for outsourced services and joint computations. A developer considering the use of secure computation should also be aware of possible *programming paradigms* for a chosen method. Either the programmer designs boolean or arithmetic circuits or is able to write programs that will be interpreted by the computation framework. The programming paradigms for different secure computation techniques are collected in Table 2.

Different SC paradigms are represented by various concrete protocol sets and frameworks for secure computation. We describe these using an *implementation maturity* category in Table 3 together with a Technological Readiness Level (TRL)[2]. This category indicates the engineering level of the best publicly known

[2]U.S. Department of Defense Technological Readiness Assessment (TRA) Guidance. Available at `http://www.acq.osd.mil/chieftechnologist/publications/docs/TRA2011.pdf`

Category	*Criteria of belonging*	*Example*
Outsourced processing	A client uses external resources (e.g., a cloud) to process its own data, and seeks protection against resource controller.	Salesforce, Erply
Outsourced services	A client uses external resources (e.g., a cloud) to process data collected from multiple data owners, while protecting this data from the resource controller and from itself.	Google Forms, Survey-Monkey.
Joint processing	Multiple clients collaborate to process data collected from among themselves, protecting their own data from each other.	Tinder, Doodle

Table 1: Usage model for secure computing systems

Category	*Criteria of belonging*	*Example*
Circuits	Task expressed as a fully formed boolean or arithmetic circuit.	Yao-style GC, TinyOT
Programs	Task expressed as a continuously interpreted program of arbitrarily complex primitive operations.	LSS-based SC

Table 2: Programming paradigm used by secure computing systems

implementations of different paradigms. Currently, LSS is the most used secure computation method, however GC approaches are also featured in market-ready and real-world deployment levels. Similarly, we describe the *developer tool maturity* in Table 4 to show which tools are available for developing applications with different SC frameworks. Most systems propose a special language that can be used to program applications but some also provide various levels of libraries to ease the development.

Level	*Criteria of belonging*	*Examples*
Academic prototype	Implementation demonstrated in a laboratory setting (TRL 1-4).	Fairplay (GC) [MNPS04, BNP08], HElib (FHE) [HS14], SEPIA (LSS) [BSMD10], TASTY (GC & HE) [HKS+10], ShareMonad (LSS), VIFF (LSS) [DGKN09]
Real-world deployment	Implementation demonstrated in a real-world setting (TRL 5-6).	FastGC (GC) [HEKM11], FRESCO (LSS) [Han15]
Market-ready	Commercial services available based on technology (TRL 7-9).	Dyadic (GC & LSS), Partisia (LSS), Sharemind (LSS)

Table 3: Implementation maturity of secure computation systems

Level	*Criteria of belonging*	*Examples*
Programming library	A hand-modified implementation or a library of primitives for integration.	FRESCO, HElib, ShareMonad, SEPIA, VIFF
Domain-specific language	An embedded or compilable domain-specific language targeted to secure computing.	L1 [SKB+09, SKM11], OblivC [ZE15], PCF [KMsB13], SecreC [Jag10, BLR14], SFDL [MNPS04, BNP08]
High-level libraries & tools	Reusable application-specific functionalities or tool integrations.	SecreC standard library, Rmind [BKLS14]

Table 4: Developer tool maturity of secure computation systems

There are many secure computation frameworks proposed in the literature, but not all of them have practical implementations in software or hardware. The classification in Table 5 assigns the *performance level category* to secure computation frameworks to denote the performance level of more mature implementations available in practice. In the following, Sec. 5 compares the performance of different systems in more depth based on concrete benchmarks.

Level	*Criteria of belonging*	*Examples*
Single-operation-level	Performed primitive operations on input arrays of non-trivial size.	HElib [GHJR14]
Algorithm-level	Runs an algorithm built of multiple primitive operations on an input structure of non-trivial size.	FRESCO [DDN+15], SEPIA [BD11, MBD12]
Business-process-level	Runs a multi-algorithm business process on an input database of non-trivial size.	Sharemind [Kam15, Sec. 6.4]

Table 5: Performance levels of secure computing systems

5 Performance of SC Implementations

A comprehensive survey performed near the end of the DARPA PROCEED program characterized the performance of the three SC paradigms described in Sec. 2. We describe salient results of that survey here. First, we aim to provide a pragmatic answer to the question, "How fast is FHE anyway?" using a number of benchmarks. Second, we aim to compare all three SC paradigms using the comparison benchmark of computing the AES-128 cipher.

5.1 How Fast is FHE?

Table 6 shows results from several publications on FHE performance for basic algorithms. The artifacts referenced here were all created using fully homomorphic (FHE) techniques. In each case, the artifact is tolerant of passive adversaries. Results show a wide range of performance for the same operation computed using different FHE approaches, as shown in the first two table entries. Other results in this table show that the relative performance for different artifacts using the same FHE method is sometimes non-intuitive. For example, the difference of 3 orders of magnitude between 32-bit addition and multiplication is not analogous to typical computation results "in the clear".

Although details are somewhat non-intuitive, the table offers an intuitive feel for current FHE performance. This and other data gathered during the PROCEED program suggest that as of early 2015, FHE computation is often between 5 and 10 orders of magnitude slower than computing "in the clear".

FHE Method	*Security level, bits*	*Operation*	*Time in seconds*	*Platform*
[XBY12]	80	32-bit add	0.0001	2.1GHz Core2 Duo
[BLLN13]	80	32-bit add	0.000024	2.9GHz Core i7
[FSF+13]	40	quadratic discriminant	108	2.0GHz Core2 Duo
[XBY12]	80	32-bit multiply	0.108	2.1GHz Core2 Duo
[FSF+13]	40	sum ten 4-bit numbers	36.3 - 51.2	2.0GHz Core2 Duo
[LN14]	80	SIMON 32/64 cipher, per block	0.8-1.1	3.4GHz Core i7
[LN14, Gal14]	128	SIMON 64/128 cipher, per block	2-8	3.4GHz Core i7

Table 6: Performance of FHE on various benchmarks

5.2 Cross-paradigm SC Performance on AES-128

The computation of the AES cipher (represented as a circuit) is a convenient benchmark for comparing the performance of SC techniques. For example, Damgård et al. report on a pre-2012 comparison of several GC and linear secret sharing systems, comparing them by AES performance [DKL+12]. In Fig. 2, we report on amortized cost for block evaluation results from the 2012-2015 time frame. Across a range of 9-bit to 128-bit security levels, performance results span almost seven orders of magnitude, from roughly 50 microseconds per block to roughly 200 seconds per block. Note that some of these results are based on running multiple threads in parallel or need a preprocessing computation which is not included in the online runtime.

LSS implementations tend to achieve the highest throughputs shown on the chart, ranging from roughly 50 microseconds to 0.5 seconds per block [NNOB12, Gal13a, DKL+12, LTW13, Cyb15]. Results of Sharemind [LTW13, Cyb15] and TinyOT [NNOB12] also demonstrate that increased amortization gives efficiency gains, especially for TinyOT this is even true when increasing the security parameter. GC implementations cover the middle range of performance shown, ranging from roughly 20 milliseconds per block to 60 seconds per block [FN13, HS13, KsS12, HKE12, sS13]. FHE implementations demonstrate the slowest throughputs in the time frame of our comparison, ranging from roughly 8 seconds per block to 200 seconds per block [DHS14, CLT14b, GHS15]. The best pictured amortized cost of secure AES is about two orders of magnitude slower than non-amortized software AES.

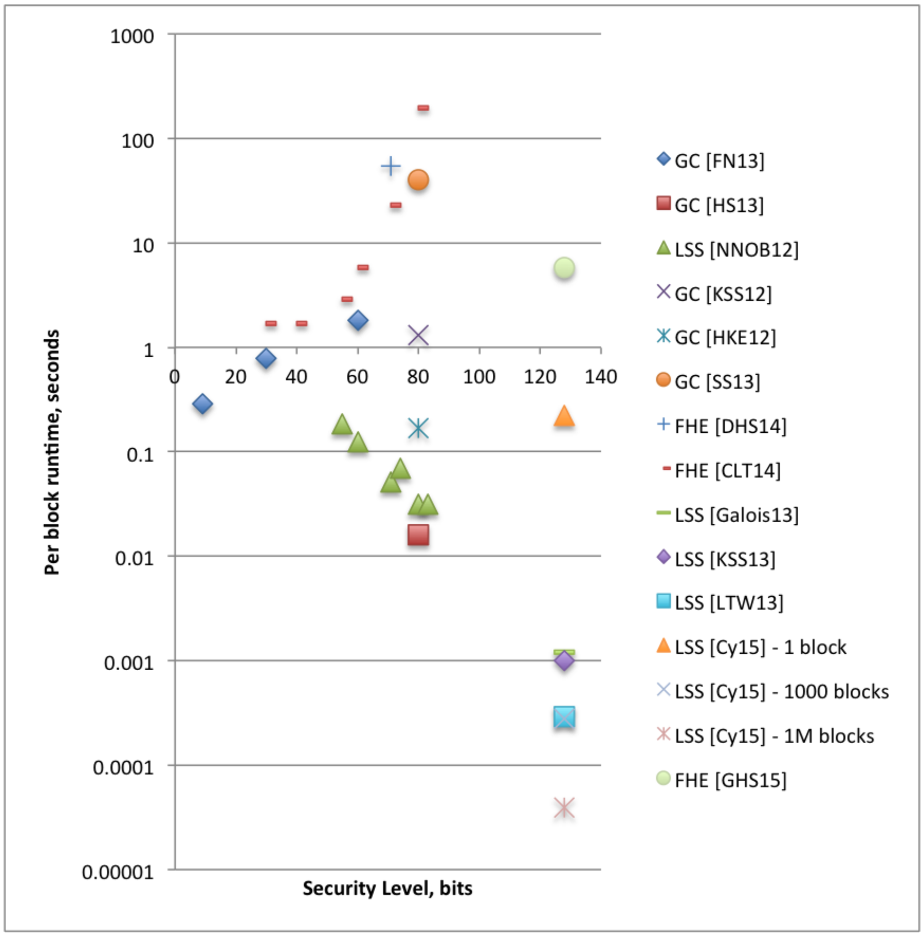


Figure 2: Comparison of SC paradigms using the AES-128 block cipher

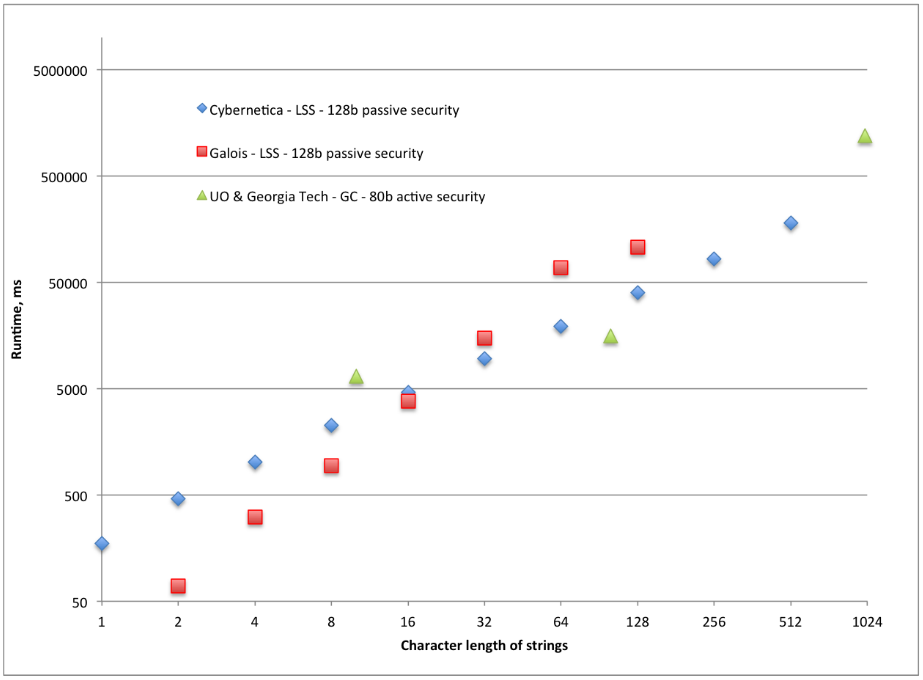


Figure 3: Comparison of SC paradigms using Levenshtein edit distance computation

5.3 Levenshtein Distance Computation

Edit distance is a sought after primitive in the processing genomic data. Therefore, Levenshtein algorithm was useful as a SC benchmark in the PROCEED program, though we found no other published results on this benchmark. In Fig. 3, we compare performance results as a function of input string length for linear secret sharing and garbled circuit constructions. In the chart, the red datapoints represent a ShareMonad (LSS) solution developed by Galois, Inc., the blue data points represent a Sharemind (LSS) solution developed by Cybernetica, and the green points represent a GC solution developed by a team at University of Oregon and Georgia Tech. Measurements were made on a 2.1GHz Intel e7 (Xeon) CPU with 1TB physical memory. The consistency of the results is surprising, given the variability of FHE-specific data and of the AES comparisons discussed above.

5.4 SC in Complex Applications

Applications more complex than simple operations or algorithms are difficult to compare across SC paradigms. Partly, because the choice of underlying algorithms can vary substantially between implementations, since SC is typically not the only measurable contributor to performance in such applications. In addition, user expectations of performance of complex applications are typically subjective rather than absolute. Thus in this paper we list complex applications that have been successfully implemented in SC with "reasonable" performance, but do not report on specific performance measurements of those implementations. In Table 7 we show for each considered SC paradigm a selection of complex applications that have been implemented.

LSS	VoIP with encrypted server signal processing [LADM14, AR15] E-mail filtering using regular expressions [LADM14, AR15] Naive Bayes spam filtering [Cyb15] Linear best-fit regression [NWI+13] [Gal13b] Satellite collision analysis [KW14] SHA2-based web service authentication [Gal13b] Genome SNP correlation to medical conditions [KBLV13] Tax fraud detection [BJSV15] Private statistics to analyse ICT students dropping out and working during studies [Kam15, Sec. 6.4] Linear programming based credit ranking [DDN+15]
GC	Route mapping [CLT14a] Fingerprint identification [BS15] Finding similar patients [WHZ+15]
FHE	VoIP with encrypted server voice addition [AR15] E-mail filtering using string match only [AR15] Genetic association study algorithms [LLAN15] Forensic image recognition [BPHJ14]

Table 7: Complex SC applications implemented with reasonable performance

6 Maturity of Secure Computation

We collected information on the performance results and published prototypes and used it to estimate the maturity of programmable secure computation techniques. Table 8 evaluates the five most popular technologies with their current state of the art estimations. The schemes that achieve practical efficiency are currently mostly passively secure. Hence, currently secure computation is a good approach for tasks with multiple parties who trust each other to behave honestly but still need to deploy means to ensure the privacy of the computations. However, ongoing research and development is largely focused on active security and it is likely to gain more efficiency in the near future.

It is still not clear, which fields will benefit the most from programmable secure computation or whether it will find industrial acceptance. However, continued research in the field as well as increasingly larger real-world deployments suggest that anyone looking for privacy-preserving computing technology keep an eye on the development of secure computation.

Technique	*Usage model*	*Implementation maturity*	*Programming paradigm*	*Developer tool maturity*	*Performance*
LSS passive	Outsourced services / Joint processing	Market-ready[a]	Programs	High-level libraries & tools[b]	Business-process-level[c]
LSS active	Outsourced services / Joint processing	Market-ready[d]	Programs	Programming library	Algorithm-level[e]
GC passive	Outsourced services / Joint processing	Real-world deployment [f]	Circuits	Domain-specific language	Business-process-level[g]
GC active	Outsourced services / Joint processing	Market-ready[h]	Circuits	Domain-specific language	Algorithm-level[i]
FHE passive	Outsourced processing / Outsourced services	Academic prototype	Circuits	Programming library	Single-operation-level

[a] Partisia Market Design ApS `www.partisia.dk/`, Cybernetica AS `sharemind.cyber.ee/`

[b] SecreC standard library `github.com/sharemind-sdk/secrec` and `github.com/sharemind-sdk/stdlib`, Rmind [BKLS14]

[c] Sugar beet auction [BCD+09] `http://alexandra.dk/uk/cases/simap`, ICT companies economic indicators [BTW12], tax fraud detection [BJSV15], government statistics [Kam15]

[d] Dyadic Security `www.dyadicsec.com`

[e] Simplex algorithm for linear programming [DDN+15]

[f] FastGC `http://www.mightbeevil.com/framework/`

[g] CommonContacts `www.mightbeevil.com/contacts/`

[h] Dyadic Security `www.dyadicsec.com`

[i] Dijkstra's shortest path algorithm on PCF `github.com/cryptouva/pcf` and Whitewash [CLT14a]

Table 8: Maturity of most popular programmable SC techniques

References

[ABPP16] David W. Archer, Dan Bogdanov, Benny Pinkas, and Pille Pullonen. Maturity and performance of programmable secure computation. *IEEE Security and Privacy*, 14(5):48–56, Sept 2016.

[AR15] David W. Archer and Kurt Rohloff. Computing with Data Privacy: Steps toward Realization. *IEEE Security & Privacy*, 13(1):22–29, 2015.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure Multiparty Computation Goes Live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 325–343. Springer-Verlag, Berlin, Heidelberg, 2009.

[BD11] Martin Burkhart and Xenofontas Dimitropoulos. Privacy-preserving Distributed Network Troubleshooting — Bridging the Gap Between Theory and Practice. *ACM Transactions on Information and System Security (TISSEC)*, 14(4):31:1–31:30, December 2011.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly Auditable Secure Multi-Party Computation. In *Security and Cryptography for Networks - 9th International Conference, SCN 2014*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196. Springer, 2014.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, USA, 2012. ACM.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, USA, 1988. ACM.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 478–492, Washington, DC, USA, 2013. IEEE Computer Society.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of Garbled Circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 784–796, New York, USA, 2012. ACM.

[BJSV15] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation. In *Financial Cryptography and Data Security, FC 2015*, volume 8975 of *LNCS*, pages 227–234. Springer, 2015.

[BKLPV13] Dan Bogdanov, Liina Kamm, Sven Laur, and Pille Pruulmann-Vengerfeldt. Secure multi-party data analysis: end user validation and practical experiments. Cryptology ePrint Archive, Report 2013/826, 2013.

[BKLS14] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Rmind: a tool for cryptographically secure statistical analysis. Cryptology ePrint Archive, Report 2014/512, 2014.

[Bla79] George Robert Blakley. Safeguarding Cryptographic Keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[BLLN13] Joppe W. Bos, Kristin E. Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *Cryptography and Coding - 14th IMA International Conference, IMACC 2013*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2013.

[BLR14] Dan Bogdanov, Peeter Laud, and Jaak Randmets. Domain-Polymorphic Programming of Privacy-Preserving Applications. In *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security*, PLAS'14, pages 53–65. ACM, 2014.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer Berlin Heidelberg, 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.

[BNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *Proceedings* *of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 257–266, New York, USA, 2008. ACM.

[Bog13] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications.* PhD thesis, Institute of Computer Science, University of Tartu, Estonia, 2013.

[BPHJ14] Chrisoph Bosch, Andreas Peter, Pieter H. Hartel, and Willem Jonker. SOFIR: Securely outsourced Forensic image recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 2694–2698, May 2014.

[BS15] Marina Blanton and Siddharth Saraph. Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification. In Günther Pernul, Peter Y. A. Ryan, and Edgar Weippl, editors, *Computer Security ESORICS 2015*, volume 9326 of *LNCS*, pages 384–406. Springer International Publishing, 2015.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 15–15, Berkeley, USA, 2010. USENIX Association.

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying Secure Multi-Party Computation for Financial Data Analysis - (Short Paper). In *Financial Cryptography*, pages 57–64, 2012.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty Unconditionally Secure Protocols. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 11–19, New York, USA, 1988. ACM.

[CLT14a] Henry Carter, Charles Lever, and Patrick Traynor. Whitewash: outsourcing garbled circuit generation for mobile devices. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 266–275. ACM New York, USA, 2014.

[CLT14b] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Scale-Invariant Fully Homomorphic Encryption over the Integers. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography*, pages 311–328, 2014.

[Cyb15] Cybernetica. Sharemind Performance. Private communication in the PROCEED program, 2015.

[DDN+15] Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential Benchmarking based on Multiparty Computation. Cryptology ePrint Archive, Report 2015/1006, 2015.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography: PKC '09*, Irvine, pages 160–179, Berlin, Heidelberg, 2009. Springer-Verlag.

[DHS14] Yarkin Doröz, Yin Hu, and Berk Sunar. Homomorphic AES Evaluation using NTRU. *IACR Cryptology ePrint Archive*, 2014:39, 2014.

[DKL+12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an Actively/Covertly Secure Dishonest-majority MPC Protocol. In *Proceedings of the 8th International Conference on Security and Cryptography for Networks*, SCN'12, pages 241–263, Berlin, Heidelberg, 2012. Springer-Verlag.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Science*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, pages 643–662, 2012.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. Cryptology ePrint Archive, Report 2012/629, 2012. http://eprint.iacr.org/.

[Elg85] Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, USA, 1985. Springer-Verlag New York, Inc.

[FN13] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and Maliciously Secure Two-Party Computation Using the GPU. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013*, pages 339–356, 2013.

[FSF+13] Simon Fau, Renaud Sirdey, Caroline Fontaine, Carlos Aguilar Melchor, and Guy Gogniat. Towards Practical Program Execution over Fully Homomorphic Encryption Schemes. In *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, pages 284–290, 2013.

[Gal13a] Galois. ShareMonad Performance. Private communication in the PROCEED program, 2013.

[Gal13b] Galois. Unpublished ShareMonad Experiments during DARPA PROCEED. Private communication in the PROCEED program, 2013.

[Gal14] Galois. Block Ciphers, Homomorphically, 2014. `http://galois.com/blog/2014/12/block-ciphers-homomorphically/`, last checked October 21, 2015.

[Gen09a] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729.

[Gen09b] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, New York, USA, 2009. ACM.

[GHJR14] Craig Gentry, Shai Halevi, Charanjit Jutla, and Mariana Raykova. Private Database Access With HE-over-ORAM Architecture. Cryptology ePrint Archive, Report 2014/345, 2014.

[GHS15] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic Evaluation of the AES Circuit (Updated implementation). Cryptology ePrint Archive, Report 2012/099, 2015.

[GMW87] Oded Goldreich, Silvio M. Micali, and Avi Wigderson. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, USA, 1987. ACM.

[Han15] Edited by Isabelle Hang. State-of-the-Art Analysis. Technical Report ICT-609611 / D22.1 / 1.1, PRACTICE: Privacy-Preserving Computation in the Cloud, 2015. `http://www.practice-project.eu/downloads/publications/D22.1-State-of-the-art-analysis-PU-V1.1.pdf`.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-party Computation Using Garbled Circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, USA, 2011. USENIX Association.

[HFKV12] Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith. Secure Two-party Computations in ANSI C. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 772–783, New York, USA, 2012. ACM.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening Semi-honest Protocols with Dual Execution. In *IEEE Symposium on Security and Privacy, SP 2012*, pages 272–284, 2012.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 451–462, New York, USA, 2010. ACM.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *Proceedings of the Third International Symposium on Algorithmic Number Theory*, ANTS-III, pages 267–288, London, UK, UK, 1998. Springer-Verlag.

[HS13] Wilko Henecka and Thomas Schneider. Faster Secure Two-party Computation with Less Memory. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 437–446, New York, USA, 2013. ACM.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. Cryptology ePrint Archive, Report 2014/106, 2014.

[Jag10] Roman Jagomägis. SecreC: a Privacy-Aware Programming Language with Applications in Data Mining. Master's thesis, Institute of Computer Science, University of Tartu, Estonia, 2010.

[Kam15] Liina Kamm. *Privacy-preserving statistical analysis using secure multi-party computation*. PhD thesis, Institute of Computer Science, University of Tartu, Estonia, 2015.

[KBLV13] Liina Kamm, Dan Bogdanov, Sven Laur, and Jaak Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics*, 29(7):886–893, 2013.

[KMsB13] Ben Kreuter, Benjamin Mood, abhi shelat, and Kevin Butler. PCF: A Portable Circuit Format for Scalable Two-party Secure Computation. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 321–336, Berkeley, CA, USA, 2013. USENIX Association.

[KsS12] Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. Billion-gate Secure Computation with Malicious Adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 285–300, Berkeley, CA, USA, 2012. USENIX Association.

[KSS13] Marcel Keller, Peter Scholl, and Nigel P. Smart. An Architecture for Practical Actively Secure MPC with Dishonest Majority. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 549–560, New York, USA, 2013. ACM.

[KW14] Liina Kamm and Jan Willemson. Secure Floating-Point Arithmetic and Private Satellite Collision Analysis. *International Journal of Information Security*, pages 1–18, 2014.

[LADM14] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-Scale Secure Multiparty Computation. In Zhong Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014*, volume 8410 of *Lecture Notes in Computer Science*, pages 8–26. Springer, 2014.

[LDDAM12] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient Lookup-table Protocol in Secure Multiparty Computation. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*, ICFP '12, pages 189–200, New York, USA, 2012. ACM.

[Lin13] Yehuda Lindell. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[LLAN15] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. Private Computation on Encrypted Genomic Data. In Diego F. Aranha and Alfred Menezes, editors, *Progress in Cryptology - LATINCRYPT 2014*, volume 8895 of *LNCS*, pages 3–27. Springer International Publishing, 2015.

[LN14] Tancrède Lepoint and Michael Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa*, volume 8469 of *Lecture Notes in Computer Science*, pages 318–335, 2014.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest Majority Multi-Party Computation for Binary Circuits. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, pages 495–512, 2014.

[LP07] Yehuda Lindell and Benny Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[LTW13] Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, pages 84–101, Berlin, Heidelberg, 2013. Springer-Verlag.

[LWN+15] Chang Liu, Xiao Shaun Wang, K. Nayak, Yan Huang, and E. Shi. Oblivm: A programming framework for secure computation. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 359–376, May 2015.

[MBD12] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. Fast Private Set Operations with SEPIA . Technical Report TIK report no. 345, ETH Zurich, 2012.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a Secure Two-party Computation System. In *Proceedings of the 13th Conference on USENIX Security Symposium*, volume 13 of *SSYM'04*, pages 20–20, Berkeley, USA, 2004. USENIX Association.

[MR13] Payman Mohassel and Ben Riva. Garbled Circuits Checking Garbled Circuits: More Efficient and Secure Two-Party Computation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 36–53. Springer Berlin Heidelberg, 2013.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A New Approach to Practical Active-Secure Two-Party Computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, pages 681–700, 2012.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. LEGO for Two-Party Secure Computation. In Omer Reingold, editor, *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 368–386. Springer Berlin Heidelberg, 2009.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 334–348, Washington, DC, USA, 2013. IEEE Computer Society.

[Pai99] Pascal Paillier. Public-key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[PGFW14] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. Systematizing Secure Computation for Research and Decision Support. In *Security and Cryptography for Networks, SCN 2014*, pages 380–397, 2014.

[Pik14] Tiit Pikma. Auditing of Secure Multiparty Computations. Master's thesis, Institute of Computer Science, University of Tartu, Estonia, 2014.

[Sec14] Dyadic Security. Dyadic Security White Paper. Technical report, Dyadic Security Ltd, 2014. Published online at `https://www.dyadicsec.com/`.

[Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[SHS+15] E.M. Songhori, S.U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 411–428, May 2015.

[SKB+09] Axel Schröpfer, Florian Kerschbaum, Debmalya Biswas, Steffen Geißinger, and Christoph Schütz. L1-Faster Development and Benchmarking of Cryptographic Protocols. In *ECRYPT Workshop on Software Performance Enhancements for Encryption and Decryption and Cryptographic Compilers (SPEED-CC09)*, pages 12–13. IEEE Computer Society, 2009.

[SKM11] Axel Schröpfer, Florian Kerschbaum, and Günter Müller. L1 - An Intermediate Language for Mixed-Protocol Secure Computation. In *Proceedings of the 35th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2011*, pages 298–307. IEEE Computer Society, 2011.

[sS13] abhi shelat and Chih-hao Shen. Fast Two-party Secure Computation with Minimal Assumptions. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 523–534, New York, USA, 2013. ACM.

[VDGHV10] Marten Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Advances in cryptology–EUROCRYPT 2010 - 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010.

[WHZ+15] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient Genome-Wide, Privacy-Preserving Similar Patient Query Based on Private Edit Distance. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer & Communications Security*, CCS '15, pages 492–503, New York, USA, 2015. ACM.

[XBY12] Liangliang Xiao, Osbert Bastani, and I-Ling Yen. An Efficient Homomorphic Encryption Protocol for Multi-User Systems. *IACR Cryptology ePrint Archive*, 2012:193, 2012.

[Yao82] Andrew C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[ZE15] Samee Zahur and David Evans. Obliv-C: A Language for Extensible Data-Oblivious Computation (whitepaper), 2015. `http://oblivc.org/downloads/oblivc.pdf`.