AFRL-RI-RS-TR-2015-241

SYSTEMIZATION OF SECURE COMPUTATION

YALE UNIVERSITY

NOVEMBER 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2015-241 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
CARL R. THOMAS
Work Unit Manager

/ S /
MARK H. LINDERMAN
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
NOV 2015	FINAL TECHNICAL REPORT	JAN 2014 – JUN 2015

4. TITLE AND SUBTITLE

SYSTEMIZATION OF SECURE COMPUTATION

5a. CONTRACT NUMBER
FA8750-13-2-0058

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)

Joan Feigenbaum (Yale University)

Rebecca N. Wright (Rutgers University)

5d. PROJECT NUMBER
PROC

5e. TASK NUMBER
ED

5f. WORK UNIT NUMBER
YL

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Yale University
51 Prospect St.,
New Haven CT 06511-8937

Rutgers University
611 George St.,
New Brunswick, NJ 08901

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

DARPA
675 North Randolph Street
Arlington, VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-241

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We propose a framework for organizing and classifying secure multiparty computation (SMC) research results. Using the framework, we classify a large number of MPC protocols on the axes and develop an interactive tool for exploring the problem space of secure computation. We also consider the use of MPC in three real-world scenarios: structural audits of cloud services, policy compliance for outsourced data, and lawful surveillance. Finally, we explore the reuse of encrypted values to increase efficiency in garbled-circuit computation, which is a well studied MPC paradigm.

15. SUBJECT TERMS

Garbled Circuits, Secure Multiparty Computation, SMC, Multiparty Computation, MPC, Server-aided computation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**CARL R. THOMAS**
U	U	U	UU	93	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1.0 SUMMARY

The highly active field of secure, multiparty computation (MPC) provides a diverse, powerful set of techniques for *using data without revealing it*: Multiple parties each at a different node in a network and each with a private data set can engage in a distributed computation in such a manner that all parties learn the results of the computation, but none learns the others' private inputs. MPC is a rich, highly promising theory, but to date it has not seen much use in practice.

We propose a framework for organizing and classifying MPC research results. Our *systematization of secure computation* consists of

- a set of definitions circumscribing the MPC protocols to be considered,
- a set of quantitative axes for classifying and comparing MPC protocols, and
- a knowledge base of propositions specifying the known relations among axis values.

Using the framework, we classify a large number of MPC protocols on the axes and develop an interactive tool for exploring the problem space of secure computation. Our tool can help potential users of MPC learn which existing protocols and implementations best match the private-data computations they would like to perform, help new researchers get up to speed in a complex area by providing an overview of the "lay of the land," and help experienced MPC researchers explore the problem space and discover gaps to fill with new protocols or impossibility results.

We also consider the use of MPC in three real-world scenarios: structural audits of cloud services, policy compliance for outsourced data, and lawful surveillance. Finally, we explore the reuse of encrypted values to increase efficiency in garbled-circuit computation, which is a well studied MPC paradigm.

2.0 INTRODUCTION

As computers and networks become ubiquitous, and valuable data are available from numerous and diverse sources, the potential to make use of these data grows explosively. In many cases, it is desirable to prevent the data or the results of computing with and mining them from being directly shared with other parties. While data encryption can protect data from unauthorized access, straightforward encryption renders the data unusable until decrypted. For many years, computing with encrypted data, along with the strongly related technologies of secure two-party and multiparty computation and other supporting technologies (which we collectively refer to as "secure computation" or "SC") have promised the seemingly paradoxical ability to enable computation on encrypted or otherwise obscured data while still providing the desired data protection. SC techniques have been known for decades, but have mostly been investigated within the realm of theoretical research. The theory of SC is elegant and powerful, but nonetheless SC has rarely been used in practice and has had limited impact on the real world.

In this project, we addressed what we believe to be one of the key reasons for this lack of adoption: the lack of a systematic characterization (aka a "systematization") that allows decision makers to understand the essential strengths and weaknesses of different SC solutions and determine which best applies in a given scenario. Specifically, we enumerated important properties of existing SC solutions, including:

- their methods of representing functions,
- their adversarial models,
- whether or not they guarantee fairness,
- requirements on their execution environments,
- prerequisites regarding correctness, auditability, and compliance,
- intractability assumptions,
- communication latency, and
- computational overhead.

We developed a multidimensional systematization of SC protocols and used it to classify a large number of protocols in the literature. We also built an interactive tool that researchers and decision makers can use to identify protocols with desired combinations of properties, to learn that there are no known protocols with the desired combination of properties, or in some cases to learn that it is provably impossible to achieve the desired combination.

With a systematization in hand, we considered three application scenarios in which secure computation could be useful: structural-reliability auditing of cloud services, policy compliance in database access, and lawful surveillance by police and intelligence agencies. For each scenario, we designed, prototyped, and tested a solution that builds on the diverse set of protocols and techniques in the extensive SC literature.

Finally, we explored a technical issue that our systematization efforts showed was a bottleneck to efficient secure computation, namely the need to retain state after a computation has been performed so that expensive processing does not have to be redone from scratch if a similar computation is done subsequently. We designed, prototyped, and built PartialGC, a tool that allows the reuse of encrypted values, thus making secure computation significantly more efficient, with respect to both time and bandwidth.

Five published papers were supported in whole or in part by this award. They are listed in the Bibliography below, which is part of the Appendix. The papers themselves are also included in the Appendix.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1. Systematizing Secure Computation for Research and Decision Support

Since the groundbreaking work of Yao [1, 2] and Goldreich *et al.* [3], hundreds of research papers on *Secure, Multiparty Computation* (MPC) have appeared. Many ingenious protocols for carrying out general secure computation under varying sets of assumptions have been proposed. In this project, we systematically organized the main results in the secure-computation literature, with the following goals in mind:

- Help potential users of MPC learn which existing protocols and implementations best match the sensitive-data computations they would like to perform. This may stimulate adoption of MPC in areas where it would be beneficial.
- Help new researchers get up to speed in a complex area by providing an overview of the "lay of the land."
- Help MPC researchers explore the problem space and discover remaining openings for protocols with new combinations of requirements and security features – or for new impossibility results that preclude the existence of such protocols.

There are a handful of introductory surveys and textbook-like treatments of MPC; they have (justifiably) focused on a narrow region of the problem space or specific security model, in order to present the material in a pedagogically clean way. However, none of these earlier works has attempted to organize the larger problem space in order to meet the goals listed above. In contrast, we do not limit ourselves to one model or set of definitions but instead provide a framework for examining their variations.

The main conceptual object in our systematization is a set of linear *axes*, where each axis represents an ordering of values of a single feature of MPC protocols. Every axis has at least two labeled values, at the endpoints. Some axes are continuous and others discrete. MPC protocols can be scored on these axes, allowing them to be compared quantitatively. This is the first attempt we are aware of to factor research results in MPC into such a representation. The axes were selected based on our literature survey, using two guiding principles: (1) The axes should be as orthogonal as possible, minimizing overlap (although some logical dependencies between axes are unavoidable); (2) The set of axes should be *complete* in the sense that they can express all distinctions of security and (asymptotic) efficiency between any two protocols.

Figure 1 contains all of the axes that we considered in our systematization, grouped in four categories. Figure 2 shows the usefulness of our systematization: Using just four axes, we can succinctly compare two of the best studied MPC protocols in the literature [3, 4], clearly demonstrating that neither is strictly "better" than the other; rather, each dominates the other on exactly two of the four axes that are most important in these protocols.

We developed an interactive GUI tool, called SysSC-UI, for exploring our systematization and MPC-protocol database. It is, to the best of our knowledge, the first tool designed to enable people to come to grips with the multi-dimensional landscape of MPC protocols. To explore the protocol database, a user adjusts a set of sliders and checkboxes corresponding to our axes of systematization. Figure 3 shows a sample SysSC-UI screenshot.

For a more detailed account, please refer to [5], a copy of which can be found in the Appendix.

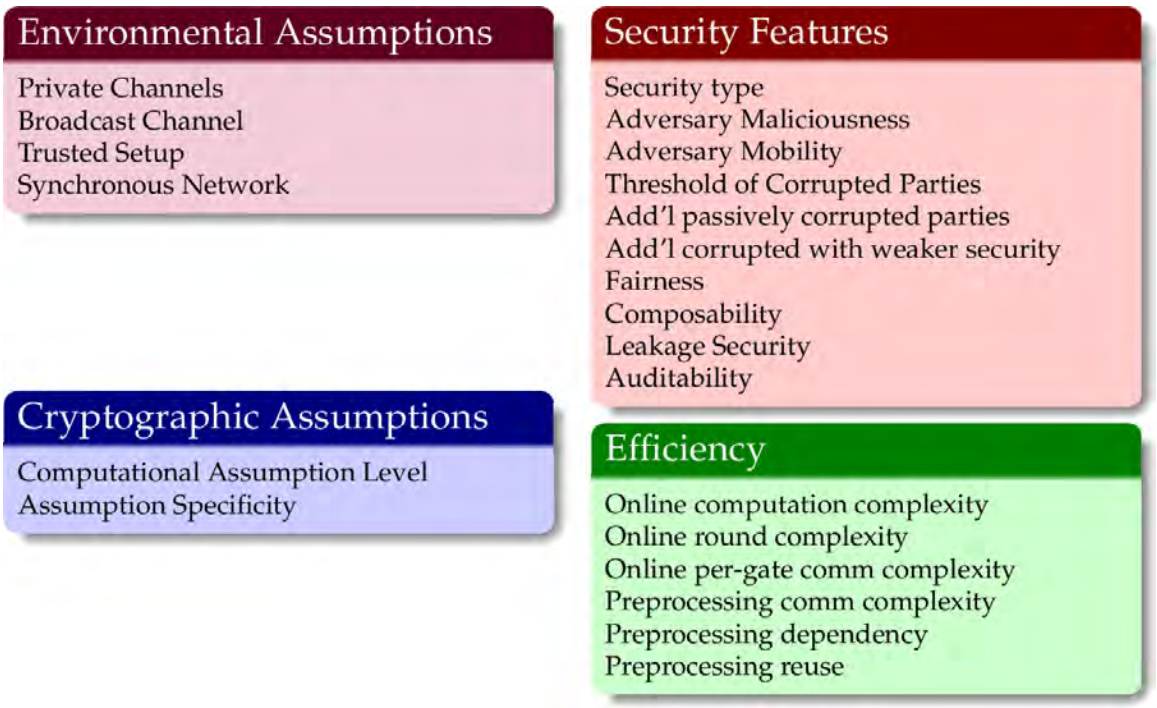


Figure 1: Axes of MPC Systematization

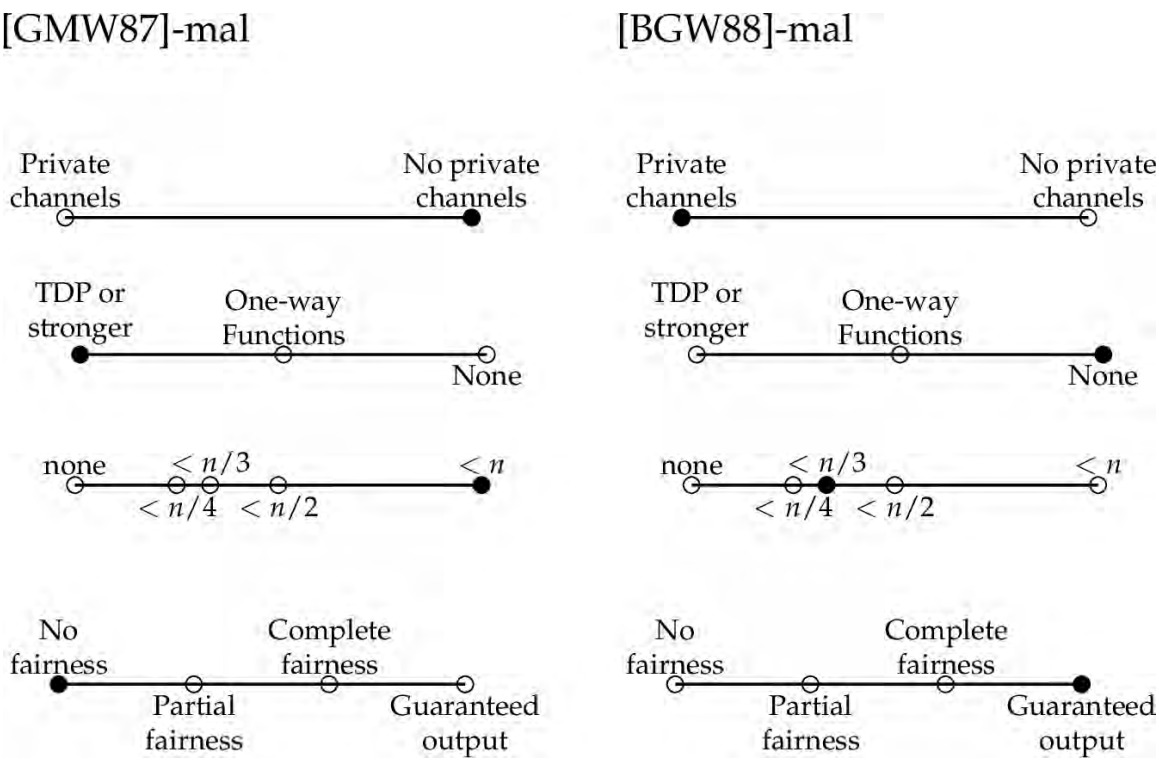


Figure 2: Sample Protocol Comparison using Axes

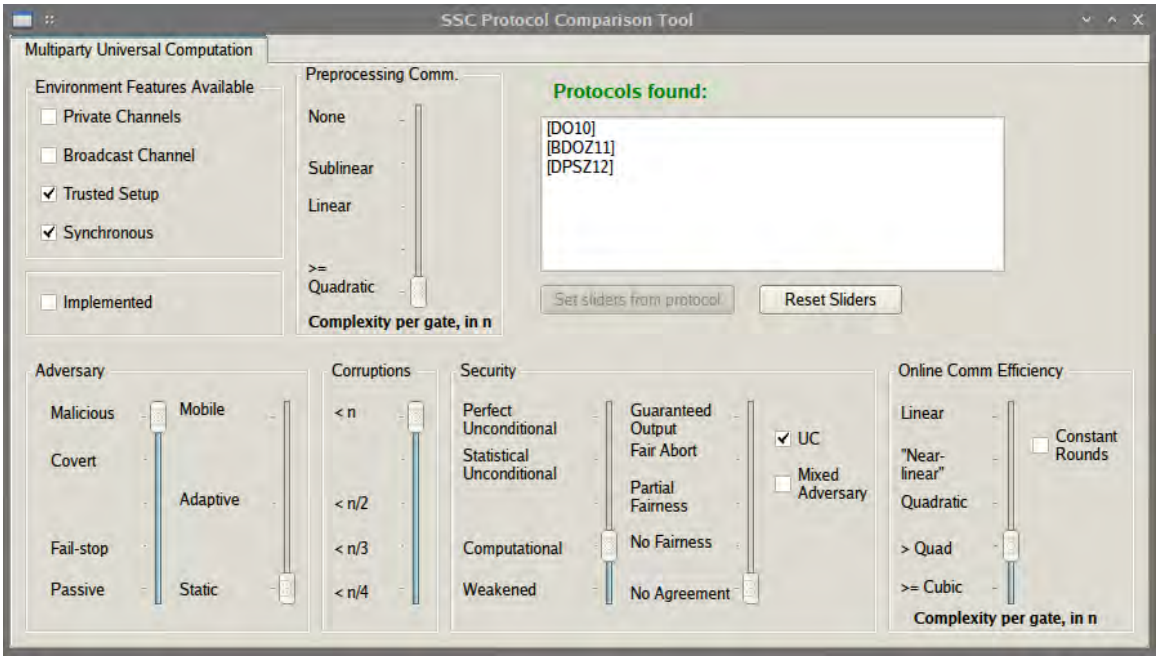


Figure 3: Sample Screenshot of Secure-Computation Systematization Tool

3.2. Structural Cloud Audits that Protect Private Information

Cloud computing and cloud storage now play a central role in the daily lives of individuals and businesses. For example, more than a billion people use Gmail and Facebook to create, share, and store personal data, 20% of all organizations use the commercially available cloud-storage services provided both by established vendors and by cloud-storage start-ups [6, 7], and programs run on Amazon EC2 and Microsoft Azure perform essential functions.

As people and organizations perform more and more critical tasks “in the cloud," reliability of cloud-service providers grows in importance. It is natural for cloud-service providers to use redundancy to achieve reliability. For example, a provider may replicate critical state in two data centers. If the two data centers use the same power supply, however, then a power outage will cause them to fail simultaneously; replication *per se* does not, therefore, enable the cloud-service provider to make strong reliability guarantees to its users. This is not merely a hypothetical problem: Although Amazon EC2 uses redundant data storage in order to boost reliability, a lightning storm in northern Virginia took out both the main power supply and the backup generator that powered all of Amazon's data centers in the region [8]. The lack of power not only disabled EC2 service in the area but also disabled Netflix, Instagram, Pinterest, Heroku, and other services that relied heavily on EC2. This type of dependence on common components is a pervasive source of vulnerability in cloud services that believe (erroneously) that they have significantly reduced vulnerability by employing simple redundancy.

Zhai *et al.* [9] propose structural-reliability auditing (SRA) as a systematic way to discover and quantify vulnerabilities that result from common infrastructural dependencies. They showed how to use sensitive, "internal" information about the structure and operational procedures of cloud-service providers to estimate the likelihood that the providers can live up to their service-level agreements. Prototype implementation and testing presented in [9] indicates that the SRA approach to evaluation of cloud-service reliability can be practical.

A potential barrier to adoption of SRA is the sensitive nature of both its input and its output. Infrastructure providers justifiably regard the structure of their systems, including the components and the dependencies among them, as proprietary information. They may be unwilling to disclose this information to a customer so that the latter can improve its reliability guarantees to *its* customers. Failure-probability estimates computed by an SRA are also proprietary and potentially damaging (to the cloud-service provider as well as the infrastructure providers). All of the parties to an SRA computation thus have an incentive not to participate. On the other hand, they have a countervailing incentive *to* participate: Each party stands to lose reputation (and customers) if it promises more reliability than it can actually deliver because it is unaware of common dependencies in its supposedly redundant infrastructure.

We investigate the use of secure multi-party computation (SMPC) to perform SRA computations in a privacy-preserving manner. We refer to our system as P-SRA (for "private structural-reliability auditor"). An overview of the system is given in Figure 4. The data-acquisition unit (DAU) collects from a target cloud-service provider *S* and all of the service providers that *S* depends on the details of network dependencies, hardware dependencies, and software dependencies, as well as the failure probability of each component. The local-execution unity (LEU) performs structural-reliability analysis within a subsystem of a particular provider. The secret-sharing unit (SSU) splits dependency information into shares in order to enable SMPC. Our preliminary experiments indicate that our P-SRA approach can be practical. The experiments were performed on the Sharemind SecreC platform [10].

SRA computation is a novel and challenging application of SMPC, which has not often been used for graph computations (or, more generally, for computations on complex, linked data structures). We introduce a novel data-partitioning and abstraction technique in order to achieve acceptable running times for SMPC even on large inputs; this approach to SMPC efficiency may be applicable in other contexts. Figures 5 and 6 show the effect of data partitioning and abstraction on the dependency graph of a cloud-service provider. In Figure 5, "Core" represents a core router, "Agg" an aggregation switch, and "ToR" a top-of-rack switch.

For a more detailed account, please refer to [11], a copy of which can be found in the Appendix.

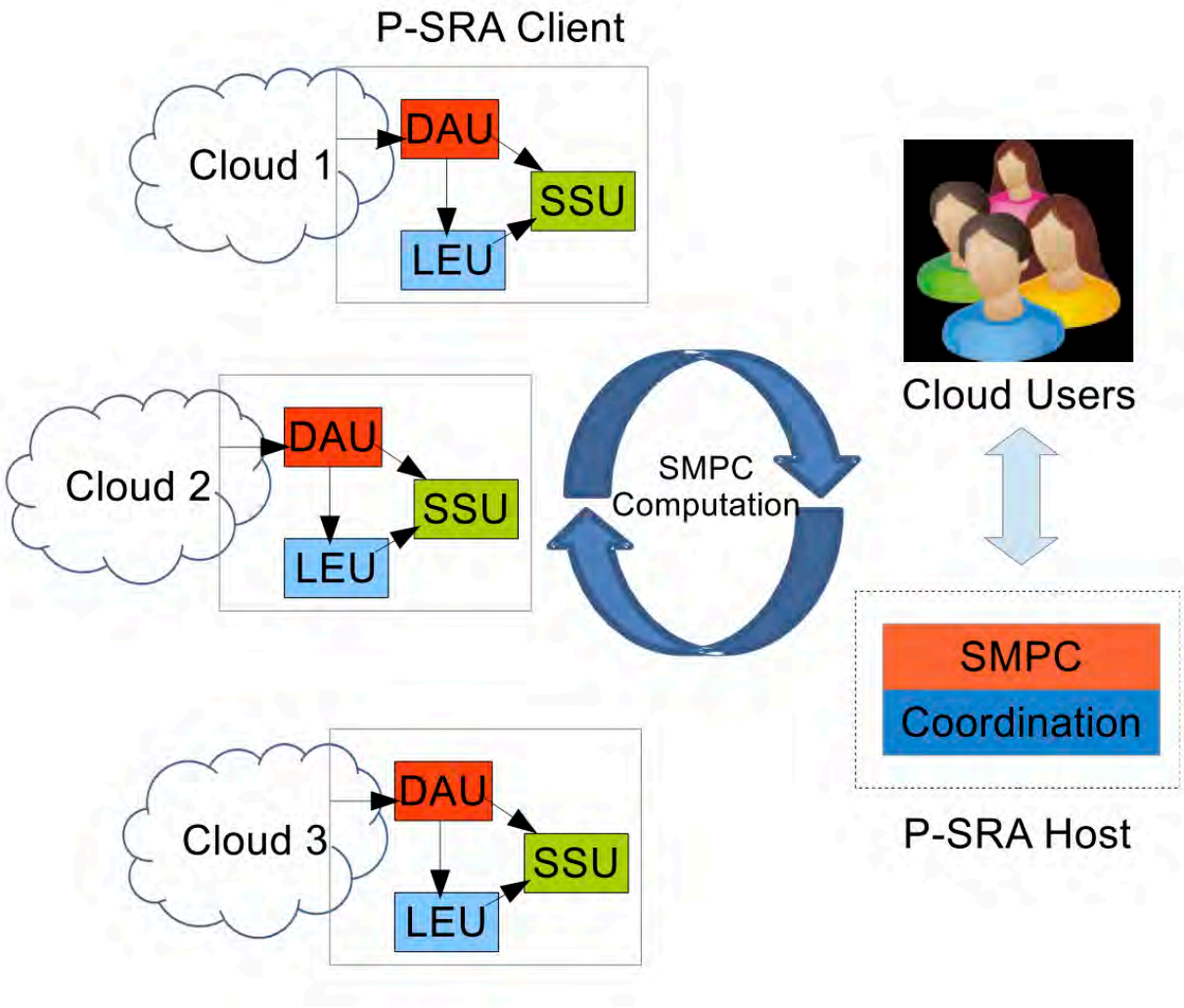


Figure 4: P-SRA System Overview

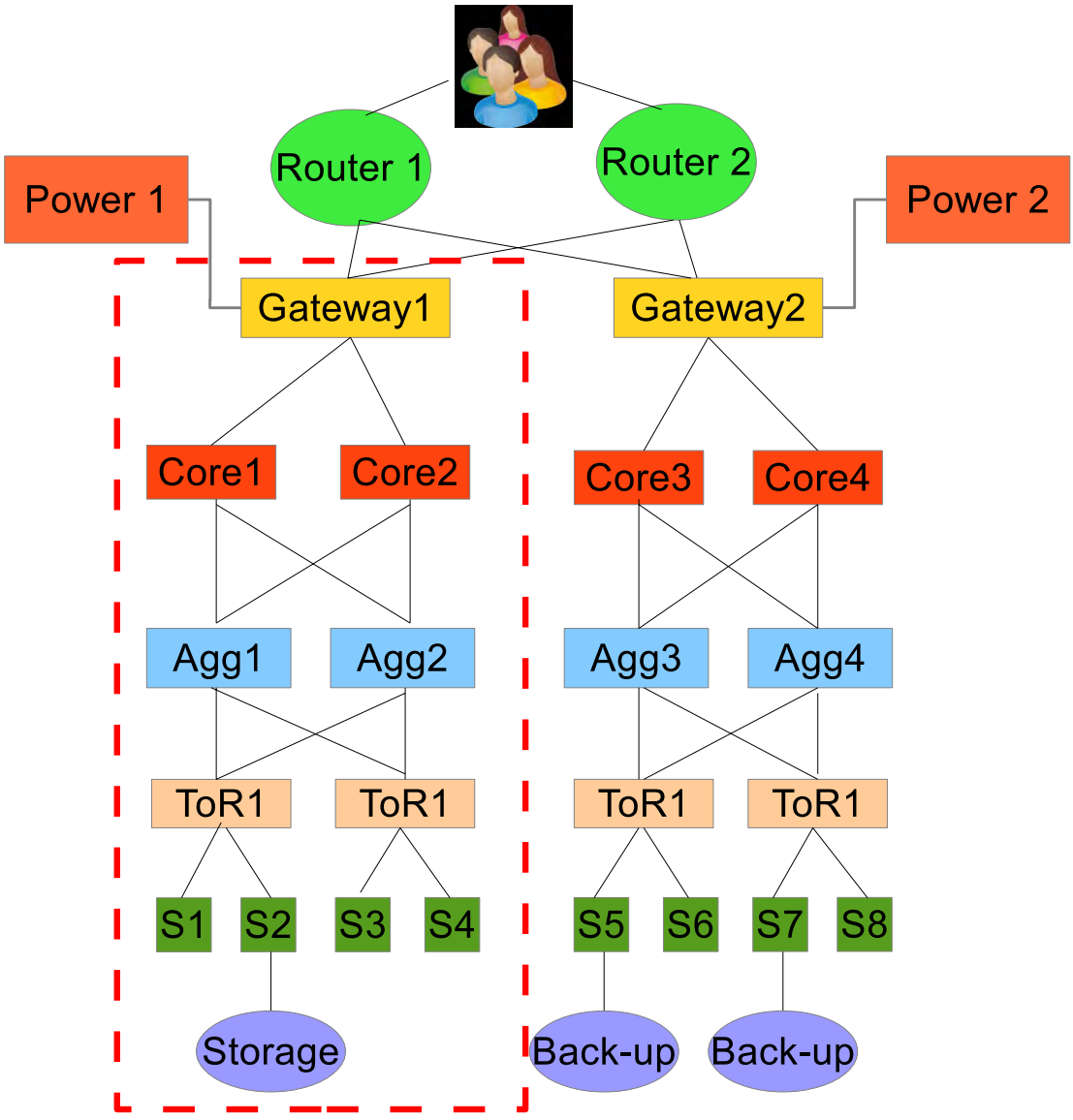


Figure 5: Full Dependency Graph of a Cloud-Service Provider C

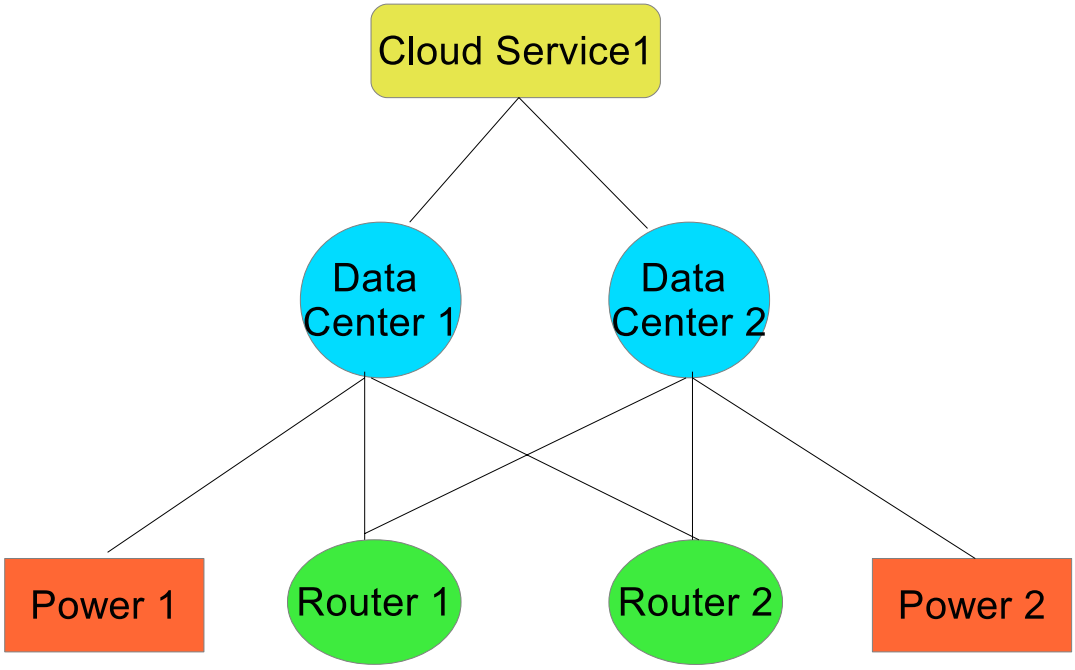


Figure 6: Abstracted Dependency Graph of C, Suitable for SMPC

3.3. Practical and Privacy-Preserving Policy Compliance for Outsourced Data

The information-technology trend of outsourcing "big data" to the cloud has been embraced in banking, finance, government, and other areas. Cloud storage provides tremendous efficiency and utility for users (as exemplified by the "database-as-a-service" application paradigm), but it also creates privacy risks. To mitigate these risks, database-management systems can use privacy-preserving data-retrieval (DR) protocols that allow users to submit queries and receive results in such a way that users learn nothing about the contents of a database except the results of their queries and data owners do not learn which queries are submitted. However, such protocols do not deal with the potentially sensitive nature of the data-access policies of the data owner. In order to address this crucial aspect of cloud-storage systems, we investigate the use of SMPC techniques in policy-compliance protocols for outsourced data.

For consistency with the database-as-a-service model, and to achieve practical solutions, we consider a three-party model, featuring a client C (interested in private data retrieval), a data owner D (offering data for retrieval conditioned on compliance with an access policy), and a server S (*e.g.*, a cloud server) that helps both parties to achieve their goals.

Our focus is on designing and implementing a privacy-preserving database policy-compliance (DPC) protocol. Our DPC solutions can be composed in a modular fashion with DR protocols in our three-party model, provided the DR protocols satisfy some natural properties. (DR protocols with the required properties can be found in, *e.g.*, [12, 13, 14].) See Figures 7 and 8 for the structure of this protocol composition.

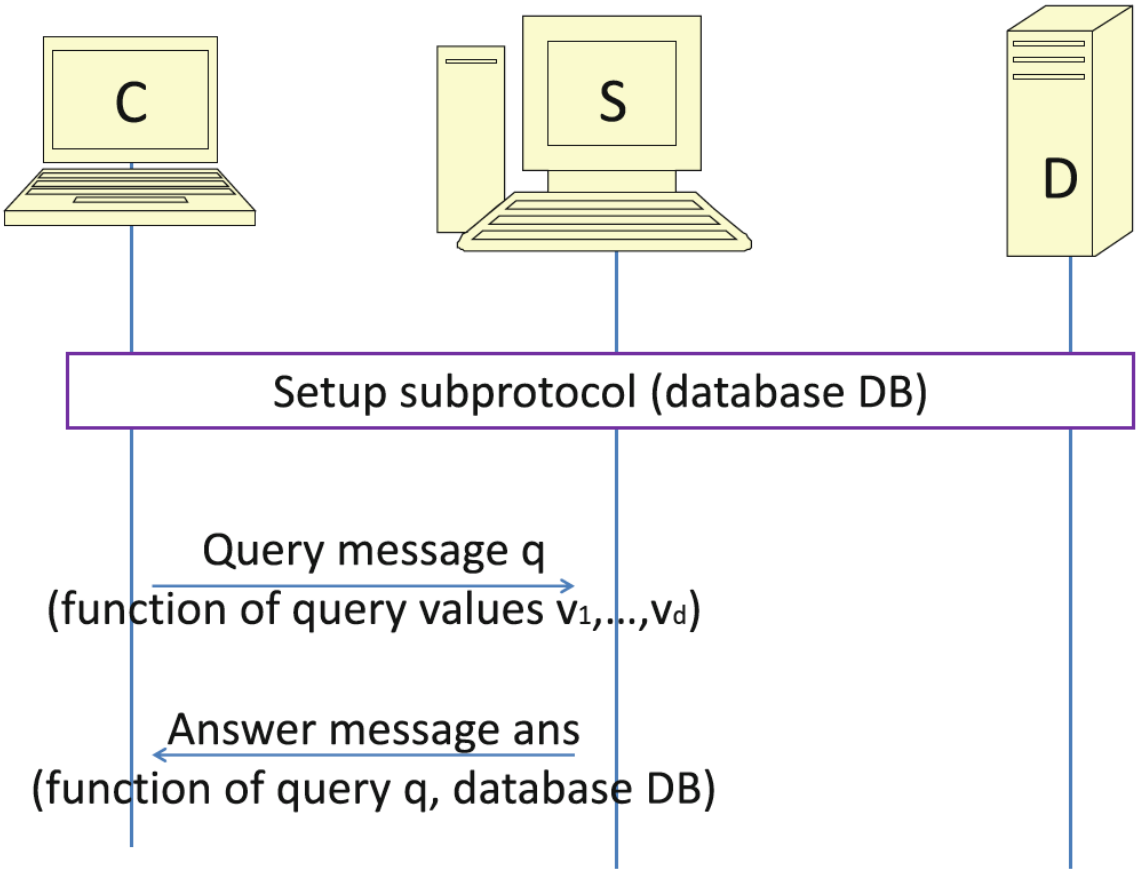


Figure 7: Structure of a DR Protocol

We present a set of DPC protocols with the following properties. Provided the client C submits a policy-compliant query, he can retrieve all records that he could have retrieved using the DR protocol without compliance checking. All queries that satisfy (resp., do not satisfy) the policy are found to be compliant with probability 1 (resp., with negligible probability). No efficient, malicious adversary impersonating the client can fool the data owner into answering a non-compliant query with non-negligible probability. Privacy of database values, policy-compliance values, and query values is preserved in the presence of semi-honest adversaries. The time complexity, communication complexity, and round complexity of the protocols are low enough to render them usable for large databases.

Technical contributions of independent interest include the use of equality-preserving encryption to produce highly practical symmetric-cryptography protocols and the use of a query-rewriting technique that maintains privacy of the compliance result.

For a more detailed account, please refer to [15], a copy of which can be found in the Appendix.

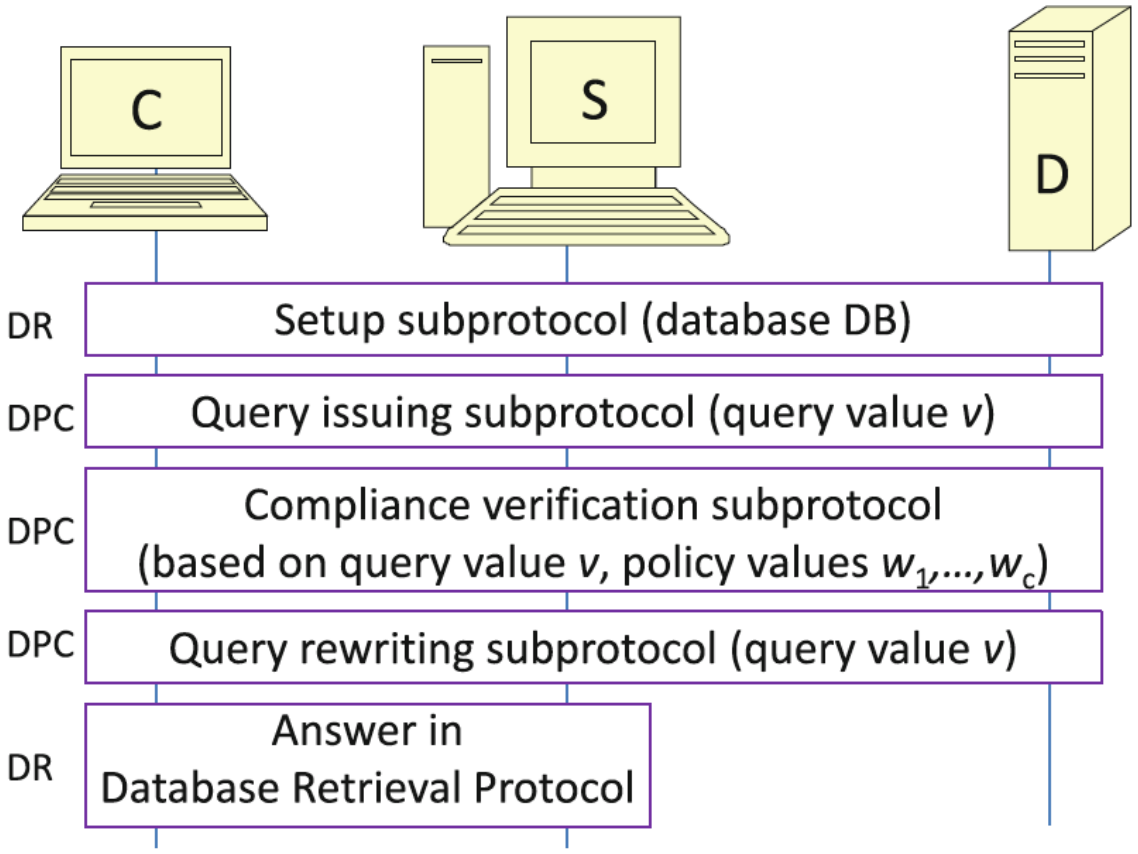


Figure 8: Composition of a DR Protocol with a DPC Protocol

3.4. Catching Bandits and *Only* Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance

Much of the Snowden-triggered debate has revolved around the "balance" between national security and personal privacy. Both sides of these "balance" arguments presume that security and privacy represent a zero-sum tradeoff, a presumption that we believe is false – not just on public-policy grounds but also for technical reasons. Specifically, we believe that with *existing* SMPC technology, deployed under the right public-policy framework, we can have both strong national security and strong privacy protections.

Consistent with both US Constitutional and human-rights principles that allow government "search and seizure" in private spaces only via warrant processes grounded in public law, we propose that any electronic-surveillance activity searching or otherwise touching private user data or metadata must likewise be implemented via *open, public* processes that protect the privacy of innocent, untargeted users (i.e., users who are not the subjects of legitimate warrants). We formulate an openness principle comprising two main planks: (1) Any surveillance or law-enforcement process that obtains or uses private information about untargeted users shall be an open, public, unclassified process, and (2) Any secret surveillance or law-enforcement processes

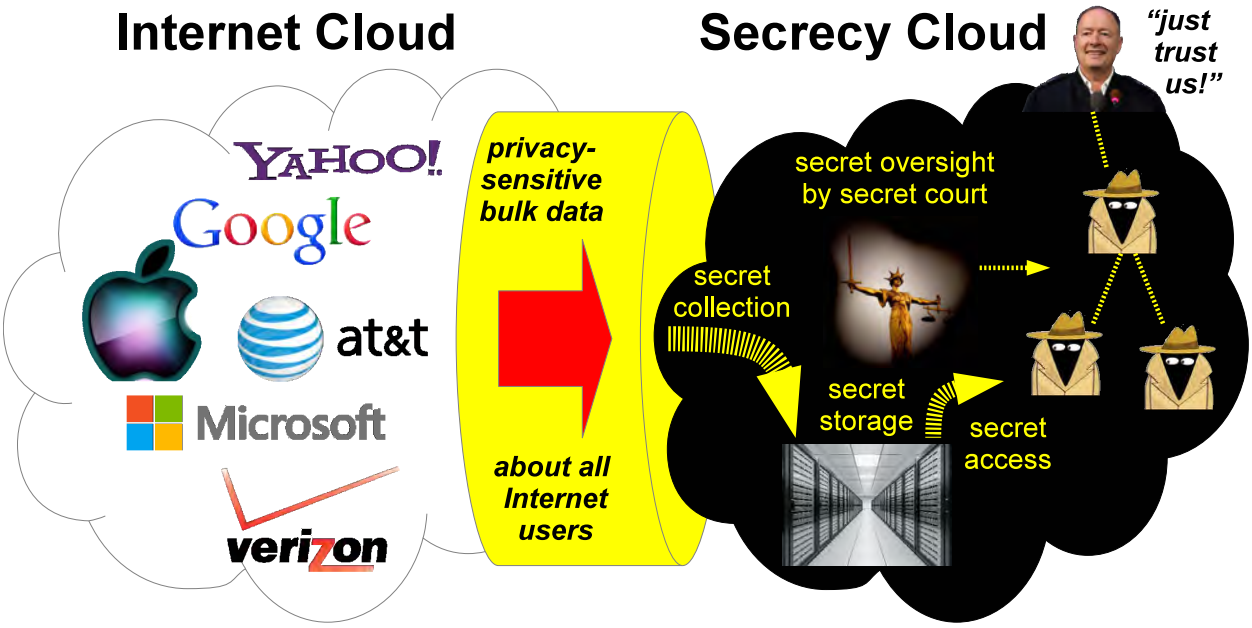


Figure 9: What We Have: A Cloud of Secret Mass Surveillance Processes

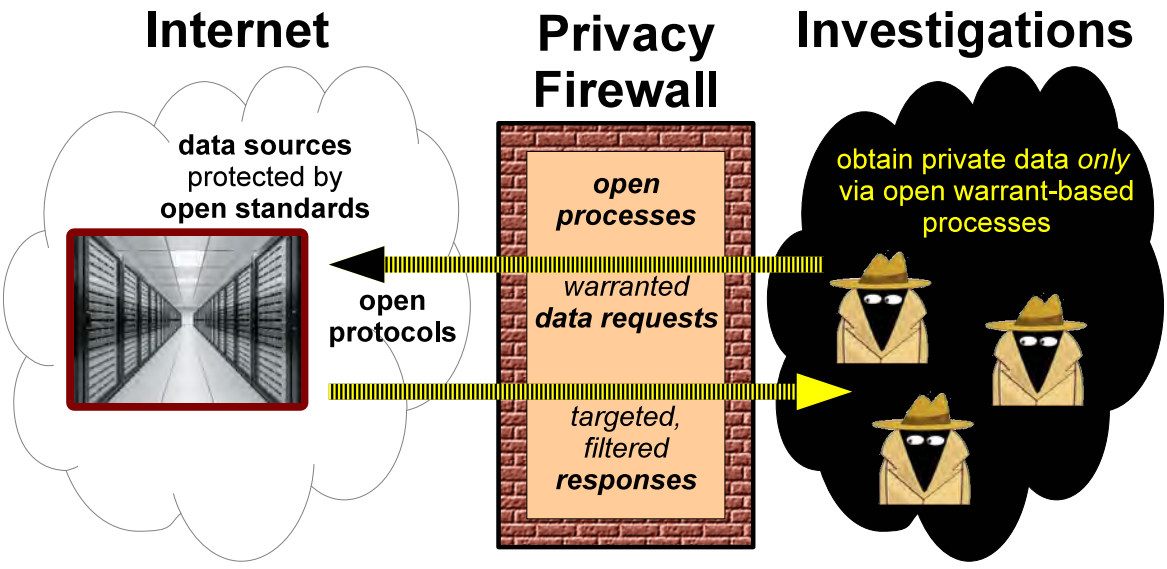


Figure 10: What We Require: Open Warrant-Based Processes for Lawful Electronic Surveillance, Creating a "Privacy Firewall"

Any secret surveillance or law-enforcement processes shall use only: (a) public information and (b) private information about targeted users obtained under authorized warrants via open surveillance processes. Instead of the deplorable situation that we have now, in which secret

Table 1: Improved Performance Results for Set-Intersection Protocol

Number of Items in Each Set	Data Sent per Node (KB)	End-to-End Runtime (secs)
10	21	1.0
25	46	1.1
50	86	1.3
75	127	1.6
100	167	1.7
250	410	2.9
500	815	4.9
750	1220	6.8
1000	1625	8.2
2500	4055	18.5
5000	8106	36.7
7500	12156	53.6
10000	16206	71.8
25000	40507	229.4
50000	81009	629.4

processes govern the bulk collection of private data about all Internet users (depicted in Figure 9), we propose the creation of a "privacy firewall" in which open processes ensure the protection of private information about innocent parties (depicted in Figure 10).

As a concrete case study, we designed, prototyped, and tested a metadata-query system based on a mature and practical SMPC protocol for set intersection [16]. Set intersection is a natural function to study in this context, because it was used successfully by the FBI in the "High-Country Bandits" case [17] and because it is the basis of the NSA's "co-traveler" program [18]. The performance results that we obtained are given in Table 1. In each execution, three sets of encrypted data were intersected, and only the elements in the intersection were decrypted; the number of items in each set is given in the leftmost column. The bottom row represents a test run with a total of 150,000 items, which is the number that the FBI handled (in a privacy-destructive manner) in the High-Country Bandits case; note that our privacy-preserving protocol was able to process an instance of this size in less than 11 minutes – clearly more than fast enough for an offline process. The performance results in Table 1 are not those in our paper [19]. In the paper, we conjectured that the quick-and-dirty implementation that we had tested at that time could be parallelized and, more generally, sped up considerably. It is the results of parallelizing and otherwise optimizing our prototype that are presented in Table 1, and they are (as conjectured) a vast improvement over the results in the paper.

In addition to privacy, our protocol achieves accountability: Trust is distributed over multiple government agencies, any of which can halt the protocol before any data items are decrypted if it is discovered that one or more of the conditions specified in the warrant are not met (*e.g.*, if the size of the intersection is discovered to be larger than the targeted set could reasonably be). To achieve privacy and accountability simultaneously, we use two cryptosystems that are *mutually commutative*: A message encrypted under a combination of encryption keys from the two cryptosystems can be decrypted by the corresponding decryption

keys, regardless of the order of encryption and decryption operations. This novel technique is of independent interest and may prove useful in other SMPC-application scenarios.

For a more detailed account, please refer to [19], a copy of which can be found in the Appendix.

3.5. Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values

The two-party variant of SMPC, known as 2P-SFE (for "two-party secure function evaluation"), was first introduced by Yao in the early '80s [1] but for a long time was largely a theoretical curiosity. Recent developments have made 2P-SFE vastly more efficient [20, 21, 22], but computing a function in a privacy-preserving manner is still usually much slower than doing so in a privacy-destructive manner.

As mobile devices become more powerful and ubiquitous, users expect more services to be accessible through them. When 2P-SFE is performed on mobile devices (where resource constraints are tight), it is extremely slow; that's *if* the computation can be run at all without exhausting the memory – something that may well happen for non-trivial input sizes and algorithms. One way to allow mobile devices to perform 2P-SFE is to use a server-aided computational model, allowing the majority of the work to be "outsourced" to a more powerful device while still preserving privacy. Past approaches, however, have not considered the ways in which mobile computation differs from the desktop. Often, the mobile device is called upon to perform *incremental* operations that are continuations of a previous computation.

We show that it is possible to reuse an encrypted value in an outsourced 2P-SFE computation even if one is restricted to primitives that are part of standard garbled circuits. Our system, PartialGC, which is based on CMTB [23], provides a way to take encrypted output wire values from one 2P-SFE computation, save them, and then reuse them as input wire values in a new garbled circuit. Our method vastly reduces the number of cryptographic operations compared to the trivial mechanism of simply XOR'ing the results with a one-time pad, which requires either generating (inside the circuit) or inputting a very large one-time pad, both complex operations. Using improved input-validation mechanisms proposed in [22] and new methods of partial-input gate checks and evaluation, we improve on previous proposals. Overviews of the operation and architecture of PartialGC are given in Figures 11 and 12.

Unlike other approaches to create reusable garbled circuits [24, 25, 26], our scheme is practical and efficient. We provide two sets of experimental results to substantiate this claim, one in which the garbled circuit is evaluated on a mobile device and one in which it is evaluated on a server. We show that using PartialGC can reduce computation time by as much as 96% and bandwidth by as much as 98% in comparison with previous schemes.

For a more detailed account, please refer to [27], a copy of which can be found in the Appendix.

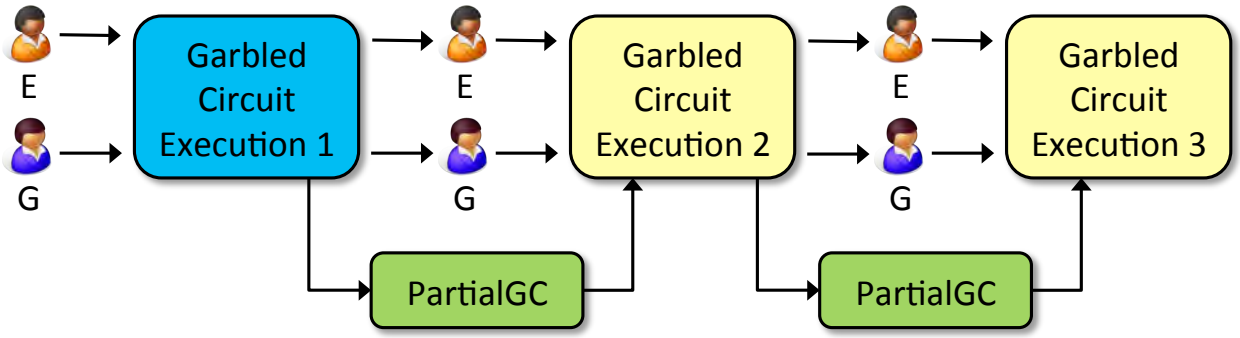


Figure 11: PartialGC Overview. E is evaluator and G is generator. The blue box is a standard execution that produces partial outputs (garbled values); yellow boxes represent executions that take partial inputs and produce partial outputs.

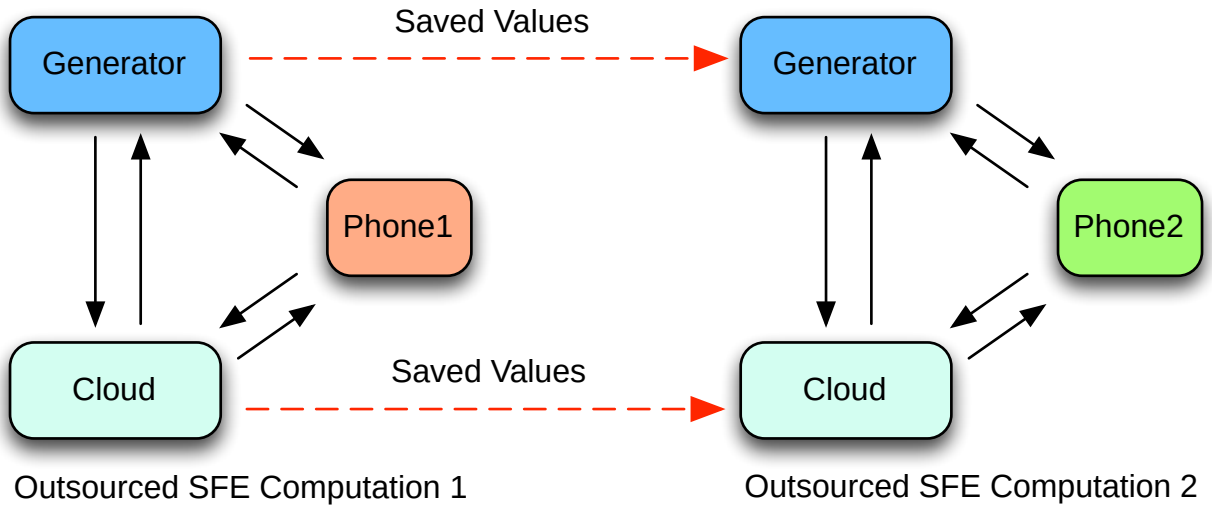


Figure 12: Our system has three parties. Only the cloud and generator have to save intermediate values - this means that we can have different phones in different computations.

4.0 CONCLUSIONS

At the beginning of this project, there was no comprehensive and systematic way to approach the literature on secure computation. Our work has shown that it is in fact quite difficult to systematize this research area; we cannot prove that the 22 axes along which we have classified SC protocols are the best ones for the purpose. Nonetheless, it is clear that our systematization is quite useful, both for newcomers to the field and for experienced SC researchers, and that even an approximate systematization is better than none.

Contrary to popular belief, SMPC is not prohibitively hard to implement, and SMPC protocols are not too slow to be of practical use. PROCEED emphasized performance and was (naturally) focused on DoD applications. However, our success in designing and implementing

efficient SMPC protocols for structural-reliability analysis, policy-compliance checking, and accountable set intersection demonstrates that there are many application scenarios in which the performance of existing technology suffices. Straightforward explanations for this fact may be that, unlike in DoD application scenarios, the data sets in our applications were of modest size, or the computations could be done offline.

The toolkit for secure computation is still primitive but has improved significantly over the course of the PROCEED program. In particular, the Sharemind SecreC platform is extremely well suited to experimental SMPC research, and garbled-circuit generators are now good enough to handle large problem instances.

5.0 RECOMMENDATIONS

Our results and conclusions are cause for optimism about the practicality and relevance of SMPC, but still neither it nor other PROCEED technologies are in widespread use. We recommend that future research in the area focus intently on overcoming obstacles to deployment, both technical and non-technical. Compelling challenges include:

- Understanding problems as well as solutions: Research in secure computation, both theoretical and experimental, has proceeded in a purely technique-driven manner for nearly 35 years. It would be highly desirable if future work in the area were solidly grounded in real applications and informed by potential adopters.

- Characterizing promising application scenarios: Despite the obvious drawbacks of centralized, trusted-party solutions, it has proven very difficult to convince users to abandon them for SMPC solutions. We conjecture that SMPC may be easier to deploy in application scenarios in which there are no incumbent solutions and no obvious third parties to trust. We believe that surveillance may be a killer app: The incumbent "trust-the-government" solution is broken, and citizens are disturbed by the Snowden revelations. Yet it is unrealistic to hope for mass-communication systems in which there is no surveillance at all. Relevant SMPC technology, including privacy-preserving set intersection, is mature and practical, and it is compatible with legal frameworks.

- Improved application-development environments: Although compilers, circuit generators, and other tools have improved significantly over the past decade, they are not yet adequate for industrial-strength application development by typical development teams. Moreover, existing tools deal exclusively with technical aspects of secure computation. More effort must be devoted to the challenge of explaining secure-computation technology to potential adopters and to enabling them to determine whether proposed solutions meet legal and organizational-policy requirements as well as technical requirements.

6.0 REFERENCES

[1] A. C.-C. Yao. **Protocols for Secure Computations (Extended Abstract).** 23rd IEEE Symposium on Foundations of Computer Science (FOCS'82), 1982.

[2] A. C.-C. Yao. **How to Generate and Exchange Secrets (Extended Abstract).** 27th IEEE Symposium on Foundations of Computer Science (FOCS'86), 1986.

[3] O**.** Goldreich, S. Micali, and A. Wigderson. **How to Play any Mental Game or a Completeness Theorem for Protocols with Honest Majority.** 19th ACM Symposium on Theory of Computing (STOC'87), 1987.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. **Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract).** 20th ACM Symposium on Theory of Computing (STOC'88), 1988.

[5] J. Perry, D. Gupta, J. Feigenbaum, and R. N. Wright. **Systematizing Secure Computation for Research and Decision Support.** 9th Conference on Security and Cryptography in Networks (SCN'14), 2014.

[6] B. Butler. **Cloud Storage Viable Option, but Proceed Carefully.** URL: http://www.networkworld.com/news/2013/010313-gartner-storage-265460.html**.** Accessed July 15, 2013.

[7] B. Butler. **Top 10 Cloud-Storage Providers.** URL: http://www.networkworld.com/news/2013/010313-gartner-cloud-storage-265459.html. Accessed July 15, 2013.

[8] W. Oremus. **Internet Outages Highlight Problem for Cloud Computing: Actual Clouds.** URL: http://www.slate.com/blogs/future_tense/2012/07/02/amazon_ec2_outage_netflix_pinterest_instagram_down_after_aws_cloud_loses_power.html. Accessed July 15, 2013.

[9] E. Zhai, D. I. Wolinsky, H. Xiao, H. Liu, X. Su, and B. Ford. **Auditing the Structural Reliability of the Clouds.** Technical Report YALEU/DCS/TR-1479. Yale University, New Haven, CT. July 2013.

[10] D. Bogdanov and A. Kalu. **Pushing Back the Rain -- How to Create Trustworthy Services in the Cloud.** *ISACA Journal,* vol. 3, pp. 49-51, 2013.

[11] H. Xiao, B. Ford, and J. Feigenbaum. **Structural Cloud Audits that Protect Private Information**. 5th ACM Cloud Computing Security Workshop (CCSW'13), 2013.

[12] A. Ceselli, E. Damiani, S. De Capitani di Vimercati, and S. Paraboschi. **Modeling and Assessing Inference Exposure in Encrypted Databases.** *ACM Transactions on Information and System Security*, vol. 8, pp. 119-152, 2005.

[13] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. **Executing SQL over Encrypted Data in the Database-Service-Provider Model.** 28th ACM SIGMOD International Conference on the Management of Data (SIGMOD'02), 2002.

[14] R. N. Wright, Z. Yang, and S. Zhong. **Privacy-Preserving Queries on Encrypted Data.** 11th European Symposium on Research in Computer Security (ESORICS'06), 2006.

[15] G. Di Crescenzo, J. Feigenbaum, D. Gupta, E. Panagos, J. Perry, and R. N. Wright. **Practical and Privacy-Preserving Policy Compliance for Outsourced Data**. 2nd Workshop on Applied Homomorphic Cryptography (WAHC'14), 2014.

[16] J. Vaidya and C. Clifton. **Secure Set-Intersection Cardinality with Application to Association Rule Minin**g. *Journal of Computer Security*, vol. 13, pp. 593–622, 2005.

[17] N. Anderson. **How "Cell Tower Dumps" Caught the High Country Bandits – and Why it Matters.** *arstechnica*, August 29, 2013.

[18] A. Soltani and B. Gellman. **New Documents Show How the NSA Infers Relationships Based on Mobile Location Data.** *The Washington Post*, December 10, 2013.

[19] A. Segal, B. Ford, and J. Feigenbaum. **Catching Bandits and *Only* Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance.** 4th USENIX Workshop on Free and Open Communications on the Internet (FOCI'14), 2014.

[20] Y. Huang, D. Evans, J. Katz, and L. Malka, **Faster Secure Two-Party Computation Using Garbled Circuits.** 20th USENIX Security Symposium, 2011.

[21] B. Kreuter, a. shelat, and C.-H. Shen. **Billion-Gate Secure Computation with Malicious Adversaries.** 21st USENIX Security Symposium, 2012.

[22] a. shelat and C.-H. Shen. **Fast Two-Party Secure Computation with Minimal Assumptions.** 20th ACM Conference on Computer and Communication Security (CCS'13), 2013.

[23] H. Carter, B. Mood, P. Traynor, and K. Butler. **Secure Outsourced Garbled Circuit Evaluation for Mobile Devices.** 22nd USENIX Security Symposium, 2013.

[24] S. Goldwasser, Y. Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. Nickolai. **Reusable Garbled Circuits and Succinct Functional Encryption.** 45th ACM Symposium on Theory of Computing (STOC'13), 2013.

[25] C. Gentry, S. Gorbunov, S. Halevi, V. Vaikuntanathan, and D. Vinayagamurthy. **How to Compress (Reusable) Garbled Circuits.** Cryptology ePrint Archive, Report 2013/687. URL: http://eprint.iacr.org/. Accessed April 10, 2014.

[26] L. T. A. N. Brandão. **Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique.** 19th International Conference on the Theory and Application of Cryptology and Information (Asiacrypt'13), 2013.

[27] B. Mood, D. Gupta, K. Butler, and J. Feigenbaum. **Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values**. 21st ACM Conference on Computer and Communications Security (CCS'14), 2014.

APPENDIX AND BIBLIOGRAPHY

Published Papers (Each Included Below)

H. Xiao, B. Ford, and J. Feigenbaum. **Structural Cloud Audits that Protect Private Information**. 5th ACM Cloud Computing Security Workshop (CCSW'13), 2013.

G. Di Crescenzo, J. Feigenbaum, D. Gupta, E. Panagos, J. Perry, and R. N. Wright. **Practical and Privacy-Preserving Policy Compliance for Outsourced Data**. 2nd Workshop on Applied Homomorphic Cryptography (WAHC'14), 2014.

A. Segal, B. Ford, and J. Feigenbaum. **Catching Bandits and *Only* Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance.** 4th USENIX Workshop on Free and Open Communications on the Internet (FOCI'14), 2014.

J. Perry, D. Gupta, J. Feigenbaum, and R. N. Wright. **Systematizing Secure Computation for Research and Decision Support.** 9th Conference on Security and Cryptography in Networks (SCN'14), 2014.

B. Mood, D. Gupta, K. Butler, and J. Feigenbaum. **Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values**. 21st ACM Conference on Computer and Communications Security (CCS'14), 2014.

These papers were not approved through DARPA/PA (DISTAR), and hence there are no PA approval numbers and dates. Pre-publication approval was not required by the contract.

Structural Cloud Audits that Protect Private Information

Hongda Xiao
Yale Univ., EE Dept.
hongda.xiao@yale.edu

Bryan Ford
Yale Univ., CS Dept.
bryan.ford@yale.edu

Joan Feigenbaum
Yale Univ., CS Dept.
joan.feigenbaum@yale.edu

ABSTRACT

As organizations and individuals have begun to rely more and more heavily on cloud-service providers for critical tasks, cloud-service reliability has become a top priority. It is natural for cloud-service providers to use redundancy to achieve reliability. For example, a provider may replicate critical state in two data centers. If the two data centers use the same power supply, however, then a power outage will cause them to fail simultaneously; replication *per se* does not, therefore, enable the cloud-service provider to make strong reliability guarantees to its users. Zhai *et al.* [28] present a system, which they refer to as a *structural-reliability auditor* (SRA), that uncovers common dependencies in seemingly disjoint cloud-infrastructural components (such as the power supply in the example above) and quantifies the risks that they pose. In this paper, we focus on the need for structural-reliability auditing to be done in a *privacy-preserving* manner. We present a privacy-preserving structural-reliability auditor (P-SRA), discuss its privacy properties, and evaluate a prototype implementation built on the Sharemind SecreC platform [6]. P-SRA is an interesting application of *secure multi-party computation* (SMPC), which has not often been used for graph problems. It can achieve acceptable running times even on large cloud structures by using a novel data-partitioning technique that may be useful in other applications of SMPC.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Reliability, Security, Measurement

Keywords

Cloud computing, reliability, secure multi-party computation

1. INTRODUCTION

Cloud computing and cloud storage now play a central role in the daily lives of individuals and businesses. For example, more than a billion people use Gmail and Facebook to create, share, and store personal data, 20% of all organizations use the commercially available cloud-storage services provided both by established vendors and by cloud-storage start-ups [8, 9], and programs run on Amazon EC2 and Microsoft Azure perform essential functions.

As people and organizations perform more and more critical tasks "in the cloud," reliability of cloud-service providers grows in importance. It is natural for cloud-service providers to use redundancy to achieve reliability. For example, a provider may replicate critical state in two data centers. If the two data centers use the same power supply, however, then a power outage will cause them to fail simultaneously; replication *per se* does not, therefore, enable the cloud-service provider to make strong reliability guarantees to its users. This is not merely a hypothetical problem: Although Amazon EC2 uses redundant data storage in order to boost reliability, a lightning storm in northern Virginia took out both the main power supply and the backup generator that powered all of Amazon's data centers in the region [16]. The lack of power not only disabled EC2 service in the area but also disabled Netflix, Instagram, Pinterest, Heroku, and other services that relied heavily on EC2. This type of dependence on common components is a pervasive source of vulnerability in cloud services that believe (erroneously) that they have significantly reduced vulnerability by employing simple redundancy.

Zhai *et al.* [28] propose *structural-reliability auditing* as a systematic way to discover and quantify vulnerabilities that result from common infrastructural dependencies. To use their SRA system, a cloud-service provider proceeds in three stages: (1) It collects from all of its infrastructure providers (*e.g.*, ISPs, power companies, and lower-level cloud providers) a comprehensive inventory of infrastructure components and their dependencies; (2) it constructs a service-wide fault tree; and (3) using fault-tree analysis, it estimates the likelihood that critical sets of components will cause an outage of the service. Prototype implementation and testing presented in [28] indicates that the SRA approach to evaluation of cloud-service reliability can be practical.

A potential barrier to adoption of SRA is the sensitive nature of both its input and its output. Infrastructure providers justifiably regard the structure of their systems, including the components and the dependencies among them, as proprietary information. They may be unwilling to disclose this information to a customer so that the latter can improve its reliability guarantees to *its* customers. Fault trees and failure-probability estimates computed by the SRA are also proprietary and potentially damaging (to the cloud-service provider as well as the infrastructure providers). All of the parties to SRA computation thus have an incentive not to participate. On the other hand, they have a countervailing incentive *to* participate: Each party stands to lose reputation (and customers) if it promises more reliability than it can actually deliver because it is unaware of common dependencies in its supposedly redundant infrastructure.

In this paper, we investigate the use of *secure multi-party computation* (SMPC) to perform SRA computations in a privacy-preser-

CCSW'13, November 8, 2013, Berlin, Germany.
ACM 978-1-4503-2490-8/13/11.
http://dx.doi.org/10.1145/2517488.2517493.

ving manner. SRA computation is a novel and challenging application of SMPC, which has not often been used for graph computations[1] (or, more generally, for computations on complex, linked data structures). We introduce a novel data-partitioning technique in order to achieve acceptable running times for SMPC even on large inputs; this approach to SMPC efficiency may be applicable in other contexts. Our preliminary experiments indicate that our P-SRA (for "private structural-reliability auditing") approach can be practical.

2. RELATED WORK

2.1 Secure Multi-Party Computation

The study of secure multi-party computation (SMPC) began with the seminal papers of Yao [25,26] and has been pursued vigorously by the cryptographic-theory community for more than 30 years. SMPC allows n parties $P_1,\ldots,P_n$ that hold private inputs $x_1,\ldots,x_n$ to compute $y = f(x_1,\ldots,x_n)$ in such a way that they all learn y but no P_i learns anything about x_j, $i \neq j$, except what is logically implied by the result y and the particular input x_i that he already knew. Typically, the *input providers* $P_1,\ldots,P_n$ wish not only to compute y in a privacy-preserving manner but also to do so using a protocol in which they all play equivalent roles; in particular, they don't want simply to send the x_i's to one trusted party that can compute y and send it to all of them. Natural applications include voting, survey computation, and set operations. One of the crowning achievements of cryptographic theory is that such privacy-preserving protocols can be obtained for any function f, provided one is willing to make some reasonable assumptions, *e.g.*, that certain cryptographic primitives are secure or that some fraction of the P_i's do not cheat (*i.e.*, that they follow the protocol scrupulously).

Many SMPC protocols have the following structure: In the first round, each P_i splits its input x_i into *shares*, using a *secret-sharing scheme*, and sends one share to each P_j; the privacy-preserving properties of secret sharing guarantee that the shares do not reveal x_i to the other parties (or even to coalitions of other parties, provided that the coalitions are not too large). The parties then execute a multi-round protocol to compute shares of y; the protocol ensures that the shares of intermediate results computed in each round also do not reveal x_i. In the last round, the parties broadcast their shares of y so that all of them can reconstruct the result. Alternatively, they may send the shares of y to an outside entity (resp., to a subset of the P_j's) if none (resp., only a subset) of the P_j's is supposed to learn the result. The maximum size of a coalition of cheating parties that the protocol must be able to thwart and the "adversarial model," *i.e.*, the capabilities and resources available to the cheaters, determine which secret-sharing scheme the P_i's should use. Because secret-sharing-based SMPC is common (and for ease of exposition), we will refer to parties' "sharing" or "splitting" their inputs. Note, however, that some SMPC protocols use other techniques to encode inputs and preserve privacy in multi-round computation.

The past decade has seen great progress on general-purpose platforms for SMPC, including Fairplay [14], FairplayMP [4], SEPIA [7], VIFF [10], and Tasty [13]. For our prototype implementation of P-SRA, we use the Sharemind SecreC platform [6]. Thorough comparison of SMPC platforms is beyond the scope of this paper, but we note briefly the properties of SecreC that make it a good choice in this context. Because it has a C-like programming language and optimizing compiler, assembler, and virtual machine, programmers can more easily write efficient programs with SecreC than with most of the other SMPC tools. Scalability to large numbers of input providers and reliable predictions of running times of programs are better in SecreC than in other SMPC environments. SecreC makes it easy for programs to use both private data (known to only one input provider) and public data (known to all parties to the computation) in the same program – something that is useful in our reliability-auditing context but is not provided by all SMPC platforms. On the downside, SecreC is not especially flexible or easily configurable.

[1] A notable exception is the work of Gupta *et al.* [12] on SMPC for interdomain routing.

2.2 Cloud Reliability

The case for "audits" as a method of achieving reliability in cloud services was originally put forth by Shah *et al.* [17], who advocated both *internal* and *external* auditing. Internal audits use information about the structure and operational procedures of a cloud-service provider to estimate the likelihood that the provider can live up to its service-level agreements. To the best of our knowledge, the first substantial effort to design and implement a general-purpose internal-auditing system that receives the structural and operational information directly from the cloud-service providers is the recent work of Zhai *et al.* [28]; the privacy issue and the possibility of addressing it with SMPC were raised in [28] but were not developed in detail.[2] External audits use samples of the cloud-service output provided by third parties through externally available interfaces to evaluate the quality of service; they have been investigated extensively, *e.g.*, in [18, 20–24]. Bleikertz *et al.* [5] present a cloud-auditing system that Shah *et al.* [17] would probably classify as "internal," because it uses structural and operational information about the cloud services to estimate reliability rather than using sampled output, but it obtains that structural and operational information through external interfaces rather than receiving it directly from the cloud-service providers.

In addition to auditing, technical approaches that researchers have taken to cloud reliability include *diagnosis*, the purpose of which is to discover the causes of failures after they occur and, in some cases, to mitigate their effects, *accountability*, the purpose of which is to place blame for a failure after it occurs, and *fault tolerance*. Further discussion of these approaches and pointers to key references can be found in Section 6 of [28].

2.3 Fault Trees

Fault-tree analysis [19] is a deductive-reasoning technique in which an undesirable event in a system is represented as a boolean combination of simpler or "lower-level" events. Each node in a fault tree[3] represents either an event or a logic gate. Event nodes depict failure events in the system, and logic gates depict the logical relationships among these events. The links of a fault tree illustrate the dependencies among failure events. The root node represents a "top event" that is the specific undesirable state that this tree is designed to analyze. The leaf nodes are "basic events," *i.e.*, fail-

[2] Concurrently with this work, Zhai, Chen, Wolinsky, and Ford [27] also explored the problem of privacy in cloud-reliability analysis, with a different goal of recommending good cloud configurations in a privacy-preserving manner. Their work was done independently of ours and differs from ours both in its target problem and in its technical approach. Briefly, Zhai *et al.* [27] simply compute the set of components that are common to two cloud-service systems, while our P-SRA system involves more participants and a richer set of outputs. The main technical tool in [27] is privacy-preserving set intersection, whereas P-SRA does a variety of privacy-preserving distributed computations using a general-purpose SMPC platform.

[3] What are called "fault trees" in the literature are not, in general, trees but rather DAGs. Because it is standard and widely used, we adopt the term "fault tree" in this paper.

ures that may trigger the top event; in order to use a fault tree, one must be able to assess (at least approximately accurately) the probabilities of these basic failures. Figure 4 is an example of a fault tree.

Fault-tree analysis has been applied very widely, *e.g.*, in aerospace, nuclear power, and even social services [11], but, to the best of our knowledge, was first used for cloud-service-reliability auditing by Zhai *et al.* [28]. It is an appropriate technique in this context for at least two reasons. First, the architectures of many cloud platforms can be accurately represented as leveled DAGs; therefore, potential cloud-service failures are naturally modeled by fault trees. Second, to construct a fault tree, one must uncover and represent the dependency relationships among components in a cloud system, and this inventory of dependencies is itself helpful in identifying potential failures (especially correlated failures).

3. PROBLEM FORMULATION

In order to specify in full detail the goals of our P-SRA system and how it achieves them, we start with a brief explanation of the SRA system of Zhai *et al.* [28]. Here and throughout the rest of this paper, a *failure set* (FS) is a set of components whose simultaneous failure results in cloud-service outage.[4] For example, the main and backup power supplies in the Amazon EC2 example described in Section 1 are an FS. A *minimal FS* is an FS that contains no proper subset that is also an FS.

The first necessary and nontrivial task of SRA is *data acquisition*. SRA's *data-acquisition unit* (DAU) collects from a target cloud-service provider S and all of the service providers that S depends on the details of network dependencies, hardware dependencies, and software dependencies, as well as the failure probability of each component. Using this inventory of components and the dependencies among them, SRA builds a model of S and the services on which it depends in the form of a *dependency graph*. Zhai *et al.* [28] assume that the dependency graph of a cloud service is a leveled DAG, and we also make this assumption; we are aware that it is a simplification (see Section 6), but it is an important first step. There are many potential technical and administrative challenges involved in modeling cloud components, discovering their dependencies, and assigning realistic failure probabilities; in particular, all cloud-service providers that participate in an SRA computation must agree on a taxonomy of components and types of dependencies. We defer to Subsection 3.2 of Zhai *et al.* [28] for discussion of these challenges and for details about data acquisition and dependency-graph construction in SRA. Here, we merely assume that cloud providers have *some* usable modeling and dependency-gathering infrastructure.

The next step in SRA is fault-tree analysis for the target cloud service S. Ideally, the output of this step is a complete set of minimal FSes for S. Note that an outage may occur because multiple entities that S relied on for redundancy had a common dependency on a set of components that failed; so accurate reporting of all minimal FSes requires information about all of the other service providers (*e.g.*, ISPs, power suppliers, and lower-level cloud services) that S uses. For some of these other services, the information might be publicly available (or, in any case, available to S) and thus not require the other service to participate in the computation; for example, SRA makes the simplifying assumption that it can obtain the information it needs about the ISPs and power suppliers that S uses without their participation, and we continue with that assumption in P-SRA. In other cases, participation in the SRA computation by other service providers is required; this is true, for example, of lower-level and peer cloud services on which S depends. If the ideal of reporting all minimal FSes is unattainable because it is too time-consuming, then SRA may produce a collection of (not necessarily minimal) FSes using a *failure-sampling algorithm*; this algorithm uses both the dependency graph and the individual components' failure probabilities.

Figure 3 depicts a simple dependency graph. Figure 4 depicts a corresponding fault tree. The semantics of an OR gate in the fault tree are that, if any input fails, the output of the gate is "fail." For an AND gate, only if all of the inputs fail does the gate output "fail." So Data Center #1 fails if Power #1 fails or if both Router #1 and Router #2 fail. Note that the logic-gate nodes in Figure 4 cannot be inferred from Figure 3; SRA collects additional information during its data-acquisition phase that is needed for fault-tree construction. The minimal FSes for Cloud Service #1 are {Data Center #1, Data Center #2}, {Router #1, Router #2}, {Power #1, Power #2}, {Power #1, Data Center #2}, and {Data Center #1, Power #2}.

The goal of P-SRA is to perform structural-reliability auditing in a privacy-preserving manner; to do this, we must modify all phases of SRA – data acquisition, fault-tree construction and analysis, and delivery of output. Our basic approach to the first two is to use SMPC. Instead of sending their data to one machine that integrates them and performs fault-tree analysis, P-SRA participants split their data into shares and perform fault-tree construction and analysis in a distributed, privacy-preserving fashion. However, the output of this computation cannot simply be a comprehensive list of S's minimal FSes, as it was in SRA, because these sets may contain infrastructural components that are used only by other service providers (*i.e.*, not by S). So the first technical challenge in the design of P-SRA is to specify SMPC outputs that reveal to S the components of its own infrastructure that could cause an outage while not revealing private information about other service providers' infrastructure. The second technical challenge is to reduce the size of the data sets that are input to the SMPC; the complete dependency graph of a cloud-service provider could have millions of nodes, which is more than current SMPC technology can handle, even in an off-line procedure like reliability auditing. P-SRA deals with this challenge by requiring each service provider that participates in the SMPC to partition its components into those that are known to be "private" and those that might be shared with other participants. For example, if the storage devices in a data center owned and operated by S are not accessible by anyone outside of S, then their failure cannot cause any service other than S to fail – they can be marked "private" by S and not entered individually into the SMPC. Rather, S can collapse certain "private" subgraphs of its dependency graph into single nodes, treat each such node as a "component" when entering its input to the SMPC, and perform SRA-style fault-tree analysis on the private subgraph locally. We refer to this data-partitioning technique as *subgraph abstraction*. Finally, P-SRA must provide useful, privacy-preserving output to cloud-service users as well as cloud-service providers. These three technical challenges are addressed in detail in Section 4.

In Section 5, we present a P-SRA prototype implemented on the Sharemind SecreC platform. The properties of this platform guarantee security in the semi-honest (or honest-but-curious) adversarial model. See the beginning of Section 5 for a more detailed explanation of Sharemind's computational model and adversarial model.

4. SYSTEM DESIGN

4.1 System Overview

There are three types of participants in the P-SRA system: the P-SRA host, cloud-service providers, and cloud-service users; see

[4]These are called *cut sets* in the fault-tree literature.

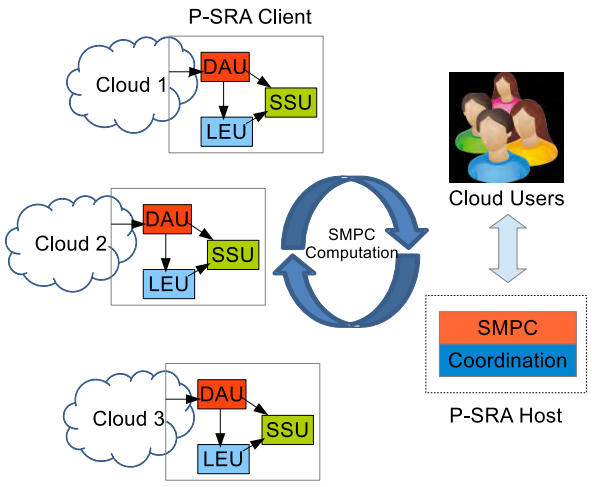


Figure 1: System Overview

Figure 1. The input supplied by each cloud-service provider is its topology information; this is private information and cannot be revealed to any other participants. The input supplied by the P-SRA host is the SMPC protocol. The inputs supplied by the cloud-service users are the set of cloud-service providers that they use or plan to use. The inputs of the P-SRA host and the cloud-service users are not private.

The P-SRA host consists of two modules. One is the SMPC execution unit (SMPC), which is responsible for execution of the SMPC protocol. The other is the coordination unit, which is responsible for establishing the SMPC protocol and coordinating the communication among the P-SRA host and the other participants.

Each cloud-service provider installs and controls a **P-SRA client** that processes local data and communicates with the P-SRA host. The P-SRA client consists of three modules: the Data-Acquisition Unit (DAU), the Secret-Sharing Unit (SSU), and the Local-Execution Unit (LEU). The DAU collects component and dependency information from the cloud-service provider and stores it in a local database. The SSU (1) abstracts the dependency information of "private" components in order to reduce the size of the input to the SMPC, (2) splits the dependency information into secret shares, and (3) connects to the P-SRA host and the SSUs of other cloud-service providers to execute the SMPC. The LEU performs local structural-reliability analysis within each "abstracted" macro-component.

Step 1: Privacy-preserving dependency acquisition: The DAU of the P-SRA client in each cloud-service provider S collects as much dependency information as possible from S, including network dependencies, hardware dependencies, software dependencies, and component-failure probabilities. The DAU stores this information in a local database. Because the P-SRA client is fully controlled by S, and the DAU does not communicate with any other cloud-service providers or the P-SRA host, there is no risk that private information will leak through the DAU.

Step 2: Subgraph abstraction. After data acquisition by the DAU, the SSU processes the dependency information and creates the macro-components to generate the SMPC input according to some abstraction policy. For instance, if cloud-service provider S_1 uses cloud-service provider S_2 as a lower-level infrastructure provider, S_1 can abstract S_2 as a macro-component that its services depend on. The SSU treats macro-components, the number of which is much smaller than total number of components in a cloud-service provider, as individual inputs to the SMPC. We leave the choices of abstraction policies to the cloud-service providers, which can tailor the policies based on the features of their architectures. However, we provide a standard example of subgraph abstraction in Subsection 4.3.

Step 3: SMPC protocol execution and local computation. After the subgraph-abstraction step, the SSUs of the cloud-service providers and the P-SRA host execute the SMPC protocol. The SSU of each cloud-service provider first adds some randomness to conceal statistical information about the input (without changing the output) and splits the randomized input into secret shares. It then establishes connections with the SSUs of other providers and the P-SRA host to execute the SMPC protocol, which identifies common dependency, performs fault-tree analysis, and computes reliability measures in a privacy-preserving manner. Meanwhile, the SSU passes the dependency graphs of the macro-components to the LEU, which performs local computation. The LEU mainly performs fault-tree analysis to obtain minimal FSes of the macro-components. After both the SSU and the LEU finish their execution, the SSU combines the results of the SMPC protocol and local computation to generate the comprehensive outputs for the cloud-service providers and users.

Step 4: Privacy-preserving output delivery. The output of the P-SRA system should satisfy two requirements: preserving privacy of the cloud-service providers and illustrating reliability risk caused by correlated failure. The SRA system of Zhai *et al.* [28] fully reveals all minimal FSes; P-SRA cannot do this, because the full specification of all minimal FSes may compromise the privacy of cloud-service providers. Although P-SRA is flexible in that cloud-service providers can specify the output sets that are most appropriate for them, we recommend some sets of benchmark outputs for cloud-service providers and users. For cloud-service providers, we recommend *common dependency* and *partial failure sets*. For cloud-service users, we recommend *common dependency ratio*, *failure probabilities of relevant cloud services*, and a small set of *top-ranked FSes*. All the outputs are delivered by an SMPC protocol in a privacy-preserving manner. We discuss these recommended outputs in Subsection 4.5.

4.2 Privacy-preserving Data Acquisition

The DAU of each cloud-service provider collects as much information as possible about the components and dependencies of this provider and then stores the information in a local database for later use by the P-SRA's other modules. The DAU can collect network dependencies, hardware dependencies, software dependencies, and failure probabilities of each component. For network dependencies, it collects information about a variety of components in the cloud structure including servers, racks, switches, aggregation switches, routers, and power stations, as well as the connections between these components within the cloud infrastructure and from the cloud infrastructure to the Internet. For hardware dependencies, the DAU inventories the CPUs, network cards, memory, disks, and drivers, and collects product information about each piece of hardware, including vendor, machine life, model number, and uptime. For software dependencies, the DAU analyzes the cloud-service provider's software stacks to determine the correlations between programs within the applications running on servers and the calls and libraries used by these programs. Failure probabilities can be obtained via a variety of methods, including examining the warranty documents of a vendor or searching online based on hardware type and serial number.

The dependency information can be encoded in XML files to store in the local databases of the cloud providers. We use the *topology-path form* to store graph information. The definition of the topology-path form and our reasons for choosing it are given in Subsection 4.4.

4.3 Subgraph Abstraction

Recall that a macro-component is an abstracted (or virtual) node in the dependency graph of a cloud-service provider that can be considered an atomic unit for the purpose of SMPC protocol execution. Creating macro-components allows us to reduce the input-set size to something that is feasible for SMPC execution. A subgraph H of the full dependency graph of a cloud-service provider S should have two properties in order to be eligible for abstraction as a macro-component. First, all components in H must be used only by S; intuitively, this is a "private" part of S's infrastructure. Second, for any two components v and w in H, the dependency information of v with respect to components outside of H is identical to that of w; that is, if v has a dependency relationship (as computed by the DAU) with a component y outside of H, then w has exactly the same dependency relationship with y. (Note that y may be inside or outside of S.) Abstraction of a subgraph that does not satisfy these properties would destroy dependency information that is needed for structural-reliability auditing.

Recall that different cloud-service providers may wish to use different abstraction policies. That is, we do not *require* that all subgraphs that satisfy the two properties given above be abstracted – some providers may wish to use a more stringent definition of a macro-component.

Suppose that G is the full dependency graph of cloud-service provider S and that G contains macro-component H. To transform G into a smaller graph G' via subgraph abstraction of H, S "collapses" H to a single node in G'; that is, S replaces H with a single node, say h, and, for every node y in G but not in H, replaces all dependency relationships in G of the form (w,y,ℓ), where w was a node in H and ℓ is a label that describes the nature of the dependency relationship between w and y, with a single dependency relationship (h,y,ℓ) in G'. (Note that there will, in general, be many nodes y that have no dependency relationships with nodes in H.) Of course, there may be more than one subgraph H that is abstracted before the reduced dependency graph is entered into the SMPC. After S receives the results of the SMPC, it combines them with the results of local fault-tree analysis of the macro-components H. For example, if F is an FS of G', $h \in F$, and f is an FS of H, then $(F-\{h\}) \cup f$ is an FS of G.

As promised, we now provide a standard example in order to illustrate the abstraction process. In this example, the SSU creates a macro-component to represent all of the components in a data center. In most cloud structures, the data centers are eligible for subgraph abstract. First, all the nodes in the data centers are owned and used by exactly one cloud-service provider. Second, all nodes in a data center communicate with the rest of the world only through the data-center gateways; they therefore have identical dependency relationships with components outside of the data center.

Figures 2 and 3 illustrate this process. Suppose that Figure 2 is the full dependency graph of cloud-service provider C_1, which contains a storage-cloud service. C_1's users' files are stored in server S_2, with two backup copies stored in server S_5 and S_7. The components inside the red box belong to a data center DC_1, which has the two properties required for asbtraction. After abstracting both DC_1 and another data-center subgraph, the SSU obtains Figure 3 as the input to the SMPC. After the abstraction process, the SSU executes the SMPC protocol with the abstracted inputs and passes

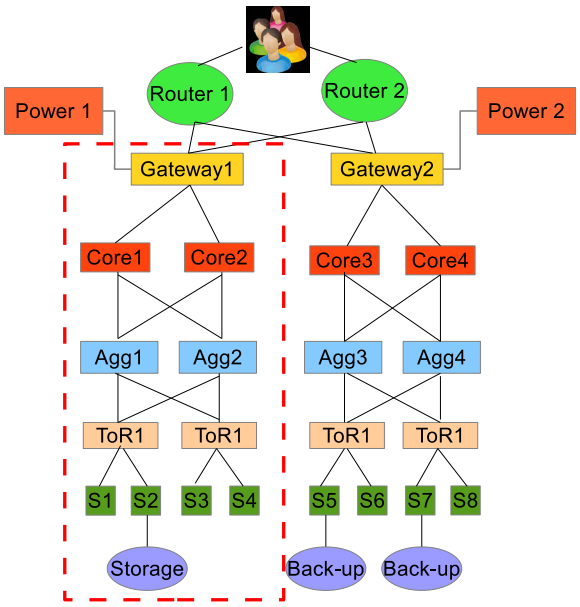


Figure 2: Full Dependency Graph of C_1

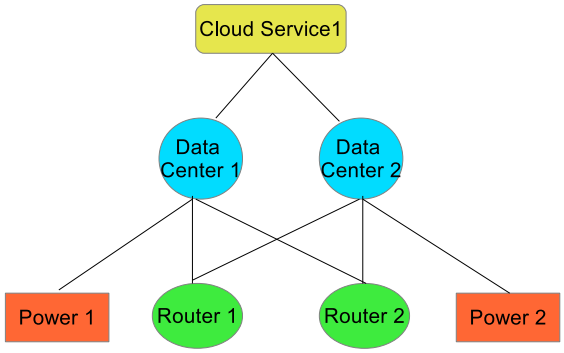


Figure 3: Abstracted Dependency Graph, suitable for SMPC

the dependency information within the macro-components (such as the red box in Figure 2 for DC_1) to the LEU for local computation.

Because the number of components in a data center is often huge, this kind of abstraction can be a crucial step toward the feasibility of SMPC.

4.4 SMPC and Local Computation

4.4.1 SSU Protocol:

Fault-tree construction: Recall that a fault tree contains two kinds of information: dependency information about components (events and links) and logical relationships among components (represented as logic gates). As we said in Subsection 4.2, we use the *topology-path form* to store dependency information. That is, we represent a leveled DAG as a set of (directed) paths in which the first node of each path is the root node of the leveled DAG, the last node is one of the leaf nodes of the leveled DAG, and the other nodes form a path from the root to the leaf. Figure 5 depicts the topology-path form of the dependency graph in Figure 3. The topology-path form of a DAG can, in the worst case, be ex-

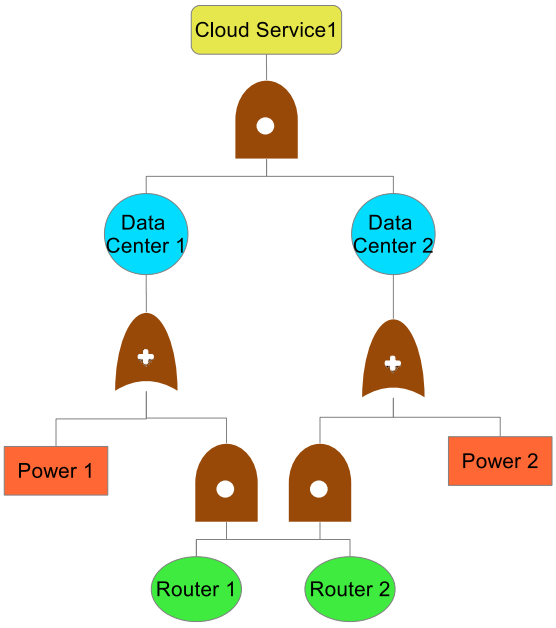


Figure 4: Fault Tree Based on Dependency Graph in Figure 3

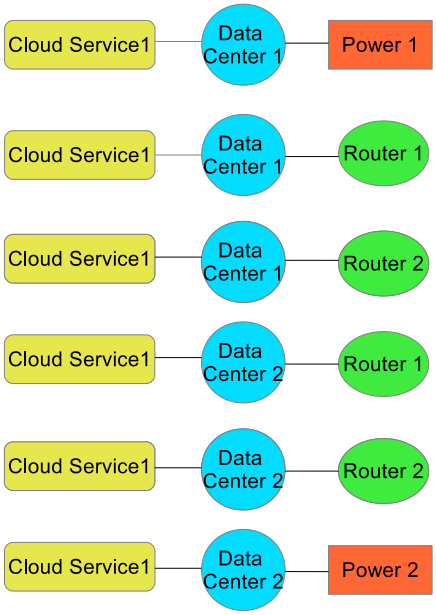


Figure 5: Topology-path Form of Dependency Graph in Figure 3.

ponentially larger than the DAG itself; thus, subgraph abstraction is crucially important, because we need to start with modest-sized DAGs. On the positive side, the topology-path form enables us to avoid using conditional statements in our SecreC code – something we must do to avoid leaking private information.

In order to capture the logical relationships among components of a cloud-service provider, we extend this representation to what we call the *topology-path form with types*. The SSU builds a "disjunction of conjunctions of disjunctions" data structure by assigning different "types" to the topology paths. Failure of the top event in the fault tree is the OR of a set of "type failures"; if any "type" that is an input to this OR fails, then the top event fails. Each "type failure" is the AND of failures of individual topology paths in the type; the "type failure" occurs only if all of the topology paths in that type fail. Failure of a topology path is the OR of failures of individual nodes on the path.

The SSU assigns a "type ID" to each topology path; the type ID is a function of the component IDs in the nodes on the path. Type IDs and the mapping from sets of component IDs to type IDs can be agreed upon by all of the relevant cloud-service providers and stored in a table before the P-SRA execution starts; so the SSU simply needs to look up type IDs during the protocol execution. To construct the fault tree from the topology-path form with types, the SSU traverses each path and constructs an OR gate for each path, the inputs to which are the nodes on the path. It then constructs an AND gate for each type of path, the inputs to which are the outputs of the OR gates of the paths in the type. Finally, the SSU constructs an OR gate whose inputs are the outputs of all the AND gates in the previous step.

For example, starting with the fault tree of Figure 4, the SSU can classify the topology paths of Figure 5 into two types. Type 1 includes the two topology paths ($Cloud\ Service_1$, DC_1, $Power_1$) and ($Cloud\ Service_1$, DC_2, $Power_2$). Type 2 includes the other four topology paths. It can be verified that the minimal FSes of the fault tree generated by the topology path form with types are the same as the minimal FSes of the fault tree in Figure 4; we defer a formal statement and proof of this fact to a future, longer version of this paper that includes all of the necessary details of fault-tree analysis given in [28].

Generate input for the SMPC: After constructing the topology paths with types, the SSU "pads" the paths so that they all have the same length L, where L is an agreed-upon global parameter distributed by the P-SRA host. Padding is accomplished by adding the required number of "dummy" nodes in which the component ID is 0. (Here, "0" is any fixed value that is *not* a valid, real component ID.) Similarly, the SSU adds a random number of "0 paths," which are topology paths with types in which all of the nodes have component ID 0. The types of these 0 paths can be assigned randomly, because they do not affect the result – the 0 paths never fail. The purpose of this padding step is to prevent leakage of structural information about the cloud-service providers' architectures, including the number of topology paths or the size of each path. Finally, the SSU splits the padded paths into secret shares that are input to the SMPC protocol.

Identify common dependencies: A component is in the *common dependency* of cloud-service provider S_i if it is in the fault tree of S_i *and* in the fault tree of at least one other cloud-service provider S_j, $j \neq i$. Conceptually, the common dependency is very easy to compute by doing multiple (privacy-preserving) set intersections, followed by one (privacy-preserving) union. However, we need to do this computation without conditional statements; see Algorithm 1 for a method of doing so.

Calculate failure sets: Finally, the SMPC protocol integrates the fault trees of all participating cloud-service providers into a unified, global fault tree and performs fault-tree analysis. It can execute either algorithm 2, which computes minimal FSes, or algorithm 3, a heuristic "failure-sampling" algorithm that is faster than algorithm 2 and computes FSes but does not guarantee that the FSes returned are minimal.

Algorithm 2 works as follows. Let T denote the unified, global fault tree; because we represent fault trees as padded, topology

Algorithm 1: Common-Dependency Finder

Input: Fault tree T_i, $i = 1$ *to* N, where N is the number of participating cloud-service providers
Output: Common Dependency

foreach T_I *and* $T_J, I \neq J$ **do**
  private mask.clear();
  foreach $node_i \in T_I$ *and* $node_j \in T_J$ **do**
    $private\ mask[i][j] = (node_i.ID == node_j.ID)$;
  private CommonDep.clear();
  foreach $node_i \in T_i$ *and* $node_j \in T_j$ **do**
    $private\ CommonDep[i] = mask[i][j] \times node_j.ID + CommonDep[i]$;
  private CommonDependent.append(CommonDep);
return private CommonDependent;

Algorithm 2: Minimal-FS algorithm

Input: Global Fault tree T
Output: MinimalFS

foreach $private\ path_i \in T$ **do**
  foreach $private\ node_j \in private\ path_i$ **do**
    $private\ path_i.FS.apped(node_j)$;
    /* each path corresponds to an OR gate with input as the nodes along the path */
foreach $AndGate_i \in T$ **do**
  $AndGate_i.FS.clear()$;
  foreach $path_j \in AndGate_i$ **do**
    $AndGate_i.FS \leftarrow AndGate_i.FS \times path_j.FS$;
    /* process the AndGate for each type of topology paths */
    /* FS of $AndGate_i$ is the Cartesian Product of $AndGate_i.FS$ and $path_j.FS$. */
private minimalFS.clear();
foreach $AndGate_i \in T$ **do**
  $minimalFS.append(AndGate_i.FS)$;
  /* process the OR gate connecting to the And Gates */
/* reduce redundant items in $minimumFS$ and assign the result to $minimalFS$, and then simplify minimalFS. */
$minimalFS \leftarrow reduce\_redundancy(minimalFS)$;
$minimalFS \leftarrow simplify(minimalFS)$;
return minimalFS;

Algorithm 3: Failure-Sampling Algorithm

Input: Global Fault tree T and the number of samples N
Output: FSes

private FSes.clear();
for $i \leftarrow 1$ *to* N **do**
  foreach $private\ path_j \in T$ **do**
    private tmp = 0;
    foreach $private\ node_s \in path_j$ **do**
      foreach $private\ node_k \in T$ **do**
        private random = 0 or 1 based on randomly flipping a fair coin;
        $tmp+ = random \times (node_s.ID == node_k.ID)$;
    /* calculate whether $path_j$ fails */
    $path_j.failure = (tmp > 0)$;
  foreach $AndGate_i \in T$ **do**
    $AndGate_i.failure = true$;
    foreach $path_j \in AndGate_i$ **do**
      $AndGate_i.failure = AndGate_i.failure$ && $path_j.failure$;
  private serviceFailure = false;
  foreach $AndGate_i \in T$ **do**
    $serviceFailure = AndGate_i.failure \parallel serviceFailure$;
  open(serviceFailure);
  if *serviceFailure* **then**
    FS.clear();
    foreach $path_i \in T$ **do**
      FS.append($path_i.failure$);
    FSes.append(FS);
return FS;

paths with types, T is simply the union of the fault trees of the individual cloud-service providers. The algorithm traverses T, producing FSes for each of the visited events. Basic events generate FSes containing only themselves, while non-basic events produce FSes based on the FSes of their child events and their gate types. For an OR gate, any FS of one of the input nodes is an FS of the OR. For an AND gate, we first take the cartesian product of the sets of FSes of the input nodes and then combine each element of the cartesian product into a single FS by taking a union. The last step of algorithm 2 reduces the top event's FSes to minimal FSes.

Algorithm 3 works as follows. For each sampling round, the algorithm randomly assigns 1 or 0 to the basic events (leaves) of the fault tree T, where 1 represents failure and 0 represents non-failure. Starting from such an assignment, the algorithm can assign 1s and 0s to all non-basic events in T, using the logic gates. At the end of each sampling round, the algorithm checks whether the top event fails. If the top event fails, then the failure nodes in this sampling round are an FS. The algorithm runs for a large number of sampling rounds to find FSes. In [28], it is proven that most of the critical FSes can be found in this fashion but that the FSes are not necessarily minimal.

4.4.2 *LEU Protocol:*

The LEU in the P-SRA client of cloud-service provider S performs fault-tree analysis on S's macro-components. The LEU can use algorithm 2 or algorithm 3. Note that these computations are done locally and do not involve SMPC; so, large macro-components are not necessarily bottlenecks in P-SRA computation. It is very advantageous when a cloud-service provider can partition its infrastructure in a way that produces a modest number of large macro-components, each one of which is a "virtual node" in the SMPC.

4.5 Privacy-preserving Output Delivery

Recall that P-SRA performs an SMPC on dependency information that is potentially shared by multiple cloud-service providers and performs local computation on dependency information that is definitely relevant to only one provider. The intermediate results include common dependency and minimal FSes (or FSes if algorithm 3 was used). We now turn our attention to the outputs that P-SRA delivers to cloud-service providers and to cloud-service users. P-SRA gives cloud services the flexibility to choose exactly what should be output. However, we argue that the outputs should not compromise the privacy of cloud-service providers and must be illustrative of correlated-failure risk and reliability. We propose some specific outputs that satisfying these two requirements.

4.5.1 *Output for Cloud-Service Providers*

Common dependency: The common dependency set, as defined in Subsection 4.4, includes components shared by more than one cloud-service provider. It is useful for cloud-service providers, in that it can make them aware of unexpected correlation with other

providers. They can then deploy independent components as backups to mitigate the impact of the common dependency or switch to independent components to improve the reliability of their service and decrease the correlation with other cloud-service providers.

Partial failure sets: If F is a (minimal) FS for cloud-service provider S, then the corresponding partial (minimal) FS is simply all of the components in F that are used by S. Such a partial FS gives S information about components whose failure may lead to an outage because equipment that is controlled by some other service provider fails. If S can build enough redundancy into its internal infrastructure to avoid failure of all of the components in this partial FS, then it will not suffer an outage because of F, regardless of what happens outside.

Sometimes the number of FSes is huge. If this is the case, we need to rank the FSes first and only output the partial failure sets of the top-ranked FSes. Ranking of comprehensive failure sets can be either probability-based or size-based [28].

4.5.2 Output for Cloud-Service Users

Common-dependency ratio: Cloud-service users can obtain a *common-dependency ratio* for each cloud-service provider. We define the common-dependency ratio of cloud-service provider S as the fraction of components in S that are shared with at least one other cloud-service provider. Intuitively, the larger the common-dependency ratio, the higher the risk of correlated failure. In the extreme case, if a cloud service is deployed entirely on an external cloud infrastructure (as is the case with some Software-as-a-Service providers), then its common-dependency ratio is 1. If a cloud-service provider shares no components with other providers, then its common-dependency ratio is 0. Cloud-service users can evaluate risk and choose cloud providers in part based on this ratio. This common-dependency ratio does not reveal any information about internal architecture of the cloud providers.

Overall failure probabilities of cloud services: Cloud-service users can compare these failure probabilities with the reliability measures promised by the providers in their service-level agreements and evaluate whether they are subject the risk of unexpected, correlated failure. Failure probabilities, like common-dependency ratios, do not reveal the architectures of the service providers.

Top-ranked failure sets: Recall from Subsection 2.1 that, in its SMPC, P-SRA computes the *secret shares* of the (minimal) FSes of the cloud-service providers. As we have seen, an SMPC program can compute from those shares the partial (minimal) FSes that are delivered to the providers. However, an alternative SMPC program could use those shares to rank the (minimal) FSes based on failure probability or size. Then a small set of *top-ranked* (minimal) FSes can be delivered to cloud-service users. Just a few top-ranked sets can give users useful information about how to avoid correlated failures; they reveal some information about the cloud-service architectures, but this may be tolerable in some markets.

5. IMPLEMENTATION

5.1 P-SRA Prototype

The Sharemind SecreC platform includes a set of **miners** to execute the SMPC protocols and a controller to coordinate the miners. The SMPC protocols run by the miners are coded in **SecreC**, a C-like programming language for SMPC programs. Variables in SecreC may be declared **public** or **private**. The language supports basic arithmetic, and some matrix and vector operations. SecreC uses a client/server model, with multiple clients providing (secret-shared) input to the miners, which execute the SMPC protocol.

Our implementation of P-SRA is illustrated in Figure 6. The miners are installed in the SMPC module of the P-SRA host. The P-SRA clients and P-SRA host upload their SecreC scripts to the miners. The SecreC scripts are executed by the P-SRA clients remotely through the C++ interface of the controller or by the P-SRA host locally. The P-SRA clients execute the SecreC scripts to split their inputs into secret shares and to read and write shares of inputs or intermediate results from the miners' secure databases. The P-SRA host executes the SecreC scripts to perform the SMPC protocol that identifies common dependencies and performs fault-tree analysis. SecreC uses SSL for secure communication between miners and clients.

From Figure 6, it is not immediately obvious what one gains from using the Sharemind platform and SMPC instead of a trusted-party SRA as in [28]: All of the miners, *i.e.*, the nodes that execute the SMPC protocol, run inside the P-SRA host; if they share information, then together they constitute a trusted party. However, this system configuration is merely the default of the currently available Sharemind "demo," and we have used it only in order to be able to build this proof-of-concept prototype as quickly as possible. In a real, deployed P-SRA (or any real SMPC-based application coded in SecreC), the miners would run on separate, independently administered machines and communicate over a network; no substantive changes to the SecreC compiler are needed to create executables that run on separate networked nodes, and we expect future Sharemind releases to create them. Thus, moving to P-SRA from the SRA of Zhai *et al.* [28], in which one trusted auditor handles all of the sensitive information supplied by the cloud-service providers, is tantamount to "distributing trust" over a number of independently administered auditors no one of which is trusted with any sensitive information, in the sense that each receives only a secret share of every input; if the independent owners of the networked nodes that run the auditors do not collude, then the clients' inputs will remain private. This SMPC architecture, in which clients (or "input providers"), rather than executing an SMPC protocol themselves, instead send their input shares to independently administered computational agents that then execute the SMPC protocol, is known as *secure outsourcing* in the SMPC literature; see, *e.g.*, Gupta *et al.* [12] for more information about secure outsourcing's history, its practical advantages, and its use in a routing application.

The SecreC compiler relieves programmers of the need to code standard cryptographic functionality. In particular, it generates secret-sharing code automatically. Currently, it uses additive secret sharing and thus guarantees privacy only against honest-but-curious adversaries. We expect future releases to incorporate more elaborate secret-sharing techniques and hence to protect input providers against stronger classes of adversaries.

The DAU and LEU in the P-SRA client are written in Python. The DAU uses the SNMPv2 library support from NetSNMP to collect network dependencies; it uses lshw, a lightweight tool that extracts detailed hardware configuration from the local machines, to collect hardware dependencies; it uses ps and gprof to collect software dependencies. The LEU uses the Network-X library [2] to process the dependency-graph data structures.

5.2 Case Study

This section outlines a case study to illustrate the prototype's operation. Let CS_1 denote a cloud service provided by cloud provider C_1. To improve the reliability of CS_1, C_1 decides to use providers C_2 and C_3 for redundant storage. Only C_1 serves users directly, while C_2 and C_3 provide lower-level services to C_1. This architec-

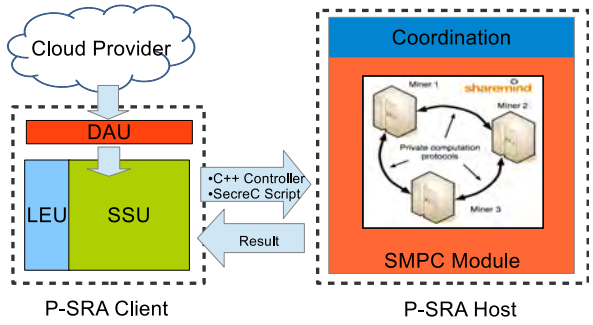


Figure 6: Implementation in Sharemind SecreC

ture is analogous to iCloud, Apple's storage service, which uses Amazon EC2 and Microsoft Azure for redundant backup storage.

Suppose Alice, a user of CS_1, wants to deploy a MapReduce function using CS_1. Alice deploys the MapReduce Master on a data center DC_1 of C_1, and C_1 uses a data center DC_2 of C_2 and a data center D_3 of C_3 as backup for the MapReduce Master. However, as in Figure 7, C_1, C_2, and C_3 depend on the same power station P_1. Alice and all three cloud providers are unaware of this situation. Therefore, they may overestimate the reliability of the MapReduce Master and underestimate the risk of correlated failure. If P_1 goes down, Alice's MapReduce may not work, because all the backup data centers may fail simultaneously.

The P-SRA system can help to identify P_1 as the common dependency in the cloud structure supporting CS_1 and provide multiple measures of reliability and correlated failure risk (the failure probability for Alice and partial FSes for C_1), without revealing significant private information about C_1, C_2, and C_3. Alice need not learn private topological information about the three cloud providers (or even learn of the existence of C_2 and C_3) but can accurately assess the failure risk via the P-SRA system. Meanwhile, C_1 can improve the reliability of CS_1 by connecting to alternative power stations or seeking redundancy from cloud providers other than C_2 and C_3, without learning private topological information about C_2 and C_3.

To further illustrate P-SRA, we display the details within a data center. There are a large number of components in data centers including servers, racks, switches, aggregate switches and routers. For simplicity, we generate the same topology for all the data centers and show only the components in DC_1 – see Figure 8. The MapReduce Master is installed on server 5 of DC_1. The DAUs of C_1, C_2, and C_3 collect the dependency information of each cloud provider. Then the SSUs abstract macro-components for each cloud provider using standard data-center abstraction. The SSUs pass the information within the data centers to the LEUs and establish connections with each other and the P-SRA host to execute the SMPC protocol. The LEUs perform fault-tree analysis on the dependency information within the data centers locally. The results of the SMPC and the local computation are then combined as explained in Subsection 4.3.

The P-SRA system is practical in this case study. Even using a laptop with little computational power, equipped only with a 2.5GHz 2-core Intel i5 CPU and 2.00GB of memory, the running time used by the SSUs and P-SRA host to find the common dependency was approximately 20 seconds; the time to perform the fault-tree analysis was approximately 13 minutes using the minimal-FS algorithm and 55 seconds using the failure-sampling algorithm with 100 rounds. The running time for the LEUs deployed on

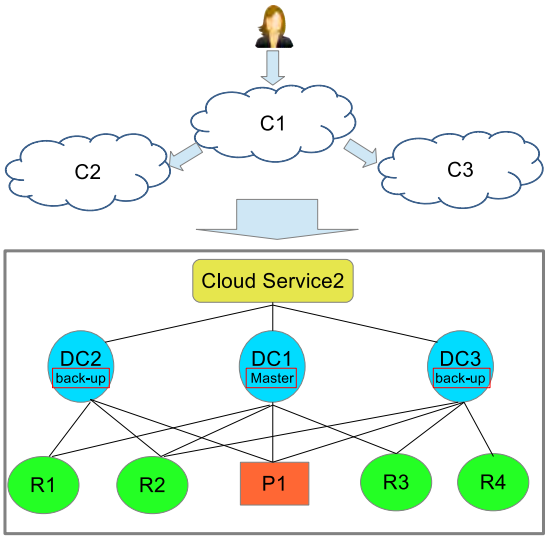


Figure 7: Multi-level Structure of Cloud Service

servers equipped with two 2.8GHz 4-core Intel Xeon CPUs and 16GB of memory was less than 30 seconds for both the minimal-FS algorithm and the failure-sampling algorithm.

5.3 Large-Scale Simulation

This section evaluates the P-SRA prototype using larger-scale simulations. Our data set is synthesized based on the widely accepted three-stage fat-tree cloud model [15] and scaled up to what we expect to find in real cloud structures. For the SMPC protocol run by the P-SRA host and the SSUs of P-SRA clients, we test the running time of the common-dependency-finder Algorithm 1, the minimal-FS Algorithm 2, and the failure-sampling Algorithm 3. Our output for both cloud-service providers and users can be computed efficiently from the common dependency and the (minimal) FSes.

We test the five cases summarized in Table 1. For simplicity, we generate only homogeneous cloud providers. In Table 1, the numbers of data centers, Internet routers, and power stations are numbers per cloud provider. The common-dependency ratio is as defined in Subsection 4.5.2. The padding ratio is the number of zeros with which the topology paths were padded divided by the total number of nodes on the topology paths after padding.

The five cases are intended to be illustrative of configurations broadly comparable to realistic multi-cloud services. To the best of our knowledge, it is uncommon for any cloud services to be deployed on more than three cloud providers or distributed over more than 10 data centers, because the total number of data centers worldwide is limited, and cloud-service management costs increase quickly as data centers are added. Amazon, one of the giant cloud providers, owns only 15 data centers globally [1]; Microsoft Azure has fewer than 10 data centers [3].

Figure 9 summarizes measured P-SRA computation performance. The P-SRA host and SSUs of the P-SRA clients were run on laptops with 2.5GHz 2-core Intel i5 CPU and 2.00GB of memory. We used these machines because the SecreC platform supported only Microsoft Windows when we started this work. We expect that performance would improve using higher-powered machines.

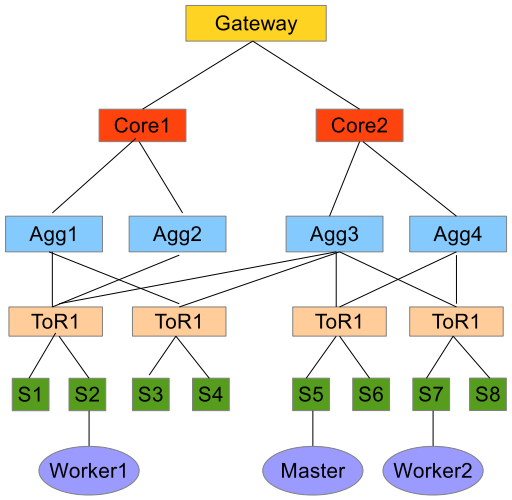


Figure 8: Components in Data Center DC_1: Core, Agg, and ToR represent core router, aggregation switch, and top-of-rack switch.

	Case 1	Case 2	Case 3	Case 4	Case 5
# of cloud providers	2	2	3	3	2
# of data center	1	3	8	10	3
# of internet router	3	5	10	15	5
# of power stations	1	2	3	5	2
ratio of common dep.	0.8	0.2	0.2	0.2	0.2
ratio of padding	0.0	0.0	0.0	0.0	0.5

Table 1: Configuration of Test Data Sets

The common-dependency finder exhibits reasonable efficiency in all five cases, the runtimes of which are all less than 3 minutes. The minimal-FS algorithm yields exact minimal FSes (but takes exponential time in the worst case, because the problem in NP-hard), while the failure-sampling algorithm produces FSes approximating the minimal FSes and runs in polynomial time. In Cases 4 and 5, the minimal-FS algorithm was aborted before it finished, and thus no results are shown for them in Figure 9. The runtimes of other simulations of the minimal-FS algorithm and the failure-sampling algorithm range from 1 to 50 hours depending on the configuration. As the number of nodes increases, the efficiency of fault-tree analysis drops quickly. Case 5 shows that the cost of padding to conceal the statistical information of each topology path is high. Therefore, subgraph abstraction to reduce the size of the dependency graphs is important for the efficiency of fault-tree analysis in P-SRA.

For the LEUs in the P-SRA clients performing local computations, we also test the running times of both the minimal-FS algorithm and the failure-sampling algorithm. For the LEUs running on servers with two 2.8GHz 4-core Intel Xeon CPUs and 16GB of memory, the failure-sampling algorithm with 10^6 rounds on a data center with 13,824 servers and 3000 switches takes around 6 hours. For details, see Table 2. "FS round 10^n" denotes the runtime (in minutes) of the failure-sampling algorithm running 10^n rounds. "Minimal FS" denotes the runtime of the minimal-FS algorithm.

6. CONCLUSIONS AND FUTURE WORK

We have designed P-SRA, a private, structural-reliability auditor for cloud services based on secure, multi-party computation, and

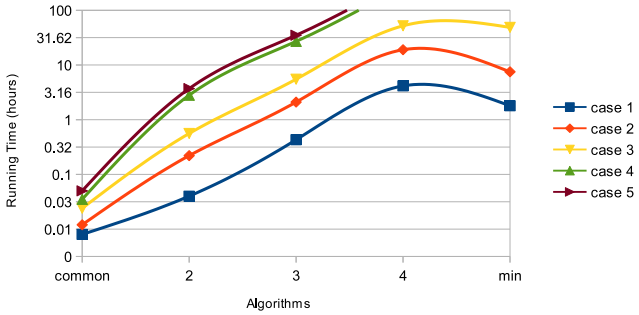


Figure 9: Performance of algorithms. On the X axis, "Common" represents the common-dependency finder, 2 through 4 represent the failure-sampling algorithm with sampling rounds at various powers of 10, and "Min" represents the minimal-FS algorithm.

Table 2: Performance of the LEU of a P-SRA client

Configuration	Case 1	Case 2	Case 3	Case 4	Case 5
# of switch ports	4	8	16	24	48
# of core routers	4	16	64	144	576
# of agg switches	8	32	128	288	1152
# of ToR switches	8	32	128	288	1152
# of servers	16	128	1024	3456	13824
Total # components	40	216	1360	4200	16752
Runtime (minutes)					
FS round 10^3	< 0.7	< 0.7	< 0.7	< 0.7	< 0.7
FS round 10^4	0.7	0.7	1.7	2.3	6.9
FS round 10^5	0.8	0.9	5.3	28.1	6.9
FS round 10^6	1.7	4.5	65.0	243.5	462.9
FS round 10^7	28.3	56.6	512.1	NA	NA
Minimal FS	0.8	14.8	309.7	NA	NA

prototyped it using the Sharemind SecreC platform. In addition, we have explored the use of data partitioning and subgraph abstraction in secure, multi-party computations on large graphs, with promising results. Our preliminary experiments and simulations indicate that P-SRA could be a practical, off-line service, at least for small-scale cloud services or for ones that permit significant subgraph abstraction. There are many interesting directions for future work, including: (1) Although our preliminary experiments indicate that the cost of privacy in structural reliability auditing (*i.e.*, the additional cost of using P-SRA instead of SRA) is not prohibitive, it would be useful to measure this cost more precisely with more exhaustive experiments. (2) It will be interesting to seek more efficient algorithms for fault-tree analysis and/or a more efficient P-SRA implementation; both would enable us to test P-SRA on larger cloud architectures. (3) Note that we assumed, following Zhai *et al.* [28], that dependency graphs of cloud services are acyclic, but they need not be. Tunneling-within-tunneling of the type already in use in MPLS and corporate VPNs could (perhaps unintentionally) create cyclic dependencies if used in clouds. Thus, it will be worthwhile to develop structural-reliability auditing techniques that apply to cyclic dependency graphs. (4) P-SRA partitions components based on the fact that some physical equipment is used by exactly one service provider and hence cannot cause the failure of another provider's service, but this type of partitioning has limitations. If, for example, two cloud-service providers purchase large numbers of hard drives of the same make and model from the same batch, and that batch is discovered to be faulty, then the two services have

a common dependency on this faulty batch of drives. P-SRA's data partitioning could hide this common dependency, because the hard drives could be considered "private" equipment by both services. It will be worthwhile to extend P-SRA so that it can discover this type of common dependency while retaining the efficiency provided by data partitioning and subgraph abstraction.

7. ACKNOWLEDGEMENTS

We thank Ennan Zhai, Aaron Segal, Debayan Gupta, and the anonymous reviewers for their helpful comments. This material is based on research sponsored by NSF grants CNS-1016875 and CNS-1149936, ONR grant N00014-12-1-0478, DARPA contract FA8750-13-2-0058, and a gift from Google Research.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, ONR, DARPA, or the U.S. Government.

8. REFERENCES

[1] Amazon web services global infrastructure. `http://aws.amazon.com/en/about-aws/globalinfrastructure/`.

[2] NetworkX. `http://networkx.github.com/`.

[3] Windows azure. `http://en.wikipedia.org/wiki/Windows_Azure`.

[4] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Symposium on Computer and Communication Security*, pages 257–266, 2008.

[5] S. Bleikertz, M. Schunter, C. W. Probst, D. Pendarakis, and K. Eriksson. Security audits of multi-tier virtual infrastructures in public infrastructure clouds. In *ACM Cloud Computing Security Workshop*, pages 93–102, 2010.

[6] D. Bogdanov and A. Kalu. Pushing back the rain – how to create trustworthy services in the cloud. *ISACA Journal*, 3:49–51, 2013. Available at `http://www.isaca.org/Journal/Past-Issues/2013/Volume-3/Pages/default.aspx`.

[7] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, 2010.

[8] B. Butler. Cloud storage viable option, but proceed carefully, 2013. Available at `http://www.networkworld.com/news/2013/010313-gartner-storage-265460.html`.

[9] B. Butler. Top 10 cloud storage providers, 2013. Available at `http://www.networkworld.com/news/2013/010313-gartner-cloud-storage-265459.html`.

[10] I. Damgård, M. Geisler, M. Krøigård, and J. Nielsen. Asynchronous multiparty computation: Theory and implementation. In S. Jarecki and G. Tsudik, editors, *Public Key Cryptography – PKC 2009*, pages 160–179. Springer Verlag, LNCS 5443, 2009.

[11] C. A. Ericson II. *Hazard analysis techniques for system safety*. John Wiley and Sons, 2000.

[12] D. Gupta, A. Segal, A. Panda, G. Segev, M. Schapira, J. Feigenbaum, J. Rexford, and S. Shenker. A New Approach to Interdomain Routing Based on Secure Multi-Party Computation. In *ACM SIGCOMM Workshop on Hot Topics in Networks*, 2012.

[13] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462, 2010.

[14] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, pages 298–302, 2004.

[15] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-tolerant Layer 2 Data Center Network Fabric. In *ACM SIGCOMM*, pages 39–50, 2009.

[16] W. Oremus. Internet outages highlight problem for cloud computing: Actual clouds, 2012. Available at `http://www.slate.com/blogs/future_tense/2012/07/02/amazon_ec2_outage_netflix_pinterest_instagram_down_after_aws_cloud_loses_power.html`.

[17] M. A. Shah, M. Baker, J. C. Mogul, and R. Swaminathan. Auditing to keep online storage services honest. In *USENIX Workshop on Hot Topics in Operating Systems*, 2007.

[18] M. A. Shah, R. Swaminathan, and M. Baker. Privacy-preserving audit and extraction of digital contents. Cryptology ePrint Archive, Report 2008/186, 2008. Available at `http://eprint.iacr.org/2008/186/`.

[19] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. US Nuclear Regulatory Commission, 1981.

[20] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers*, 62(2):362–375, 2013.

[21] C. Wang, K. Ren, W. Lou, and J. Li. Toward publicly auditable secure cloud data storage services. *IEEE Network*, 24(4):19–24, 2010.

[22] C. Wang, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *IEEE INFOCOM*, pages 525–533, 2010.

[23] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(5):847–859, 2011.

[24] K. Yang and X. Jia. Data storage auditing service in cloud computing: challenges, methods and opportunities. *World Wide Web*, 15(4):409–428, 2012.

[25] A. C. Yao. Protocols for secure computation. In *IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[26] A. C. Yao. How to generate and exchange secrets. In *IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[27] E. Zhai, R. Chen, D. I. Wolinsky, and B. Ford. An untold story of redundant clouds: Making your service deployment truly reliable. In *ACM Workshop on Hot Topics in Dependable Systems*, 2013.

[28] E. Zhai, D. I. Wolinsky, H. Xiao, H. Liu, X. Su, and B. Ford. Auditing the structural reliability of the clouds. Technical Report YALEU/DCS/TR-1479, July 2013. Available at `http://www.cs.yale.edu/publications/techreports/tr1479.pdf`.

Practical and Privacy-Preserving Policy Compliance for Outsourced Data

Giovanni Di Crescenzo[1(✉)], Joan Feigenbaum[2], Debayan Gupta[2], Euthimios Panagos[1], Jason Perry[3], and Rebecca N. Wright[3]

[1] Applied Communication Sciences, Middlesex County, NJ, USA
{gdicrescenzo,epanagos}@appcomsci.com
[2] Yale University, New Haven, CT, USA
{joan.feigenbaum,debayan.gupta}@yale.edu
[3] Rutgers University, Middlesex County, NJ, USA
{jasperry,rebecca.wright}@cs.rutgers.edu

Abstract. We consider a scenario for data outsourcing that supports performing database queries in the following three-party model: a client interested in making database queries, a data owner providing its database for client access, and a server (e.g., a cloud server) holding the (encrypted) outsourced data and helping both other parties. In this scenario, a natural problem is that of designing efficient and privacy-preserving protocols for checking compliance of a client's queries to the data owner's query compliance policy. We propose a cryptographic model for the study of such protocols, defined so that they can compose with an underlying database retrieval protocol (with no query compliance policy) in the same participant model. Our main result is a set of new protocols that satisfy a combination of natural correctness, privacy, and efficiency requirements. Technical contributions of independent interest include the use of equality-preserving encryption to produce highly practical symmetric-cryptography protocols (i.e., *two orders of magnitude faster* than "Yao-like" protocols), and the use of a query rewriting technique that maintains privacy of the compliance result.

1 Introduction

The recent information technology trend of outsourcing "big data" in the "cloud" is being embraced in banking, finance, government and other areas. Banks and financial institutions need to process huge data volumes on a daily basis; in government, large databases are needed in many contexts (e.g., no-fly lists, metadata of communication records, etc.). Cloud storage and computing provide tremendous efficiency and utility for users (as exemplified by the "database-as-a-service" application paradigm), but they also create privacy risks. To mitigate these risks, database-management systems can use *privacy-preserving* data-retrieval protocols that allow users to submit queries and receive results in a way that users learn nothing about the contents of a database except the results of their queries, data owners do not learn which queries are submitted. Of critical importance

R. Bohme et al. (Eds.): FC 2014 Workshops, LNCS 8438, pp. 181-194, 2014
DOI: 10.1007/978-3-662-44774-1_15

for the success of database management systems is the notion of *access control*, which requires carefully crafted *data-access policies*. Compliance of these policies can be enforced by the database-management system but might need to be confidential, as the policies themselves may reveal sensitive facts about the data and its owner. In this paper, we formalize, design, and analyze practical and privacy-preserving policy compliance protocols for outsourced data.

Our Problem: Our goal is to augment natural encrypted database retrieval solutions with a query authorization property based on compliance to a policy, while preserving the privacy and efficiency properties of the basic database retrieval solution. For consistency with the database-as-a-service model, and to achieve practical solutions, we consider a 3-party model, shown in Fig. 1, including a client C (interested in private data retrieval), a data owner D (offering data for retrieval conditioned on compliance to a query-specific policy), and a server S (e.g., a cloud server) helping both parties to achieve their goals. In this paper, we focus on the policy-compliance building block. Our solutions can be combined in a modular fashion with database retrieval (DR) protocols in this 3-party model, provided that they satisfy some natural structure and properties (described later). Such protocols already exist in the literature (e.g., [5,11,18]).

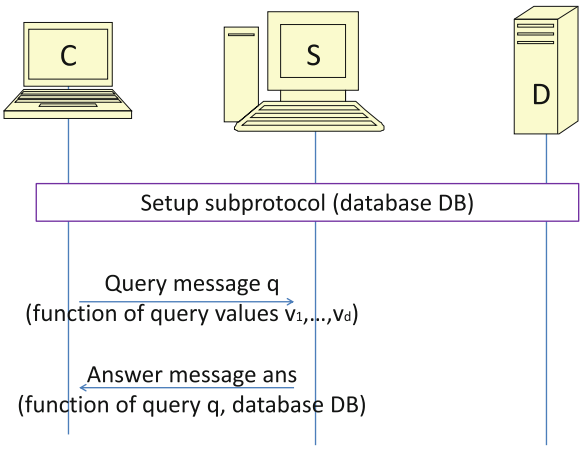


Fig. 1. Structure of a database-retrieval protocol

Our Contributions. We investigate the modeling and design of database policy compliance (DPC) protocols that combine with known DR protocols, as shown in Fig. 2, and that satisfy the following novel set of requirements:

1. *Preservation of Query Correctness*: A client that could retrieve all of the records that satisfy its query using a DR protocol can still do so if the query is compliant;

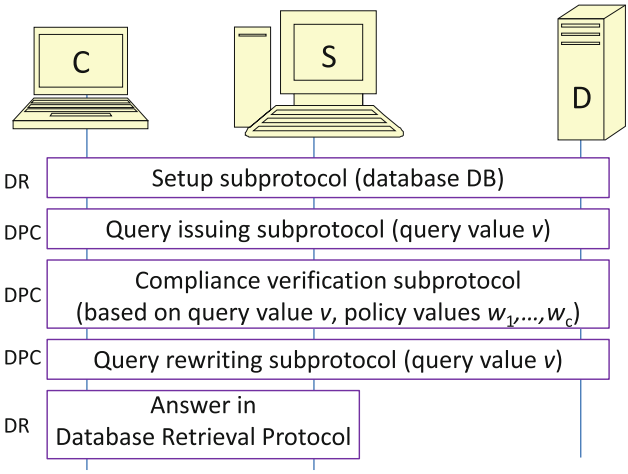


Fig. 2. Composition of a DR protocol with a DPC protocol

2. *Compliance Completeness*: All queries that satisfy (resp., do not satisfy) the policy are found to be compliant with probability 1 (resp., with negligible[1] probability);
3. *Compliance Soundness*: For any efficient (and even malicious) adversary impersonating the client, the data owner can correctly compute (except with negligible probability) the policy compliance of whichever query message is received and answered by the server according to the DR protocol;
4. *Privacy*: Privacy of database values, policy values and query values is preserved, in that no efficient semi-honest adversary corrupting one among the parties C, S, D learns more information at the end of the protocol than whatever is efficiently computable from the following: the system parameters (which are intended to be known by all parties), the compliance bit b (if the corrupted party is D, who is intended to learn b), the query message Q' (if the corrupted party is S, who is intended to learn Q'), where Q' has the same distribution as the query message Q in the DR protocol when the query is compliant, or otherwise represents a query that does not match any records in DB (to reduce leakage of the policy-compliance result to S or C);
5. *Efficiency*: The protocol should have low time, communication and round complexity. One of the most significant design criteria we target to reduce computational overhead of query compliance checking is to minimize or eliminate costly public-key cryptographic operations and to achieve protocols faster than a direct application of secure function evaluation techniques.

Implicit in the above privacy requirement is the fact that the protocol does not reveal new information about the data owner's policy to client and server,

[1] A function is *negligible* if for any positive polynomial p and all sufficiently large natural numbers $\sigma \in \mathcal{N}$, it is smaller than $1/p(\sigma)$.

other than what is revealed to clients by fulfilled queries. Still, to hide some additional information about the policy, the privacy requirement also demands that the result of a non-compliant query is indistinguishable from a query that matches zero records in the database, so that the protocol does not reveal to clients whether a query that returns no matches does so because it is non-compliant or because there are actually no matching records[2].

We design three protocols for enforcing compliance of *keyword search* queries. We only consider *whitelist* (resp., *blacklist*) policy types, where the query is compliant only if the query value is equal to one (resp., none) of the policy values. For such query and policy types, we provide highly efficient and scalable database policy compliance protocols that satisfy all our requirements, as detailed below (the PRP assumption being the existence of pseudo-random permutations).

Requirement	Protocol π_1	Protocol π_2	Protocol π_3
Correctness preservation	If DR protocol satisfies Added Property 1	If DR protocol satisfies Added Property 2	If DR protocol satisfies Added Property 1
Compliance completeness	Under no complexity assumption	Under no complexity assumption	Under no complexity assumption
Compliance soundness	(not satisfied)	Under no complexity assumption	Under no complexity assumption
Privacy	Under PRP assumption (some leakage to S)	Under PRP assumption (some leakage to S)	Under PRP assumption And based on SFE
Time	Linear in policy size	Linear in policy size	Linear in policy size
Communication	Linear in policy size	Linear in policy size	Linear in policy size
Rounds	$O(1)$ in policy size	$O(1)$ in policy size	$O(1)$ in policy size

An additional important property is that all our 3 DPC protocols only require $O(1)$ cryptographic operations per query and policy value, and are about two orders of magnitude faster than 2-party arbitrary function evaluation protocols [19], which require at least $\Omega(\ell)$ cryptographic operations per query and policy value, where ℓ denotes the length of these values (even in recent optimized solutions). Just like achieved previously for DR protocols in the 3-party model, our DPC protocols not only minimize or eliminate costly public-key cryptography operations, but they provide concrete time efficiency, which we document through performance numbers from our implementations (in Sect. 4). Our solutions rest on two main technical contributions: (1) using equality-preserving symmetric encryption with multiple keys shared among different subsets of

[2] Of course, sometimes a client is able to distinguish these cases due to auxiliary information.

parties (building on [3]) for efficient 3-party computation on encrypted data, and (2) performing policy-based query rewriting to make the results of non-compliant queries indistinguishable from queries matching no records. Formal definitions and proofs are omitted due to space restrictions.

Related Work. To the best of our knowledge, there is no previous work on privacy-preserving, efficient, *query* policy compliance checking for database queries. That is, although there has been previous work on 3-party protocols in which the data set being searched is encrypted, the query is kept private, and queries are only allowed if they satisfy certain structural conditions, we are unaware of previous work in which the restriction on allowable queries (i.e., the policy) depends on the query and/or is kept private from the clients. Existing work in this area focuses on policy conditions that mainly depend on the database attributes (see, e.g., [5,8,11,18]) or on the identity of the clients (see, e.g., [4,10,13,16], or consider different kinds of access control in such systems (see, e.g., [12,14]).

Our work is also somewhat related (in a complementary way) to a number of areas in theoretical and applied cryptography, including private information retrieval [6], searchable symmetric encryption [17], searchable public-key encryption [2] and oblivious RAMs [9]. Previous cryptographic work in a 3-party model (also referred as commodity-based, server-assisted, server-aided model) seems to have originated in [1], with respect to oblivious transfer protocols, and [7], with respect to private information retrieval.

2 Models, Definitions and Properties

We discuss models and DR protocol properties used in the rest of the paper, and further clarify the privacy and security properties that our DPC protocols must satisfy.

Data, Query and Policy Models. We model a *database table* (briefly, database) as a matrix DB with n rows and m columns, where each row is associated with a data *record*, each column is associated with a data *attribute*, and each database entry $\mathrm{DB}(i, j)$ is the value of the j-th attribute of the i-th record. The database *schema* consists of n, m, and the *domains* of each of the m attributes (*i.e.*, the j-th domain is the set of values that the j-th attribute of a record can take), and is assumed to be known by all parties that participate in the protocol. We assume that domains are large in that a randomly chosen domain element is, with very high probability, not in DB. (If DB does not satisfy these conditions, then simple padding of domain strings can be used to make it so.)

A *query* q contains a database attribute and a *query value* v from the corresponding attribute domain. We consider keyword-match queries of the following form (using SQL notation): "SELECT $*$ FROM *main* WHERE attribute_name $= v$".

A data owner's *query compliance policy* (briefly, *policy*) contains, for each attribute $j \in \{1, \ldots, m\}$, a set $W_j = \{w_{j,1}, \ldots, w_{j,c_j}\}$ of *policy values* drawn from the j-th domain. All of the clients that access DB through this data owner

are subject to the same policy. On input a query value v, an attribute name (or, equivalently, an attribute index j), and a set of attribute values W_j, the policy returns 1 (resp., 0) to denote query compliance (resp., non-compliance). We mainly consider the *whitelist* and *blacklist* policies:

1. *Whitelist*: If query q refers to the j-th attribute, then p returns 1 iff $v \in W_j$;
2. *Blacklist*: If query q refers to the j-th attribute, then p returns 1 iff $v \notin W_j$.

Intuitively, a blacklist policy captures the notion of a set of forbidden query values, while a whitelist policy restricts queries to a specified set of allowed values. We assume that the lengths c_j of whitelists and blacklists and the lengths of the policy values $w_{j,k}$ are system parameters known to all parties (although our protocols will keep the latter values hidden from C).

DR Protocol Properties. We consider DR protocols, as depicted in Fig. 1, with the following structure:

1. C, D and S run a preliminary setup subprotocol
   (this enables S to later answer C's query on the database owned by D)
2. Given a query q, C constructs a *query message* Q and sends it to S
3. S computes an *answer message* ans and sends it to C
4. Based on Q and ans, C can compute database records that satisfy q, if any.

The *unique-query* property requires that, for any database DB and any properly formatted query message Q, there is at most one pair (attribute_name, v) for which C could have generated query message Q. When such a pair exists, we refer to v as the "query value associated with Q."

The *query-correctness* property requires that, for any database DB, any input pair (attribute_name, v), and any Q with associated query value v, at the end of the DR protocol, C can compute all records in DB that satisfy query attribute_name $= v$.

We also impose some additional structural properties on DR protocols:

1. *Added DR Property 1:* At the end of step 1, S stores $F(k_{\mathrm{c,d}}; DB(i,j))$, for each database entry $DB(i,j)$, where F is a pseudo-random permutation and $k_{\mathrm{c,d}}$ denotes a key shared between C and D;
2. *Added DR Property 2:* At the end of step 1, for each database entry $DB(i,j)$, S stores the triple encryption $F(k_{\mathrm{c,d}}; F(k_{\mathrm{c,s}}; F(k_{\mathrm{c,d}}; DB(i,j))))$, where F is a pseudo-random permutation and $k_{\mathrm{c,d}}$ (resp., $k_{\mathrm{c,s}}$) denotes a key shared between C and D (resp., C and S).

Note the following simple DR protocol satisfying Property 1: query values and data values are encrypted via a pseudo-random permutation, a query message contains the encrypted query value, and the answer message contains the records with encrypted data values equal to the encrypted query value. Other examples can be found in the literature (see, e.g., [11,18]). It should also be noted that any protocol satisfying Property 1 can be turned into one that satisfies Property 2, and that our techniques will work with a number of variations of these example properties.

DPC Protocol Properties. Our DPC protocols compose with DR protocols as follows (see Fig. 2): after the DR setup subprotocol, instead of a single query message Q sent from C to S, we now have three subprotocols (a query subprotocol, a compliance-verification subprotocol, and a query rewriting subprotocol) after which a query message is sent to S, and then the answer step of the DR protocol can be executed.

The requirements we demand from any DPC protocol were already informally described and motivated in Sect. 1. Here, we only further clarify its input/output behavior and privacy requirement. The inputs to a DPC protocol are a security parameter 1^σ (known to all parties), an attribute name and query value v (private inputs to C), and a database DB (schema known to all parties, but contents private to D). The outputs of a DPC protocol are a query message Q' (communicated privately to S) and a bit b (communicated privately to D) indicating whether the query complies with the policy ($b = 1$) or not ($b = 0$). We consider privacy in multiple runs of the DPC protocol against a semi-honest probabilistic polynomial-time adversary *Adv* (with history as auxiliary input) corrupting up to one party, by a natural adaptation of the real/ideal security framework, as typically used in the cryptography literature. Briefly speaking, a (real-world) execution of multiple runs of the DPC protocol are executed, does not leak to *Adv* more than the ideal-world leakage, defined as follows. On input of a query value v given by C, a database DB, policy values $w_1, \ldots, w_c$, and policy p input by D, each ideal execution of a single DPC protocol returns:

1. the output b of policy p on input query value v and policy values $w_1, \ldots, w_c$ to D
2. a random query message Q' to S, where Q' has no matching records if $b = 0$ or has associated query value v if $b = 1$.

In our first two protocols, we admit some additional leakage to S, and consider the variant of the above definition, where such leakage is also admitted in the ideal world.

Our design also targets a number of additional security properties, which can be obtained using network security protocols such as TLS: *confidentiality* of the communication between all participants, message *sender authentication*, message *receiver authentication*, and *communication integrity* protection.

3 DPC Protocols

In this section we present our three DPC protocols (whose properties are detailed in Sect. 1). Our first protocol π_1 falls short of satisfying all requirements formulated in Sects. 1 and 2 in two ways: (a) it does not satisfy compliance soundness (i.e., a malicious C could send inconsistent encryptions for compliance verification and query rewriting; thus, the compliance verification test would pass on a query value different than the one used for query rewriting); (b) privacy against D is only satisfied if the protocol is allowed to leak any repeated occurrences of the same query value. Our second protocol π_2 extends π_1 so to eliminate (a), and

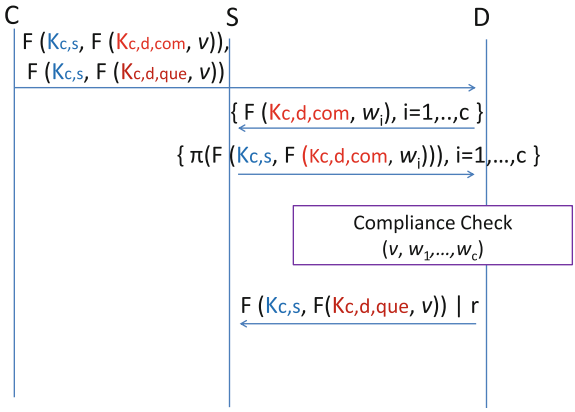


Fig. 3. The basic keyword match policy compliance protocol π_1

protocol π_3 eliminates both (a) and (b). In the following protocol descriptions, keys are named with two subscripts indicating by which parties they are shared. For example, a private key shared by C and S would be named $k_{c,s}$. There may optionally be a third subscript *com* or *que*, to indicate whether the key is used for policy compliance checking or query rewriting. Thus, $k_{c,d,com}$ means a key shared by C and D and used for compliance checking. We assume a standard secure 2-party key agreement protocol is executed in an initialization phase to produce these keys.

Protocol π_1. Our most basic protocol π_1 allows efficient enforcement of policy compliance for keyword search queries with whitelist policies (blacklist policies can be supported with minor modifications).

A pictorial description of the protocol can be found in Fig. 3. In the first step, C sends to D two double encryptions of its query value v, once using key $k_{c,d,que}$ as the inner layer, and a second time using key $k_{c,d,com}$. Then, D and S interact to analogously compute ciphertexts for the policy values, as follows: first, D encrypts each of the policy values $w_1, \ldots, w_c$ using key $k_{c,d,com}$ and sends the resulting ciphertexts to S; then, S further encrypts each of these ciphertexts using key $k_{c,s}$, and returns the resulting ciphertexts, *reordered using a random permutation* π, to D. At this point, D computes the whitelist policy output by simply checking whether one or zero of the policy value ciphertexts is equal to the ciphertext received by C. After the policy compliance calculation, if the query is compliant, D simply forwards the received encryption $k_{c,d,que}$ to S, who can remove the outer layer of encryption and fulfill the query. Otherwise, D performs *query rewriting*, sending S a random value indistinguishable from a double-encrypted query.

As described in the introduction, the two main technical ideas embedded in this protocol are: (1) using "equality-preserving encryption" to allow D to calculate the policy output without revealing the policy values to S or C and without learning why the policy was or was not satisfied (i.e., which policy

value(s) w_i may have textually matched value(s) in the query); (2) using "query rewriting" to allow D to rewrite the query q obtained by C into a query q' which guarantees that the same database records match q and q' if q is compliant, or no records match q' otherwise, without S or C obtaining any additional information on which is the case.

Protocol π_2: Soundness Against Malicious Clients. One problem with protocol π_1 is that a malicious C can violate the soundness property by sending two different queries for compliance verification and query rewriting. Protocol π_2 prevents this attack with minimal modifications from π_1. As a preliminary observation, we see that since C only sends one query message, the only opportunity for C to provide malicious input is before the compliance verification subprotocol. This naturally leads us to examine ways in which we could modify protocol π_1 to require only one input from C. Note that we cannot use the same encryption for both compliance checking and query rewriting, since that would allow S to identify encrypted query values that match policy items it has seen during the setup phase.

We can resolve this by storage of a triple encryption $F(k_{\mathrm{c,d}}, F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v)))$ of each database value, as in Added DR Property 2, instead of a single encryption $F(k_{\mathrm{c,d}}, v)$, as in π_1. The structure of the protocol is similar as for π_1. At query time, C encrypts the query value with *both* of its keys and sends the resulting doubly-encrypted value $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v))$ to D. Then D encrypts each of the policy values w_i using key $k_{\mathrm{c,d}}$ and sends them to S, which then re-encrypts each of these using key $k_{\mathrm{c,s}}$, randomly permutes the order of keywords, and returns the re-encrypted values to D.

As before, D checks the encrypted query for equality with the double-encrypted policy values. If the query is non-compliant, D sends to S a random query indistinguishable from a triple-encrypted real query; otherwise, D re-encrypts $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v))$ using $k_{\mathrm{c,d}}$, and sends the triple-encrypted value $F(k_{\mathrm{c,d}}, F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v)))$ to S for its answer generation in the DR protocol. Note that the outermost layer of encryption prevents S from identifying whether the query matches policy items it had previously encrypted from D—thus eliminating the need for separate *com* and *que* encryptions.

The resulting DPC protocol π_2 inherits the same properties as π_1, plus compliance soundness under a different assumption on the method used to encrypt the database values in the DR protocol (namely, Added Property 2). The triply-encrypted database of Added DR Property 2 can be generated during the setup phase as follows. First, D encrypts all items in the database with $k_{\mathrm{c,d}}$ and sends them to S, which re-encrypts them using $k_{\mathrm{c,s}}$ and returns them to D. Then D adds a third layer of encryption, again using $k_{\mathrm{c,d}}$, and sends the triply-encrypted database to S. This interaction between D and S may be expensive, as it involves every item in the database being encrypted and sent over the network three times; this may render this method undesirable to practitioners, especially when dealing with large databases. We address this issue as well in π_3.

Protocol π_3: Privacy Across Multiple Queries. Protocols π_1, π_2 come with some leakage to D across multiple query executions: D learns, by checking for

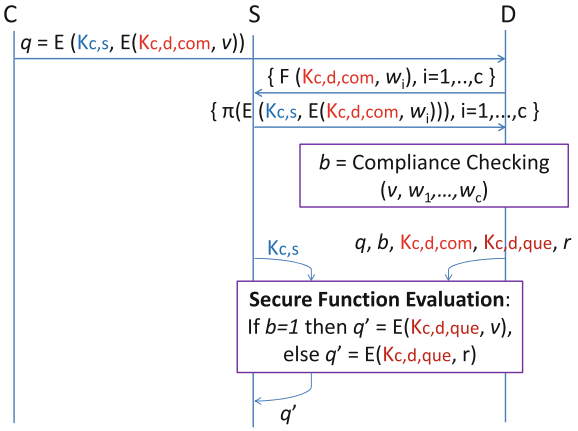


Fig. 4. Protocol π_3: Keyword search policy compliance with (multi-query) security against D

repetitions in the first message sent by C to D, whether the query value in the current execution is equal to a previously executed query. Although not a major form of leakage, it remains of interest to see if we can prevent it at some affordable efficiency cost.

We now describe a protocol π_3 that keeps all properties in π_1, the compliance soundness property achieved in π_2 and satisfies privacy against D without the mentioned leakage. (It also avoids the database setup inefficiency mentioned at the end of the description of π_2.) Protocol π_3 uses an additional cryptographic tool: 2-party Secure Function Evaluation (SFE) protocols [19]. Recall that in such protocols, two parties P_1 and P_2, with private inputs x_1 and x_2, respectively, can jointly compute a functionality $f(x_1, x_2) = (f_1, f_2)$, such that P_1 receives $f_1(x_1, x_2)$, and P_2 receives $f_2(x_1, x_2)$, and it is required that nothing is learned by either party other than the output.

Instead of using a triple encryption as in π_2, protocol π_3 uses a different shared key $k'_{\mathrm{c,d}}$ for query rewriting. After the policy check, which remains unchanged, D and S perform a two-party SFE protocol, returning to S a newly-encrypted form of the query.

First, C sends $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v))$ to D, at which point the same policy compliance check as in π_2 takes place. After the compliance check, D and S engage in a two-party SFE protocol, where D inputs $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v)), k_{\mathrm{c,d}}, k'_{\mathrm{c,d}}$, a random query r, and the (one-bit) result of the compliance check. S inputs $k_{\mathrm{c,s}}$. Together, the two parties securely compute the following output, which is received only by S: if the query was non-compliant, random value r is output; if the query was compliant, the doubly-encrypted value $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,d}}, v))$ is decrypted twice to produce v, which is then encrypted using key $k'_{\mathrm{c,d}}$, and the result, $F(k'_{\mathrm{c,d}}, v)$ is released to S. S then proceeds to compute the answer based on the DR protocol.

The resulting DPC protocol π_3, illustrated in Fig. 4, inherits the same properties as π_1 and π_2, plus multi-query privacy against D. However, π_3 is not strictly better than π_1, π_2 since two-party SFE protocols come with added running time, even when considering recent implementation advances (see, e.g., [15] and follow-up work). Accordingly, we only used two-party SFE executions on very short ℓ-bit inputs (as opposed to a generic SFE solution which would require inputs as large as the policy itself).

4 Performance Results

In this section we report initial performance results related to implementations of our basic DPC protocols. We focus on results for π_1 as π_2 and π_3 exhibit similar behaviour. Specifically, π_2 is only slower than π_1 by a small constant multiplicative factor and π_3 is only slower than π_2 by a small constant additive factor.

Setup. The Data Owner and Server processes were running on a Dell PowerEdge R710 server with two Intel Xeon 2.66 Ghz processors and 48 GB of memory, running 64-bit Ubuntu 12.04.1. The R710 server was connected to a Dell PowerVault MD1200 disk array containing 12 2TB 7.2 K RPM SAS drives arranged in a RAID6 configuration. The Client was running on a Dell PowerEdge R810 server with two Intel Xeon 2.40 GHz processors and 64 GB of memory, running 64-bit Red Hat Enterprise Linux Server release 6.3 and connected to the R710 server via switched Gigabit Ethernet.

The database was populated by generating random values about fictitious people using demographic information from the US Census Bureau. A single table with 23 columns was used (e.g., last name *lname*, state *state*, and zip code *zip*), including several columns containing large text fields and one column containing binary data (*fingerprint*). We considered the following policies:

Policy	Compliant queries must include:
F	All queries are rejected as non compliant
T	All queries are accepted as compliant
B1	A conjunction of at least 3 keyword queries on *state*, *lname*, and *zip*
B2	A conjunction of at least 3 keyword queries on *state*, *lname*, and any one of the remaining columns, excluding *fingerprint*
W1	A keyword query on *lname* with query value in a 1-entry whitelist
W2	A keyword query on *lname* with query value in a 100-entry whitelist
W3	A keyword query on *lname* with query value in a 1000-entry whitelist
W4	A keyword query on *lname* with query value in a 10000-entry whitelist
W5	A keyword query on *lname* with query value in a 20000-entry whitelist

Compliance policy $B2$ was expressed as a disjunction of 23 sub-policies of $B1$ type, each of them requiring keyword query conjunctions on *state*, *lname*, and an additional (and different) database column. We considered the following queries:

Query	Template
Q1	SELECT * FROM main WHERE lname=value
Q2	SELECT * FROM main WHERE state=value AND lname=value AND zip=value

Results. Each query template was executed several times using different values. We note that policies F, T, $B1$ and $B2$ only refer to the query structure or database attributes and do not depend on query values, contrarily to queries $W1, \ldots, W5$, which depend on values in the query and in the (variable-length) whitelist.

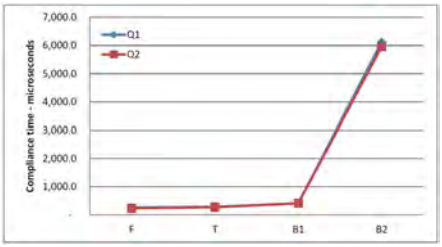


Fig. 5. Query compliance checking overhead for policies F, T, $B1$, and $B2$.

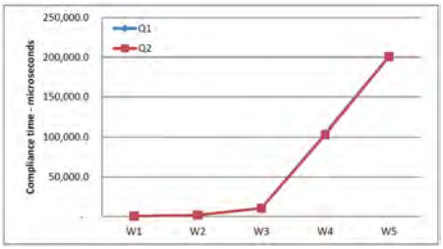


Fig. 6. Query compliance checking overhead for policies $W1$, $W2$, $W3$, $W4$, and $W5$.

Figure 5 shows results when checking compliance for policies F, T, $B1$, and $B2$ for $Q1$ and $Q2$ queries. Such checking was based on the query structure only and, thus, there is no impact from cryptographic operations on the measured running time. Three main observations can be derived from this figure: (1) running time for $B1$ is almost the same as for the trivial policies T and F; (2) running time for $B1$ and $B2$ is almost the same for the two policy types $Q1$ and $Q2$, with differences smaller than 3 %; and (3) running time for $B1$ and $B2$ is essentially linear with policy size.

Figure 6 shows computation results when running protocol π_1 for query classes $Q1$ and $Q2$ and policies $W1, \ldots, W5$. These policies depend on query values and, hence, trigger execution of π_1, with its cryptographic operations. Two main observations can be derived from this figure. First, running time is almost the same again for the two policy types $Q1$ and $Q2$, with differences of less than 7 % for shorter policies and less than 2 % as policies get longer. Second, the time required by π_1 grows linearly with the size of the whitelist. Specifically,

as the size of the whitelist grows, so does the time it takes to doubly encrypt its values, send/receive them between D and S, and checking by using sequential scan whether an attribute value referenced in C's query belongs to the doubly encrypted and permuted whitelist values. (Here, a speedup from the use of binary search does not seem to impact the running time substantially, due to the double encryption and network communication required).

When comparing the two figures, we observe that the impact of running π_1 when checking compliance is essentially minimal for policies with short-size whitelists (i.e., a factor of about 10, calculated by comparing the running time for $F, T, B1$ with the running time of policies $W1, W2, W3$).

Acknowledgement. Supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D13PC00003. The second, third, fifth and sixth authors also acknowledge DARPA contract FA8750-13-2-0058 for some of the time spent on revising this paper. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DARPA, DoI/NBC, or the U.S. Government.

References

1. Beaver, D.: Commodity-based cryptography (extended abstract), pp. 446–455. In: STOC (1997)
2. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
3. Brickell, E., Di Crescenzo, G., Frankel, Y.: Sharing block ciphers. In: Clark, A., Boyd, C., Dawson, E.P. (eds.) ACISP 2000. LNCS, vol. 1841, pp. 457–470. Springer, Heidelberg (2000)
4. Camenisch, J., Kohlweiss, M., Rial, A., Sheedy, C.: Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 196–214. Springer, Heidelberg (2009)
5. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Paraboschi, S.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC **8**, 119–152 (2005)
6. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
7. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Universal service-providers for database private information retrieval, pp. 91–100. In: PODC (1998)
8. Evdokimov, S., Günther, O.: Encryption techniques for secure database outsourcing. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 327–342. Springer, Heidelberg (2007)
9. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)

10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS Conference, pp. 89–98 (2006)
11. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model, pp. 216–227. In: SIGMOD Conference (2002)
12. Hamlen, K.W., Kagal, L., Kantarcioglu, M.: Policy enforcement framework for cloud data management. IEEE Data Eng. Bull. **35**(4), 39–45 (2012)
13. Jarecki, S., Lincoln, P.: Negotiated privacy. In: Okada, M., Babu, C.S., Scedrov, A., Tokuda, H. (eds.) ISSS 2002. LNCS, vol. 2609, pp. 96–111. Springer, Heidelberg (2003)
14. Li, M., Yu, S., Cao, N., Lou, W.: Authorized private keyword search over encrypted data in cloud computing, pp. 383–392. In: ICDCS (2011)
15. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system, pp. 287–302. In: USENIX Security Symposium (2004)
16. Miklau, G., Suciu, D.: Controlling access to published data using cryptography, pp. 898–909. In: VLDB (2003)
17. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data, pp. 44–55. In: IEEE Symposium on Security and Privacy (2000)
18. Yang, Z., Zhong, S., Wright, R.N.: Privacy-preserving queries on encrypted data. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 479–495. Springer, Heidelberg (2006)
19. Yao, A.C.C.: How to generate and exchange secrets (extended abstract), pp. 162–167. In: FOCS (1986)

Catching Bandits and *Only* Bandits: Privacy-Preserving Intersection Warrants for Lawful Surveillance

Aaron Segal, Bryan Ford, and Joan Feigenbaum
Yale University

Abstract

Motivated in part by the Snowden revelations, we address the question of whether intelligence and law-enforcement agencies can gather actionable, relevant information about unknown electronic targets without conducting dragnet surveillance. We formulate principles that we believe effective, lawful surveillance protocols should adhere to in an era of big data and global communication networks. We then focus on intersection of cell-tower dumps, a specific surveillance operation that the FBI has used effectively. As a case study, we present a system that computes such intersections in a *privacy-preserving, accountable* fashion. Preliminary experiments indicate that such a system could be efficient and usable, suggesting that privacy and accountability need not be barriers to effective intelligence gathering.

1 Introduction

Much of the Snowden-triggered debate has revolved around the "balance" between national security and personal privacy. Both sides of these "balance" arguments presume that security and privacy represent a zero-sum tradeoff, a presumption that we believe is false – not just on policy grounds [16] but also for technical reasons. With the right *existing* technology deployed under the right policy framework, we can have both strong national security and strong privacy protections [8].

We observe and accept that certain demonstrably effective electronic-surveillance processes require "bulk" access to privacy-sensitive metadata. For example, by obtaining *cell-tower dumps* from related bank robbery sites – sets of about 150,000 total users whose cell phones had been in the area at particular times – the FBI *intersected* these sets to discover a phone used by the High Country Bandits [2]. Although the FBI's dragnet proved effective in catching these particular criminals, their incidental ingestion of many innocent users' phone numbers raises important privacy concerns.

Consistent with both US Constitutional and human-rights principles that allow government "search and seizure" in private spaces only via warrant processes grounded in *public* law, we propose that any electronic-surveillance activity searching or otherwise touching private user data or metadata must likewise be implemented via *open, public* processes that protect the privacy of innocent, untargeted users. These open processes can and should, for example, enable agencies like the FBI to catch criminals such as the High Country Bandits without ingesting 149,999 unrelated users' phone numbers into internal databases for potential use in arbitrary, secret surveillance activities now or in the future.

This paper takes preliminary steps toward enunciating basic principles for *open, privacy-preserving, accountable surveillance processes* and explains why this phrase is not an oxymoron. As a concrete case study, we present a prototype metadata-query system based on mature and practical *privacy-preserving set-intersection* methods [9, 13, 18]. This design supports warrant-based surveillance targeting not just known but *unknown* users, as in the FBI's targeting of the High Country Bandits and the NSA's targeting of the CO-TRAVELERs of terrorism suspects [17]. Our preliminary experimental results suggest that intersection of cell-tower dumps can indeed be implemented in a manner consistent with the principles of privacy-preserving, accountable surveillance.

Before proceeding, we wish to address the question of why "privacy-preserving, accountable surveillance" is an appropriate topic for a workshop on "free and open communications on the Internet." While it may be interesting and appealing to contemplate an Internet in which there is little or no surveillance, it would not be an effective way to increase the degree to which "Internet freedom" is a lived experience for ordinary people. Law-enforcement and intelligence agencies have been and currently are active in *every* national- or global-scale mass-communication system, and the Internet will be no exception. The Snowden revelations may have provided an opportunity to design protocols that allow government agencies to collect and use data that are demonstrably relevant to their missions *while respecting the privacy of ordinary citizens and being democratically accountable*. The FOCI community should seize that opportunity.

In Section 2, we present principles that could guide the development of privacy-preserving, accountable surveil-

lance protocols; we also explain why the intersection of cell-tower dumps is a natural domain in which to apply these principles. In Section 3, we flesh out our operational model of privacy-preserving, accountable set intersection and present the specific protocol that we used for our preliminary experiments. Section 4 contains the results of those experiments. Section 5 outlines related work, and Section 6 concludes with a (non-exhaustive) list of directions for further research in this area.

2 Privacy Principles for Surveillance

This section outlines several principles that we believe should govern electronic surveillance. We start with a basic principle stating that processes that use private data in bulk must be *open*, and we then outline several related properties that we expect such open processes to have. Finally, we summarize how these principles might be applied in the case of "set-intersection warrants."

2.1 Open Processes for Law Enforcement

A basic tenet of democratic society is that law enforcement must follow *open processes*: procedures laid out in public law and subject to debate and revision through deliberation. Police need not disclose *whom* they may suspect of a particular crime or other details of an ongoing investigation, but their investigation must nevertheless follow rules and procedures established in *open* law books that everyone has a right to know and understand. And it is accepted that searching a person's home or personal records requires a narrowly targeted and properly authorized warrant based on probable cause.

We wish to formulate an openness principle for electronic surveillance that distinguishes between two classes of Internet users. A *targeted user* is one who is under suspicion and is the subject of a properly authorized warrant. All others are *untargeted users* – the vast majority of Internet users (and cell-phone users and users of any general-purpose, mass-communication system).

Just as search-warrant processes in free societies are grounded in open law, we believe that any "bulk" electronic-surveillance process that ingests, searches, or otherwise touches private[1] data of untargeted users must likewise be an open process. We refer to processes that are not open, public, and unclassified as *secret processes*, and we seek to limit their use (while admitting that there are circumstances in which they may be needed). Once law enforcement has legitimately employed an open process to identify, target, and obtain information about an Internet user suspected of a crime, however, it may potentially subject that targeted user's data to the full range of secret analysis tools and techniques in its arsenal.

One of the key reasons the NSA's mass-surveillance activities disclosed by Snowden are so troubling is that they tap into "bulk" data and metadata about untargeted users and ingest these private bulk data into secret processes that are codified only in secret FISA law and are subject only to secret oversight and accountability procedures (Figure 1a). In short, the public must simply "trust" the US government's evidence-free assertions that its mass ingestion and secret processing of privacy-sensitive data are (secretly) lawful and subject to adequate (secret) privacy protections and effective (secret) oversight. We cannot remotely envision the framers of the US Constitution being comfortable with such blind faith in secret mass-surveillance processes of this nature.

We therefore propose that a basic openness principle, comprising two main planks, should govern electronic-surveillance processes in a modern democracy:

I Any surveillance or law-enforcement process that obtains or uses private information about untargeted users shall be an open, public, unclassified process.

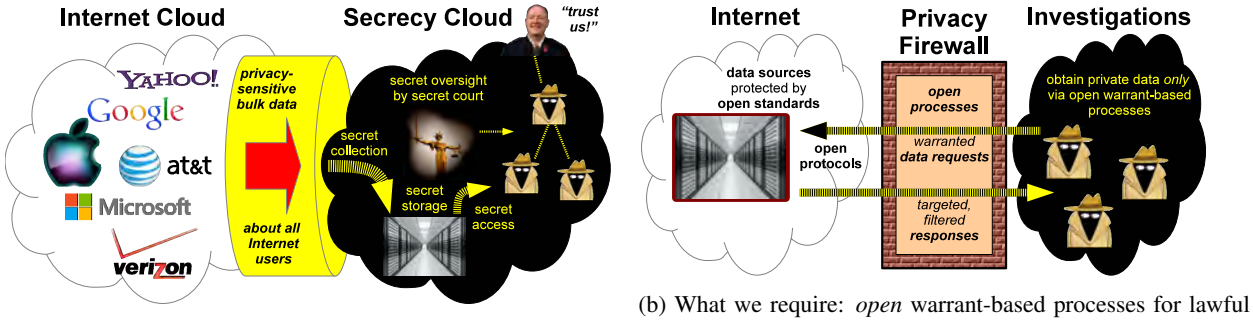
II Any secret surveillance or law-enforcement processes shall use only:
   (a) public information, and
   (b) private information about targeted users obtained under authorized warrants via open surveillance processes.

We view this openness principle as demanding that an open *privacy firewall* be placed in the path of private information flowing from the Internet to law enforcement (Figure 1b). Processes that search or ingest private data of untargeted users "through the firewall" must be open processes, but, once a user is targeted by a legitimate warrant and his data have been acquired via open processes, these targeted user data may potentially be subject to secret investigative processes.

Openness conceived in this manner may sound incompatible with the requirement that government agencies be able to keep secret the targets and details of active investigations, but it is not. Using appropriate security technology, a data-collection or surveillance *process* used in an investigation may be made fully public without revealing the *content* of any particular investigation.

Our focus here is on general electronic surveillance principles for law enforcement purposes, independent of any particular government or agency. The hot-button case of the NSA is complicated by the fact that the NSA was founded as a foreign-intelligence agency but has acquired *de facto* characteristics of law-enforcement agencies by: (a) increasingly serving to support and feed surveillance data to law-enforcement agencies such as the FBI and the DEA; (b) collecting and storing both

[1] Rigorous definitions of the term "private" are the subject of extensive study in computer security, law, philosophy, and many other fields; as such, they are beyond the scope of this paper.



(a) What we have: a cloud of secret mass surveillance processes

(b) What we require: *open* warrant-based processes for lawful electronic surveillance, creating a "privacy firewall."

Figure 1: Secret versus open electronic surveillance processes

US and non-US surveillance data alike, even if internal "searches" are allowed only on "non-US persons"; (c) being increasingly employed not just against wartime adversaries but against citizens of peaceful, allied, democratic states, who common sense dictates should have protection against "unreasonable search and seizure" regardless of the letter of US law [11]. To whatever extent the NSA or any government agency behaves like a domestic or international law-enforcement agency, we believe the above openness principle should apply.

2.2 Mass Surveillance

How should this openness principle be applied to *mass-surveillance* processes, *i.e.*, processes such as the cell-phone records-collection program that have the potential to collect or use *all* data in a particular category about *all* users? (As currently implemented, the cell-phone records-collection program realizes this potential [6, 10], but we do not think it should.) We refer to data sets collected and used in mass surveillance as *bulk data sets*.

We identify four particular "sub-principles" that we believe should apply to mass-surveillance processes:

Division of trust: No single agency or branch of government should have either the authority *or the technical means* to compromise the privacy of bulk data about untargeted users. Mass-surveillance processes must require the sign-off, oversight, and active participation of multiple independent authorities representing each branch of government.

Enforced scope limiting: Surveillance processes must incorporate scope-limiting mechanisms ensuring that no particular warranted-surveillance activity captures data from an overly broad group of users. For example, each warrant might have a specified limit on the number of users whose data may be touched by the warrant-authorized process.

Sealing time and notification: Surveillance processes that capture privacy-sensitive user data must impose a limit on the length of time that the users in question may be kept ignorant of the fact that their data were captured. After this time has expired, the process must ensure that the users are notified of the data access and given means to investigate the justification and/or obtain recompense for any unjust effects of the investigation. Higher levels of authority should be required to authorize longer sealing times. No level of authority should permit indefinite sealing times (even indirectly, on an "installment" basis).

Accountability: Surveillance processes must incorporate accounting mechanisms that enable all three branches of government, as well as civilian participants, to maintain and safely disclose relevant statistics on how frequently and extensively warranted-access mechanisms are used, *e.g.*, number of warrants per month of a given type, maximum number of individuals affected under any warrant, total number of individuals affected by all warrants in one month, or maximum secrecy period applied to any outstanding warrant in one month.

2.3 Case Study: Intersection Warrants Using Cell-Tower Dumps

Given a properly authorized warrant, we wish to enable law-enforcement agencies to target not just *known* users (those whose cell-phone numbers they already have and are covered by the warrant) but also *unknown* users (in our case, those whose cell-phone numbers they do not have but may be able to discover by *intersecting* several relevant cell-tower dumps). It may appear nonsensical to describe a user as both "unknown" and lawfully "targeted," but it is not. We may view such an *intersection warrant* as a type of "John Doe" warrant [3]: one in which the names or phone numbers of the person(s) of interest are unknown, but for which relevant times and locations are known, and for which there is sufficient evidence to convince a judge that there is probable cause to believe the given times and locations uniquely identify the unknown person(s) who committed a crime.

For example, the FBI caught the High Country Ban-

dits [2] by intersecting three cell-tower dumps, representing the sets of cell-phone numbers that had been used near three different bank-robbery sites at the times of the robberies. In total, these dumps contained 150,000 cell-phone numbers, but their intersection contained only one: that of a High Country Bandit. Similarly, the NSA's CO-TRAVELER program [17] searches for unknown associates of known surveillance targets by first intersecting cell-tower dumps from times and locations at which a particular known target appeared and then interpreting the intersection as the set of cell-phone numbers of people who may be "traveling with" the known target.

In Sections 3 and 4, we present and evaluate a protocol that computes the intersection of cell-tower dumps and obeys the principles articulated above. This is a natural test case for us for at least two reasons. First, intersections of cell-tower dumps have proven useful in catching criminals; this distinguishes them from many of the other surveillance activities featured in the Snowden revelations, the practical utility of which is at best unclear. Second, privacy-preserving set intersection is a well-studied, mature, and practical technology [9,13,18].

Note that our protocol is not specific to cell-tower dumps and could also be used to query other "time-and-place" metadata collections in an open, lawful manner.

3 Lawful Intersection Attacks

This section first outlines the assumptions and principals involved in our lawful intersection-warrant protocol, then describes the operation of the protocol, and finally summarizes its key security properties.

3.1 Principals

Our model for lawful intersection attack involves the following three types of principals. For simplicity, we assume here that these principals will participate in an intersection-attack mechanism in an honest-but-curious way. That is, they will not attempt to violate the rules of the mechanism, but they may use their own views of all data they see to acquire additional information.[2]

Sources: entities that produce metadata records embodying information of the form, "user X was observed to be near location Y at time Z." The obvious examples are phone companies whose cell towers produce logs of the users who appeared in the vicinity of a given cell tower at a given time, but our model extends to other producers of metadata of this general form.

[2] This assumption could be relaxed significantly by requiring all principals to produce zero-knowledge correctness proofs of their intermediate results, using standard and well-known techniques. We leave these details to future work, however, and we would still need to assume the correctness of the original inputs – *e.g.*, logged phone numbers.

Repository: any entity tasked with storing metadata for surveillance or law-enforcement purposes. This may be the phone companies that produced the records (*i.e.*, the same as the metadata sources), a government agency, or some specialized independent agency. While "who stores the data" is an important question in general, it is orthogonal to our goals, and our model is agnostic with respect to its answer.

Agencies: a set of *multiple* independent but cooperating government agencies across whom our model divides surveillance authority. While our model is formally agnostic with respect to the number or specific natures of the authorities across whom trust is divided, we will use the US's 3-branch constitutional model as a concrete example, in which it might be appropriate to divide surveillance authority across three agencies:

- The **Executive Agency** represents the executive branch and is responsible for *requesting* surveillance warrants – *e.g.*, an agency like the NSA or FBI.
- The **Judicial Agency** represents the judicial branch and is responsible for *authorizing* requested warrants, after verifying independently that they are legally justified and suitably scoped.
- The **Legislative Agency** reports to the legislative branch and is responsible for ensuring that accurate and sufficiently detailed data are gathered and regularly reported to Congress on how and to what extent these surveillance capabilities are employed.

3.2 Lawful Set-Intersection Protocol

The lawful set intersection protocol we present is similar in structure to the protocol of Vaidya and Clifton [18].

Our protocol is built on two commutative encryption schemes: ElGamal and Pohlig-Hellman. A commutative encryption scheme has the property that a message encrypted sequentially under multiple encryption keys can be decrypted by applying the corresponding decryption keys *in any order*. The ElGamal and Pohlig-Hellman encryption schemes are not only commutative but mutually commutative - that is, a message encrypted under a combination of encryption keys from the two cryptosystems can still be decrypted by the corresponding decryption keys, again regardless of order.

We use the randomized, public-key ElGamal encryption scheme for long-term encryption of stored data. Each agency, or "participant," in the protocol needs an ElGamal key pair, the public key for which is known to the sources of private information. The Pohlig-Hellman encryption scheme is symmetric-key and deterministic, and the participants in the protocol use it to blind the data prior to intersection. Because Pohlig-Hellman is deterministic and commutative, there is a one-to-one correspondence between data items and their encryptions un-

der any fixed set of Pohlig-Hellman keys, regardless of the order in which those keys are applied. Short-term Pohlig-Hellman keys are generated by each participant during the protocol execution and discarded at the execution's end.

Each participant's input is its ElGamal private key and a set of data that has been encrypted under the ElGamal public keys of all agencies. The agencies do not generate these sets – rather, they are distributed to the agencies by the repositories. If there are k agencies, they are given numbers 1 through k so that, when the j^{th} agency is done acting on a set of data, it can pass the set on to the $(j+1)^{st}$, and it can receive new sets from the $(j-1)^{st}$.

Assuming all agencies execute the protocol with honest-but-curious behavior, the protocol's output for each participant will be the intersection of all sets. Optionally, each agency may also supply a threshold limiting the size of the intersection it is willing to reveal. If the size of the intersection of all sets would be above any agency's threshold, no agency will learn anything except the cardinality of the intersection, which agency's threshold was violated, and some intermediate values discussed in Section 3.3. The protocol runs as follows:

Initialize. Each agency first generates a temporary Pohlig-Hellman key to be used only during this execution of the protocol and then discarded. The first agency then obtains the ElGamal-encrypted sets to be intersected from the Repository.

Phase 1. Each agency in turn uses its ElGamal private key to remove a layer of ElGamal encryption from each item in each set to be intersected; it then adds a layer of Pohlig-Hellman encryption to each item, using the temporary key it generated in the Initialize step. The agency then randomly shuffles each encrypted set independently, while keeping the sets separate, and forwards the sets to the next agency. The phase is complete when agency k decrypts the final layer of ElGamal encryption from all items in all sets, leaving all sets encrypted under every agency's Pohlig-Hellman keys only.

Phase 2. Agency k broadcasts the resulting Pohlig-Hellman-encrypted data sets to all other participants. Each participant then computes the desired intersection: *i.e.*, the encrypted elements that appear in all sets – or, more generally, the elements that appear in some threshold number of the sets as defined by the intersection warrant. Because Pohlig-Hellman is a commutative and deterministic encryption scheme, two identical data items will have identical encryptions at this stage, making computation of the intersection trivial despite the encryption.

Phase 3. If any agency sees that the number of distinct items (*e.g.*, phone numbers) appearing in the resulting set intersection is above a warrant-specified limit on the number of individuals the warrant is permitted to target, the agency deletes its Pohlig-Hellman key, sends a message to all other agencies, and refuses to continue with the protocol. (The agency requesting the warrant might then be required to produce a new, more narrowly targeted warrant and try again.)

Phase 4. If the intersection's cardinality meets the requirements of the warrant, then the agencies collectively decrypt the items in the intersection. As in Phase 1, each agency in turn uses its Pohlig-Hellman key to decrypt each element of the intersection set, shuffles the intersection set, then forwards it to the next agency. The phase completes when when agency k decrypts the last layer of Pohlig-Hellman encryption and forwards the plaintext result to the other agencies.

For simplicity, we describe Phases 1 and 4 above as strict "cascades," each agency processing the full data set before passing it to the next. A simple performance optimization our prototype implements is for different input sets to start at different agencies – i.e., to start and end at different "points" around a circle – thus spreading computational load and increasing parallelism. This is only one of many potential optimizations, however.

3.3 Protocol Properties

We now analyze our cell-tower-dump intersection mechanism with respect to our openness principle for mass-surveillance processes. We accomplish division of trust by having all data be encrypted in advance with the public keys of the agencies that request, authorize, or oversee the surveillance. Without participation of all of these agencies, the data cannot be decrypted – even if the Repository is compromised, for example – and no unauthorized surveillance can be performed unilaterally.

The protocol also provides the means to enforce a limited scope of investigation. The sizes of all sets and intersections are visible to all participants in Phase 3; so the warrant can specify a limit on the number of users whose data may be revealed. Any participant can stop the protocol, *before* any metadata records are revealed in cleartext, if the size of the intersection is above this limit.

If the protocol completes, it gives the same output to each participant. This makes it easy to notify users whose data were viewed after some sealing time and to maintain statistics for the purpose of accountability. These processes are beyond the scope of the intersection protocol, but one of the participants in the protocol can be responsible for maintaining them.

Finally, this process protects the privacy of untargeted users. The only information leaked apart from the output are the *sizes* of intersections of any two or more sets involved in the protocol. (This property is proven in [18] for a protocol using the same structure as ours; the proof generalizes straightforwardly to our case.)

This small information leakage reveals how many

users appear in multiple sets but does not reveal any specific user identities or metadata other than those in the requested intersection. Because only aggregate properties of the sets are leaked, we feel this leak should not represent a major privacy issue – and when a query fails because of an empty or too-large intersection, the leaked statistics may help the agency that requested the warrant formulate a revised request for a better scoped warrant.

4 Implementation and Evaluation

This section presents the results of our preliminary experiments with the lawful set-intersection protocol of Section 3. Recall that each network node that participates in this protocol acts on behalf of an "agency," in the terminology of Section 3.1. It is this set of agencies across whom trust, *e.g.*, the power to authorize an intersection attack, is divided in our model. Because the public keys of *all* agencies are used to encrypt the data stored in the Repository, each agency effectively uses its private key to "authorize" the selection and decryption of results responsive to a particular intersection warrant.

4.1 Prototype Implementation

Our implementation of the lawful set-intersection protocol is written in Java and available on GitHub.[3] The prototype does not use any external libraries for cryptography beyond Java's standard BigInteger class.

The program is run on multiple servers. The servers connect to each other over TCP sockets, and, for simplicity, we use a directed cycle as the connection graph. Each server sends data only to the next server in the cycle and receives data only from the previous server in the cycle. Each participant takes one set of encrypted data and a 1024-bit ElGamal private key as input. The data must have been sequentially encrypted under all participants' public keys before it can be used in the protocol; because this encryption is done offline and in advance, the protocol itself does not require access to public keys.

To test the protocol, we ran it many times on data sets of various sizes. We used PlanetLab [5] on a network of three computers located across the United States in order to take into account the potential effects of latency on end-to-end running time. The computers we used are located at Yale in Connecticut, University of Texas in Dallas, and University of California in Riverside.

The tests were all run using only these three nodes, each node with one data set. We expect the running time would increase considerably as a function of the number of participants, but three is a natural number of nodes in this context, representing a distribution of authority across three branches of government (Section 3.1).

[3] https://github.com/DeDiS/Surveillance

Items	Data sent per node (KB)	CPU time per node (s)	End-to-End runtime (s)
10	21	0.6	4.1
25	46	1.3	6.0
50	86	2.6	9.6
75	127	3.8	12.6
100	167	5.0	15.5
250	410	12.4	38.2
500	815	24.7	69.1
750	1220	36.9	103.0
1000	1625	49.3	137.2
2500	4055	123.0	369.9
5000	8106	245.6	724.9
7500	12156	369.4	1034.9
10000	16206	493.8	1402.3
50000	81009	2560.5	6971.2

Table 1: Experimental Results

4.2 Query Efficiency

Prior to execution, we randomly generated data sets for each trial for each node. We ran the protocol 10 times each with different-sized data sets, ranging from 10 items per set to 50,000 items per set. We measured three variables: the bandwidth, or amount of data each node transmitted during the protocol; the CPU time each node used in performing calculations; and the total end-to-end time, from the start of the protocol's execution to the production of output. After running each test 10 times, we averaged the results; these averages are presented in Table 1.

In the High Country Bandits case, the FBI processed information from about 150,000 users total [2]. Our largest test, with 50,000 data items per set, tested our protocol's efficiency with an equally large amount of data. The average amount of time needed to run the protocol in this experiment was 6971.2 seconds, just under two hours. Considering the amount of time it would take a law-enforcement agency to set up its own set-intersection program, two hours seems quite reasonable. Further, because all the key computations in this protocol are "embarrassingly parallel," delay could probably be reduced by orders of magnitude with a moderate and readily feasible investment in processing power at each agency.

These tests were run with an intersection size of three. We also tested these benchmarks with an intersection size of 10 and found that the average times did not change by more than one second in any case and that the data sent per node always increased by 3 KB.

Our results indicate that the amount of data sent over the network, CPU time, and end-to-end time all increase linearly with the size of the data sets, which is what we

would expect from this protocol.

Further tests showed that total data sent and total CPU usage across participants were not affected if the data were concentrated in one or two sets, as opposed to being spread equally over all three sets. However, we found that the end-to-end delay can increase by up to a factor of two if the data are spread out. This result is unsurprising, because unbalanced sets render less effective the optimization mentioned at the end of Section 3.2, wasting time while the small-set-input participants idle, waiting for data to be sent by large-set-input participants.

5 Related Work

Private set intersection [9, 13, 18] is but a sliver of a large body of work on privacy-preserving algorithms [1]. We are not the first to propose employing such algorithms for targeted lawful surveillance. Kamara recently explored the use of Private Information Retrieval (PIR) for metadata queries [12]. Kroll, Felten, and Boneh explore mechanisms to distribute trust and improve privacy and accountability in queries [14]. These methods focus on queries for *known* targets, whereas we wish to demonstrate that proper use of cryptography can support powerful privacy-preserving surveillance of *unknown* targets. Non-cryptographic techniques have also been explored to protect privacy in video surveillance [7].

6 Conclusions and Open Problems

From the experimental results in Section 4, we conclude that privacy-preserving, accountable set intersection may indeed be achievable at scale. This in turn leads us to be optimistic about the feasibility of the broader goal articulated in Section 1: maintaining constitutional rights in powerful, evolving, digital-communication systems while simultaneously equipping law-enforcement and intelligence agencies to use these systems to combat and prevent crime and terrorism. There is a great deal of further work to be done along these lines, and we briefly describe a portion of it here.

The principles given in Section 2 are a first stab at an appropriate foundation for privacy-preserving, accountable surveillance. We hope that they will stimulate discussion and be refined and revised by the relevant research communities.

6.1 Enhancements and Generalizations

The protocol that we have implemented leaks the sizes of pairwise intersections (but not the contents of those intersections) in the case of three participants; more generally, it leaks the sizes of the j-wise intersections, where $1 < j < k$, and k is the number of participants. As explained in Sections 3 and 4, this is not a show stopper on privacy or efficiency grounds, but it leaves open the question of whether there is a similarly efficient protocol with the same accountability properties that reveals no information except the k-wise intersection.

In principle, one could achieve this ideal level of privacy by starting with a general secure, multiparty computation (SMPC) protocol and augmenting it with the appropriate accountability features. How well such an approach would scale is an open question.

Starting with the Fairplay platform for secure, two-party computation [15], there has been much work on general-purpose SMPC platforms. The goal of this research is to provide languages, compilers, run-time environments, and other platform elements that enable programmers who are not experts in cryptography or SMPC to write ordinary code and transform it into executable, multiparty protocols with the desired security properties. There are now many such platforms whose performance and usability are improving (see, *e.g.,* [19]). One could, in principle, achieve the goals put forth in Section 3 simply by writing a set-intersection program and using, say, Sharemind [4] to translate it into a privacy-preserving, distributed set-intersection protocol (rather than implementing privacy-preserving set intersection "from scratch" as in Section 4). Open questions include whether the resulting protocol would be efficient enough to use at scale and how to make it accountable.

Of course, intersection of cell-tower dumps is but one of many computations that could be of use to law-enforcement and intelligence agencies. It would be interesting to identify other such computations and to apply to them the principles and computational approaches that we have explored in this paper.

6.2 Openness in Lawful Surveillance

One remaining high-level issue is the tension between the openness principle proposed in Section 2.1 – requiring that processes handling "bulk" electronic surveillance data be open – and the traditional desire of intelligence agencies to protect "sources *and methods*," especially from the knowledge of criminals or terrorists being investigated. We emphasize that satisfying our openness principle by no means demands exposing *all* intelligence methods – only those few involved in implementing the "privacy firewall" in Figure 1b. All the details of any particular investigation – who is being investigated, when, the details of a particular warrant such as which metadata sets are to be intersected, how those sets were chosen, and how the decrypted results are processed *after* being lawfully queried through the "privacy firewall" – could still rely on closely guarded intelligence methods.

A generic open process such as the set-intersection primitive tends to be usable in many different, specific ways – as in the contrasting High Country Bandits [2] and NSA CO-TRAVELER [17] examples. Had CO-TRAVELER not been disclosed by Snowden, for example, then this specific method of using set intersection to find unknown associates of known targets as they travel might well remain a closely guarded secret, even if the basic intersection-warrant mechanism were well-known, openly debated, and instituted in public policy.

Finally, a basic tenet of democratic society and the rule of law is that it is better to risk a few criminals' going uncaught, because they know and understand public law-enforcement processes "too well," than to risk that secret law-enforcement processes, however well intentioned at the outset, might become unaccountable and evolve into "star-chamber" tools of political repression and authoritarianism. This democratic principle of openness must be carried into the electronic world; with the right tools, the principle need not tie the hands of legitimate, accountable law-enforcement processes.

7 Acknowledgements

We thank the FOCI reviewers for their valuable and insightful feedback. Our prototype intersection-warrant implementation is derived from private set-intersection code by Ennan Zhai. This work was supported in part by the National Science Foundation under grant 1016875, the Office of Naval Research under grant N00014-12-1-0478, the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory under contract FA8750-13-2-0058, and DARPA and SPAWAR Systems Center Pacific under contract N66001-11-C-4018. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the office policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

[1] Charu C. Aggarwal and Philip S. Yu. A general survey of privacy-preserving data mining models and algorithms. In *Privacy-Preserving Data Mining: Models and Algorithms*, chapter 2, pages 11–52. Springer, 2008.

[2] Nate Anderson. How "cell tower dumps" caught the High Country Bandits—and why it matters. *arstechnica*, August 29, 2013.

[3] Meredith A. Bieber. Meeting the statute or beating it: Using John Doe indictments based on DNA to meet the statute of limitations. *University of Pennsylvania Law Review*, 150(3):1079–1098, 2002.

[4] Dan Bogdanov and Aivo Kalu. Pushing back the rain – how to create trustworthy services in the cloud. *ISACA Journal*, 3:49–51, 2013.

[5] Brent Chun et al. PlanetLab: An overlay testbed for broad-coverage services. In *ACM Computer Communications Review*, July 2003.

[6] Ryan Devereaux, Glenn Greenwald, and Laura Poitras. Data Pirates of the Caribbean: The NSA Is Recording Every Cell Phone Call in the Bahamas. *The Intercept*, May 20, 2014.

[7] Frédéric Dufaux and Touradj Ebrahimi. Scrambling for privacy protection in video surveillance systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1168–1174, 2008.

[8] Joan Feigenbaum and Bryan Ford. Is Data Hoarding Necessary for Lawful Surveillance? *The Huffington Post*, April 19, 2014.

[9] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT (EUROCRYPT)*. Springer, May 2004.

[10] Glenn Greenwald. NSA collecting phone records of millions of Verizon customers daily. *The Guardian*, June 6, 2013.

[11] Human Rights Council. The right to privacy in the digital age: Report of the Office of the United Nations High Commissioner for Human Rights, June 2014.

[12] Seny Kamara. Restructuring the NSA Metadata Program. In *Workshop on Applied Homomorphic Cryptography (WAHC)*, March 2014.

[13] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Annual International Cryptology Conference (CRYPTO)*. Springer, August 2005.

[14] Joshua A. Kroll, Edward W. Felten, and Dan Boneh. Secure protocols for accountable warrant execution, April 2014.

[15] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay – a secure two-party computation system. In *USENIX Security Symposium*, August 2004.

[16] Daniel J. Solove. *Nothing to Hide: The False Tradeoff Between Privacy and Security*. Yale University Press, 2011.

[17] Ashkan Soltani and Barton Gellman. New documents show how the NSA infers relationships based on mobile location data. *The Washington Post*, December 10, 2013.

[18] Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *Journal of Computer Security*, 13(4):593–622, 2005.

[19] Yihua Zhang, Aaron Steele, and Marina Blanton. Picco: A general-purpose compiler for private distributed computation. In *ACM Conference on Computer and Communications Security*, November 2013.

Systematizing Secure Computation for Research and Decision Support

Jason Perry[1], Debayan Gupta[2], Joan Feigenbaum[2]
and Rebecca N. Wright[1]

[1] Rutgers University, NJ, USA. {jason.perry|rebecca.wright}@rutgers.edu
[2] Yale University, CT, USA. {debayan.gupta|joan.feigenbaum}@yale.edu

Abstract. We propose a framework for organizing and classifying research results in the active field of secure multiparty computation (MPC). Our *systematization of secure computation* consists of (1) a set of definitions circumscribing the MPC protocols to be considered; (2) a set of quantitative axes for classifying and comparing MPC protocols; and (3) a knowledge base of propositions specifying the known relations between axis values. We have classified a large number of MPC protocols on these axes and developed an interactive tool for exploring the problem space of secure computation. We also give examples of how this systematization can be put to use to foster new research and the adoption of MPC for real-world problems.

1 Introduction

For more than 30 years, since the groundbreaking work of Yao [30,31] and Goldreich *et al.* [18], hundreds of research papers on *Secure Multiparty Computation* (MPC) have appeared, many of them proposing original protocols for carrying out general secure computation under varying sets of assumptions. In this paper, we systematically organize the main research results in this area, in order to:

- Help potential users of MPC learn which existing protocols and implementations best match the sensitive-data computations they would like to perform. This may stimulate adoption of MPC in areas where it would be beneficial.
- Help new researchers get up to speed in a complex area by providing an overview of the "lay of the land."
- Help MPC researchers explore the problem space and discover remaining openings for protocols with new combinations of requirements and security features—or for new impossibility results that preclude the existence of such protocols.

Most research papers in MPC include comparisons of their results to related work, often with tables related to the most significant protocol features, in order to provide context for understanding the paper's contributions. However, none has attempted to organize the larger problem space in order to meet the goals listed above. There are a handful of introductory surveys and textbook-like

treatments of MPC [17,11,23,12]; these have (justifiably) focused on a narrower region of the problem space or specific security model, in order to present the material in a pedagogically clean way. In contrast, we do not limit ourselves to one model or set of definitions but instead provide a framework for examining their variations.

Our work shares some common goals with research meant to foster the real-world adoption of secure MPC. Such papers include the MPC-in-the-field experiments of Feigenbaum *et al.* and Bogetoft *et al.* [15,8] and the end-user survey work of Kamm *et al.* [7].

Since an effort such as this can never comprehensively account for every piece of research in a sprawling and active field, the framework has been deliberately designed to be extensible; it can accommodate new results and refined definitions without breaking.

In Section 2, we present the three major components that we believe are needed to systematize the main body of MPC results: (1) a set of definitions delineating the boundaries of the problem space; (2) a set of quantitative features for describing protocols; and (3) a knowledge base of propositions specifying the known relationships and dependencies among features. In Section 3, we describe the construction of a systematization database of more than 60 significant MPC protocols, and in Section 4 we present a user interface designed to aid the exploration of the database of systematized MPC protocols and show how our systematization can be put to work to facilitate new research.

2 The Systematization

As a necessary prerequisite for this work, we have carried out an extensive literature survey producing an annotated bibliography of MPC research. It contains over 180 papers from the MPC literature, as well as a sampling of important papers for related problems, including secret sharing and oblivious transfer.

In addition to a written paragraph annotating each paper, the BibTeX source of our Annotated Bibliography includes tags for each paper indicating which aspects of MPC it treats, *e.g.*, `2party` for a paper with a specifically two-party protocol, or `uncond` for a protocol with unconditional security. These tags make it possible to write scripts to automatically generate a bibliography for any specific sub-problem or aspect of MPC.

The bibliography continues to be updated on an ongoing basis. The most recent version is available online [27].

2.1 Definitions

In this section, we provide definitions to delimit the scope of the secure computation protocols that we are concerned with. These definitions are purposely quite broad, so that a large range of work can potentially be captured by them.

Variables that determine the fundamental nature of an MPC protocol include: (1) whether the protocol is for a fixed number of parties (most commonly,

two) or is for any number $n \geq 2$ of parties, (2) whether the protocol is for computing one specific functionality (*e.g.*, set intersection) or any of a class of functionalities, and (3) whether the protocol treats exact computation only or secure computation of approximations, which is a generalization of exact MPC [14]. Our initial literature survey encompassed all of these, although not exhaustively. For the current systematization, we consider only protocols for exact computation, and thus we have four definitions, one for each setting of the two variables.

Since the way that security is defined varies from protocol to protocol, and indeed a primary purpose of our systematization is to examine such variations, our definitions necessarily cannot give any fixed definition of security. What matters is that, in the literature, protocols are proven to meet rigorous definitions of security and that our systematization indicates which definitions are actually in use for a given protocol. Therefore, for an MPC protocol to be considered as a candidate for systematization, in addition to fitting into one the definitions listed below, it must be accompanied by rigorous definitions of security, including privacy of inputs and correctness of outputs, that the protocol has been proven to meet. The nature of these security definitions is elaborated in the next subsection.

Definition 1. *A protocol for* Secure n-party Computation of a functionality f *is a specification of an interactive process by which a fixed number n of players, each holding a private input x_i, can compute a specific, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$.*

Definition 2. *A protocol for* Secure Multiparty Computation of a functionality f *is a specification of an interactive process by which any number $n \geq 2$ of players, each holding a private input x_i, can compute a specific, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$.*

Definition 3. *A protocol for* Secure n-party Computation of a class $\mathcal{C}$ of functionalities *allows a fixed number n of players, each holding a private input x_i, to compute an agreed-upon, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$, where f is any member of the class $\mathcal{C}$ of functionalities.*

Definition 4. *A protocol for* Secure Multiparty Computation of a class $\mathcal{C}$ of functionalities *allows any number $n \geq 2$ of players, each holding a private input x_i, to compute an agreed-upon, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$, where f is any member of the class $\mathcal{C}$ of functionalities.*

The class $\mathcal{C}$ is typically used to refer to the model of computation in which a protocol's functionalities are represented, such as circuits or RAM programs. A majority of the work in MPC has been concerned with universal (Turing-complete) computation, but there has been work exploring secure computation specifically for restricted computation classes, such as AC0 or NC1 circuits or regular or context-free languages.

All of the protocols we surveyed for the systematization fall under one of these definitions, with the majority coming under the most general definition (Definition 4). Yao-like two-party computation protocols fall under Definition 3.

2.2 Linear axis representation of MPC protocol features

The main conceptual object in our systematization is a set of linear *axes*, where each axis represents an ordering of values of a single feature of MPC protocols. Every axis has at least two labeled values, at the endpoints. Some axes are continuous and others discrete. MPC protocols can be scored on these axes, allowing them to be compared quantitatively. This is the first attempt we are aware of to factor research results in MPC into such a representation. The axes were selected based on our literature survey, using two guiding principles:

1. The axes should be as orthogonal as possible, minimizing overlap (although some logical dependencies between axes are unavoidable.)
2. The set of axes should be *complete* in the sense that they can express all distinctions of security and (asymptotic) efficiency between any two protocols.

For any discrete axis that is not inherently binary, the number of occupied intermediate values on the axis is subject to change. The diagrams below show intermediate values that are known to have been achieved by MPC protocols. However, this should not be seen as finally determining the number of points on the axis. Indeed, one of the objectives of the axis representation is to highlight the possibility of future protocols with new intermediate values. This has already happened for several axes over the history of MPC research. For example, the appearance of protocols tailored for covert adversaries, such as those of Aumann *et al.* and Goyal *et al.* [1,22], showed that there are intermediate values along the "Maliciousness" axis (Axis 7), whereas previously only the passive/malicious distinction had been considered.

We orient all the axes in the same direction, such that moving from left to right on a given axis indicates an improved protocol—*e.g.*, one that is more efficient, has a stronger security guarantee, or requires a weaker setup or computational-hardness assumption. In drawing the non-numeric axes, the points have been placed with equal spacing; the relative distances between points on these axes should not be considered significant.

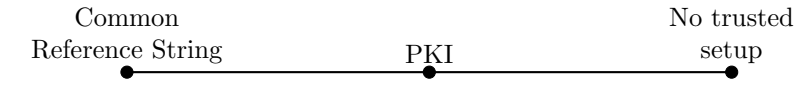
Our axes do not include the model of computation in which a protocol is expressed. This is a categorical feature which is indicated by the definition (Section 2.1) under which the protocol falls. The model of computation for each protocol is included in its entry in our MPC protocol database, described in Section 3. We have not generated axes for proof techniques, because a proof technique is not a function of the protocol; a protocol's security may be proven in a number of ways.

We now proceed to describe the axes and values in detail. The axes are informally grouped into four categories, which serves to highlight the tradeoffs inherent in achieving secure multiparty computation. The axes in categories I

and II, Environmental features and Assumptions, can be thought of as what one "pays" to enable secure MPC, and categories III (Security) and IV (Efficiency) can be seen as what one is "buying." A similar tradeoff structure can also be seen on a smaller scale among axes within the efficiency category. When it is helpful, our description of an axis also cites particular MPC solutions that instantiate points on the axis.

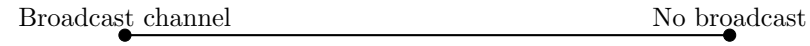
I. Environmental Features Axes
This is the category of features assumed to be provided by the execution environment. The right endpoint of these axes indicates that the feature is not required in any form.

1. Trusted setup



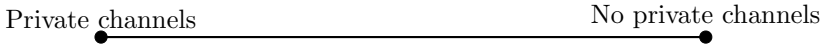
Protocols achieving the highest composable security levels require some type of trusted data to be shared by all the parties prior to the protocol execution. The middle point, PKI, is occupied by protocols such as that of Barak *et al.* [2], who showed how to use public-key-like assumptions instead of a polynomial-length common reference string in any case where computational security suffices.

2. Broadcast



The broadcast-channel assumption means that each party has the ability to send a message to all other parties simultaneously, and that all parties receiving the broadcast have assurance that the same message was received by all parties.

3. Private channels



The private channel assumption is only significant for unconditionally secure protocols, because cryptography using basic computational-hardness assumptions can be used to emulate private and authenticated communication channels.

4. Synchronization

A basic assumption of the early MPC protocols is that they operate on a *synchronous* network, in which all sent messages arrive on time and in order. The asynchronous case was first considered in Ben-Or *et al.* [4], where messages may be arbitrarily delayed and arrive in any order. Note that, in such a case, it is impossible to know whether a corrupted party has failed to send a message or, rather, the message is simply delayed. Later works, such as that of Damgård *et al.* [13], have staked

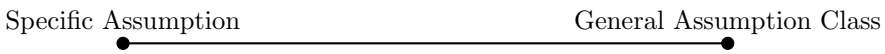
out intermediate points on this axis by giving protocols that require a smaller number of synchronization points (typically a single one).

II. Assumption Axes

5. Assumption level

A total (linear) ordering for cryptographic assumptions is not known, and the separation of assumptions cannot currently be unconditionally proven. We therefore use broader categories of assumption type, because these are usually sufficient to distinguish protocols. If a protocol makes no such assumptions, it is said to have *unconditional* security (see Axis 10). Some work specifies protocols in a *hybrid* model, with no concrete computational assumptions, but in which some high-level cryptographic operation (such as oblivious transfer) is assumed to exist as a black box. See Section 3 for a discussion of how such protocols are treated in this systematization.

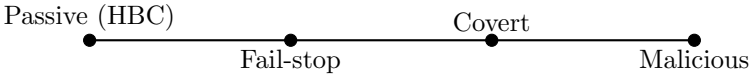
6. Specific or general assumption



Some more efficient protocols have been designed by making use of specific number-theoretic assumptions. This axis indicates whether the protocol requires such assumptions or whether it is stated so as to use any assumption from a given class, *e.g.*, trapdoor permutations.

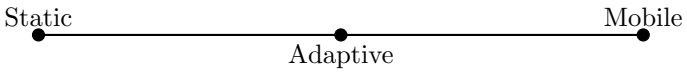
III. Security Axes

7. Adversary maliciousness



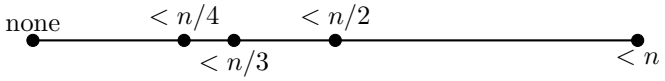
A passive, or honest-but-curious, adversary is one that follows the protocol but may use the data of corrupted parties to attempt to break the protocol's privacy. A fail-stop adversary follows the protocol except for the possibility of aborting. A malicious adversary is one whose behavior is arbitrary, and a covert adversary is like a malicious one, except that it only deviates from the protocol if the probability of being caught is low. Not present on the axis is the value "rational," since the class of rational adversaries is in fact a generalization that can encompass the entire axis, except for fully malicious, because malicious behavior can be truly arbitrary. The position of a rational adversary on the axis is determined by its utility function.

8. Adversary mobility



A static adversary must choose which parties to corrupt before the protocol begins. An adaptive adversary can choose which parties to corrupt, up to the security threshold (see Axis 9), over the course of the computation, after observing the state of previously corrupted parties. A mobile adversary is able to move corruptions from one party to another in the course of the computation.

9. Number of corrupted parties tolerated

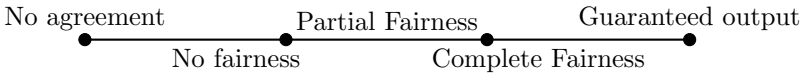


This is the maximum number of corrupted parties for which the (strongest) security guarantees of the protocol hold. The values shown are chosen merely to be representative of the most well known protocols; any value along the axis is possible.
Some protocols tolerate additional corrupted parties at a lower level of maliciousness; see axes 13 and 14.

10. Security type

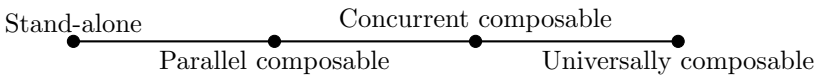
Both statistical and perfect security are unconditional, that is, not based on computational hardness assumptions. Note that true unconditional security typically cannot be achieved through the internet, even if an unconditionally-secure protocol is used, since all unconditionally secure protocols require the assumption of either private or broadcast channels, which on the internet must be emulated by cryptography.

11. Fairness guarantee



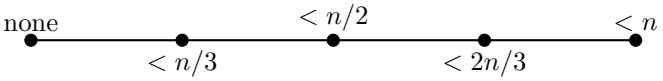
A protocol is *fair* if all honest parties receive the output if any party does. Agreement means that either all honest parties receive the output or none of them do. Protocols without agreement were introduced by Goldwasser *et al.* [20]. Some authors use the term "with abort" to refer to the no-fairness situation, in which dishonest parties can abort after receiving the correct output.

12. Composability



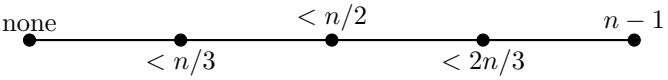
The composability guarantees of a protocol indicate whether that protocol remains secure when executed in an environment where other protocols may be executed sequentially or in parallel. The strongest guarantee, *universal composability* (UC), implies that the security properties of a protocol hold regardless of the environment in which it is executed.

13. Bound for additional passively corrupted parties tolerated



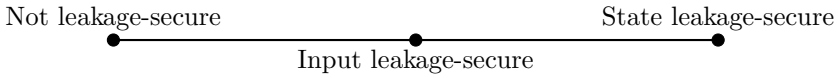
This axis applies to protocols achieving "mixed adversary" security. A protocol that tolerates a certain proportion of maliciously/covertly corrupted parties may also tolerate an additional number of passively corrupted parties, up to a certain threshold. The values on this axis represent that upper threshold. This and the following axis relate to MPC protocols with "graceful degradation", which is surveyed in Hirt *et al.* [24].

14. Corrupted parties tolerated with weakened security



This axis applies to protocols with "hybrid security" results. A protocol that tolerates a certain proportion of corrupted parties (Axis 9) may in fact tolerate a larger number of corruptions, but with a weaker security type, *e.g.*, computational vs. unconditional security.

15. Leakage Security



Leakage security is an additional guarantee that an adversary cannot gain an advantage even if it can force all honest players to "leak" some bits of information about their state in the course of the computation. See definitions in Bitansky *et al.* [6].

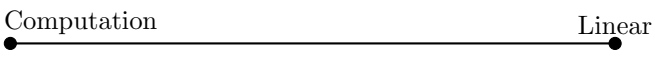
16. Auditability

This axis indicates whether the protocol includes computations that allow for examining the transcript of computation after it is finished, to prove that the parties have correctly followed the protocol. This may be the most recent axis to come into existence, starting with the work of Baum *et al.* [3]

IV. Efficiency Axes

Our efficiency axes are concerned primarily with the asymptotic efficiency of the protocol in question.

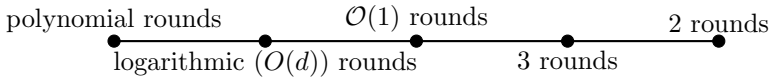
17. Online computational overhead



Historically, the main efficiency concern in MPC has been with communication rather than computational complexity; thus the lack of elaboration of this axis. More recently, Ishai *et al.* have notably shown how to

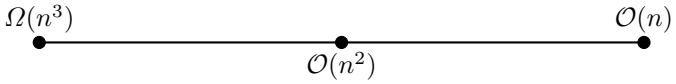
achieve MPC with constant computational overhead [25], and the RAM-model results of Gordon *et. al* [21] have shown the possibility, in the RAM model, of MPC with amortized computation that is sublinear in the input size.

18. Online communication complexity (rounds)



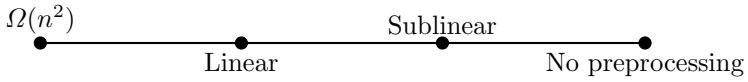
Here, d is the depth of the circuit representing the functionality. Minimizing the number of rounds of computation, independently of the total amount of bytes communicated, is crucial for efficiency in a high-latency or asynchronous network environment. Fully general MPC was shown to require at least three rounds in Gennaro *et al.* [16], although for some functionalities a 2-round protocol is possible.

19. Online communication complexity (per-gate)



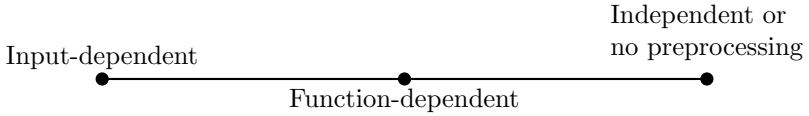
This is the most significant measure of efficiency for MPC protocols. It can represent either bits or field elements of communication. The original BGW protocol has a communication complexity of $O(n^6)$ bits per multiplication gate in the worst case. Anything cubic or worse occupies the lowest position on our axis, as finer distinctions at that level would have little value for distinguishing current, more practical protocols.

20. Preprocessing Communication complexity



Many recent protocols achieve improved online communication efficiency by means of a preprocessing phase. In the case where the functionality is represented as an arithmetic circuit, the preprocessing phase is typically a simulation of a trusted dealer that distributes *multiplication triples*, which allow local evaluation of multiplication gates in the online phase. Sublinear preprocessing typically indicates that the preprocessing consists only of exchange of public keys, which is also indicated as a setup assumption (Axis 1).

21. Preprocessing Dependency



In some protocols, the preprocessing phase depends on the specific functionality to be computed, while in others it only depends on the upper bound of the size of the circuit. In all cases, the preprocessing is independent of the parties' inputs.

22. Preprocessing Reuse

Not Reusable — Reusable

This indicates whether the information computed in the preprocessing stage, of whatever type or amount, can be reused for multiple computations. Data of the nature of a public key typically can be reused, while *e.g.,* garbled circuits traditionally cannot be reused without breaking security. But see recent work of Goldwasser *et al.* [19].

As mentioned above, we have endeavored to make this selection of axes as complete as possible. The value of completeness in a systematization can be illustrated as follows: Suppose there are two MPC protocols whose scores along the axes are identical for every axis except one. If the set of axes is complete, then we can be confident that the protocol with the higher value on that axis is strictly better.

2.3 Dependencies Between Axes

Since many MPC protocols involve essential tradeoffs in order to achieve security or efficiency, a systematization of secure computation also needs to model what is known about how features of protocols interact. In this section, we present the second major aspect of our systematization: a list of each of the theorems known to imply constraints among the axes' values, each accompanied by a statement of the constraint. References are given to the paper in which the theorem implying the constraint was proven.

Theorem 1 ([5]). *If statistical or perfect security is obtained, then either a broadcast channel or private channels must be assumed.* ***Axis constraint****: If Axis 10's value is to the right of "Computational," then either Axis 3's value is "Private channel" or Axis 2's value is "Broadcast channel".*

Theorem 2 ([29]). *No protocol with security against malicious adversaries can tolerate more than $n/2$ corrupted parties without losing the complete fairness property.* ***Axis constraint****: If Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/2$, then Axis 11's value must be to the left of "Complete fairness".*

Theorem 3 ([10]). *No protocol unconditionally secure against malicious adversaries can guarantee output delivery with $n/3$ or more corrupted parties.* ***Axis constraint****: If Axis 7's value is "Malicious," Axis 10's value is to the right of "Computational," and Axis 9's value is to the right of "$n/3$," then Axis 11's value must be to the left of "Guaranteed output".*

Theorem 4 ([5]). *No protocol can have perfect security against more than $n/3$ maliciously corrupted adversaries.* ***Axis constraint****: If Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then Axis 10's value must be to the left of "Perfect".*

Theorem 5 ([16]). *Any general MPC protocol with complete fairness against a malicious adversary must have at least three rounds.* ***Axis constraint****: If Axis 7's value is "Malicious" and Axis 11's value is at or to the right of "complete fairness", then Axis 18's value must be to the left of "2 rounds".*

Theorem 6 ([5]). *For unconditional security against t maliciously corrupted players, $n/3 \leq t < n/2$, a broadcast channel is required.* ***Axis constraint****: If Axis 10's value is to the right of "Computational" and Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then Axis 2's value must be "Broadcast channel".*

Theorem 7 ([18]). *For (even cryptographic) security against $\geq n/3$ maliciously corrupted players, either a trusted key setup or a broadcast channel is required.* ***Axis constraint****: If axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then either Axis 2's value must be "Broadcast channel," or else Axis 1's value is to the left of "No trusted setup."*

Theorem 8 ([5]). *There can be no unconditionally secure protocol against an adversary controlling a majority of parties.* ***Axis constraint****: Axis 10's value can be to the right of "Computational" only if Axis 9's value is at or to the left of $n/2$.*

Theorem 9 ([9]). *There is no protocol with UC security against a dishonest majority without setup assumptions.* ***Axis constraint****: If Axis 9's value is to the right of $n/2$ and Axis 12's value is "Universally composable," then axis 1's value must be to the left of "No trusted setup".*

Theorem 10 ([4]). *In an asynchronous environment, there is no protocol with guaranteed output secure against a fail-stop adversary corrupting $n/3$ or more parties.* ***Axis constraint****: If Axis 4's value is "Asynchronous," Axis 7's value is at or to the right of "Fail-stop," and Axis 11's value is at "Guaranteed output," then Axis 9's value must be at or to the left of $n/3$.*

Theorem 11 ([4]). *In an asynchronous environment, there is no protocol with guaranteed output secure against a malicious adversary corrupting $n/4$ or more parties.* ***Axis constraint****: If Axis 4's value is "Asynchronous," Axis 7's value is "Malicious," and Axis 11's value is at "Guaranteed output," then Axis 9's value must be at or to the left of $n/4$.*

One validation of our choice of axes is that these theorems are directly and compactly expressible in terms of them, thus giving a unified representation of the central body of knowledge of MPC. The axis constraints can easily be represented in a programming or knowledge representation language, as Section 4 shows.

3 An Extensible Protocol Database

We scored more than 60 of the most significant protocols in secure multiparty computation on our axes, integrating information from our annotated bibliography, resulting in an extensible *MPC protocol database*.

Many papers in the area include multiple protocols. We give each protocol a separate entry in the database, which is labeled by adding a suffix to the usual "alpha"-style reference. For instance, "[GMW87]-mal" refers to the protocol of [18] that is secure against a malicious adversary. The database also indicates whether an implementation of the protocol is known to exist.

The work of constructing the database motivated many revisions of our set of axes and highlighted difficulties in systematizing MPC results, some of which we discuss here.

Efficiency. As mentioned in the axis descriptions, our efficiency axes are concerned primarily with asymptotic efficiency measurements. When we populated our database, we relied on evaluations in the literature, frequently from the paper actually introducing the protocol.

The model of computation in which a protocol's functionalities are expressed can have a large impact on concrete efficiency. Historically, MPC functionalities have been expressed as circuits. The original Yao model considers Boolean circuits, while most of the current state-of-the-art MPC protocols are in the arithmetic-circuit model. Implementing these requires performing field arithmetic, and, although the size of the field elements is a constant in the security parameter, the time taken to perform field operations can have a significant impact on efficiency. Although this difference in concrete efficiency is not captured by the axes, our protocol database notes the model of computation for each protocol.

Even in the asymptotic case, comparing the efficiency of MPC protocols is an extremely difficult problem because of multiple interacting aspects of efficiency present in MPC. In selecting an actual implementation, a concrete analysis and/or empirical efficiency measurements should also be consulted.

Substitutability. One factor that makes it nontrivial to enumerate a list of MPC protocols is that many protocols described in the literature make use of subprotocols for cryptographic operations in a black-box fashion, making it possible to substitute different protocols implementing that operation. This can alter not only the performance characteristics but also the computational and environmental assumptions and security and composability guarantees of the resultant protocol. In some cases, a new and improved subprotocol can trivially be used to improve an older MPC protocol, but no published work explicitly presents the improvement; in other cases a protocol explicitly allows for black-box substitution of subprotocols, in which case it is said to be stated in a *hybrid* model. In the case of the *OT-hybrid* model, in which oblivious transfer is a black box, recent work in *OT extension* has produced significant performance gains.

An extreme case of this "substitutability" factor is the IPS compiler of Ishai *et al.* [26], which is not only in the OT-hybrid model but also allows any of a wide of of honest-majority MPC protocols to be plugged in as an "outer protocol," with the resulting protocol inheriting some (but not all) of the security properties of the outer protocol.

To address this complex issue, we have limited our systematization to represent only concrete instantiations of hybrid protocols. The axes are such that

a protocol in a hybrid model must first be "instantiated" with concrete subprotocols in order to be scored. In our database, we have sought to enumerate as many such concrete protocols as possible that are based on well-known hybrid-model protocols.

Two-party secure computation. Although much research has been done specifically addressing two-party secure computation, starting from Yao's original garbled circuits idea, it can be considered as a special case of multiparty computation, in which, if security against a malicious adversary is sought, no honest majority can be assumed (Axis 9 at $n - 1$). Thus, two-party protocols can at least theoretically be compared with multiparty results in this category.

In reality, however, vast improvements in efficiency have been made for the two-party case. We note that these optimizations often come at the expense of *symmetry*: The security guarantee against cheating by one of the two parties may be weaker than for the other. For instance, one of the two parties may be able to cheat with an inverse-polynomial probability, while the other may only be able to cheat with negligible probability. Asymmetry is not included in our axis definitions, and so two-party protocols are scored by the weaker of the two sets of security guarantees. Of course, the database indicates which protocols and implementations are strictly for two parties.

4 Putting the Systematization to Work

As mentioned in Section 1, the main theorems of secure multiparty computation demonstrate essential tradeoffs involved in securely computing a functionality among distrustful parties. To leverage the information that our systematization captures about these tradeoffs and gain insight into the problem space, we experimented with visually plotting the protocol axis scores of the MPC database. However, we found that the highly categorical nature of the data makes it difficult to gain insight from a static visualization. In this section, we describe an an interactive tool we have developed for interacting with the systematization and MPC protocol database, which provides a better way of coming to grips with the multi-dimensional landscape of MPC protocols.

4.1 A prototype decision-making support tool

We have developed a prototype GUI tool, *SysSC-UI*, which reads in a protocol database of axis values and enables the user to adjust a set of sliders and checkboxes corresponding to our axes of systematization. For the tool's interface, the axes are oriented vertically rather than horizontally as in this paper, so a higher position of a slider corresponds to a stronger result. A dynamically updated *results window* displays the protocols from the database that match the specified axis values. See the screenshot in Figure 1. The source code for the desktop version is available online `https://code.google.com/p/syssc-ui/`, as well as a beta web-based version `http://work.debayangupta.com/ssc/`.

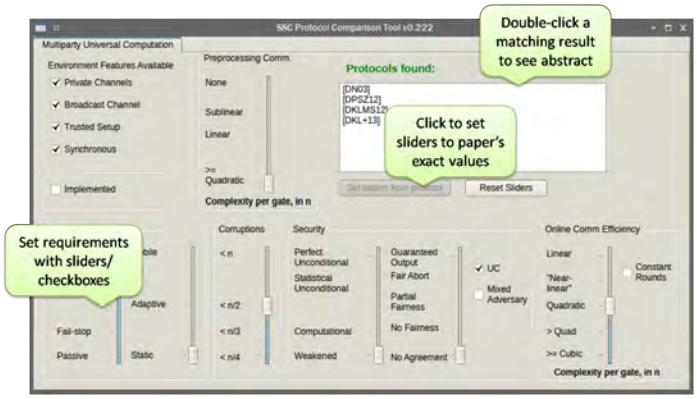


Fig. 1. Screenshot of the SysSC-UI tool for interacting with the protocol database.

For a given setting of the sliders and checkboxes, the results window shows all papers whose axis values are at the same level *or higher* than the settings. Thus, the tool presents all protocols that are *at least as good* as the specified settings. When the tool is started, the sliders and checkboxes are all set to the least constraining position, such that every protocol in the database is displayed in the results window. There is also a button to reset the tool to this state. Another button sets the sliders to the exact values of the protocol in the currently highlighted protocol, allowing the precise achievements of a protocol to be examined. This has the side effect of changing the output to the results window, so that only protocols that are at least as good as the selected one are displayed.

Double-clicking on a reference in the results window will display a pop-up window giving the authors' full names and the description of the paper containing the protocol from the annotated bibliography. The GUI also indicates when the positioning of the sliders is such that a secure computation protocol is known to be impossible, by means of an encoding of the theorems in Section 2.3.

The axes and values displayed by SysSC-UI are a subset of those in the full systematization. This was done in order to simplify the interaction and avoid confusion from the visual display of too much information at once. For example, for the composability axis, there is only a single checkbox, to indicate whether the protocol is proven universally composable or not.

We now present sample use cases highlighting the features of SysSC-UI.

4.2 Sample use cases for SysSC-UI

Finding the best protocol for a known problem. Consider a scenario in which a technology consultant is hired by a company to find a way to compute some function of a distributed set of sensitive data residing on servers owned by different divisions of the company. We show how she can use the SysSC-UI tool to find an appropriate MPC protocol for achieving this secure computation.

In the initial state of the UI tool, all four environmental assumption boxes are checked, and all sliders are in the lowest position, so that every protocol in the database is displayed in the results window. To begin, our consultant unchecks "Private Channels", knowing that the computation will be carried out over an ordinary internet connection, which should always be assumed to be tapped. She wishes to protect against adversaries that are covertly malicious, so she moves the leftmost Adversary Type slider up to "Covert". The consultant is suspicious of protocols that use a weaker model to prove security, so she moves the first Security slider up to "Computational." Furthermore, she suspects that universal composability is necessary to guarantee security in a heterogenous environment, so she checks the "UC" box. The protocols resulting from these selections can be seen in the results window in Figure 2.

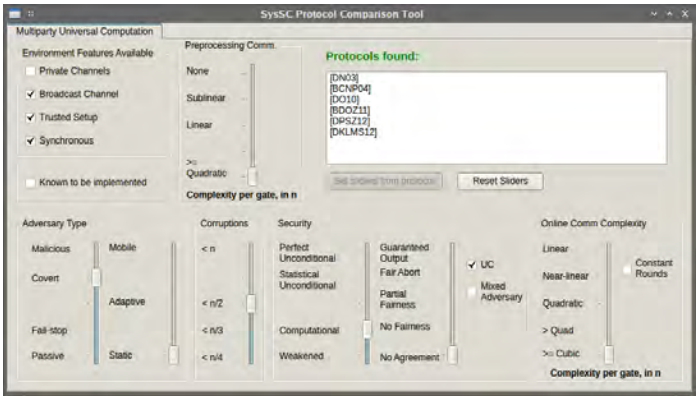


Fig. 2. Results shown by the SysSC-UI tool after selections have been made.

The consultant wishes to determine the most efficient protocol that meets these requirements, so next she moves the "Online Comm Complexity" slider up to the highest level for which the results box is not empty. We would like to achieve this online complexity with the minimum preprocessing complexity, so she tries sliding the "Preprocessing comm" slider up. If it is moved up too far, the results window becomes empty. So she readjusts both this and the Online slider to find an agreeable tradeoff between online and preprocessing complexity. The results of this exploration are shown in Figure 2.

Exposing directions for cryptographic research. In using the tool, one invariably stumbles upon a setting of the sliders / boxes that is not in the "known impossible" range, and yet has no papers with a matching protocol listed. Of course, one reason for this could simply be the incompleteness of the protocol database. Another possible reason is that the combination of features is not desirable from a security or efficiency standpoint. However, a third possibility exists, which is that a genuine opening for new research has been revealed. Two kinds of such "holes" may reveal new research directions: (1) Gaps between achieved

and proven impossible security levels, and (2) settings in which a weakening of security parameters may allow greater efficiency. An example of the second type of advance is the case of positing composable security definitions weaker than universal composability, as in [28].

5 Ongoing Work

As it is unlikely that the authors will permanently be able to stay abreast of the growing MPC literature, we have developed a web-based survey that allows researchers to submit descriptions of new protocols and their features on the axes so that they can be integrated into the protocol database and SysSC-UI. The survey is available at `http://goo.gl/T4ORzr`. Author participation is vital to the continuing usefulness of this effort.

The set of theorems in Section 2.3 should also be expanded as knowledge of secure computation increases. This work has highlighted several remaining unknowns in characterizing the possibility of secure computation; for example, perhaps some of the malicious impossibility results hold for the covert case as well. A longer-term goal is to characterize the efficiency of the protocols more precisely, in terms of the number of elementary operations, to make the efficiency of all protocols directly comparable.

Systematizations of knowledge are especially needed in research fields where a large body of results have been generated in a short time, and secure multiparty computation is undoubtedly such a field. Without an effort to systematically organize results, there may be unnecessary duplication of research efforts, the barriers to entry for new researchers may be needlessly high, and results may not see useful applications as early as they could. Our systematization of secure computation is a tool that can significantly ease the task of coming to grips with this sprawling body of results, and potentially speed its adoption in fields where it would be useful.

Acknowledgments

This material is based on research sponsored by DARPA under agreement number FA8750-13-2-0058. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the office policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

References

1. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Theory of Cryptography – TCC 2007. pp. 137–156 (2007)

2. Barak, B., Canetti, R., Nielsen, J.B., Pass, R.: Universally composable protocols with relaxed set-up assumptions. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004). pp. 186–195. IEEE (2004)
3. Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Report 2014/075 (2014), http://eprint.iacr.org/
4. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC '93). pp. 52–61 (1993)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88). pp. 1–10 (1988)
6. Bitansky, N., Canetti, R., Halevi, S.: Leakage-tolerant interactive protocols. In: Theory of Cryptography – TCC 2012. pp. 266–284 (2012)
7. Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P.: Secure multi-party data analysis: end user validation and practical experiments. Cryptology ePrint Archive, Report 2013/826 (2013), http://eprint.iacr.org/2013/826
8. Bogetoft, P., Christensen, D., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J., Nielsen, J., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 5628, pp. 325–343. Springer Berlin Heidelberg (2009)
9. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Advances in Cryptology – EUROCRYPT 2003, pp. 68–86. Springer (2003)
10. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC. pp. 364–369 (1986)
11. Cramer, R., Damgård, I.: Multiparty computation, an introduction. In: Contemporary cryptology, pp. 41–87. Springer (2005)
12. Cramer, R., Damgård, I., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing: An Information Theoretic Approach. Self-published manuscript (2013), https://users-cs.au.dk/jbn/mpc-book.pdf
13. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Public Key Cryptography. pp. 160–179 (2009)
14. Feigenbaum, J., Ishai, Y., Malkin, T., Nissim, K., Strauss, M.J., Wright, R.N.: Secure multiparty computation of approximations. ACM Transactions on Algorithms 2(3), 435–472 (2006)
15. Feigenbaum, J., Pinkas, B., Ryger, R., Saint-Jean, F.: Secure computation of surveys. In: EU Workshop on Secure Multiparty Protocols. Citeseer (2004)
16. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002, Lecture Notes in Computer Science, vol. 2442, pp. 178–193. Springer Berlin Heidelberg (2002)
17. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
18. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Proceedings of the Nine-

teenth Annual ACM Symposium on Theory of Computing (STOC '87). pp. 218–229 (1987)
19. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13). pp. 555–564 (2013)
20. Goldwasser, S., Lindell, Y.: Secure computation without agreement. In: Proceedings of the 16th Int'l Symposium on DIStributed Computing (DISC). pp. 17–32 (2002)
21. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: ACM Conference on Computer and Communications Security (ACM CCS 2012). pp. 513–524 (2012)
22. Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Advances in Cryptology – EUROCRYPT 2008. pp. 289–306 (2008)
23. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols – Techniques and Constructions. Information Security and Cryptography, Springer (2010)
24. Hirt, M., Lucas, C., Maurer, U., Raub, D.: Graceful degradation in multi-party computation (extended abstract). In: 5th International Conference on Information Theoretic Security (ICITS 2011). pp. 163–180 (2011)
25. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08). pp. 433–442. ACM, New York, NY, USA (2008)
26. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Advances in Cryptology – CRYPTO 2008. pp. 572–591 (2008)
27. Perry, J., Gupta, D., Feigenbaum, J., Wright, R.N.: The secure computation annotated bibliography (2014), `http://paul.rutgers.edu/~jasperry/ssc-annbib.pdf`
28. Prabhakaran, M., Sahai, A.: New notions of security: achieving universal composability without trusted setup. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04). pp. 242–251 (2004)
29. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89). pp. 73–85 (1989)
30. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982). pp. 160–164 (1982)
31. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1986). pp. 162–167 (1986)

Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values

Benjamin Mood
University of Florida
bmood@ufl.edu

Debayan Gupta
Yale University
debayan.gupta@yale.edu

Kevin R. B. Butler
University of Florida
butler@ufl.edu

Joan Feigenbaum
Yale University
joan.feigenbaum@yale.edu

ABSTRACT

Two-party secure-function evaluation (SFE) has become significantly more feasible, even on resource-constrained devices, because of advances in server-aided computation systems. However, there are still bottlenecks, particularly in the input-validation stage of a computation. Moreover, SFE research has not yet devoted sufficient attention to the important problem of retaining state after a computation has been performed so that expensive processing does not have to be repeated if a similar computation is done again. This paper presents PartialGC, an SFE system that allows the reuse of encrypted values generated during a garbled-circuit computation. We show that using PartialGC can reduce computation time by as much as 96% and bandwidth by as much as 98% in comparison with previous outsourcing schemes for secure computation. We demonstrate the feasibility of our approach with two sets of experiments, one in which the garbled circuit is evaluated on a mobile device and one in which it is evaluated on a server. We also use PartialGC to build a privacy-preserving "friend-finder" application for Android. The reuse of previous inputs to allow stateful evaluation represents a new way of looking at SFE and further reduces computational barriers.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection – *Cryptographic Controls*

Keywords

Garbled Circuits, Cut-and-Choose, Server-Aided Computation

1. INTRODUCTION

Secure function evaluation, or *SFE*, allows multiple parties to jointly compute a function while maintaining input and output privacy. The two-party variant, known as 2P-SFE, was first introduced by Yao in the 1980s [39] and was largely a theoretical curiosity. Developments in recent years have made 2P-SFE vastly more efficient [18, 27, 38]. However, computing a function using SFE is still usually much slower than doing so in a non-privacy-preserving manner.

As mobile devices become more powerful and ubiquitous, users expect more services to be accessible through them. When SFE is performed on mobile devices (where resource constraints are tight), it is extremely slow – *if* the computation can be run at all without exhausting the memory, which can happen for non-trivial input sizes and algorithms [8]. One way to allow mobile devices to perform SFE is to use a server-aided computational model [8, 22], allowing the majority of an SFE computation to be "outsourced" to a more powerful device while still preserving privacy. Past approaches, however, have not considered the ways in which mobile computation differs from the desktop. Often, the mobile device is called upon to perform *incremental* operations that are continuations of a previous computation.

Consider, for example, a "friend-finder" application where the location of users is updated periodically to determine whether a contact is in proximity. Traditional applications disclose location information to a central server. A privacy-preserving friend-finder could perform these operations in a mutually oblivious fashion. However, every incremental location update would require a full re-evaluation of the function with fresh inputs in a standard SFE solution. Our examination of an outsourced SFE scheme for mobile devices by Carter et al. [8] (hereon CMTB), determined that the cryptographic consistency checks performed on the inputs to an SFE computation *themselves* can constitute the greatest bottleneck to performance.

Additionally, many other applications require the ability to save state, a feature that current garbled-circuit implementations do not possess. The ability to save state and reuse an intermediate value from one garbled circuit execution in another would be useful in many other ways, *e.g.*, we could split a large computation into a number of smaller pieces. Combined with efficient input validation, this becomes an extremely attractive proposition.

In this paper, we show that it is possible to reuse an encrypted value in an outsourced SFE computation (we use a cut-and-choose garbled circuit protocol) even if one is restricted to primitives that are part of standard garbled circuits. Our system, PartialGC, which is based on CMTB, provides a way to take encrypted output wire values from one SFE computation, save them, and then reuse them as

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660285.

input wires in a new garbled circuit. Our method vastly reduces the number of cryptographic operations compared to the trivial mechanism of simply XOR'ing the results with a one-time pad, which requires either generating inside the circuit, or inputting, a very large one-time pad, both complex operations. Through the use of improved input validation mechanisms proposed by shelat and Shen [38] (hereon sS13) and new methods of *partial input* gate checks and evaluation, we improve on previous proposals. There are other approaches to the creation of reusable garbled circuits [13, 10, 5], and previous work on reusing encrypted values in the ORAM model [30, 11, 31], but these earlier schemes have not been implemented. By contrast, we have implemented our scheme and found it to be both practical and efficient; we provide a performance analysis and a sample application to illustrate its feasibility (Section 6), as well as a simplified example execution (Appendix C).

By breaking a large program into smaller pieces, our system allows interactive I/O throughout the garbled circuit computation. To the best of our knowledge this is the first practical protocol for performing interactive I/O in the middle of a cut-and-choose garbled circuit computation.

Our system comprises three parties - a generator, an evaluator, and a third party ("the cloud"), to which the evaluator outsources its part of the computation. Our protocol is secure against a malicious adversary, assuming that there is no collusion with the cloud. We also provide a semi-honest version of the protocol.

Figure 1 shows how PartialGC works at a high level: First, a standard SFE execution (blue) takes place, at the end of which we "save" some intermediate output values. All further executions use intermediate values from previous executions. In order to reuse these values, information from both parties – the generator and the evaluator – has to be saved. In our protocol, it is the cloud – rather than the evaluator – that saves information. This allows multiple distinct evaluators to participate in a large computation over time by saving state in the cloud between different garbled circuit executions. For example, in a scenario where a mobile phone is outsourcing computation to a cloud, PartialGC can save the encrypted intermediate outputs to the cloud instead of the phone (Figure 2). This allows the phones to communicate with each other by storing encrypted intermediate values in the cloud, which is more efficient than requiring them to directly participate in the saving of values, as required by earlier 2P-SFE systems. Our friend finder application, built for an Android device, reflects this usage model and allows multiple friends to share their intermediate values in a cloud. Other friends use these saved values to check whether or not someone is in the same map cell as themselves without having to copy and send data.

By incorporating our optimizations, we give the following contributions:

1. *Reusable Encrypted Values* – We show how to reuse an encrypted value, using only garbled circuits, by mapping one garbled value into another.
2. *Reduced Runtime and Bandwidth* – We show how reusable encrypted values can be used in practice to reduce the execution time for a garbled-circuit computation; we get a 96% reduction in runtime and a 98% reduction in bandwidth over CMTB.
3. *Outsourcing Stateful Applications* – We show how our system increases the scope of SFE applications by allowing multiple evaluating parties over a period of time to operate on the saved state of an SFE computation without the need for these parties to know about each other.

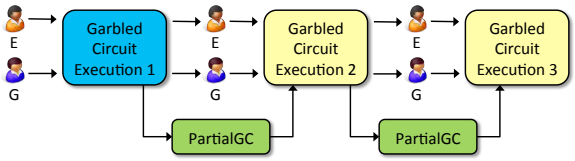


Figure 1: PartialGC Overview. E is evaluator and G is generator. The blue box is a standard execution that produces partial outputs (garbled values); yellow boxes represent executions that take partial inputs and produce partial outputs.

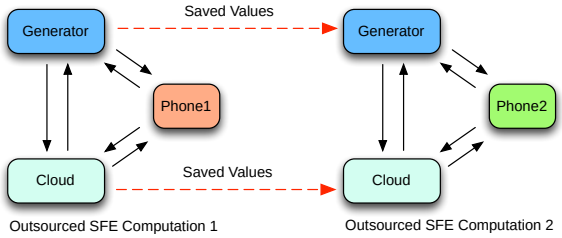


Figure 2: Our system has three parties. Only the cloud and generator have to save intermediate values - this means that we can have different phones in different computations.

The remainder of our paper is organized as follows: Section 2 provides some background on SFE. Section 3 introduces the concept of partial garbled circuits in detail. The PartialGC protocol and its implementation are described in Section 4, while its security is analyzed in Section 5. Section 6 evaluates PartialGC and introduces the friend finder application. Section 7 discusses related work and Section 8 concludes.

2. BACKGROUND

Secure function evaluation (SFE) addresses scenarios where two or more mutually distrustful parties $P_1, \ldots, P_n$, with private inputs $x_1, \ldots, x_n$, want to compute a given function $y_i = f(x_1, \ldots, x_n)$ (y_i is the output received by P_i), such that no P_i learns anything about any x_j or y_j, $i \neq j$ that is not logically implied by x_i and y_i. Moreover, there exists no trusted third party – if there was, the P_is could simply send their inputs to the trusted party, which would evaluate the function and return the y_is.

SFE was first proposed in the 1980s in Yao's seminal paper [39]. The area has been studied extensively by the cryptography community, leading to the creation of the first general purpose platform for SFE, Fairplay [32] in the early 2000s. Today, there exist many such platforms [6, 9, 16, 17, 26, 37, 40].

The classic platforms for 2P-SFE, including Fairplay, use garbled circuits. A garbled circuit is a Boolean circuit which is encrypted in such a way that it can be evaluated when the proper input wires are entered. The party that evaluates this circuit does not learn anything about what any particular wire represents. In 2P-SFE, the two parties are: the *generator*, which creates the garbled circuit, and the *evaluator*, which evaluates the garbled circuit. Additional cryptographic techniques are used for input and output; we discuss these later.

A two-input Boolean gate has four truth table entries. A two-input garbled gate also has a truth table with four entries representing 1s and 0s, but these entries are encrypted and can only be retrieved when the proper keys are used.

The values that represent the 1s and 0s are random strings of bits. The truth table entries are permuted such that the evaluator cannot determine which entry she is able to decrypt, only that she is able to decrypt an entry. The entirety of a garbled gate is the four encrypted output values.

Each garbled gate is then encrypted in the following way: Each entry in the truth table is encrypted under the two input wires, which leads to the result, $truth_i = Enc(input_x || input_y) \oplus output_i$, where $truth_i$ is a value in the truth table, $input_x$ is the value of input wire x, $input_y$ is the value of input wire y, and $output_i$ is the non-encrypted value, which represents either 0 or 1.We use AES as the Enc function. If the evaluator has $input_x$ and $input_y$, then she can also receive $output_i$, and the encrypted truth tables are sent to her for evaluation.

For the evaluator's input, 1-out-of-2 oblivious transfers (OTs) [1, 20, 34, 35] are used. In a 1-out-of-2 OT, one party offers up two possible values while the other party selects one of the two values without learning the other. The party that offers up the two values does not learn which value was selected. Using this technique, the evaluator gets the wire labels for her input without leaking information.

The only way for the evaluator to get a correct output value from a garbled gate is to know the correct decryption keys for a specific entry in the truth table, as well as the location of the value she has to decrypt.

During the permutation stage, rather than simply randomly permuting the values, the generator permutes values based on a specific bit in $input_x$ and $input_y$, such that, given $input_x$ and $input_y$ the evaluator knows that the location of the entry to decrypt is $bit_x * 2 + bit_y$. These bits are called the *permutation bits*, as they show the evaluator which entry to select based on the permutation; this optimization, which does not leak any information, is known as *point and permute* [32].

2.1 Threat Models

Traditionally, there are two threat models discussed in SFE work, semi-honest and malicious. The above description of garbled circuits is the same in both threat models. In the semi-honest model users stay true to the protocol but may attempt to learn extra information from the system by looking at any message that is sent or received. In the malicious model, users may attempt to change anything with the goal of learning extra information or giving incorrect results without being detected; extra techniques must be added to achieve security against a malicious adversary.

There are several well-known attacks a malicious adversary could use against a garbled circuit protocol. A protocol secure against malicious adversaries must have solutions to all potential pitfalls, described in turn:

Generation of incorrect circuits If the generator does not create a correct garbled circuit, he could learn extra information by modifying truth table values to output the evaluator's input; he is limited only by the external structure of the garbled circuit the evaluator expects.

Selective failure of input If the generator does not offer up correct input wires to the evaluator, and the evaluator selects the wire that was not created properly, the generator can learn up to a single bit of information based on whether the computation produced correct outputs.

Input consistency If either party's input is not consistent across all circuits, then it might be possible for extra information to be retrieved.

Output consistency In the two-party case, the output consistency check verifies that the evaluator did not modify the generator's output before sending it.

2.1.1 Non-collusion

CMTB assumes non-collusion, as quoted below:
"The outsourced two-party SFE protocol securely computes a function f(a,b) in the following two corruption scenarios: (1)The cloud is malicious and non-cooperative with respect to the rest of the parties, while all other parties are semi-honest, (2)All but one party is malicious, while the cloud is semi-honest."

This is the standard definition of non-collusion used in server-aided works such as Kamara et al. [22]. Non-collusion does not mean the parties are trusted; it only means the two parties are not working together (i.e. both malicious). In CMTB, any individual party that attempts to cheat to gain additional information will still be caught, but collusion between multiple parties could leak information. For instance, the generator could send the cloud the keys to decrypt the circuit and see what the intermediate values are of the garbled function.

3. PARTIAL GARBLED CIRCUITS

We introduce the concept of *partial garbled circuits* (PGCs), which allows the encrypted wire outputs from one SFE computation to be used as inputs to another. This can be accomplished by *mapping* the encrypted output wire values to valid input wire values in the next computation. In order to better demonstrate their structure and use, we first present PGCs in a semi-honest setting, before showing how they can aid us against malicious adversaries.

3.1 PGCs in the Semi-Honest Model

In the semi-honest model, for each wire value, the generator can simply send two values to the evaluator, which transforms the wire label the evaluator owns to work in another garbled circuit. Depending on the point and permute bit of the wire label received by the evaluator, she can map the value from a previous garbled circuit computation to a valid wire label in the next computation.

Specifically, for a given wire pair, the generator has wires w_0^{t-1} and w_1^{t-1}, and creates wires w_0^t and w_1^t. Here, t refers to a particular computation in a series, while 0 and 1 correspond to the values of the point and permute bits of the $t-1$ values. The generator sends the values $w_0^{t-1} \oplus w_0^t$ and $w_1^{t-1} \oplus w_1^t$ to the evaluator. Depending on the point and permute bit of the w_i^{t-1} value she possesses, the evaluator selects the correct value and then XORs her w_i^{t-1} with the $(w_i^{t-1} \oplus w_i^t)$ value, thereby giving her w_i^t, the valid partial input wire.

3.2 PGCs in the Malicious Model

In the malicious model we must allow the evaluation of a circuit with partial inputs and verification of the mappings, while preventing a selective failure attack. The following features are necessary to accomplish these goals:

1. Verifiable Mapping

The generator G is able to create a secure mapping from a saved garbled wire value into a new computation that can be checked by the evaluator E, without E being able to reverse the mapping. During the evaluation and check phase,

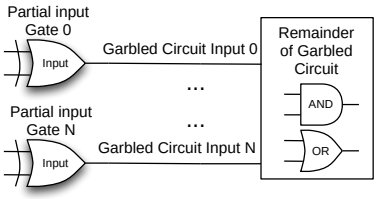


Figure 3: This figure shows how we create a single *partial input gate* for each input bit for each circuit and then link the *partial input gates* to the remainder of the circuit.

E must be able to verify the mapping G sent. G must have either committed to the mappings before deciding the partition of evaluation and check circuits, or never learned which circuits are in the check versus the evaluation sets.

2. Partial Generation and Partial Evaluation

G creates the garbled gates necessary for E to enter the previously output intermediate encrypted values into the next garbled circuit. These garbled gates are called *partial input gates*. As shown in Figure 3 each garbled circuit is made up of two pieces: the partial input gates and the remainder of the garbled circuit.

3. Revealing Incorrect Transformations

Our last goal is to let E inform G that incorrect values have been detected. Without a way to limit leakage, G could gain information based on whether or not E informs G that she caught him cheating. This is a selective failure attack and is not present in our protocol.

4. PARTIALGC PROTOCOL

We start with the CMTB protocol and add cut-and-choose operations from sS13 before introducing the mechanisms needed to save and reuse values. We defer to the original papers for full details of the outsourced oblivious transfer [8] and the generator's input consistency check [38] subprotocols that we use as primitives in our protocol.

Our system operates in the same threat model as CMTB (see Section 2.1.1): we are secure against a malicious adversary under the assumption of non-collusion. A description of the CMTB protocol is available in Appendix A.

4.1 Preliminaries

There are three participants in the protocol:

Generator – The generator is the party that generates the garbled circuit for the 2P-SFE.

Evaluator –The evaluator is the other party in the 2P-SFE, which is outsourcing computation to a third party, the cloud.

Cloud – The cloud is the party that executes the garbled circuit outsourced by the evaluator.

Notation

C_i - The ith circuit.

$CKey_i$ - Circuit key used for the free XOR optimization [25]. The key is randomly generated and then used as the difference between the 0 and 1 wire labels for a circuit C_i.

$CSeed_i$ - This value is created by the generator's PRNG and is used to generate a particular circuit C_i.

$POut\#_{i,j}$ - The *partial output* values are the encrypted wire values output from an SFE computation. These are encrypted garbled circuit values that can be reused in another garbled circuit computation. # is replaced in our protocol description with either a 0, 1, or x, signifying whether it represents a 0, 1, or an unknown value (from the cloud's point of view). i denotes the circuit the $POut$ value came from and j denotes the wire of the $POut_i$ circuit.

$PIn\#_{i,j}$ - The *partial input* values are the re-entered $POut$ values after they have been obfuscated to remove the circuit key from the previous computation. These values are input to the *partial input gates*. #, i, and j, are the same as above.

$GIn\#_{i,j}$ - The *garbled circuit input* values are the results of the partial input gates and are input into the remaining garbled circuit, as shown in Figure 3. #, i, and j, are the same as above.

Partial Input Gates - These are garbled gates that take in PIn values and output GIn values. Their purpose is to transform the PIn values into values that are under $CKey_i$ for the current circuit.

4.2 Protocol

Each computation is self-contained; other than what is explicitly described as saved in the protocol, each value or property is only used for a single part of the computation (*i.e.* randomness is different across computations).

Algorithm 0: PartialComputation

Input : Circuit_File, Bit_Security, Number_of_Circuits, Inputs, Is_First_Execution
Output: Circuit File Output
`Cut_and_Choose`(*is_First_Execution*)
Eval_Garbled_Input ← `Evaluator_Input`(*Eval_Select_Bits, Possible_Eval_Input*)
`Generator_Input_Check`(*Gen_Input*)
Partial_Garbled_Input ← `Partial_Input`($Partial\_Output_{time-1}$)
Garbled_Output, Partial_Output ← `Circuit_Execution`(*Garbled_Input (Gen, Eval, Partial)*)
`Circuit_Output`(*Garbled_Output*)
`Partial_Output`(*Partial_Output*)

Common Inputs: The program circuit file, the bit level security, the circuit level security (number of circuits) S, and encryption and commitment functions.

Private Inputs: The evaluator's input $evlInput$ and generator's input $genInput$.

Outputs: The evaluator and generator can both receive garbled circuit outputs.

Phase 1: Cut-and-choose

We modify the cut-and-choose mechanism described in sS13 as we have an extra party involved in the computation. In this cut-and-choose, the cloud selects which circuits are evaluation circuits and which circuits are check circuits,

$$circuitSelection = rand()$$

where $circuitSelection$ is a bit vector of size S; N evaluation circuits and $S-N$ check circuits are selected where $N = \frac{2}{5}S$. The generator does not learn the circuit selection.

The generator generates garbled versions of his input and circuit seeds for each circuit. He encrypts these values using unique 1-time XOR pad keys. For $0 \leq i < S$,

$$CSeed_i = rand()$$
$$garbledGenInput_i = garble(genInput, rand())$$
$$checkKey_i = rand()$$
$$evlKey_i = rand()$$
$$encSeedIn_i = CSeed_i \oplus evlKey_i$$
$$encGarbledIn_i = garbledGenInput_i \oplus checkKey_i$$

Algorithm 1: Cut_and_Choose

Input : is_First_Execution
if *is_First_Execution* **then**
  circuitSelection ← rand() // bit-vector of size S
$N \leftarrow \frac{2}{5}S$ // Number of evaluation circuits
//Generator creates his garbled input and circuit seeds for each circuit
for $i \leftarrow 0$ **to** S **do**
  $CSeed_i \leftarrow rand()$
  $garbledGenInput_i \leftarrow garble(genInput, rand())$
  //generator creates or loads keys
  if *is_First_Execution* **then**
    $checkKey_i \leftarrow rand()$
    $evlKey_i \leftarrow rand()$
  else
    loadKeys();
    $checkKey_i \leftarrow hash(loadedCheckKey_i)$
    $evlKey_i \leftarrow hash(loadedEvlKey_i)$
  // encrypts using unique 1-time XOR pads
  $encSeedIn_i \leftarrow CSeed_i \oplus evlKey_i$
  $encGarbledIn_i \leftarrow garbledGenInput_i \oplus checkKey_i$
if *is_First_Execution* **then**
  // generator offers input OR keys for each circuit seed
  $selectedKeys \leftarrow$
  $OT(circuitSelection, \{evlKey, checkKey\})$
else
  loadSelectedKeys()
for $i \leftarrow 0$ **to** S **do**
  $genSendToEval(hash(checkKey_i),$
  $hash(evaluationKey_i))$
for $i \leftarrow 0$ **to** S **do**
  $cloudSendToEval(hash(selectedKey_i), isCheckCircuit_i)$
// If all values match, the evaluator learns split, else abort.
for $i \leftarrow 0$ **to** S **do**
  $j \leftarrow isCheckCircuit_i$
  $correct \leftarrow (recievedGen_{i,j} == recievedEvl_i)$
  if *!correct* **then**
    abort()

The cloud and generator perform an oblivious transfer where the generator offers up decryption keys for his input and decryption keys for the circuit seed for each circuit. The cloud can select the key to decrypt the generator's input or the key to decrypt the circuit seed for a circuit but not both. For each circuit, if the cloud selects the decryption key for the circuit seed in the oblivious transfer, then the circuit is used as a check circuit.

$$selectedKeys = OT(circuitSelection, \{evlKey, checkKey\})$$

If the cloud selects the key for the generator's input then a given circuit is used as an evaluation circuit. Otherwise, the key for the circuit seed was selected and the circuit is a check circuit. The decryption keys are saved by both the generator and cloud in the event a computation uses saved values from this computation.

The generator sends the encrypted garbled inputs and check circuit information for all circuits to the cloud. The cloud decrypts the information he can decrypt using its keys.

The evaluator must also learn the circuit split. The generator sends a hash of each possible encryption key the cloud could have selected to the evaluator for each circuit as an ordered pair. For $0 \leq i < S$,

$$genSend(hash(checkKey_i), hash(evaluationKey_i))$$

The cloud sends a hash of the value received to the evaluator for each circuit. The cloud also sends bits to indicate which circuits were selected as check and evaluation circuits to the evaluator. For $0 \leq i < S$,

$$cloudSend(hash(selectedKey_i), isCheckCircuit_i)$$

The evaluator compares each hash the cloud sent to one of the hashes the generator sent, which is selected by the circuit selection sent by the cloud. For $0 \leq i < S$,

$$j = isCheckCircuit_i$$
$$correct = (receivedGen_{i,j} == receivedEvl_i)$$

If all values match, the evaluator uses the $isCheckCircuit_i$ to learn the split between check and evaluator circuits. Otherwise the evaluator safely aborts.

We only perform the cut-and-choose oblivious transfer for the initial computation. For any subsequent computations, the generator and evaluator hash the saved decryption keys and use those hashes as the new encryption and decryption keys. The circuit split selected by the cloud is saved and stays the same across computations.

Phase 2: Oblivious Transfer

Algorithm 2: Evaluator_Input

Input : Eval_Select_Bits, Possible_Eval_Input
Output: Eval_Garbled_Input
// cloud gets selected input wires // generator offers both possible input wire values for each input wire; evaluator selects its input
$outSeeds = BaseOOT(bitsEvl, possibleInputs)$.
// the generator sends unique IKey values for each circuit to the evaluator
for $i \leftarrow 0$ **to** S **do**
  $genSendToEval(IKey_i)$
// the evaluator sends IKey values for all evaluation circuits to the cloud
for $i \leftarrow 0$ **to** S **do**
  if *!isCheckCircuit(i)* **then**
    $EvalSendToCloud(IKey_i)$
// cloud uses this to learn appropriate inputs
for $i \leftarrow 0$ **to** S **do**
  for $j \leftarrow 0$ **to** $len(evlInputs)$ **do**
    if *!isCheckCircuit(i)* **then**
      $inputEvl_{ij} \leftarrow hash(IKeys_i, outSeeds_j)$
return inputEvl

We use the base outsourced oblivious transfer (OOT) of CMTB. In this transfer the generator inputs both possible input wire values for each evaluator's input wire while the evaluator inputs its own input. After the OOT is performed, the cloud has the selected input wire values, which represent the evaluator's input.

As with CMTB, which uses the results from a single OOT as seeds to create the evaluator's input for all circuits, the cloud in our system also uses seeds from a single base OT (called "BaseOOT" below) to generate the input for the evaluation circuits. The cloud receives the seeds for each input bit selected by the evaluator.

$$outSeeds = BaseOOT(evlInputSeeds, evlInput).$$

The generator creates unique keys, $IKey$, for each circuit and sends each key to the evaluator. The evaluator sends the keys for the evaluation circuits to the cloud. The cloud then uses these values to attain the evaluator's input. For $0 \leq i < S$, for $0 \leq j < len(evlInputs)$ where $!isCheckCircuit(i)$,

$$inputEvl_{ij} = hash(IKey_i, outSeeds_j)$$

Phase 3: Generator's Input Consistency Check

Algorithm 3: Generator_Input_Check

```
Input  : Generator_Input
// The cloud takes a hash of the generator's input or each
evaluation circuit for i ← 0 to S do
    if isCheckCircuit(i) then
        t_i ← UHF(garbledGenInput_i)
//If a single hash is different then the cloud knows the generator
tried to cheat.
correct ← ((t_0 == t_1)&(t_0 == t_2)& ... &(t_0 == t_{N-1}))
if !correct then
    abort()
```

We use the input consistency check of sS13. In this check, a universal hash is used to prove consistency of the generator's input across each evaluation circuit. Simply put, if the hash is different in any of the evaluation circuits, we know the generator did not enter consistent input. More formally, a hash of the generator's input is taken for each circuit. For $0 < i < S$ where $!isCheckCircuit(i)$,

$$t_i = UHF(garbledGenInput_i, C_i)$$

The results of these universal hashes are compared. If a single hash is different then the cloud knows the generator tried to cheat and safely aborts.

$$correct = ((t_0 == t_1)\&(t_0 == t_2)\& \ldots \&(t_0 == t_{N-1}))$$

Phase 4: Partial Input Gate Generation, Check, and Evaluation

Generation

For $0 \leq i < S$, for $0 \leq j < len(savedWires)$, the generator creates a *partial input gate*, which transforms a wire's saved values, $POut0_{i,j}$ and $POut1_{i,j}$, into wire values that can be used in the current garbled circuit execution, $GIn0_{i,j}$ and $GIn1_{i,j}$. For each circuit, C_i, the generator creates a pseudorandom transformation value R_i, to assist with the transformation.

For each set of $POut0_{i,j}$ and $POut1_{i,j}$, the generator XORs each value with R_i. Both results are then hashed, and put through a function to determine the new permutation bit, as hashing removes the old permutation bit.

$$t0 = hash(POut0_{i,j} \oplus R_i)$$
$$t1 = hash(POut1_{i,j} \oplus R_i)$$
$$PIn0_{i,j}, PIn1_{i,j} = setPPBitGen(t0, t1)$$

This function, *setPPBitGen*, pseudo-randomly finds a bit that is different between the two values of the wire and notes that bit to be the permutation bit. *setPPBitGen* is seeded from $CSeed_i$, allowing the cloud to regenerate these values for the check circuits.

For each $PIn0_{i,j}, PIn1_{i,j}$ pair, a set of values, $GIn0_{i,j}$ and $GIn1_{i,j}$, are created under the master key of C_i, $CKey_i$, – where $CKey_i$ is the difference between 0 and 1 wire labels for the circuit. In classic garbled gate style, two truth table values, $TT0_{i,j}$ and $TT1_{i,j}$, are created such that:

$$TT0_{i,j} \oplus PIn0_{i,j} = GIn0_{i,j}$$
$$TT1_{i,j} \oplus PIn1_{i,j} = GIn1_{i,j}$$

The truth table, $TT0_{i,j}$ and $TT1_{i,j}$, is permuted so that the permutation bits of $PIn0_{i,j}$ and $PIn1_{i,j}$ tell the cloud which entry to select. Each *partial input gate*, consisting of the permuted $TT0_{i,j}$, $TT1_{i,j}$ values and the bit location from *setPPBitGen* is sent to the cloud. Each R_i is also sent to the cloud.

Algorithm 4: Partial_Input

```
Input  : Partial_Output
Output: Partial_Garbled_Input
// Generation: the generator creates a partial input gate, which
transforms a wire's saved values, POut0_{i,j} and POut1_{i,j}, into
values that can be used in the current garbled circuit execution,
GIn0_{i,j} and GIn1_{i,j}.
for i ← 0 to S do
    R_i ← PRNG.random()
    for j ← 0 to len(savedWires) do
        t0 ← hash(POut0_{i,j} ⊕ R_i)
        t1 ← hash(POut1_{i,j} ⊕ R_i)
        PIn0_{i,j}, PIn1_{i,j} ← setPPBitGen(t0, t1)
        GIn0_{i,j} ← TT0_{i,j} ⊕ PIn0_{i,j}
        GIn1_{i,j} ← TT1_{i,j} ⊕ PIn1_{i,j}
        GenSendToCloud( Permute([TT0_{i,j}, TT1_{i,j}]),
        permute_bit_locations )
    GenSendToCloud(R_i)
// Check: The cloud checks the gates to make sure the generator
didn't cheat
for i ← 0 to S do
    if isCheckCircuit(i) then
        for j ← 0 to len(savedWires) do
            // the cloud has received the truth table
            information, TT0_{i,j}, TT1_{i,j}, bit locations from
            setPPBitGen, and R_i
            correct ← (generateGateFromInfo() ==
            receivedGateFromGen())
            // If any gate does not match, the cloud knows the
            generator tried to cheat.
            if !correct then
                abort();
// Evaluation
for i ← 0 to S do
    if !isCheckCircuit(i) then
        for j ← 0 to len(savedWires) do
            //The cloud, using the previously saved POutx_{i,j}
            value, and the location (point and permute) bit sent
            by the generator, creates PInx_{i,j}
            PInx_{i,j} ←
            setPPBitEval(hash(R_i ⊕ POutx_{i,j}), location)
            // Using PInx_{i,j}, the cloud selects the proper
            truth table entry TTx_{i,j} from either TT0_{i,j} or
            TT1_{i,j} to decrypt
            // Creates GInx_{i,j} to enter into the garbled circuit
            GInx_{i,j} ← TTx_{i,j} ⊕ POutx_{i,j}
return GIn;
```

Check

For $0 \leq i < S$ where $isCheckCircuit(i)$, for $0 \leq j < len(savedWires)$, the cloud receives the truth table information, $TT0_{i,j}, TT1_{i,j}$, and bit location from *setPPBitGen*, and proceeds to regenerate the gates based on the check circuit information. The cloud uses R_i (sent by the generator), $POut0_{i,j}$ and $POut1_{i,j}$ (saved during the previous execution), and $CSeed_i$ (recovered during the cut-and-choose) to generate the *partial input gates* in the same manner as described previously. The cloud then compares these gates to those the generator sent. If any gate does not match, the cloud knows the generator tried to cheat and safely aborts.

Evaluation

For $0 \leq i < S$ where $!isCheckCircuit(i)$, for $0 \leq j < len(savedWires)$ the cloud receives the truth table information, $TTa_{i,j}, TTb_{i,j}$ and bit location from *setPPBitGen*. a and b are used to denote the two permuted truth table values. The cloud, using the previously saved $POutx_{i,j}$ value, creates the $PInx_{i,j}$ value:

$$PInx_{i,j} = setPPBitEval(hash(R_i \oplus POutx_{i,j}), location)$$

location is the location of the point and permute bit sent by the generator. Using the point and permute bit of $PInx_{i,j}$, the cloud selects the proper truth table entry $TTx_{i,j}$ from either $TTa_{i,j}$ or $TTb_{i,j}$ to decrypt, creates $GInx_{i,j}$ and then enters $GInx_{i,j}$ into the garbled circuit.

$$GInx_{i,j} = TTx_{i,j} \oplus POutx_{i,j}$$

Phase 5: Circuit Generation and Evaluation

Algorithm 5: Circuit_Execution

```
Input  : Generator_Input, Evaluator_Input, Partial_Input
Output: Partial_Output, Garbled_Output
// The generator generates each garbled gate and sends it to the
cloud. Depending on whether the circuit is a check or evaluation
circuit, the cloud verifies that the gate is correct or evaluates the
gate.
for i ← 0 to S do
    for j ← 0 to len(circuit) do
        g ← genGate(C_i, j)
        send(g)
// the cloud receives all gates for all circuits, and then checks
OR evaluates each circuit
for i ← 0 to S do
    for j ← 0 to len(circuit) do
        g ← recvGate()
        if isCheckCircuit(i) then
            if ! verifyCorrect(g) then
                abort()
        else
            eval(g)
return Partial_Output, Garbled_Output
```

Circuit Generation

The generator generates each garbled gate for each circuit and sends them to the cloud. Since the generator does not know the check and evaluation circuit split, nothing changes for the generation for check and evaluation circuits. For $0 \leq i < S$, For $0 \leq j < len(circuit)$,

$$g = genGate(C_i, j), send(g)$$

Circuit Evaluation and Check

The cloud receives each garbled gate for all circuits. For evaluation circuits the cloud evaluates those garbled gates. For check circuits the cloud generates the correct gate, based on the circuit seed, and is able to verify it is correct.

For $0 \leq i < S$, For $0 \leq j < len(circuit)$, $g = recvGate()$, $if(isCheckCircuit(j))$ $verifyCorrect(g)$ $else$ $eval(g)$

If a garbled gate is found not to be correct, the cloud informs the evaluator and generator of the incorrect gate and safely aborts.

Phase 6: Output and Output Consistency Check

Algorithm 6: Circuit_Output

```
Input  : Garbled_Output
// a MAC of the output is generated inside the garbled circuit,
and both the resulting garbled circuit output and the MAC are
encrypted under a one-time pad.
outEvlComplete = outEvl||MAC(outEvl)
result = (outEvlMAC == MAC(outEvl))
if !result then
    abort() // output check fail
```

As the final step of the garbled circuit execution, a MAC of the output is generated inside the garbled circuit, based on a k-bit secret key entered into the function.

$$outEvlComplete = outEvl||MAC(outEvl)$$

Both the resulting garbled circuit output and the MAC are encrypted under a one-time pad. The generator can also have output verified in the same manner. The cloud sends the corresponding encrypted output to each party.

The generator and evaluator then decrypt the received ciphertext, perform a MAC over real output, and verify the cloud did not modify the output by comparing the generated MAC with the MAC calculated within the garbled circuit.

$$result = (outEvlMAC == MAC(outEvl))$$

Phase 7: Partial Output

Algorithm 7: Partial_Output

```
Input  : Partial_Output
for i ← 0 to S do
    for j ← 0 to len(Partial_Output) do
        //The generator saves both possible wire values
        GenSave(Partial_Output0_{i,j})
        GenSave(Partial_Output1_{i,j})
for i ← 0 to S do
    for j ← 0 to len(Partial_Output) do
        if isCheckCircuit(i) then
            EvlSave(Partial_Output0_{i,j})
            EvlSave(Partial_Output1_{i,j})
        else
            // circuit is evaluation circuit
            EvlSave(Partial_OutputX_{i,j})
```

The generator saves both possible wire values for each partial output wire. For each evaluation circuit the cloud saves the partial output wire value. For check circuits the cloud saves both possible output values.

4.3 Implementation

As with most garbled circuit systems there are two stages to our implementation. The first stage is a compiler for creating garbled circuits, while the second stage is an execution system to evaluate the circuits.

We modified the KSS12 [27] compiler to allow for the saving of intermediate wire labels and loading wire labels from a different SFE computation. By using the KSS12 compiler, we have an added benefit of being able to compare circuits of almost identical size and functionality between our system and CMTB, whereas other protocols compare circuits of sometimes vastly different sizes.

For our execution system, we started with the CMTB system and modified it according to our protocol requirements. PartialGC automatically performs the output consistency check, and we implemented this check at the circuit level. We became aware and corrected issues with CMTB relating to too many primitive OT operations performed in the outsourced oblivious transfer when using a high circuit parameter and too low a general security parameter in general. The fixes reduced the run-time of the OOT.

5. SECURITY OF PARTIALGC

In this section, we provide a basic proof sketch of the PartialGC protocol, showing that our protocol preserves the standard security guarantees provided by traditional garbled circuits - that is, none of the parties learns anything about the private inputs of the other parties that is not logically implied by the output it receives. Since we borrow heavily from [8] and [38], we focus on our additions, and defer to the original papers for detailed proofs of those protocols. Due to

space constraints, we do not provide a formal proof here; a complete proof will be provided in the technical report.

We know that the protocol described in [8] allows us to garble individual circuits and securely outsource their evaluation. In this paper, we modify certain portions of the protocol to allow us to transform the output wire values from a previous circuit execution into input wire values in a new circuit execution. These transformed values, which can be checked by the evaluator, are created by the generator using circuit "seeds."

We also use some aspects of [38], notably their novel cut-and-choose technique which ensures that the generator does not learn which circuits are used for evaluation and which are used for checking - this means that the generator must create the correct transformation values for all of the cut-and-choose circuits.

Because we assume that the CMTB garbled circuit scheme can securely garble any circuit, we can use it individually on the circuit used in the first execution and on the circuits used in subsequent executions. We focus on the changes made at the end of the first execution and the beginning of subsequent executions which are introduced by PartialGC.

The only difference between the initial garbled circuit execution and any other garbled circuit in CMTB is that the output wires in an initial PartialGC circuit are stored by the cloud, and are not delivered to the generator or the evaluator. This prevents them from learning the output wire labels of the initial circuit, but cannot be less secure than CMTB, since no additional steps are taken here.

Subsequent circuits we wish to garble differ from ordinary CMTB garbled circuits only by the addition, before the first row of gates, of a set of partial input gates. These gates don't change the output along a wire, but differ from normal garbled gates in that the two possible labels for each input wire are not chosen randomly by the generator, but are derived by using the two labels along each output wire of the initial garbled circuit.

This does not reduce security. In PartialGC, the input labels for partial input gates have the same property as the labels for ordinary garbled input gates: the generator knows both labels, but does not know which one corresponds to the evaluator's input, and the evaluator knows only the label corresponding to its input, but not the other label. This is because the evaluator's input is exactly the output of the initial garbled circuit, the output labels of which were saved by the evaluator. The evaluator does not learn the other output label for any of the output gates because the output of each garbled gate is encrypted. If the evaluator could learn any output labels other than those which result from an evaluation of the garbled circuit, the original garbled circuit scheme itself would not be secure.

The generator, which also generated the initial garbled circuit, knows both possible input labels for all partial evaluation gates, because it has saved both potential output labels of the initial circuit's output gates. Because of the outsourced oblivious transfer used in CMTB, the generator did not know which input labels to use for the initial garbled circuit, and therefore will not have been able to determine the output labels for that circuit. Therefore, the generator will likewise not know which input labels are being used for subsequent garbled circuits.

Generator's Input Consistency Check

We use the generator's input consistency check from sS13. We note there is no problem with allowing the cloud to perform this check; for the generator's inconsistent input to pass the check, the cloud would have to see the malicious input and ignore it, which would violate the non-collusion assumption.

Correctness of Saved Values

Scenarios where either party enters incorrect values in the next computation reduce to previously solved problems in garbled circuits. If the generator does not use the correct values, then it reduces to the problem of creating an incorrect garbled circuit. If the evaluator does not use the correct saved values then it reduces to the problem of the evaluator entering garbage values into the garbled circuit execution; this would be caught by the output consistency check.

Abort on Check Failure

If any of the check circuits fail, the cloud reports the incorrect check circuit to both the generator and evaluator. At this point, the remaining computation and any saved values must be abandoned. However, as is standard in SFE, the cloud cannot abort on an incorrect evaluation circuit, even when she knows that it is incorrect.

Concatenation of Incorrect Circuits

If the generator produces a single incorrect circuit and the cloud does not abort, the generator learns that the circuit was used for evaluation, and not as a check circuit. This leaks no information about the input or output of the computation; to do that, the generator must corrupt a majority of the evaluation circuits without modifying a check circuit. An incorrect circuit that goes undetected in one execution has no effect on subsequent executions as long the total amount of incorrect circuits is less than the majority of evaluation circuits.

Using Multiple Evaluators

One of the benefits of our outsourcing scheme is that the state is saved at the generator and cloud allowing the use of different evaluators in each computation. Previously, it was shown a group of users working with a single server using 2P-SFE was not secure against malicious adversaries, as a malicious server and last k parties, also malicious, could replay their portion of the computation with different inputs and gain more information than they can with a single computation [15]. However, this is not a problem in our system as at least one of our servers, either the generator or cloud, must be semi-honest due to non-collusion, which obviates the attack stated above.

Threat Model

As we have many computations involving the same generator and cloud, we have to extend the threat model for how the parties can act in different computations. There can be no collusion in each singular computation. However, the malicious party can change between computations as long as there is no chain of malicious users that link the generator and cloud – this would break the non-collusion assumption.

6. PERFORMANCE EVALUATION

We now demonstrate the efficacy of PartialGC through a comparison with the CMTB outsourcing system. Apart from the cut-and-choose from sS13, PartialGC provides other benefits through generating partial input values after the first execution of a program. On subsequent executions, the partial inputs act to amortize overall costs of execution and bandwidth.

We demonstrate that the evaluator in the system can be a

mobile device outsourcing computation to a more powerful system. We also show that other devices, such as server-class machines, can act as evaluators, to show the generality of this system. Our testing environment includes a 64-core server containing 1 TB of RAM, which we use to model both the Generator and Outsourcing Proxy parties. We run separate programs for the Generator and Outsourcing Proxy, giving them each 32 threads. For the evaluator, we use a Samsung Galaxy Nexus phone with a 1.2 GHz dual-core ARM Cortex-A9 and 1 GB of RAM running Android 4.0, connected to the server through an 802.11 54 Mbps WiFi in an isolated environment. In our tests, which outsource the computation from a single server process we create that process on our 64-core server as well. We ran the CMTB implementation for comparison tests under the same setup.

6.1 Execution Time

The PartialGC system is particularly well suited to complex computations that require multiple stages and the saving of intermediate state. Previous garbled circuit execution systems have focused on single-transaction evaluations, such as computing the "millionaires" problem (i.e., a joint evaluation of which party inputs a greater value without revealing the values of the inputs) or evaluating an AES circuit.

Our evaluation considers two comparisons: the improvement of our system compared with CMTB without reusing saved values, and comparing our protocol for saving and reusing values against CMTB if such reuse was implemented in that protocol. We also benchmark the overhead for saving and loading values on a per-bit basis for 256 circuits, a necessary number to achieve a security parameter of 2^{-80} in the malicious model. In all cases, we run 10 iterations of each test and give timing results with 95% confidence intervals. Other than varying the number of circuits our system parameters are set for 80-bit security.

The programs used for our evaluation are exemplars of differing input sizes and differing circuit complexities:

Keyed Database: In this program, one party enters a database and keys to it while the other party enters a key that indexes into the database, receiving a database entry for that key. This is an example of a program expressed as a small circuit that has a very large amount of input.

Matrix Multiplication: Here, both parties enter 32-bit numbers to fill a matrix. Matrix multiplication is performed before the resulting matrix is output to both parties. This is an example of a program with a large amount of inputs with a large circuit.

Edit (Levenstein) Distance: This program finds the distance between two strings of the same length and returns the difference. This is an example of a program with a small number of inputs and a medium sized circuit.

Millionaires: In this classic SFE program, both parties enter a value, and the result is a one-bit output to each party to let them know whether their value is greater or smaller than that of the other party. This is an example of a small circuit with a large amount of input.

Gate counts for each of our programs can be found in Table 1. The only difference for the programs described above is the additional of a MAC function in PartialGC. We discuss the reason for this check in Section 6.4.

Table 2 shows the results from our experimental tests. In the best case, execution time was reduced by a factor of 32 over CMTB, from 1200 seconds to 38 seconds, a 96% speedup over CMTB. Ultimately, our results show that our system outperforms CMTB when the input checks are the bottleneck. This run-time improvement is due to improvements we added from sS13 and occurs in the keyed database, millionaires, and matrix multiplications programs. In the other program, edit distance, the input checks are not the bottleneck and PartialGC does not outperform CMTB. The total run-time increase for the edit distance problem is due to overhead of using the new sS13 OT cut-and-choose technique which requires sending each gate to the evaluator for check circuits and evaluation circuits. This is discussed further in Section 6.4. The typical use case we imagine for our system, however, is more like the keyed database program, which has a large amount of inputs and a very small circuit. We expand upon this use case later in this section.

	CMTB	PartialGC
KeyedDB 64	6,080	20,891
KeyedDB 128	12,160	26,971
KeyedDB 256	24,320	39,131
MatrixMult8x8	3,060,802	3,305,113
Edit Distance 128	1,434,888	1,464,490
Millionaires 8192	49,153	78,775
LCS Incremental 128	4,053,870	87,236
LCS Incremental 256	8,077,676	160,322
LCS Incremental 512	16,125,291	306,368
LCS Full 128	2,978,854	-
LCS Full 256	13,177,739	-

Table 1: Non-XOR gate counts for the various circuits. In the first 6 circuits, the difference between CMTB and PartialGC gate counts is in the consistency checks. The explanation for the difference in size between the incremental versions of longest common substring (LCS) is given in *Reusing Values*.

Reusing Values

For a test of our system's wire saving capabilities we tested a dynamic programming problem, longest common substring, in both PartialGC and CMTB. This program determines the length of the longest common substring between two strings. Rather than use a single computation for the solution, our version incrementally adds a single bit of input to both strings each time the computation is run and outputs the results each time to the evaluator. We believe this is a realistic comparison to a real-world application that incrementally adds data during each computation where it is faster to save the intermediate state and add to it after seeing an intermediate result than rerun the entire computation many times after seeing the result.

For our testing, PartialGC uses our technique to reuse wire values. In CMTB, we save each desired internal bit under a one-time pad and re-enter them into the next computation, as well as the information needed to decrypt the ciphertext. We use a MAC (the AES circuit of KSS12) to verify that the party saving the output bits did not modify them. We also use AES to generate a one-time pad inside the garbled circuit. We use AES as this is the only cryptographically secure function used in CMTB. Both parties enter private keys to the MAC functions. This program is labeled *CMTB-Inc*, for CMTB *incremental*. The size of this program represents the size of the total strings. We also created a circuit that computes the complete longest common substring in one computation labeled *CMTB-Full*.

The resulting size of the PartialGC and CMTB circuits are shown in Table 1, and the results are shown in Figure 4. This result shows that saving and reusing values in PartialGC is more efficient than completely rerunning the com-

	16 Circuits			64 Circuits			256 Circuits		
	CMTB	PartialGC		CMTB	PartialGC		CMTB	PartialGC	
KeyedDB 64	18 ± 2%	3.5 ± 3%	5.1x	72 ± 2%	8.3 ± 5%	8.7x	290 ± 2%	26 ± 2%	11x
KeyedDB 128	33 ± 2%	4.4 ± 8%	7.5x	140 ± 2%	9.5 ± 4%	15x	580 ± 2%	31 ± 3%	19x
KeyedDB 256	65 ± 2%	4.6 ± 2%	14x	270 ± 1%	12 ± 6%	23x	1200 ± 3%	38 ± 5%	32x
MatrixMult8x8	48 ± 4%	46 ± 4%	1.0x	110 ± 8%	100 ± 7%	1.1x	400 ± 10%	370 ± 5%	1.1x
Edit Distance 128	21 ± 6%	22 ± 3%	0.95x	47 ± 7%	50 ± 9%	0.94x	120 ± 9%	180 ± 6%	0.67x
Millionaires 8192	35 ± 3%	7.3 ± 6%	4.8x	140 ± 2%	20 ± 2%	7.0x	580 ± 1%	70 ± 2%	8.3x

Table 2: Timing results comparing PartialGC to CMTB without saving any values. All times in seconds.

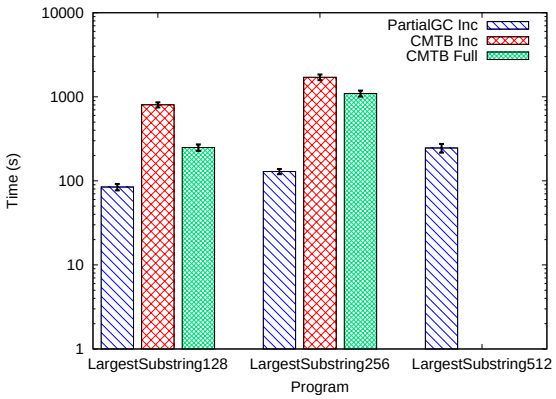


Figure 4: Results from testing our largest common substring (LCS) programs for PartialGC and CMTB. This shows when changing a single input value is more efficient under PartialGC than either CMTB program. CMTB crashed on running LCS Incremental of size 512 due to memory requirements. We were unable to complete the compilation of CMTB Full of size 512.

putation. The input consistency check adds considerably to the memory use on the phone for *CMTB-Inc* and in the case of input bit 512, the *CMTB-Inc* program will not complete. In the case of the 512-bit *CMTB-Full*, the program would not complete compilation in over 42 hours. In our *CMTB-Inc* program, we assume the cloud saves the output bits so that multiple phones can have a shared private key. We do not provide a full program due to space requirements.

Note that the growth of *CMTB-Inc* and *CMTB-Full* are different. *CMTB-Full* grows at a larger rate (4x for each 2x factor increase) than *CMTB-Inc* (2x for each 2x factor increase), implying that although at first it seems more efficient to rerun the program if small changes are desired in the input, eventually this will not be the case. Even with a more efficient AES function, *CMTB-Inc* would not be faster as the bottleneck is the input, not the size of the circuit.

The overhead of saving and reusing values is discussed further in Appendix B.

Outsourcing to a Server Process

PartialGC can be used in other scenarios than just outsourcing to a mobile device. It can outsource garbled circuit evaluation from a single server process and retain performance benefits over a single server process of CMTB. For this experiment the outsourcing party has a single thread. Table 4 displays these results and shows that in the KeyedDB 256 program, PartialGC has a 92% speedup over CMTB. As with the outsourced mobile case, keyed database problems perform particularly well in PartialGC. Because the computationally-intensive input consistency check is a greater bottleneck on mobile devices than servers, these improvements for most programs are less dramatic. In particular,

	256 Circuits		
	CMTB	PartialGC	
KeyedDB 64	64992308	3590416	18x
KeyedDB 128	129744948	3590416	36x
KeyedDB 256	259250228	3590416	72x
MatrixMult8x8	71238860	35027980	2.0x
Edit Distance 128	2615651	4108045	0.64x
Millionaires 8192	155377267	67071757	2.3x

Table 3: Bandwidth comparison of CMTB and PartialGC. Bandwidth counted by instrumenting PartialGC to count the bytes it was sending and receiving and then adding them together. Results in bytes.

both edit distance and matrix multiplication programs benefit from higher computational power and their bottlenecks on a server are no longer input consistency; as a result, they execute faster in CMTB than in PartialGC.

6.2 Bandwidth

Since the main reason for outsourcing a computation is to save on resources, we give results showing a decrease in the evaluator's bandwidth. Bandwidth is counted by making the evaluator to count the number of bytes PartialGC sends and receives to either server. Our best result gives a 98% reduction in bandwidth (see Table 3). For the edit distance, the extra bandwidth used in the outsourced oblivious transfer for all circuits, instead of only the evaluation circuits, exceeds any benefit we would otherwise have received.

6.3 Secure Friend Finder

Many privacy-preserving applications can benefit from using PartialGC to cache values for state. As a case study, we developed a privacy-preserving friend finder application, where users can locate nearby friends without any user divulging their exact location. In this application, many different mobile phone clients use a consistent generator (a server application) and outsource computation to a cloud. The generator must be the same for all computations; the cloud must be the same for each computation. The cloud and generator are two different parties. After each computation, the map is updated when PartialGC saves the current state of the map as wire labels. Without PartialGC outsourcing values to the cloud, the wire labels would have to be transferred directly between mobile devices, making a multi-user application difficult or impossible.

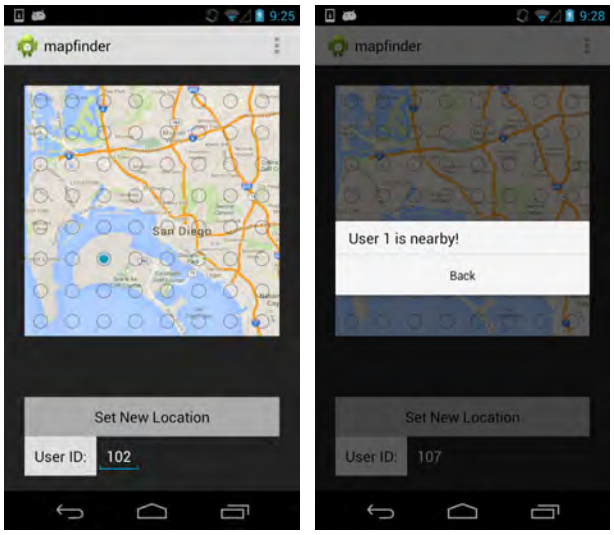
We define three privacy-preserving operations that comprise the application's functionality:

MapStart - The three parties (generator, evaluator, cloud) create a "blank" map region, where all locations in the map are blank and remain that way until some mobile party sets a location to his or her ID.

MapSet - The mobile party sets a single map cell to a new value. This program takes in partial values from the generator and cloud and outputs a location selected by the

	16 Circuits			64 Circuits			256 Circuits		
	CMTB	PartialGC		CMTB	PartialGC		CMTB	PartialGC	
KeyedDB 64	6.6 ± 4%	1.4 ± 1%	4.7x	27 ± 4%	5.1 ± 2%	5.3x	110 ± 2%	24.9 ± 0.3%	4.4x
KeyedDB 128	13 ± 3%	1.8 ± 2%	7.2x	54 ± 4%	5.8 ± 2%	9.3x	220 ± 5%	27.9 ± 0.5%	7.9x
KeyedDB 256	25 ± 4%	2.5 ± 1%	10x	110 ± 7%	7.3 ± 2%	15x	420 ± 4%	33.5 ± 0.6%	13x
MatrixMult8x8	42 ± 3%	41 ± 4%	1.0x	94 ± 4%	79 ± 3%	1.2x	300 ± 10%	310 ± 1%	0.97x
Edit Distance 128	18 ± 3%	18 ± 3%	1.0x	40 ± 8%	40 ± 6%	1.0x	120 ± 9%	150 ± 3%	0.8x
Millionaires 8192	13 ± 4%	3.2 ± 1%	4.1x	52 ± 3%	8.5 ± 2%	6.1x	220 ± 5%	38.4 ± 0.9%	5.7x

Table 4: Timing results from outsourcing the garbled circuit evaluation from a single server process. Results in seconds.



(a) Location selected. (b) After computation.

Figure 5: Screenshots from our application. (a) shows the map with radio buttons a user can select to indicate position. (b) show the result after "set new position" is pressed when a user is present. The application is set to use 64 different map locations. Map image from Google Maps.

mobile party.

MapGet - The mobile party retrieves the contents of a single map cell. This program retrieves partial values from the generator and cloud and outputs any ID set for that cell to the mobile.

In the application, each user using the *Secure Friend Finder* has a unique ID that represents them on the map. We divide the map into 'cells', where each cell is a set amount of area. When the user presses "Set New Location', the program will first look to determine if that cell is occupied. If the cell is occupied, the user is informed he is near a friend. Otherwise the cell is updated to contain his user ID and remove his ID from his previous location. We assume a maximum of 255 friends in our application since each cell in the map is 8 bits.

Figure 6 shows the performance of these programs in the malicious model with a 2^{-80} security parameter (evaluated over 256 circuits). We consider map regions containing both 256 and 2048 cells. For maps of 256 cells, each operation takes about 30 seconds.[1] As there are three operations for each "Set New Location" event, the total execution time is about 90 seconds, while execution time for 2048 cells is about 3 minutes. The bottleneck of the 64 and 256 cell maps is the outsourced oblivious transfer, which is not affected by the number of cells in the map. The vastly larger circuit associated with the 2048-cell map makes getting and setting values slower operations, but these results show such an application is practical for many scenarios.

[1]Our 64-cell map, as seen the application screenshots, also takes about 30 seconds for each operation.

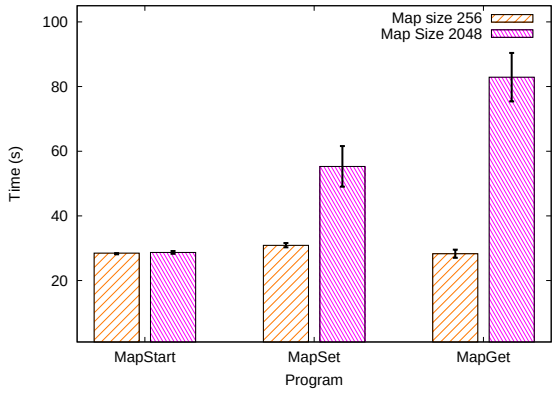


Figure 6: Run time comparison of our map programs with two different map sizes.

Example - As an example, two friends initiate a friend finder computation using Amazon as the cloud and Facebook as the generator. The first friend goes out for a coffee at a café. The second friend, riding his bike, gets a message that his friend is nearby and looks for a few minutes and finds him in the café. Using this application prevents either Amazon or Facebook from knowing either user's location while they are able to learn whether they are nearby.

6.4 Discussion

Analysis of improvements

We analyzed our results and found the improvements came from three places: the improved sS13 consistency check, the saving and reusing of values, and the fixed oblivious transfer. In the case of the sS13 consistency check, there are two reasons for the improvement, first there is less network traffic and second it does not use exponentiations. In the case of saving and reusing values, we save time by the faster input consistency check and not requiring a user to recompute a circuit multiple times. Lastly, we reduced the runtime and bandwidth by fixing parts of the OOT. The previous outsourced oblivious transfer performed the primitive OT S times instead of a single time, which turn forced many extra exponentiations. Each amount of improvement varies depending upon the circuit.

Output check

Although the garbled circuit is larger for our output check, this check performs less cryptographic operations for the outsourcing party, as the evaluator only has to perform a MAC on the output of the garbled circuit. We use this check to demonstrate using a MAC can be an efficient output check for a low power device when the computational power is not equivalent across all parties.

Commit Cut-and-Choose vs OT Cut-and-Choose

Our results unexpectedly showed that the sS13 OT cut-and-choose used in PartialGC is actually slower than the KSS12 commit cut-and-choose used in CMTB in our experimental setup. Theoretically, sS13, which requires fewer cryptographic operations, as it generates the garbled circuit only once, should be the faster protocol. The difference between the two cut-and-choose protocols is the network usage – instead of $\frac{2}{5}$ of the circuits (CMTB), *all* the circuits must be transmitted in sS13. The sS13 cut-and-choose is required in our protocol so that the cloud can check that the generator creates the correct gates.

7. RELATED WORK

SFE was first described by Yao in his seminal paper [39] on the subject. The first general purpose platform for SFE, Fairplay [32], was created in 2004. Fairplay had both a compiler for creating garbled circuits, and a run-time system for executing them. Computations involving three or more parties have also been examined; one of the earliest examples is FairplayMP [2]. There have been multiple other implementations since, in both semi-honest [6, 9, 16, 17, 40] and malicious settings [26, 37].

Optimizations for garbled circuits include the free-XOR technique [25], garbled row reduction [36], rewriting computations to minimize SFE [23], and pipelining [18]. Pipelining allows the evaluator to proceed with the computation while the generator is creating gates.

KSS12 [27] included both an optimizing compiler and an efficient run-time system using a parallelized implementation of SFE in the malicious model from [37].

The creation of circuits for SFE in a fast and efficient manner is one of the central problems in the area. Previous compilers, from Fairplay to KSS12, were based on the concept of creating a complete circuit and then optimizing it. PAL [33] improved such systems by using a simple template circuit, reducing memory usage by orders of magnitude. PCF [26] built from this and used a more advanced representation to reduce the disk space used.

Other methods for performing MPC involve homomorphic encryption [3, 12], secret sharing [4], and ordered binary decision diagrams [28]. A general privacy-preserving computation protocol that uses homomorphic encryption and was designed specifically for mobile devices can be found in [7]. There are also custom protocols designed for particular privacy-preserving computations; for example, Kamara et al. [21] showed how to scale server-aided Private Set Intersection to billion-element sets with a custom protocol.

Previous reusable garbled-circuit schemes include that of Brandão [5], which uses homomorphic encryption, Gentry *et al.* [10], which uses attribute-based functional encryption, and Goldwasser *et al.* [13], which introduces a succinct functional encryption scheme. These previous works are purely theoretical; none of them provides experimental performance analysis. There is also recent theoretical work on reusing encrypted garbled-circuit values [30, 11, 31] in the ORAM model; it uses a variety of techniques, including garbled circuits and identity-based encryption, to execute the underlying low-level operations (program state, read/write queries, etc.). Our scheme for reusing encrypted values is based on completely different techniques; it enables us to do new kinds of computations, thus expanding the set of things that can be computed using garbled circuits.

The Quid-Pro-Quo-tocols system [19] allows fast execution with a single bit of leakage. The garbled circuit is executed twice, with the parties switching roles in the latter execution, then running a secure protocol to ensure that the output from both executions are equivalent; if this fails, a single bit may be leaked due to the selective failure attack.

8. CONCLUSION

This paper presents PartialGC, a server-aided SFE scheme allowing the reuse of encrypted values to save the costs of input validation and to allow for the saving of state, such that the costs of multiple computations may be amortized. Compared to the server-aided outsourcing scheme by CMTB, we reduce costs of computation by up to 96% and bandwidth costs by up to 98%. Future work will consider the generality of the encryption re-use scheme to other SFE evaluation systems and large-scale systems problems that benefit from the addition of state, which can open up new and intriguing ways of bringing SFE into the practical realm.

Acknowledgements: This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory under contracts FA8750-11-2-0211 and FA8750-13-2-0058. It is also supported in part by the U.S. National Science Foundation under grant numbers CNS-1118046 and CNS-1254198. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, NSF, or the U.S. Government.

9. REFERENCES

[1] M. Bellare and S. Micali. Non-Interactive Oblivious Transfer and Applications. In *Proceedings of CRYPTO*, 1990.

[2] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *Proceedings of the ACM conference on Computer and Communications Security*, 2008.

[3] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-Homomorphic Encryption and Multiparty Computation. In *Proceedings of EUROCRYPT*, 2011.

[4] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, 2008.

[5] L. T. A. N. Brandão. Secure Two-Party Computation with Reusable Bit-Commitments, via a Cut-and-Choose with Forge-and-Lose Technique. Technical report, University of Lisbon, 2013.

[6] M. Burkhart, M. Strasser, D. Many, and X. 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, CA, USA, 2010. USENIX Association.

[7] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. For your phone only: custom protocols for efficient

secure function evaluation on mobile devices. In *Journal of Security and Communication Networks (SCN)*, To appear 2014.

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

[9] I. Damgård, M. Geisler, M. Krøigaard, and J. B. 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.

[10] C. Gentry, S. Gorbunov, S. Halevi, V. Vaikuntanathan, and D. Vinayagamurthy. How to compress (reusable) garbled circuits. Cryptology ePrint Archive, Report 2013/687, 2013. `http://eprint.iacr.org/`.

[11] C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled ram revisited. In *Advances in Cryptology–EUROCRYPT 2014*, pages 405–422. Springer Berlin Heidelberg, 2014.

[12] C. Gentry, S. Halevi, and N. P. Smart. Homomorphic Evaluation of the AES Circuit. In *Proceedings of CRYPTO*, 2012.

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

[14] 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 annual international conference on Advances in cryptology*, 2008.

[15] S. Halevi, Y. Lindell, and B. Pinkas. Secure Computation on the Web: Computing without Simultaneous Interaction. In *CRYPTO'11*, 2011.

[16] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: tool for automating secure two-party computations. In *Proceedings of the ACM conference on Computer and Communications Security*, 2010.

[17] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ansi c. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 772–783, New York, NY, USA, 2012. ACM.

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

[19] Y. Huang, J. Katz, and D. Evans. Quid-Pro-Quo-tocols: Strengthening Semi-Honest Protocols with Dual Execution. *IEEE Symposium on Security and Privacy*, (33rd), May 2012.

[20] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Proceedings of CRYPTO*, 2003.

[21] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. Technical Report MSR-TR-2013-63, Microsoft Research, 2013.

[22] S. Kamara, P. Mohassel, and B. Riva. Salus: A system for server-aided secure function evaluation. In *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[23] F. Kerschbaum. Expression rewriting for optimizing secure computation. In *Conference on Data and Application Security and Privacy*, 2013.

[24] M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *Proceedings of Symposium on Information Theory in the Benelux*, 2006.

[25] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *Proceedings of the international colloquium on Automata, Languages and Programming, Part II*, 2008.

[26] B. Kreuter, B. Mood, a. shelat, and K. Butler. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In *Proceedings of the USENIX Security Symposium*, 2013.

[27] B. Kreuter, a. shelat, and C.-H. Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the USENIX Security Symposium*, 2012.

[28] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2006.

[29] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proceedings of the annual international conference on Advances in Cryptology*, 2007.

[30] S. Lu and R. Ostrovsky. How to garble ram programs. In *Advances in Cryptology–EUROCRYPT 2013*, pages 719–734. Springer Berlin Heidelberg, 2013.

[31] S. Lu and R. Ostrovsky. Garbled ram revisited, part ii. Cryptology ePrint Archive, Report 2014/083, 2014. `http://eprint.iacr.org/`.

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

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

[34] M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the annual ACM Symposium on Theory of Computing (STOC)*, 1999.

[35] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, 2001.

[36] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure Two-Party Computation is Practical. In *ASIACRYPT*, 2009.

[37] a. shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *Proceedings of EUROCRYPT*, 2011.

[38] a. shelat and C.-H. Shen. Fast two-party secure computation with minimal assumptions. In *Conference*

on Computer and Communications Security (CCS), 2013.

[39] A. C. Yao. Protocols for secure computations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982.

[40] Y. Zhang, A. Steele, and M. Blanton. PICCO: A General-purpose Compiler for Private Distributed Computation. In *Proceedings of the ACM Conference on Computer Communications Security (CCS)*, 2013.

APPENDIX

A. CMTB PROTOCOL

As we are building off of the CMTB garbled circuit execution system, we give an abbreviated version of the protocol. In our description we refer to the generator, the cloud, and the evaluator. The cloud is the party the evaluator outsources her computation to.

Circuit generation and check: The template for the garbled circuit is augmented to add one-time XOR pads on the output bits and split the evaluator's input wires per the input encoding scheme. The generator generates the necessary garbled circuits and commits to them and sends the commitments to the evaluator. The generator then commits to input labels for the evaluator's inputs.

CMTB relies on Goyal et al.'s [14] random seed check, which was implemented by Kreuter et al. [27] to combat generation of incorrect circuits. This technique uses a cut-and-choose style protocol to determine whether the generator created the correct circuits by creating and committing to many different circuits. Some of those circuits are used for evaluation, while the others are used as check circuits.

Evaluator's inputs: Rather than a two-party oblivious transfer, we perform a three-party *outsourced oblivious transfer*. An outsourced oblivious transfer is an OT that gets the select bits from one party, the wire labels from another, and returns the selected wire labels to a third party. The party that selects the wire labels does not learn what the wire labels are, and the party that inputs the wire labels does not learn which wire was selected; the third party only learns the selected wire labels. In CMTB, the generator offers up wire labels, the evaluator provides the select bits, and the cloud receives the selected labels. CMTB uses the Ishai OT extension [20] to reduce the number of OTs.

CMTB uses an encoding technique from Lindell and Pinkas [29], which prevents the generator from finding out any information about the evaluator's input if a selective failure attack transpires. CMTB also uses the commitment technique of Kreuter et al. [27] to prevent the generator from swapping the two possible outputs of the oblivious transfer. To ensure the evaluator's input is consistent across all circuits, CMTB uses a technique from Lindell and Pinkas [29], whereby the inputs are derived from a single oblivious transfer.

Generator's input and consistency check: The generator sends his input to the cloud for the evaluation circuits. Then the generator, evaluator, and cloud all work together to prove the input consistency of the generator's input. For the generator's input consistency check, CMTB uses the malleable-claw free construction from shelat and Shen [37].

Circuit evaluations: The cloud evaluates the garbled circuits marked for evaluation and checks the circuits marked for checking. The cloud enters in the generator and evaluator's input into each garbled circuit and evaluates each circuit. The output for any particular bit is then the majority output between all evaluator circuits. The cloud then recreates each check circuit. The cloud creates the hashes of each garbled circuit and sends those hashes to the evaluator. The evaluator then verifies the hashes are the same as the ones the generator previously committed to.

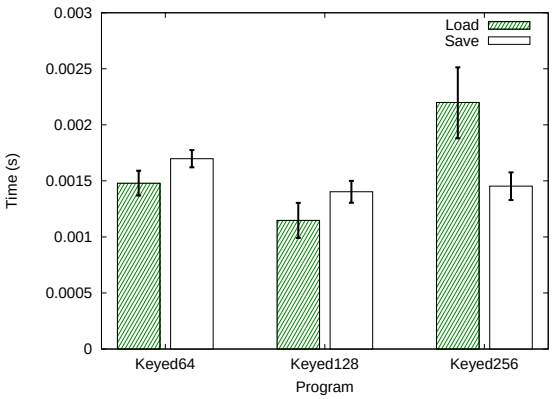


Figure 7: The amount of time it takes to save and load a bit in PartialGC when using 256 circuits.

Output consistency check and output: The three parties prove together that the cloud did not modify the output before she sent it to the generator or evaluator. Both the evaluator and generator receive their respective outputs. All outputs are blinded by the respective party's one-time pad inside the garbled circuit to prevent the cloud from learning what any output bit represents.

CMTB uses the XOR one-time pad technique from Kiraz [24] to prevent the evaluator from learning the generator's real output. To prevent output modification, CMTB uses the witness-indistinguishable zero-knowledge proof from Kreuter et al. [27].

B. OVERHEAD OF REUSING VALUES

We created several versions of the keyed database program to determine the runtime of saving and loading the database on a per bit basis using our system (See Figure 7). This figure shows it is possible to save and load a large amount of saved wire labels in a relatively short time. The time to load a wire label is larger than the time to save a value since saving only involves saving the wire label to a file and loading involves reading from a file and creating the partial input gates. Although not shown in the figure, the time to save or load a single bit also increases with the circuit parameter. This is because we need S copies of that bit - one for every circuit.

C. EXAMPLE PROGRAM

In this section we describe the execution of an *attendance application*. Imagine a building where the host wants each user to sign in from their phones to keep a log of the guests, but also wants to keep this information secret.

This application has three distinct programs. The first program initializes a counter to a number input by the evaluator. The second program, which is used until the last program is called, takes in a name and increments the counter

by one. The last program outputs all names and returns the count of users.

For this application, users (rather, their mobile phones) assume the role of evaluators in the protocol (Section 4).

First, the host runs the initial program to initialize a database. We cannot execute the second program to add names to the log until this is done, lest we reveal that there is no memory saved (*i.e.*, there is no one else present).

Protocol in Brief: In this first program, the cut-and-choose OT is executed to select the circuit split (the circuits that are for evaluation and generation). Both parties save the decryption keys: the cloud saves the keys attained from the OT and the generator saves both possible keys that could have been selected by the cloud. The evaluator performs the OOT with the other parties to input the initial value into the program. There is no input by the generator so the generator's input check does not execute. There is no partial input so that phase of the protocol is skipped. The garbled circuit to set the initial value is executed; while there is no output to the generator or evaluator, a partial output is produced: the cloud saves the garbled wire value, which it possesses, and the generator saves both possible wire values (the generator does not know what value the cloud has, and the cloud does not know what the value it has saved actually represents). The cloud also saves the circuit split.

Saved memory after the program execution (when the evaluator inputs 0 as the initial value):

Count
0
Saved Guests

Guest 1 then enters the building and executes the program, entering his name ("Guest 1") as input.

Protocol in Brief: In this second program, the cut-and-choose OT is not executed. Instead, both the generator and cloud load the saved decryption key values, hash them, and use those values for the check and evaluation circuit information (instead of attaining new keys through an OT, which would break security). The new keys are saved, and the evaluator then performs the OOT for input. The generator does not have any input in this program so the check for the generator's input is skipped. Since there exists a partial input, the generator loads both possible wire values and creates the partial input gates. The cloud loads the attained values, receives the partial input gates from the generator, and then executes (and checks) the partial input gates to receive the garbled input values. The garbled circuit is then executed and partial output saved as before (although there is more data to save for this program as there is a name present in the database).

After executing the second program the memory is as follows:

Count
1
Saved Guests
Guest1

Guest 2 then enters the dwelling and runs the program. The execution is similar to the previous one (when Guest 1 entered), except that it's executed by Guest 2's phone.

At this point, the memory is as follows:

Count
2
Saved Guests
Guest1
Guest2

Guest 3 then enters the dwelling and executes the program as before. At this point, the memory is as follows:

Count
3
Saved Guests
Guest1
Guest2
Guest3

Finally, the host runs the last program that outputs the count and the guests in the database. In this case the count is 3 and the guests are *Guest*1, *Guest*2, and *Guest*3.

LIST OF ABBREVIATIONS AND ACRONYMS

2P-SFE	Two-party Secure Function Evaluation
Agg	Aggregator Switch
BGW88	BenOr-Goldwasser-Wigderson (an MPC protocol that they published in 1988)
Core	Core Router
DAU	Data-Acquisition Unit
DPC	Data Policy Compliance
DR	Data Retrieval
GMW87	Goldreich-Micali-Wigderson (an MPC protocol that they published in 1987)
GUI	Graphical User Interface
LEU	Local-Execution Unit
MPC	Secure, Multiparty Computation (synonym of SMPC)
P-SRA	Private Structural-Reliability Auditor
SC	Secure Computation
SFE	Secure Function Evaluation (synonym of SMPC)
SMPC	Secure, Multiparty Computation
SRA	Structural-Reliability Auditor
SSU	Secret-Sharing Unit
ToR	Top-of-Rack Switch
XOR	Exclusive OR

GLOSSARY OF TERMINOLOGY

Equality-preserving encryption: An encryption method with the property that encryption of identical plaintexts with the same key produces identical ciphertexts. The simplest (but not only) case of equality-preserving encryption is deterministic encryption.

Evaluator: The party responsible for running a garbled circuit.

Garbled circuit: A scrambled representation of a low-level circuit designed to prevent its evaluator from understanding the inputs or outputs.

Generator: The party responsible for garbling a circuit and inputs.

Malicious adversary: A party that may deviate from the prescribed protocol in arbitrary ways.

Negligible probability: Probability that decreases faster than n^{-c}, for all positive constants c, as the size n of the problem instance goes to infinity.

Oblivious Transfer: A cryptographic protocol wherein a user is able to learn 1 out of n secret values held by a server, and the server is unable to determine which value the user learned.

Semi-honest adversary: A party that follows the prescribed protocol but tries to use the information revealed during executions of the protocol to compute things that he is not supposed to know.

Symmetric cryptography: Cryptographic protocols wherein each participant uses the same key for encryption and decryption.