USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

CoVault: Secure, Scalable Analytics of Personal Data

Roberta De Viti and Isaac Sheff, *Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus;* Noemi Glaeser, *Max Planck Institute for Security and Privacy (MPI-SP) and University of Maryland;* Baltasar Dinis, *Instituto Superior Técnico (ULisboa), INESC-ID;* Rodrigo Rodrigues, *Instituto Superior Técnico (ULisboa) / INESC-ID;* Bobby Bhattacharjee, *University of Maryland;* Anwar Hithnawi, *ETH Zürich;* Deepak Garg and Peter Druschel, *Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus*

https://www.usenix.org/conference/usenixsecurity25/presentation/de-viti

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

CoVault: Secure, Scalable Analytics of Personal Data

Roberta De Viti[1], Isaac Sheff[1*], Noemi Glaeser[2,4*], Baltasar Dinis[3*], Rodrigo Rodrigues[3],
Bobby Bhattacharjee[4], Anwar Hithnawi[5*], Deepak Garg[1], and Peter Druschel[1]

[1]Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus
[2]Max Planck Institute for Security and Privacy (MPI-SP)
[3]Instituto Superior Técnico (ULisboa), INESC-ID
[4]University of Maryland
[5]ETH Zürich

Abstract

There is growing awareness that the analysis of personal data, such as individuals' mobility, financial, and health data, can provide significant benefits to society. However, liberal societies have so far refrained from such analytics, arguably due to the lack of secure analytics platforms that scale to billions of records while operating in a very strong threat model. We contend that one fundamental gap here is the lack of an architecture that can scale (actively-)secure multi-party computation (MPC) horizontally without weakening security. To bridge this gap, we present CoVault, an analytics platform that leverages server-aided MPC and trusted execution environments (TEEs) to colocate the MPC parties in a single datacenter without reducing security, and scales MPC horizontally to the datacenter's available resources. CoVault scales well empirically. For example, CoVault can scale the DualEx 2PC protocol [60] to perform epidemic analytics for a country of 80M people (about 11.85B data records/day) on a continuous basis using one core pair for every 30,000 people.

1 Introduction

Data about individuals' health, diet, activity, mobility, social contacts, and finances is being captured at high resolution by smart devices and apps, by online services, and by offline services such as hospitals and banks that store customer data. Large-scale *analysis* of this data could benefit (i) public health, e.g., by studying the spread of epidemics with high spatio-temporal resolution, or new and rare diseases; (ii) sustainability, e.g., by informing transportation, energy, and urban planning; (iii) social welfare, e.g., by uncovering disparities in income and access to education or services; and (iv) economic stability, e.g., by providing insight into markets, among many others.

Yet, liberal societies have mostly refrained from making personal data available for large-scale analysis, even if doing so would clearly be in the public interest (e.g., for scientific research). Arguably, this is largely out of concern that data leaks and misuse would harm individuals and businesses, erode the public's trust and deter voluntary data contributions. There is a growing realization, however, that this conservative approach conflicts with urgent societal challenges like public health and sustainability, where savings on the order of billions of Euros are thought possible using data-driven innovation in the EU alone [30,31]. To this end, the EU Parliament approved the Data Governance Act (DGA) [4], which seeks to facilitate voluntary contributions of high-resolution data from diverse *data sources*—individuals, private companies and public bodies—for analysis by authorized parties (e.g., scientists, government agencies) in the EU. The DGA also defines *data intermediaries* who act as data clearinghouses and facilitate analytics by authorized parties securely.

Given the sensitivity and volume of the data in question, we believe that the following properties are fundamental to any data intermediary or similar secure analytics platform: (1) *Data confidentiality* at all times – while data is *in use*, at rest and in transit. Confidentiality should *not rely on a single root of trust for any component (even hardware)* because strong adversaries who can compromise or backdoor software and hardware are fathomable in this context, (2) *Scalability*: The platform should scale to datasets spanning entire nations or continents (hundreds of billions of records) and millions of data sources, (3) *Integrity* of the analytics results, which is important when analytics inform political, economic, or healthcare decisions, and (4) *Selective forward consent (SFC)*: Each data source should be able to decide which analysis queries and which analysts can use its data, and set an expiration time on the data.

To the best of our knowledge, the design of a platform that satisfies just the first three properties together remains an unsolved problem. By definition, data confidentiality without a single root of trust requires secure multi-party computation (MPC) [18,36,52,65,109]. However, owing to their onerous computation and bandwidth overheads, MPC-based systems have never been shown to scale to hundreds of billions of records. Even the best contemporary MPC systems scale only

*Affiliation at time work was done.

to millions of records [16, 29, 44, 81–83, 86, 100, 103], and that too only with passive security, where even compromised parties are assumed to follow the protocol – an unrealistic assumption in our setting. Instead, we require MPC with active security, which ensures confidentiality and integrity even when compromised parties behave arbitrarily. In the active setting, we are not aware of any MPC system that scales even to millions of records.

Accordingly, this paper presents CoVault, a new system that scales any underlying MPC protocol horizontally to the resources of a datacenter. We do not invent sophisticated new protocols. Rather, we observe that a careful combination of *existing* distributed scaling techniques and hardware features suffices to scale MPC horizontally. Furthermore, we observe that these same techniques also enforce integrity of results and SFC, yielding a system that satisfies all four properties listed above.

We validate CoVault's design by empirically demonstrating horizontal scaling in the 2-party setting for two actively-secure, garbled circuit-based MPC protocols in slightly different leakage models: malicious [66, 107] and malicious with a 1-bit leak (which leaks 1 bit of information for significantly better performance) [60]. For comparison, we also discuss how passively-secure MPC would scale horizontally with CoVault, even though this is not our intended use case. Our experiments focus specifically on MPC using garbled circuits because our target application, query-based analytics, is dominated by combinatorial operations such as integer comparison, for which garbled circuits are more efficient than other MPC protocols [90]. However, CoVault's design does not depend on garbled circuits and can be used to scale any class of MPC protocols.

Next, we briefly outline our design and how it provides all four properties, starting with horizontal scaling.

Scalability: Bandwidth problem. The main bottleneck in scaling MPC protocols horizontally is network bandwidth. A single pair of cores in the 2-party setting can saturate several Gbps of network bandwidth when evaluating garbled circuits. (Lower bandwidth increases query latency proportionally and wastes CPU resources.) The required bandwidth increases proportionally with the number of cores. For example, if each party has 1,000 cores, a cross-party bandwidth of several Tbps is needed to fully utilize the cores. Such high bandwidth is available only inside a datacenter.

Thus, in order to scale MPC horizontally, it is necessary to colocate the MPC parties in the same datacenter. However, colocation increases the parties' susceptibility to compromise by the same technical, political, and physical means, thereby negating the very purpose of using MPC.

Our first insight is that we can colocate parties in the same datacenter, while preserving their independence to compromise, by separating the parties' software stacks and isolating the parties in *trusted execution environments (TEEs) of independent hardware vendors*, such as Intel's TDX, AMD's SEV-SNP and ARM's CCA [1, 13, 62]. This design places the root of trust in the infallibility of $(n-t)$-of-n heterogeneous TEEs, i.e., it tolerates the compromise of up to t-of-n TEEs (1-of-2 in our prototype). Using this idea, we can colocate all MPC parties to obtain very high cross-party bandwidth without weakening MPC security: To compromise confidentiality or integrity, an attacker would have to compromise more vendor-independent TEEs than the underlying MPC protocol's compromise threshold, which is an extremely high bar for any attacker.

Although the isolation of MPC parties in diverse TEEs has been considered in concurrent work for other purposes [32, 39], we believe that CoVault is the first system that leverages diverse TEEs and party colocation to harness the large bisection bandwidth and core count of a datacenter to scale MPC analytics without weakening the MPC threat model.

Scalability: Core parallelism. Even with the bandwidth issue resolved, we need a way to exploit multi-core parallelism. We observe that standard scaling techniques from the distributed systems literature can be applied even when the underlying compute units run MPC. Specifically, *we implement MapReduce [43] over MPC*: Given an analytics query on a very large private dataset, we divide the query's MPC circuit into small map and reduce circuits, which are scheduled across the available cores at each party (as in standard MapReduce).

We implement mappers and reducers using standard MPC circuits for the following basic operations on lists of records: linear scans, sorting, compacting (moving marked entries to the tail) and merging sorted lists. As we explain later, this set of primitives suffices for all filter-groupby-aggregate queries. We carefully avoid leaks through network side channels by padding all communication between mappers and reducers to fixed sizes. Additionally, we support data-dependent queries through a new and simple oblivious data retrieval (ODR) scheme that enables lookups through indexes on private fields without revealing the data access pattern.

We note that MapReduce can be substituted with any other distributed scaling paradigm, as long as the paradigm's basic primitives can be efficiently implemented in MPC circuits.

Summary of CoVault's basic design. At a high-level, CoVault combines *horizontal scaling* with *server-aided MPC* [20, 33, 45, 64, 69, 94]. CoVault's analytics backbone consists of a fixed set of n administratively independent MPC parties, each with a large number of dedicated servers, all colocated in the same datacenter with high-bandwidth interconnects, and running in TEEs of n independent vendors.

Data sources – companies, individual devices and public bodies – contribute their data to CoVault by *secret-sharing* each datum among the n parties. The n parties store these shares in local databases, possibly after pre-processing in MPC (e.g., range-checks). Analytics queries are executed by the n parties on the stored shares using MPC. The MPC protocol must tolerate the malicious compromise of a subset of the parties (up to a threshold denoted t), and all MPC computations are scaled horizontally by distributing them across all available cores, as explained earlier.

CoVault is primarily designed for data donated by individuals, a setting that necessitates SFC and support for many data sources, unlike prior work like Senate [90], where a small number of sources contribute larger amounts of locally aggregated data. While CoVault supports this latter configuration as well, it is not its focus. Next, we describe how CoVault supports integrity of outputs and SFC.

Integrity of outputs. In our server-aided MPC setting, ensuring the integrity of outputs simplifies to two sub-problems: (i) Input integrity: Verifying that all MPC parties compute on *unmodified* data shares received from the data sources, and (ii) Computation integrity: Ensuring that the MPC computation executes correctly. Both sub-problems have standard solutions, which we leverage. For input integrity, we use *authenticated secret sharing* [38, 47, 93]. Each datum is secret-shared with a MAC that is created by the datum's source. A datum's MAC is checked inside MPC when the datum is used for analytics, detecting any manipulation of data shares by rogue parties. For computation integrity, we rely on actively-secure MPC protocols [18, 60, 66, 107], which detect any deviation from the intended protocol.

Selective forward consent. We further observe that TEEs, which are already a part of our design, can be harnessed to enforce SFC as follows. The code of each TEE instance (running on behalf of a MPC party) is only capable of participating in a specific set of analytics queries, authorized by a specific set of queriers up to a specific expiration time. The queries, authorized queriers, and expiration time of each instance are publicly advertised. A data source uploads data shares to a TEE instance only if it consents to the instance's queries, authorized queriers, and expiration time, and it can attest – *using TEE attestation* – that the instance's code implements these parameters. This enforces SFC.

Contributions. Our contributions include: (a) CoVault, a system that scales MPC horizontally to datacenters, fully exploiting core parallelism. CoVault leverages diverse TEEs to facilitate colocation of MPC parties with very high cross-party bandwidth, without weakening independence of the parties; (b) Three technical components of possibly independent interest: A provably secure authenticated secret-sharing scheme (§ 5), an actively secure 2PC protocol with a 1-bit leak that reveals information to a third entity, not the computing parties (§ 5), and a custom oblivious data retrieval (ODR) scheme (§ 4.2) that supports database lookups on private indexes in constant time; (c) An experimental evaluation of CoVault with epidemic analytics as an example scenario, which shows that CoVault can handle data for a country of 80M people using a single core pair per 30,000 individuals. We are not aware of any other MPC-based analytics platform that can securely perform similarly complex queries in a comparably strong threat model at this scale.

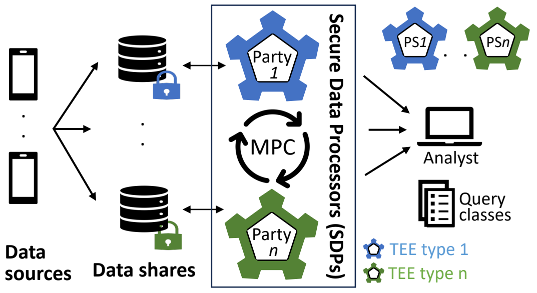

Figure 1: CoVault's secure data processing. *Conceptually*, data sources contribute their data in such a way that only queries within a given query class can be executed on their data, by analysts authorized by the query class, and for a period defined by the query class. *Concretely*, (a) Each data source secret-shares its data into n authenticated shares, encrypts each share for a different MPC party, checks that the parties have been verified, and uploads the shares to an untrusted database in a datacenter. (b) Queries are processed by a set of SDPs hosted in the same datacenter. Each of the n SDPs, one for each party, is encapsulated by a different TEE type and accesses its corresponding data share; the SDPs jointly perform MPC, and deliver their respective share of the query result to an authorized analyst. (c) Provisioning servers (PSs), one per party and implemented in the party's TEE type, attest and provision all SDPs of a party with their keys.

2 CoVault Overview

We survey relevant building blocks, sketch CoVault's architecture, and provide a roadmap for the rest of the paper.

2.1 Building blocks

Secret sharing is a method for sharing a secret value among multiple parties, such that the value can only be reconstructed if a sufficient fraction of shares (possibly all) are combined. *Authenticated* secret sharing adds auxiliary information that allows the parties to verify the authenticity of a value that they have reconstructed from their shares.

Secure multi-party computation (MPC) enables multiple parties to jointly compute a function without revealing their respective inputs. MPC protocols are either passively secure (corrupt parties are semi-honest [59, 109]) or actively secure (corrupt parties may be malicious [106]). In t-of-n active security, up to t of the n parties may be compromised. Depending on the protocol, t varies from $n/3$ to $n-1$.

Garbled circuits (GCs) are a specific class of secure 2-party computation protocols (2PC) ($n = 2, t = 1$) [89]. GCs exist with both passive and active security. A specific GC protocol, DualEx [60], achieves active security apart from leaking 1 bit of the input, in exchange for achieving performance close to that of passive security. Techniques exist both to restrict the predicate the adversary can leak and the probability that a leakage occurs [70]. All GCs are data oblivious because circuits have no control flow.

Trusted execution environments (TEEs) are supported by several recent general-purpose CPUs, e.g., Intel SGX, Intel TDX, AMD SEV and ARM CCA [9,13,35,62]. TEEs provide confidentiality and integrity of data and computation under a threat model that tolerates compromised operating systems, hypervisors, and even some physical attacks [25,35]. Newer TEEs – SEV-SNP, TDX, and CCA – encapsulate an entire VM, providing easy software migration. An *attestation protocol* verifies that a VM executes in an authentic TEE of a given type and that its initial memory state (code and data) matches an expected secure hash value (measurement). Upon successful attestation, the VM is provided with cryptographic material needed to authenticate itself to remote parties and to access data sealed for it. The security of a TEE depends on the integrity of its vendor's certificate chain, proprietary hardware, and firmware.

2.2 CoVault roadmap and threat model

We introduce CoVault's secure data processing component (SDP) (see Figure 1) in §3. In §4, we show how to scale CoVault's query processing horizontally with MapReduce [43], where each mapper and reducer is an instance of the SDP. In §5, we present our prototype SDP implementation, which relies on 2PC with garbled circuits. In §6, we evaluate CoVault in the context of epidemic analytics as an example application scenario, and compare its performance to other systems with a similar threat model. A technical report (TR) [102] provides formal security proofs for the primitives used in CoVault, and other details.

Threat model. CoVault relies on an underlying MPC protocol with t-of-n security in a deployment-chosen threat model (our intended use-case is protocols with active security or active security with fixed bit-leakage in exchange for significantly better performance). CoVault inherits the adversary model from the underlying MPC protocol used by the SDPs. CoVault places the code of the n parties inside the TEE technologies of n different vendors. Consequently, we assume that $n-t$ of these n TEE technologies/vendors are uncompromised; the remaining t TEE technologies/vendors may be compromised (statically) using any means allowed under the MPC protocol's adversarial model.

CoVault assumes that the software executed by the parties inside TEEs is correct and the TEEs are properly attested. One approach is to delegate these tasks—which have to be performed by any secure system that relies on TEEs—to a community of experts as described in the TR [102].

We assume that data sources follow the data contribution protocol and do not deliberately contribute false or biased data (data poisoning attacks). A malicious source can compromise the confidentiality and SFC of its own data (but not of others' data), and bias query results relying on its data to the extent that the query is sensitive to that source's data. Defenses against data poisoning like integrity, consistency, and plausibility checks are orthogonal to this work and can be integrated into CoVault's data ingress processing.

We make standard assumptions about the security of cryptographic primitives used in CoVault's design, and assume a PTIME adversary. We assume that data is stored in CoVault for periods short enough that the keys remain secure despite advances in cryptanalysis and compute power. Denial-of-service attacks are out of scope; they can be addressed with orthogonal techniques.

TEEs have known vulnerabilities including root-key leaks, attacks on remote attestation, vendor compromise, conventional side-channel attacks (memory-access patterns, CPU cache timing), microarchitectural side-channel attacks, and integrity attacks (power-based, Rowhammer) [25,76]. Nonetheless, our approach raises the bar for vendor compromise and implementation-specific attacks, and improves security over TEE+MPC setups that place the root of trust in a single TEE vendor (e.g., [72]). CoVault is immune to conventional side-channel attacks as the underlying MPC protocols are data oblivious. Software mitigations of microarchitectural attacks—to the extent that they exist—can be applied inside TEEs. Consequently, a successful attack on CoVault requires the compromise of TEEs from $t+1$ distinct vendors running hardened code without conventional side channels. We believe that success with such attacks is unlikely, but they are possible in principle, and mitigating them is outside the scope of this paper.

Deployment considerations with TEEs. TEE adoption for commercial use is growing rapidly. Two leading Cloud providers—Google Cloud and Azure—offer both AMD SEV-SNP and Intel TDX TEEs with high-bandwidth interconnects [7, 8]. These TEEs encapsulate entire VMs and the Cloud providers support transparent VM deployment. Hence, the additional deployment overhead is limited to code attestation (for each TEE vendor). To mitigate this burden, Cloud providers offer client libraries, code wrappers and detailed tutorials [5, 6].

3 Secure data processing

In this section, we present the design of CoVault's secure data processing, beginning with the API and then incrementally refining a strawman design.

CoVault's API is shown in Figure 2. Analysts who wish to solicit data contributions define a *query class* using the *Setup* operation. A query class is defined by the triple $[Q,D,t_e]$, where Q denotes a set of queries, D a set of authorized analysts, and t_e a time at which all contributed data should expire and no longer be available for analysis. A data source contributes data to a query class only if it is comfortable with the

$MPK[Q,D,t_e] \leftarrow$ **Setup**$(1^\kappa, Q, D, t_e)$
MPC to initialize a new query class and produce a public-private key pair $(MPK[Q,D,t_e], MSK[Q,D,t_e])$ bound to the class $[Q,D,t_e]$. $MPK[Q,D,t_e]$ is published for data sources, $MSK[Q,D,t_e]$ is retained by the n parties. **Inputs:** κ: security parameter, Q: set of allowed queries, D: set of authorized analysts, t_e: expiration time. **Outputs:** n-element public key $(MPK[Q,D,t_e])$.

$C \leftarrow$ **Contribute**$(MPK[Q,D,t_e], m)$
Executed locally at a data source to contribute data m to the query class $[Q,D,t_e]$. **Inputs:** $MPK[Q,D,t_e]$: public key, m: data to encrypt. **Outputs:** C: n encrypted shares of m.

$q(m_1,\ldots,m_k) \leftarrow$ **Query**$(q, d, C_1, \ldots, C_k)$
MPC to compute $q(m_1,\ldots,m_k)$. If the caller is d, C_is are encrypted shares of m_is in the same query class $[Q,D,t_e]$, $d \in D$, $q \in Q$, and current time $< t_e$, then return $q(m_1,\ldots,m_k)$, else return $\perp$. **Inputs:** q: query, d: analyst, C_i: n encrypted shares of data from source i. **Outputs:** $q(m_1,\ldots,m_k)$ or $\perp$.

Figure 2: CoVault's API (MPK, MSK, C are n-element vectors, where n is the number of parties.)

class's Q, D and t_e. Data sources contribute data to a query class using the *Contribute* operation, and authorized analysts may execute queries using the *Query* operation.

We aim to realize this API while satisfying the four properties listed in § 1:
1) Confidentiality. Contributed data remains confidential at all times, except for releases allowed by SFC (see below).
2) Scalability. All data processing can be horizontally scaled out within a single datacenter, without weakening our threat model.
3) Integrity. Any modifications to the data shares or the query result is detected.
4) SFC. A data source contributes its data to specific query classes. That data is used in accordance with those query classes' $[Q,D,t_e]$ triples.

Next, we describe a strawman design (S1) for core SDP functionality, which attains confidentiality, integrity, and SFC in a weak threat model. In § 3.2, we refine this strawman to our full construction (S2), which satisfies the desired strong threat model even though all parties execute in the same datacenter. In § 4, we show how to exploit this colocation to achieve the remaining property, scalability.

3.1 Strawman (S1): Server-aided MPC

Our first strawman, S1, implements the API (Figure 2) by combining secret sharing and n-party MPC (but not TEEs, which we add in § 3.2). Specifically, n independent parties *jointly* hold shares of the secret key, $MSK[Q,D,t_e]$. S1 attains confidentiality, integrity, and SFC, but in a weak threat model that assumes that parties have independent roots of trust despite being co-located in the same datacenter.

Components. To contribute its data, a source uses a n-of-n *authenticated secret-sharing scheme* (the cleartext cannot be recovered unless all n shares are available); then, it entrusts each data share to a different one of n parties by encrypting the share with the respective party's public key. To execute a query, the parties decrypt their shares of data locally, and run n-party MPC on their shares to reconstruct the uploaded plaintexts $m_1,\ldots,m_k$ and compute the query result, which is provided to the analyst. The chosen MPC scheme must be actively secure with t-of-n static compromise: no party learns anything about $m_1,\ldots,m_k$ as long as at least $n-t$ parties are honest. Furthermore, the chosen MPC scheme is data oblivious, i.e., without any control flow. A simple way to attain data obliviousness is to use circuit-based MPC.

Implementation. Next, we sketch how S1 implements the n-party API shown in Figure 2, which can be used after the n parties are initialized.

Setup: Each party executes this function locally, produces a standard asymmetric public-private key pair, keeps its private key locally alongside Q, D and t_e, and publishes the public key. MPK and MSK denote n-element vectors (one element for each party/share) of the public and private keys, respectively.

Contribute: A data source runs this function locally, secret-shares its data m into n shares with an authenticated secret-sharing scheme, and encrypts each share with the corresponding party's public key in the vector $MPK[Q,D,t_e]$. C is the vector of the encrypted shares.

Query: The analyst d calls this function separately on each party, authenticates itself as d, and provides the desired query q along with the party's encrypted shares of the vectors $C_1,\ldots,C_k$. If each party can authenticate d, finds that $d \in D$, $q \in Q$ and its local time is less than t_e, then the parties decrypt their respective encrypted shares locally and run MPC. The MPC computation first checks the authenticity of the shares (via the authenticated secret-sharing scheme), and then computes the query result (which is $\perp$ if any check fails). This result is encrypted for the analyst and signed by a private key that exists only in shared form.

Properties. S1 has the desired confidentiality, integrity, and SFC properties assuming that at least $n-t$ parties are honest. Briefly, confidentiality results from source secret-sharing plus MPC; integrity from MACs plus malicious-mode MPC; and SFC from TEEs plus attestation. S1 satisfies our threat model in theory. In practice, realizing the $n-t$ honest-party assumption of MPC requires the parties to have independent roots of trust (for compromise), i.e, their compromisability should not be correlated. This independence is questionable when parties are colocated in the same datacenter without an additional mechanism to separate their roots of trust. Since we want colocation for scalability, we introduce such a separation mechanism on top of S1 in our next construction (S2).

3.2 Full construction (S2): S1 + TEEs

Our full construction, S2, extends S1 by isolating the (colocated) parties from each other using diverse TEEs.

Components. S2 executes each of the n parties inside a TEE of a different, independent design and implementation. The code of each party is verified to be correct and the corresponding initial measurement of each party's TEE is attested.

Implementation. During initialization, the TEEs for each of the n parties are started and attested. All subsequent communication among and with the TEEs occurs via secure channels tied to the attestation. The API functions from S1 execute in TEEs. In the API call *Contribute*, the user encrypts its data shares only if the TEEs have been correctly attested.

Properties. Like S1, S2 provides the desired properties of confidentiality, integrity, and SFC within our threat model. However, S2 goes beyond S1, where colocation of the n parties correlates the parties' compromisability: in S2, the n attested TEEs provide independent roots of trust for the parties even when the parties are colocated in the same datacenter.

3.3 SDP construction details

Next, we describe the SDP construction, which is based on S2, in more detail.

General setting. The n parties execute in different, independent TEE implementations. Each party consists of a single provisioning service (PS) and one or more secure data processors (SDPs). Each PS acts on behalf of its party, attesting and provisioning the SDPs in its pipeline with the key necessary to decrypt data shares.

SDPs are the components that perform MPC: each party's SDPs *collectively* form one *party* in the MPC protocol. A corresponding set of SDPs (one from each party) handles one query class $[Q,D,t_e]$, but the same query class may be handled by several sets of SDPs (for scaling).

Each party's PS holds information about defined query classes. For each class $[Q,D,t_e]$, this information consists of the measurement hashes of the TEEs (one per TEE type/party) that jointly implement Q in MPC, the public keys of the authorized analysts in D, the data expiration time t_e, and the public keys of the SDPs that implement that query class.

PSs and SDPs can be safely shut down and re-started from their sealed state [63] without re-attestation, and the sealed state can be replicated for persistence. We discuss database rollback prevention in §4.2.

System initialization. Each party instantiates its PS; then, each PS generates its party's private-public key pair.

API call Setup$(1^\kappa, Q, D, t_e)$: A new query class can be created by interested analysts using the *Setup* call. Each party spawns a fresh SDP for the query class, and configures it with the class parameters $[Q,D,t_e]$.

The SDP is remotely attested by its PS and provisioned with cryptographic keys, including the SDP's secret key to decrypt its data shares (in the sense of standard asymmetric cryptography). Each PS stores the query class $[Q,D,t_e]$ and the public keys of the n SDPs, and advertises them publicly. The secret and public keys of the SDPs are, respectively, the vectors $MSK[Q,D,t_e]$ and $MPK[Q,D,t_e]$ of the API.

API call Contribute$(MPK[Q,D,t_e], m)$: Data sources contact the PSs to retrieve information about the available query classes $[Q,D,t_e]$, as well as the public keys of the SDP sets that implement them. Before contributing data to a query class $[Q,D,t_e]$, a data source verifies that the SDPs implementing that class have been attested.

Data sources share data using an *authenticated secret-sharing scheme*, which allows SDPs to verify that the shares have not been modified before they use the sharing (input integrity). The data source uses the authenticated secret-sharing scheme to create n shares of its data, and encrypts each share with the public key of a different member of the set of SDPs. (This set of encrypted shares is denoted C in the *Contribute* API.) The data source then provides each share to its respective SDP. Secret sharing ensures data confidentiality, while subsequently encrypting the shares prevents data misuse: Only correctly attested TEEs (i.e., attested to implement the specific $[Q,D,t_e]$ and thus provisioned with the corresponding decryption keys) can decrypt the shares.

API call Query$(q, d, C_1, \dots, C_K)$: To execute a query q over encrypted shares $C_1, \dots, C_k$ of a query class $[Q,D,t_e]$, an analyst d sends the set of SDPs of that class their shares from $C_1, \dots, C_k$ with the query q. The SDPs independently authenticate d, and check that $q \in Q$, $d \in D$ and current time $< t_e$. Then they decrypt their shares locally and run MPC to verify the shares, to reconstruct the plaintexts $m_1, \dots, m_k$ from the shares, and to compute $q(m_1, \dots, m_k)$, which is revealed only to the analyst.

4 CoVault scalable analytics

A key property of the SDP described in the previous section is that the MPC parties can execute securely within the same datacenter. In this section, we show how CoVault exploits this colocation and uses the enormous network and compute resources of a datacenter to execute many subqueries in parallel, thereby scaling out query processing. Additionally, we describe how CoVault manages its database and efficiently supports oblivious random data accesses.

4.1 Executing queries at scale

The unit of querying in CoVault is a standard SQL filter-groupby-aggregate (FGA) query of the form:

```
SELECT agg([DISTINCT] col1), ...
FROM T WHERE condition GROUP BY col2
```

where `agg` is an aggregation operator like SUM or COUNT. The query can be executed in three steps: (i) select (filter) the rows of table `T` that satisfy `condition`, (ii) group the selected rows by `col2`, and (iii) compute the required `aggregate` for each group.

Problem #1: Horizontal scaling. The colocation of MPC parties allows CoVault to leverage core parallelism and the aggregate bandwidth available in a datacenter to scale-out query processing. For this, CoVault draws inspiration from MapReduce [43]. CoVault converts FGA queries into a set of map and reduce tasks, and distributes them across the datacenter for execution by sets of SDPs using MPC. Since FGA queries tend to be highly data-parallel, most map and reduce tasks are data-independent, and can be executed in parallel using the cores, servers, and bandwidth available in a datacenter.

CoVault's map and reduce tasks for steps (i)–(iii) of a FGA query are built using basic oblivious algorithms: bitonic sort [15], bitonic merge of sorted lists [15], and a butterfly circuit for list compaction (§3 in [54]) that moves marked records to the end of a list. Each of these algorithms is slower than the fastest non-oblivious algorithm for the same task by a factor of $O(\log(N))$, where N is the input size.

Oblivious query execution additionally requires that the volume of intermediate results exchanged among map and reduce tasks not reveal private information. To this end, the data exchanged between a pair of tasks is padded to its maximum possible size, which is typically a function of the maximum number of groups in the query's output. Compaction is crucial to minimizing this size and thus to minimizing the bandwidth needed in the reduce phase.

Problem #2: Integrity of intermediate results. When intermediate data is passed between map-reduce or reduce-reduce tasks, integrity of the data must be enforced. While we could reuse the authenticated secret-sharing scheme to enforce input integrity, MAC computation in MPC is expensive. Depending on the MPC protocol used, it may be possible instead to exchange data using an in-protocol representation. The CoVault prototype uses this optimization as described in § 5.

4.2 CoVault Database

So far, we have not discussed how the encrypted data shares, denoted C in § 3, are managed in our prototype. Once a data source has obtained its encrypted data shares C by locally executing the *Contribute* API, it forwards each share to the corresponding SDP, which stores the share in a per-party untrusted database (DB). In the *Query* API call, the encrypted data shares $C_1, \ldots, C_k$ that the query runs on are retrieved directly from these DBs; they are not provided by the querying analyst. Shares stored in the untrusted DBs are encrypted and MAC'ed with keys known only inside the SDPs' TEEs. The organization of these databases and whether any processing on source data is performed by the SDPs during data ingress depend on the DB schema and the queries in the query class. We sketch an example scenario in § 6.3. In general, data shares may be stored in tables with both row- and column-level MACs to enable efficient integrity checks when the data is read during query processing.

Next, we discuss database access challenges and solutions.

Problem #3: Efficient oblivious DB random access. As explained above, the SDPs read the inputs $C_1, \ldots, C_k$ from their DBs to execute queries. For the most part, DB tables are read sequentially in their entirety. However, some queries need random access to specific rows for efficiency. The pattern of such accesses can leak secrets if the locations of the accessed rows depend on secrets read earlier from the DB (secret-dependent accesses).

In principle, one could implement ORAM [53] within MPC to solve this problem; however, even state-of-the-art implementations [40, 46, 57, 80] either do not match CoVault's requirements and threat model or are orders of magnitude slower than our solution, described below. The properties of private information retrieval (PIR) are closer to our requirements, but PIR still has substantial overhead [34, 37, 85, 104]. Another option is to randomly permute secret-shared tables [27, 58, 75], but existing techniques either have substantial overhead or assume semi-honest adversaries.

Oblivious data retrieval (ODR). CoVault instead relies on a custom malicious-secure, oblivious data retrieval (ODR) scheme, which achieves constant-time lookup at the expense of off-line work to randomly shuffle tables. (DB accesses where the accessed locations are independent of secret data need not use ODR.) Our ODR scheme combines preprocessing inside MPC with pseudorandom table shuffling *outside MPC* for efficiency. The scheme works as follows. For simplicity, we describe the protocol for the case of $n = 2$ (2PC). However, the scheme generalizes to any number of parties.

Preprocessing. During data ingress, which runs in MPC, we preprocess every table that requires ODR access. We encrypt-then-MAC (EtM) each row, and separately EtM the row's primary key. The secrets used to generate the EtMs can be recovered in MPC only.

Shuffling. A preprocessed table is shuffled by a pair of processes, P_1 and P_2, of different parties. This shuffling is done *outside MPC* for efficiency. First, P_1 locally shuffles the table by *obliviously sorting* rows, ordering them by a keyed cryptographic hash over the primary key EtMs. Oblivious sorting also creates a second layer of encryption over every row. The keys used for hashing and encryption are freshly chosen by P_1. Next, P_2 re-shuffles the already shuffled table, by re-sorting the table along a keyed hash over P_1's hashes using a fresh key. This doubly-shuffled table is stored in the DB indexed by P_2's hashes.

Row lookup. P_1 and P_2 input the secret keys they picked during shuffling into 2PC. To access a row with a given primary key, 2PC computes the position of the row in the shuffled table in constant time by applying the EtM and the two keyed hashes to the primary key. The position is revealed to both parties, which fetch the row from the shuffled table. Back in 2PC, the MAC of the row's EtM is checked, the row is decrypted and the primary key stored within the row is compared to the lookup key for equality.

Properties. The ODR scheme obfuscates the dependence of DB row locations on primary keys, and protects the integrity and confidentiality of row contents from a malicious party. First, recovering the order of rows in the doubly-shuffled table requires keys used for shuffling by both parties, which only 2PC has. Second, neither party learns any row individually since all data is encrypted by preprocessing using secret-shared keys. Third, any attempt to tamper with a row's data or swap rows is detected by the checks on the fetched row.

Unlike general, less efficient PIR techniques, our ODR scheme does not hide whether two lookups to the same shuffled table access the same row. To avoid frequency attacks, CoVault prevents such accesses. First, it prefers query plans that access each item only once (such plans are preferred for efficiency reasons as well). If such a plan does not exist, CoVault caches already fetched items inside MPC in order to eliminate multiple database accesses to the same item. If this is not possible, a different shuffle is used for each access. Table shuffles can be precomputed, so that freshly shuffled tables are readily available to queries. (Preprocessing happens once per table.)

Optimization: Public index. We can avoid the ODR overhead for queries that perform random access only on *public* attributes by creating *public indexes*. Public indexes can also speed up queries that *join* data on a public attribute (an example of this join optimization is in the TR [102], §D.2). Otherwise, data-oblivious joins can be very expensive [110], even more so in MPC. In § 6.3, we show illustrative queries that exploit public indexes.

Problem #4: Detecting database roll-back. A malicious platform operator or a corrupt party may roll back the database to an earlier version. In general, rollback can be detected by known techniques [10, 79, 88], such as a secure, persistent, monotonic counter within a TEE or TPM, or a distributed rollback detector. If a distributed rollback detector is used in CoVault, its replicas should be in different datacenters to prevent a correlated rollback. This does not impact query performance, because the detector service is consulted only during reboot of the MPC TEEs and database modifications.

In specific cases, such as the epidemic analytics scenario of § 6.3, we can instead exploit the fact that the database is append-only and continuously indexed. In this case, rollbacks other than truncation are trivially detectable. To detect truncation, CoVault additionally warns the querier whenever records within the index range specified in the query are missing.

Problem #5: Data expiration. CoVault must not return a query result if its execution finishes after the query class's expiration time. To tolerate a malicious platform operator who may manipulate the passage of time experienced by a TEE, the TEEs rely on external third-party timestamping services. Once a query execution finishes, a TEE requests a timestamping service to sign the hash of the query and its result. The TEEs deliver the result to the analyst only if the timestamp precedes the expiration time. Multiple timestamping services are contacted to distribute trust.

4.3 Query expressiveness and limitations

In principle, CoVault can run all FGA queries. Most FGA queries also execute efficiently, but there are exceptions. First, if a tight upper bound on the number of groups in a reducer's output cannot be determined, then efficiency is lost due to excessive padding of the reducer's output data. Second, a query loses efficiency if the maximum number of times the same record is accessed via ODR cannot be bounded tightly.

Finally, joins are inherently expensive in MPC. Prior systems implementing joins in MPC either assume a weaker threat model (e.g., [103]) or they are limited to modest datasets (e.g., [90]). We perform a simple join on small datasets (user uploads from a certain space-time region) during ingress processing (§D in the TR [102]). In general, it is best to do joins during ingress processing if needed and produce materialised views, which can be queried subsequently without joins.

5 CoVault prototype implementation

Next, we detail a specific implementation of the CoVault core API (Figure 2) for $n = 2$ parties, of which $t = 1$ parties may be malicious. This prototype implementation uses garbled circuits (GCs) and the DualEx protocol [60] for 2PC. We emphasize that the constructions described here generalize to any number of parties and any MPC protocol (where the adversarial model of CoVault depends on the underlying MPC protocol's threat model, be it semi-honest, malicious with a 1-bit leak, or fully malicious).

Choice of MPC protocol and number of parties. The choice of MPC protocol and number of parties in our prototype is pragmatic. Using two parties aligns with the current availability of two different hardware TEE types in public Clouds (Intel TDX and ADM SEV-SNP). Even with only two parties, an adversary must compromise both TEE types, which presents a formidable challenge. The choice of GCs with DualEx provides a very strong threat model (active compromise of one TEE with a 1-bit leak) at a resource cost that is only slightly more than twice that of a semi-honest 2PC protocol, and a similar runtime.

Choosing instead a semi-honest 2PC protocol is possible but would imply a total loss of security if one TEE is actively compromised. Choosing a fully malicious 2PC protocol is also possible but would introduce substantially higher latency for each map and reduce task. However, CoVault would still scale out up to the available resources (cores and network bandwidth) in the datacenter.

DualEx. DualEx [60] is an actively secure 2PC protocol that is nearly as fast as passively-secure protocols and needs only twice as many cores, but can leak 1 bit of information in the worst case. DualEx runs two instances of a standard passively-secure 2PC protocol concurrently on separate core pairs, with the roles of the two parties, conventionally called *generator* and *evaluator*, reversed. The results of the two runs are compared for equality using any actively-secure 2PC protocol. If the results mismatch, query execution is suspended pending a manual inspection of the system.

In principle, the leaked bit can be the result of any boolean function of the attacker's choosing, evaluated on the data consumed by a legitimate, approved query. However, an adversary faces many constraints for a practical attack, and recent work [70] has shown how to efficiently limit both the type of boolean function, and the probability of an actual leak, enabling a continuum of practical security-efficiency tradeoffs. If an application cannot tolerate the possibility of a 1-bit leak, CoVault can be configured with a full malicious-secure MPC protocol; we estimate the cost of this choice in §6.

Adapting DualEx to CoVault's needs. Like most 2PC protocols, DualEx reveals the computation's result to both parties. However, we require a protocol that reveals the result only to the analyst, not to the two parties. To address this challenge, we combine DualEx with an authenticated secret-sharing scheme (which we call MtS, see below): We run DualEx to first compute the result $r = q(m_1, \ldots, m_k)$ and then secret share r two ways using MtS, producing two authenticated shares r_1 and r_2. DualEx outputs one share to each party; each party forwards its share to the analyst, who verifies the shares and reconstructs the query result.

Authenticated secret sharing. The authenticated secret-sharing scheme used in our prototype relies on a MAC followed by secret sharing, so we call it *MAC-then-Share* (MtS).[1]

The scheme assumes a MAC function M that provides key- and message-non-malleability as well as message privacy. To split data m into two shares, the data holder generates a random key k, computes a tag $t \leftarrow \mathsf{M}_k(m)$, then generates two random strings r_k, r_m, and uses (standard) xor-secret-sharing to generate shares k_1, k_2 of the key k and shares m_1, m_2 of the data m:

$$k_1 \leftarrow k \oplus r_k, \quad k_2 \leftarrow r_k, \qquad m_1 \leftarrow m \oplus r_m, \quad m_2 \leftarrow r_m$$

[1]CoVault's design is not tied to this particular authenticated secret-sharing scheme. We could also have used other schemes [38,47].

The two MtS authenticated shares of the data are (m_1, k_1, t) and (m_2, k_2, t). Each share looks random, but parties possessing the shares can verify them *jointly* by checking that $\mathsf{M}_{k_1 \oplus k_2}(m_1 \oplus m_2) = t$. This verification fails if either share has been modified.

Our prototype instantiates MtS with $M = KMAC256$ [68]. We also tested field-based linear information-theoretic MACs [24, 71] for M, but we empirically found them to be slower than KMAC256 in our GC-based setting.

In-circuit representation of intermediate results. When passing intermediate results between successive GCs (e.g., between a map and a reduce circuit), the CoVault prototype treats successive GCs as a single, unified circuit and passes intermediate data in its native in-circuit representation, specifically in garbled form. While this choice increases the size of data transmitted or stored by a factor of 256x in our prototype (128x for the garbled encoding of bits and 2x for DualEx), we empirically found it faster than creating and verifying MACs, which would otherwise be necessary to ensure integrity when transmitting intermediate results. Our design minimizes in-2PC MAC usage in the common case: we verify one MAC for every batch of data uploaded by a data source, create per-column MACs when storing data in the database after data ingress, verify these MACs when the data is read for a query, and create one MAC for every query's output. Additional MAC and hash operations are needed for ODR-based database accesses, as detailed in §4.2.

Security analysis and proofs. Our assumption (§2.2) that at least $t = 1$ TEE implementation remains uncompromised and that data sources and analysts interact only with attested SDPs together imply that at least one SDP per query class is fully honest. As a result, all GCs run securely. To establish end-to-end security, it remains to show that our MtS scheme and the adapted DualEx protocol guarantee confidentiality and integrity, which we do in the TR [102], §F.

6 Evaluation

Next, we present an experimental evaluation of CoVault to answer the following high-level questions: What is the cost of basic query primitives and the shuffling required for ODR? How does query latency scale with the number of cores for a realistic set of epidemic analytics queries at scale? How does CoVault's performance and scaling compare to related work with a comparable threat model?

6.1 Prototype and experimental setup

We implement CoVault on top of the EMP-toolkit [105], a state-of-the-art framework for garbled circuits in C/C++. First, we implement DualEx[2] and our extension using emp-

[2]The implementation of the original protocol [60] is not publicly available, but reportedly based on a different Java framework [2].

sh2pc for the semi-honest executions, and emp-ag2pc for the actively-secure result equality check. We use emp-ag2pc also for the experiments with AGMPC (AG2PC refers to AGMPC with 2 parties). Then, we implement circuits for SHA3-256 and AES-128-CTR (which we integrate in emp-tool), and the MtS of §5 (based on KMAC-256, on top of a pre-existing circuit for Keccak). Using these building blocks, we implement our ODR scheme (§4.2), map-reduce primitives (e.g., filter, group-by, aggregate), microbenchmarks, and queries. We use Redis (6.0.16, non-persistent mode) as the DB, but during data-dependent query execution, shuffled views are held in memory on SDPs rather than in Redis to enable efficient re-shuffling.

We deploy the prototype on Google Cloud Compute Engine (GCE), which offers VMs with both AMD SEV-SNP and Intel TDX TEE hardware support, the latter available through a restricted private preview. Unless otherwise stated, we use the following instance types: n2d-standard-8 with Confidential Computing (AMD EPYC 7B13 @ 2.44GHz, 8vCPU, 4 core, 32GB RAM) and c3-standard-8 under GCE's TDX Private Preview (Intel Sapphire Rapids @ 2.7 GHz, 8vCPU, 4 core, 32GB RAM). All machines run Ubuntu 22.04 LTS, have a 100 GB balanced persistent disk, are located in the same us-central1-a zone, and use Google gVNIC (up to 16Gbps).

6.2 Microbenchmarks

We first evaluate our choice of MPC protocol by comparing the performance of DualEx with both a semi-honest and a fully malicious protocol. Then, we report the costs of basic oblivious algorithms (§ 4.1), mappers and reducers for a generic FGA query (§ 4.1), and shuffling (§ 4.2). These primitive costs provide a basis for estimating the total cost of arbitrary FGA queries, beyond the specific epidemic analytics queries we evaluate in §6.3.

GC vs. DualEx vs. AGMPC. To illustrate the performance tradeoff in the choice of a 2PC protocol, we compare DualEx to two alternatives: (i) a semi-honest 2PC protocol using GCs, which also serves as a building block for DualEx; and (ii) AGMPC, a state-of-the-art fully malicious-secure boolean MPC protocol [107], used in systems like Senate [90]. While AGMPC generally assumes $t = n - 1$, we compare against AGMPC with 2 parties (AG2PC), where $t = 1$ as in DualEx.

We sort (the most expensive query primitive) 10,000 32-bit inputs using two parties. We evaluate AG2PC, semi-honest GCs (a single execution), and DualEx (two symmetric, sequential runs plus the equality check), running each protocol on an Intel TDX and a AMD-SEV VM corresponding to the two parties, with one core per party. In addition, we run DualEx with two cores per party, thereby exploiting the natural concurrency in this protocol by executing the two semi-honest GC concurrent runs in parallel. The results are shown in Figure 3.

With the same number of cores (one per party), DualEx runs twice as long as a semi-honest GC execution, as expected, and DualEx is 11.36x faster than the fully malicious-secure AG2PC. When given extra cores to exploit its inherent parallelism, DualEx is only 8% slower than the semi-honest GCs. These results suggest that DualEx is a compelling choice in scenarios where a 1-bit leak is acceptable.

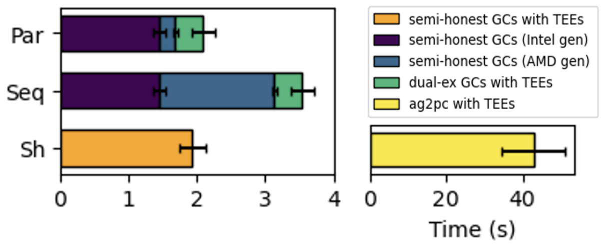

Figure 3: Cost breakdown. *Par* and *Seq* indicate parallel and sequential execution of the two GC computations in DualEx. *Sh* is the semi-honest 2PC execution in TEEs.

Cost breakdown. Figure 3 shows the latency contributions of the individual components in DualEx. The purple and blue parts show the contributions of the two parallel GC executions required for DualEx, one where the Intel CPU plays the role of the generator, and one with the AMD CPU as the generator. (The two latencies differ slightly because the load on the generator is higher, and the Intel CPU is slightly faster; the slower execution determines the total latency.) The green part of the bars indicates the additional latency incurred by executing the two parties in TEEs (SEV-SNP and TDX), along with the result comparison overhead, which is negligible.

Differentially private queries require the addition of appropriate noise to the final result of a query. The cost of adding such noise is negligible, on the order of ms.

Query primitives. Figure 4a shows the execution time for the basic oblivious query primitives from §4.1—linear scans on 32-bit and 256-bit records, sorting, merging two sorted lists, and compaction on 32-bit records—as a function of the list length (input size). Each reported number is an average of 5000 measurements, with std. dev. shown as error bars. The trends are as expected: The cost of linear scans grows linearly

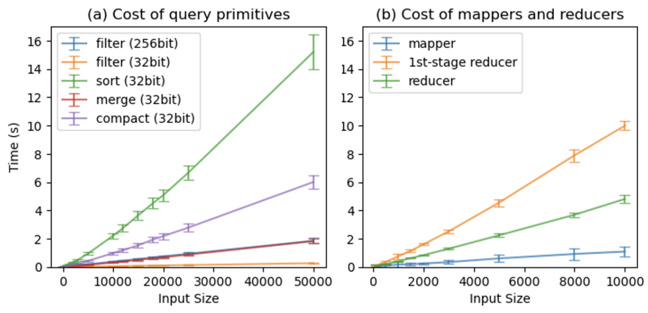

Figure 4: 2PC processing time vs. input size for basic oblivious algorithms and mappers/reducers.

with the input size, sorting, sorted merge, and compaction exhibit slightly super-linear growth. Sorting is significantly more expensive than compaction, which is why our reduce trees sort only in the first stage and merge-compact in subsequent stages.

Oblivious MapReduce. Figure 4b shows the average execution time of typical mappers and reducers on a single core pair with DualEx in TEEs. The specific operations are those of query q2 in Figure 5. The complexities of a mapper, 1st-stage reducer, and subsequent-stage reducer are $O(c)$, $O(c(\log(c))^2)$, and $O(d\log(d))$, respectively, where c is the input size and d is the maximum width of a reducer's result. The 1st-stage reducers are more expensive due to the additional sort operation. The results align with expectations: Map costs are linear in the input size, while reduce costs are slightly super-linear, with the 1st-stage reduction being the most expensive.

Shuffling. Each shuffle requires two sequential oblivious sorts in TEEs of different types, outside 2PC. Shuffling a view of 600M records takes 58min on a single core pair.[3] Shuffling one-tenth the number, 60M records, takes 4.6min. The variances are negligible. The scaling is super-linear since oblivious sort runs in $O(N(\log(N))^2)$ time even outside 2PC.

A shuffle is used only once for a query that requires ODR, but shuffles can be precomputed in parallel. Since a shuffle sorts on one core at a time, a single core pair can produce two different shuffles of 600M records in under 1h—a conservative upper bound on the pairwise encounters generated by a country of 80M people in 1h in our epidemic analytics scenario (§6.3). Thus, for a country of this size, we can prepare shuffles for q queries using $q/2$ core pairs continuously.

Bandwidth. Garbled circuits are streamed from the generator to the evaluator, along with the garbled encoding of the evaluator's inputs. During a series of sort and linear scan operations, the average bandwidth from the generator to the evaluator, measured using NetHogs [3], is ~2.8Gbps. The bandwidth from the evaluator to the generator is negligible in comparison. With DualEx, the average bandwidth is 2.8Gbps in each direction. We use 4 active cores per machine for a bandwidth of 5.6Gbps in each direction, which the GCE gVNIC can support. For comparison, the average bandwidth requirement when using AG2PC is just under 2Gbps in each direction.

6.3 End-to-end scenario: Epidemic analytics

We evaluate CoVault at scale using epidemic analytics as an example scenario. We use DualEx as the underlying 2PC protocol throughout, unless stated otherwise.

Since CoVault uses oblivious algorithms, the actual data values are irrelevant for performance: what matters is the

[3] 600M records exceed the 32GB RAM in our standard VM configuration. So, for the experiment with 600M records, we increased the RAM size to 88GB in the SNP VM, and we used a c3-standard-22 VM instance with 88GB RAM as the TDX VM.

Figure 5: Schema and queries used in epidemic analytics. The selections on the public attributes `space-time-region` and `epoch` are done outside 2PC using the public indexes of T_E and T_P. `R` is a set of space-time-regions.

T_E	*space-time-region*	eid	did1	did2	...		
T_P	*epoch*	did1, time	did2	duration	prev	next	...

(q1) Histogram of #encounters, in space-time regions `R`, of devices in set `A`

```
SELECT HISTO(COUNT(*)) FROM T_E
WHERE did1∈A AND space-time-region∈R
```

(q2) Histogram of #unique devices met, in space-time regions `R`, by each device in set `A`

```
SELECT HISTO(COUNT(DISTINCT(did2))) FROM T_E
WHERE did1∈A AND space-time-region∈R
```

(q3) Count #devices in set `B` that encountered a device in set `A` in the time interval `[start,end]`

```
WITH TT AS
 (SELECT * FROM T_P
  WHERE start<epoch<end)
SELECT COUNT(DISTINCT(did2)) FROM TT
WHERE did1∈A AND did2∈B
      AND start<time<end
```

database schema, the number of records, and the query structure. Hence, our evaluation generalizes to any scenario with similar data sizes and sequence of FGA operations.

For our epidemic analytics scenario, we use *synthetic* location and Bluetooth Low Energy (BLE) radio encounter data—which would normally be collected via smartphones—as detailed in the TR [102] (§C, D.2). Ingress processing produces two materialized views, T_E and T_P, whose schemas are shown in Figure 5. Each view includes a *public*, coarse-grained index, marked with *italics* font. T_E stores pairwise encounters, containing an encounter ID (eid) and the anonymous IDs of the two devices (did1, did2). The public index is keyed with a coarse-grained *space-time region*.

The second view, T_P, contains encounters *privately* indexed by individual device IDs and the times of the encounter reports (did1, time). Each record includes pointers to previous and next encounters, enabling timeline traversal for a given device. The public index on T_P is a coarse-grained *epoch* (~1h) during which an encounter occurred. T_P is accessed via the private index in data-dependent queries; thus, T_P is shuffled as required by the ODR scheme (§4.2).

Our evaluation uses the queries q1–q3 in Figure 5, developed in consultation with an epidemiologist. These queries can help assess the impact of contact restrictions, such as the closure of large events, on the frequency of contacts among people during epidemics. Query q3 can help determine whether two epidemic outbreaks (represented by device sets `A` and `B`) are directly connected by an encounter. A related query, discussed in the TR [102] (§D.4), extends q3 to indirect encounters between `A` and `B` via a third device.

We measured end-to-end query latency for the queries q1–q3 as a function of input size and core count.

Queries q1 and q2. Both q1 and q2 run on the view T_E.

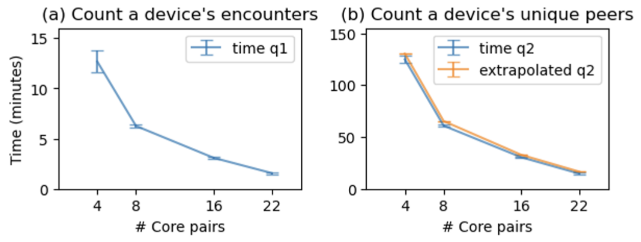

Figure 6: Query latency vs. total number of core pairs (DualEx) for the FGA queries (q1) and (q2) from Figure 5. The measurements are the average of 50 runs, with std. dev. shown as error bars. In (b), we additionally show the performance predicted by our performance model (described below).

The queries fetch only the records in the space-time region R, using the public index of T_E, which significantly reduces the amount of data processed in MPC and hence the query latency. In our synthetic database, the space-time region R contains 28M encounter records, representing a conservative upper bound on encounters generated in a space-time region with 10k data sources reporting 200 encs/day over 14 days. We process these records in 10k-sized shards, empirically minimizing latency. In both queries, we iterate over devices in the given set A. For each device *a* in A we issue a FGA query. For q1, this FGA query counts the encounters of device *a* in R. For q2, the FGA query counts the number of unique devices that *a* met in R. These FGA queries can execute in parallel for all devices in A.

Figure 6a shows the latency of q1 as a function of available core pairs (currently limited to 22 in the GCE TDX private preview). Mappers filter input encounters to those involving the specific user *a*, while reducers aggregate encounter counts. The results show that latency scales almost inversely with the number of core pairs available, showing near-linear horizontal scaling.

Figure 6b shows the latency of q2. The map phase is unchanged, but reducers must merge and deduplicate lists of encountered devices. Again, query latency varies almost inversely with the number of cores, but is higher than for q1, since 1st-stage reducers perform more work in q2.

Query q3. Query q3 runs on the view T_P, and computes the number of devices in set B that directly encountered a device in set A within the time interval `[start,end]`. A naïve 2PC implementation would require a linear scan of all encounter records in this interval to find those between two peers from sets A and B. Based on our earlier estimate, for a table of 80M people and a 14-day period, this would involve loading and filtering 165.9B records *in 2PC*.

Instead, q3 leverages T_P to pre-select relevant epochs in `[start,end]` outside 2PC. For each relevant epoch *e* and device *b* in B, the query traverses *b*'s sequence of encounters by dereferencing `prev` pointers obliviously using ODR, starting from *b*'s latest encounter. Dummy lookups are inserted as needed to hide the actual number of encounters *b* has in *e*. For each such encounter, the query checks if *b*'s time falls in `[start,end]` and if *b*'s peer is in A, using an in-circuit Bloom filter initialized with A. If so, this *b* encountered a device in A. The query repeats this traversal for every *b* in B and all relevant epochs, thus counting the number of devices in B that encountered some device in A.

For A and B of size 10 each and a 14-day period, this approach reduces the number of records processed in 2PC by five orders of magnitude, down to ~100K. The total query latency on 2 core pairs is 2.44 hours, assuming shuffled views (for ODR) for each of the 336 1h epochs are precomputed. Preparing two shuffled views of an epoch took less than one hour on a single core pair (§6.2), and shuffled views can be precomputed upfront and in parallel whenever idle cores are available. Given q3's modest latency using 2 core pairs, we did not parallelize its implementation. However, q3 is highly data-parallel. For instance, "checking whether a given device in B intersects A during a 1h epoch" can be treated as a sequential execution unit, which can be run in parallel on separate cores. This unit takes ∼2.6s in our prototype. The time required to add up the final counts is insignificant by comparison. The parallel sub-computations share one ODR view per epoch, as they access disjoint sets of records.

6.3.1 Estimating query latency at scale

Performance model. So far, we have presented measured results from a modest-sized deployment in GCE with up to 22 core pairs. Due to restrictions in the GCE TDX private preview, additional TDX cores are unavailable to us; at any rate, larger-scale experiments with thousand of cores would also exceed our monetary budget. Instead, we developed a performance model based on detailed measurements of individual mappers, reducers, and query primitives to be able to extrapolate the performance of large deployments. This model accurately predicts our measured query performance for small deployments, as shown in Figure 6(b). Details of the model are in the TR [102], §E.

Scaling to many cores. Using our model, we extrapolate the number of core pairs needed to achieve a target query latency for a given number of input records. For example, if the input was 10x larger (280M records), answering q2's basic query in 10min would need 272 core pairs (e.g., 34 16-CPU machines). Given our measured bandwidth for 2PC (§ 6.2), executing this query with 272 core pairs requires a total bisection bandwidth of ∼200Gbps between pairs of servers hosting the parties. While a datacenter can easily meet this demand among servers in the same rack, a geo-distributed MPC system would require similar bandwidth *across* datacenters! This shows that colocation is a key to scale-out in MPC-based computations.

Scaling to a country. To perform epidemic analytics for an entire country with 80M people, our model predicts that continuously ingesting encounter records (11.85B records/-day) requires 1,660 core pairs running continuously (see the TR [102], §D.3). Furthermore, running q2 on 14 days of records (165.9B total) within 24h requires an additional 1,074 core pairs engaged for those 24h. Looked at differently, a core pair is required for every ∼30,000 citizens at this scale. While this resource cost is substantial, it remains within the capacity of even medium-sized datacenters. Moreover, we believe that the monetary cost is justifiable for high-value analytics like those relevant for public health.

Performance with fully malicious and semi-honest 2PC. All reported query results used DualEx as the MPC protocol. Here, we estimate the performance impact of using AG2PC, a fully malicious-secure protocol, and GCs, a semi-honest 2PC protocol for comparison. To do so, we measured the performance of mappers, reducers, and query primitives under these protocols and used our performance model to extrapolate query performance at scale.

With AG2PC, executing q2 as described above on 14 days of records within 24h requires an estimated 13,000 core pairs, over 12x more than with DualEx. With current state-of-the-art 2PC protocols, this is the cost of a fully malicious threat model that does not admit the 1-bit leak. The bandwidth requirement per core pair remains below 2Gbps on average, peaking below 6Gbps, well within the capacity of a datacenter even at this scale (Google reportedly has more than 1Pbps bisection bandwidth [97]). Thus, CoVault's colocation of parties allows it to scale out regardless of the MPC protocol and enables analytics at scale even in a fully malicious model, provided sufficient datacenter resources.

For comparison, executing the same q2 query within 24h using a semi-honest GC protocol is possible with 537 core pairs (half of the core pairs required for DualEx) since only one of the two parallel semi-honest GC executions is required.

6.4 CoVault versus existing work

Existing secure analytics platforms with active security fall into two broad categories. In *cooperative analytics* platforms like Senate [90], large data aggregators run distributed MPC to analyse their collective, distributed data. As a result, the number of data sources participating in a (sub)query determines the number of MPC parties, and the MPC parties are geo-distributed. The overhead of AGMPC, which is used in Senate, increases super-linearly with the number of parties. Moreover, WAN bandwidth limits Senate's scalability, restricting its analytics to small databases.

The Senate prototype is not available; based on published runtime and network usage results for Senate's *m*-Sort circuit with 16 parties [90, Figs. 5a, 7b], we estimate its average bandwidth during the execution at 4.8Gbps (480GB/800s). We can use this result to extrapolate its total average bandwidth requirement if one tried to perform parallel subqueries on a large dataset. To execute our query q2 on 28M records, Senate would require 1750 (28M/16k) parallel sorts in the first stage, resulting in a total average bandwidth of 8.4Tbps! By contrast, in CoVault, the number of MPC parties is independent of the number of data sources, and the colocation of parties enables scale-out up to the available cores and bisection bandwidth within a single datacenter.

Federated analytics (FA) platforms like Arboretum [78] analyse data stored on millions of individual user devices, using a combination of homomorphic encryption, zero-knowledge proofs, MPC among small committees of user devices, and a centralized untrusted aggregator. We implemented Arboretum's `top1` query in CoVault. According to published results, for 2^{15} categories and 10^9 data sources/inputs, this query takes ∼9h with a 1,000-core aggregator in Arboretum. Using our performance model, we estimate that 250 CoVault SDPs (1,000 cores) running AG2PC (to match Arboretum's active security model) would process this query with a delay in the same order of magnitude.

FA and CoVault differ in qualitative aspects, making the two approaches suitable for different scenarios. With FA, query efficiency, results and repeatability depend on the availability, cooperation, and resource contributions of individual user devices, as well as WAN network bandwidth and delays. However, FA does not require trusted, centralized components. CoVault, in contrast, requires more substantial datacenter resources, but offers repeatable query results and additionally supports data-dependent and iterative queries.

7 Related work

Even though considerable progress has been made towards scaling secure analytics platforms and MPC broadly, no system scales horizontally to billions of records, while meeting the security properties of § 1. For example, many existing systems either provide only passive security [16, 29, 44, 81–83, 86, 100, 103], or do not fully distribute trust [11, 12, 14, 17, 42, 48, 50, 61, 73, 74, 77, 92, 96, 101, 110], and, thus, satisfy the security properties only partially. Here, we compare to closely related work, except federated analytics, which we covered in §6.4.

Collaborative / cooperative analytics refers to MPC-based analytics where the parties are also the data sources. SMCQL [16] and Conclave [103] are only passively secure, while Senate [90] provides active security. Cooperative analytics runs early stages of the computation on subsets of data sources. However, these systems have no support for data-dependent queries, and only support running queries as monolithic cryptographic computations, without horizontal scaling to cores available within each party, which is the primary problem CoVault addresses.

Combining MPC and TEEs. The encapsulation of MPC parties in TEEs has been considered in related work for various purposes but not for scaling MPC horizontally, which is what CoVault does. Encapsulation in TEEs of *different vendors* has been used to bootstrap decentralized trust by enabling stakeholders to audit the MPC configuration [39] and to recover keys securely [32].

Encapsulation of MPC parties of the *same vendor*, or without explicit consideration of TEE heterogeneity, has been used to obtain active security from passively-secure protocols [72], to offload subcomputations to a single party within MPC [108], to obtain a protocol for fair multi-party transactions in blockchains without MPC [95], and to improve trust in code verifiers [98]. Unlike CoVault, these systems cannot tolerate the failure or compromise of any TEE.

Secret-shared data analytics systems aim to decentralize trust. Obscure [56] supports aggregation queries on a secret-shared dataset outsourced by a set of owners; however, it does not support big data and nested queries. Cryptε [29] and GraphSC [82] outsource computation to two untrusted non-colluding servers, but assume only semi-honest adversaries. Waldo [41] uses secret sharing and honest-majority 3PC but focuses on outsourced analytics of *time-series* data from a *single* source. Unlike CoVault, all three systems rely on strong assumptions about the absence of correlated attacks.

TEEs without MPC [11, 14, 17, 50, 61, 73, 74, 77, 92, 96, 101] protect data at rest and in use by decrypting data only inside a TEE. This idea has been combined with *oblivious algorithms* to mitigate side-channel leaks [12, 42, 44, 48, 83, 84, 100, 110]. However, unlike CoVault, these systems do not distribute trust among independent parties. They address orthogonal problems, e.g., the design of efficient oblivious algorithms or TEE-related protocols.

Homomorphic encryption (HE) is an alternative to classic MPC. Full HE is prohibitively expensive [99]. Partial HE (PHE) restricts query expressiveness significantly, and works efficiently only in weak threat models. Seabed [86] works in a semi-honest setting. TimeCrypt [21] and Zeph [22] allow data sources to specify access control preferences (similar to SFC); however, they specifically target time-series data with a restricted set of operations (additions but not multiplications), and Zeph operates in a semi-honest threat model.

Automatic optimization of MPC has been considered in Arboretum and several MPC frameworks [16, 23, 26, 28, 49, 67, 103]. Heuristics automatically optimize the computation's data flow and, in some cases, partition the data flow and select optimal MPC protocols for all partitions. These optimization techniques are orthogonal to CoVault's design and can be applied to its future implementations.

Encrypted databases like CryptDB [91] protect data at rest only. Early work used weak encryption like deterministic or order-preserving encryption, later shown to be inadequate for security [19, 55]. Blind Seer [87] uses strong encryption and 2PC to traverse a specialized index, but leaks information about its search tree traversal. These systems do not protect data in use.

Privacy-preserving data donation systems Anonify [51] provides data-source and attribute anonymity for donated medical records. This system focuses on anonymization, assumes passive security, cannot handle millions of records, and reveals sampled *anonymized* records. In contrast, CoVault focuses on secure computation, executes queries, and reveals only query results (with SFC compliance and support for differential privacy).

8 Conclusion

CoVault relies on a set of colocated MPC parties, each executing in a TEE of a different vendor. Because CoVault is not impeded by the limits of WAN communication during query processing, it can scale out and leverage the resources of a datacenter to support big data analytics over data from individual sources, with distributed trust and malicious security. CoVault scales out to epidemic analytics queries for a country of 80M people and billions of records at a cost of a core pair for every 30,000 people.

9 Acknowledgments

We thank Gilles Barthe, Jonathan Katz, Matthew Lentz, Chang Liu, and Elaine Shi for discussions and contributions to early versions of this work; Hongjie Liu for advice on relevant epidemiological queries; Xiao Wang and Chenkai Weng for their help with emp-toolkit; Lorenzo Alvisi, Natacha Crooks, Aastha Mehta, Lillian Tsai, Vaastav Anand, and several anonymous reviewers for their feedback on earlier drafts of this paper. Finally, we are grateful to the anonymous shepherd and reviewers of Usenix Security 2025 for their insightful feedback and suggestions. This work was supported in part by the Max Planck Society, the US National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1840340, the European Research Council (ERC Synergy imPACT 610150), and the German Science Foundation (DFG CRC 1223).

10 Ethics Considerations

We carefully considered ethical implications throughout the development of CoVault. In fact, the motivation of our work is based on the principle of public welfare without compromising security (§ 1).

Our design includes explicit user consent (which we call selective forward consent or SFC). It provides full transparency and control to data owners in terms of how, by whom and until when their data can be used. We integrate technical means to enforce SFC with TEEs, and Appendix A of our TR explains how TEE attestation, which is a building block, can be made practical using community processes.

Thanks to the data obliviousness of MPC circuits, we did not need any real data to obtain accurate performance measurements. All our experiments are based on synthetic data that we generated ourselves.

11 Open Science

This paper is accompanied by an open-source artifact, which includes the code for the experiments in § 6, as well as the scripts to execute the code and produce results. The prototype uses synthetic datasets that are generated by the code. The results reported in the paper were obtained by running the scripts in Google Cloud Compute Engine following the setup described in § 6.1. The artifact is available at https://zenodo.org/records/14736568.

References

[1] GitHub: AMDSEV (sev-snp-devel). https://github.com/AMDESE/AMDSEV/tree/sev-snp-devel. Accessed: 2025-01-20.

[2] Github: FastGC. https://github.com/yhuang912/FastGC. Accessed: 2025-01-20.

[3] GitHub: Nethogs. https://github.com/raboof/nethogs. Accessed: 2025-01-20.

[4] Council of the EU. Council approves Data Governance Act. https://tinyurl.com/consilium-europa-dga, 2022. Accessed: 2025-01-20.

[5] Microsoft Azure: What is guest attestation for confidential VMs? https://learn.microsoft.com/en-us/azure/confidential-computing/guest-attestation-confidential-vms, 2022. Accessed: 2025-01-20.

[6] Google Cloud: Confidential VM attestation. https://cloud.google.com/confidential-computing/confidential-vm/docs/attestation, 2024. Accessed: 2025-01-20.

[7] Google Cloud: Confidential VM overview. https://cloud.google.com/confidential-computing/confidential-vm/docs/confidential-vm-overview, 2024. Accessed: 2025-01-20.

[8] Microsoft Azure: Azure Confidential VM options. https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-options, 2024. Accessed: 2025-01-20.

[9] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020. Accessed: 2025-01-20.

[10] S. Angel, A. Basu, W. Cui, T. Jaeger, S. Lau, S. T. V. Setty, and S. Singanamalla. Nimble: Rollback Protection for Confidential Cloud Services. In R. Geambasu and E. Nightingale, editors, *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023*, pages 193–208. USENIX Association, 2023.

[11] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security with Cipherbase. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013*. www.cidrdb.org, 2013.

[12] A. Arasu and R. Kaushik. Oblivious query processing. In N. Schweikardt, V. Christophides, and V. Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), 2014*, pages 26–37. OpenProceedings.org, 2014.

[13] ARM. ARM Confidential Compute Architecture (CCA). https://developer.arm.com/architectures/architecture-security-features/confidential-computing. Accessed 2025-01-20.

[14] S. Bajaj and R. Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011*, pages 205–216. ACM, 2011.

[15] K. E. Batcher. Sorting Networks and Their Applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[16] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. SMCQL: Secure Query Processing for Private Data Networks. *Proc. VLDB Endow.*, 10(6):673–684, 2017.

[17] A. Baumann, M. Peinado, and G. C. Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26, 2015.

[18] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 1988.

[19] V. Bindschaedler, P. Grubbs, D. Cash, T. Ristenpart, and V. Shmatikov. The Tao of Inference in Privacy-Protected Databases. *Proc. VLDB Endow.*, 11(11):1715–1728, 2018.

[20] S. Bugiel, S. Nürnberger, A. Sadeghi, and T. Schneider. Twin Clouds: Secure Cloud Computing with Low Latency - (Full Version). In B. De Decker, J. Lapon, V. Naessens, and A. Uhl, editors, *Communications and Multimedia Security, 12th IFIP TC 6 / TC 11 International Conference, CMS 2011. Proceedings*, volume 7025 of *Lecture Notes in Computer Science*, pages 32–44. Springer, 2011.

[21] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy. TimeCrypt: Encrypted Data Stream Processing at Scale with Cryptographic Access Control. In R. Bhagwan and G. Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020*, pages 835–850. USENIX Association, 2020.

[22] L. Burkhalter, N. Küchler, A. Viand, H. Shafagh, and A. Hithnawi. Zeph: Cryptographic Enforcement of End-to-End Data Privacy. In A. D. Brown and J. R. Lorch, editors, *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021*, pages 387–404. USENIX Association, 2021.

[23] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider. HyCC: Compilation of Hybrid Protocols for Practical Secure Computation. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 847–861. ACM, 2018.

[24] L. Carter and M. N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[25] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto. SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 1416–1432. IEEE, 2020.

[26] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi. EzPC: Programmable and Efficient Secure Two-Party Computation for Machine Learning. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019*, pages 496–511. IEEE, 2019.

[27] M. Chase, E. Ghosh, and O. Poburinnaya. Secret-Shared Shuffle. In S. Moriai and H. Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 342–372. Springer, 2020.

[28] E. Chen, J. Zhu, A. Ozdemir, R. S. Wahby, F. Brown, and W. Zheng. Silph: A Framework for Scalable and Accurate Generation of Hybrid MPC Protocols. In *44th IEEE Symposium on Security and Privacy, SP 2023*, pages 848–863. IEEE, 2023.

[29] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proc. SIGMOD*. ACM, 2020.

[30] European Commission. European Data Governance Act. https://digital-strategy.ec.europa.eu/en/policies/data-governance-act. Accessed 2025-01-20.

[31] European Commission. European Data Strategy. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-data-strategy_en. Accessed 2025-01-20.

[32] G. Connell, V. Fang, R. Schmidt, E. Dauterman, and R. Ada Popa. Secret Key Recovery in a Global-Scale End-to-End Encryption System. In A. Gavrilovska and D. B. Terry, editors, *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024*, pages 703–719. USENIX Association, 2024.

[33] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In A. Akella and J. Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pages 259–282. USENIX Association, 2017.

[34] H. Corrigan-Gibbs and D. Kogan. Private Information Retrieval with Sublinear Online Time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 44–75. Springer, 2020.

[35] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptol. ePrint Arch.*, page 86, 2016.

[36] R. Cramer, I. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[37] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal Service-Providers for Private Information Retrieval. *J. Cryptol.*, 14(1):37–74, 2001.

[38] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[39] E. Dauterman, V. Fang, N. Crooks, and R. A. Popa. Reflections on trusting distributed trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets 2022*, pages 38–45. ACM, 2022.

[40] E. Dauterman, V. Fang, I. Demertzis, N. Crooks, and R. A. Popa. Snoopy: Surpassing the scalability bottleneck of oblivious storage. In R. van Renesse and N. Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Koblenz, Germany, October 26-29, 2021*, pages 655–671. ACM, 2021.

[41] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica. Waldo: A Private Time-Series Database from Function Secret Sharing. In *43rd IEEE Symposium on Security and Privacy, SP 2022*, pages 2450–2468. IEEE, 2022.

[42] A. Dave, C. Leung, R. A. Popa, J. E. Gonzalez, and I. Stoica. Oblivious coopetitive analytics using hardware enclaves. In A. Bilas, K. Magoutis, E. P. Markatos, D. Kostic, and M. I. Seltzer, editors, *EuroSys '20: Fifteenth EuroSys Conference 2020*, pages 39:1–39:17. ACM, 2020.

[43] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In E. A. Brewer and P. Chen, editors, *6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150. USENIX Association, 2004.

[44] T. T. A. Dinh, P. Saxena, E. Chang, B. C. Ooi, and C. Zhang. M2R: enabling stronger privacy in mapreduce computation. In J. Jung and T. Holz, editors, *24th USENIX Security Symposium, USENIX Security 2015*, pages 447–462. USENIX Association, 2015.

[45] D. Doerner, J. Mechler, and J. Müller-Quade. Hardening the Security of Server-Aided MPC Using Remotely Unhackable Hardware Modules. In C. Wressnegger, D. Reinhardt, T. Barber, B. C. Witt, D. Arp, and Z. Mann, editors, *Sicherheit, Schutz und Zuverlässigkeit: Konferenzband der 11. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), Sicherheit 2022, Karlsruhe, Germany, April 5-8, 2022*, volume P-323 of *LNI*, pages 83–99. Gesellschaft für Informatik e.V., 2022.

[46] J. Doerner and A. Shelat. Scaling ORAM for Secure Computation. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 523–535. ACM, 2017.

[47] S. Eskandarian and D. Boneh. Clarion: Anonymous Communication from Multiparty Shuffling Protocols. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society, 2022.

[48] S. Eskandarian and M. Zaharia. ObliDB: Oblivious Query Processing for Secure Databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[49] V. Fang, L. Brown, W. Lin, W. Zheng, A. Panda, and R. A. Popa. CostCO: An automatic cost modeling framework for secure multi-party computation. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022*, pages 140–153. IEEE, 2022.

[50] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A. Sadeghi. HardIDX: Practical and secure index with SGX in a malicious environment. *J. Comput. Secur.*, 26(5):677–706, 2018.

[51] S. A. Gaballah, L. Abdullah, M. Alishahi, T. H. L. Nguyen, E. Zimmer, M. Mühlhäuser, and K. Marky. Anonify: Decentralized Dual-level Anonymity for Medical Data Donation. *Proc. Priv. Enhancing Technol.*, 2024(3):94–108, 2024.

[52] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987*, pages 218–229. ACM, 1987.

[53] O. Goldreich and R. Ostrovsky. Software Protection and Simulation on Oblivious RAMs. *J. ACM*, 43(3):431–473, 1996.

[54] M. T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In R. Rajaraman and F. Meyer auf der Heide, editors, *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pages 379–388. ACM, 2011.

[55] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why Your Encrypted Database Is Not Secure. In A. Fedorova, A. Warfield, I. Beschastnikh, and R. Agarwal, editors, *Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS 2017*, pages 162–168. ACM, 2017.

[56] P. Gupta, Y. Li, S. Mehrotra, N. Panwar, S. Sharma, and S. Almanee. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *Proc. VLDB Endow.*, 12(9):1030–1043, 2019.

[57] B. Hemenway, D. Noble, R. Ostrovsky, M. Shtepel, and J. Zhang. DORAM Revisited: Maliciously Secure RAM-MPC with Logarithmic Overhead. In G. N. Rothblum and H. Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Proceedings, Part I*, volume 14369 of *Lecture Notes in Computer Science*, pages 441–470. Springer, 2023.

[58] W. L. Holland, O. Ohrimenko, and A. Wirth. Efficient Oblivious Permutation via the Waksman Network. In Y. Suga, K. Sakurai, X. Ding, and K. Sako, editors, *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security*, pages 771–783. ACM, 2022.

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

[60] Y. Huang, J. Katz, and D. Evans. Quid-Pro-Quo-tocols: Strengthening Semi-honest Protocols with Dual Execution. In *IEEE Symposium on Security and Privacy, SP 2012*, pages 272–284. IEEE Computer Society, 2012.

[61] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Trans. Comput. Syst.*, 35(4):13:1–13:32, 2017.

[62] Intel. Intel Trust Domain Extensions (Intel TDX). https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf. Accessed 2025-01-20.

[63] Intel. Introduction to Intel SGX Sealing. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html. Accessed 2025-01-20.

[64] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi. A Framework for Outsourcing of Secure Computation. In G. Ahn, A. Oprea, and R. Safavi-Naini, editors, *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14*, pages 81–92. ACM, 2014.

[65] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. CRC Press, 2014.

[66] J. Katz, S. Ranellucci, M. Rosulek, and X. Wang. Optimizing Authenticated Garbling for Faster Secure Two-Party Computation. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 365–391. Springer, 2018.

[67] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1575–1590. ACM, 2020.

[68] J. Kelsey, S. Chang, and R. Perlner. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016. NIST Special Publication 800-185.

[69] F. Kerschbaum. Oblivious outsourcing of garbled circuit generation. In R. L. Wainwright, J. M. Corchado, A. Bechini, and J. Hong, editors, *Proceedings of the 30th Annual ACM Symposium on Applied Computing, 2015*, pages 2134–2140. ACM, 2015.

[70] V. Kolesnikov, P. Mohassel, B. Riva, and M. Rosulek. Richer Efficiency/Security Trade-offs in 2PC. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 229–259. Springer, 2015.

[71] H. Krawczyk. LFSR-based Hashing and Authentication. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.

[72] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma. CrypTFlow: Secure TensorFlow Inference. In *2020 IEEE Symposium on Security and Privacy, SP 2020*, pages 336–353. IEEE, 2020.

[73] OASIS labs. A better way to Contact Trace, Part I. https://medium.com/oasislabs/a-better-way-to-contact-trace-7beb12889017. Accessed 2025-01-20.

[74] OASIS labs. A better way to Contact Trace, Part II. https://medium.com/oasislabs/a-better-way-to-contact-trace-part-ii-code-to-back-it-up-50046c4fa6e1. Accessed 2025-01-20.

[75] S. Laur, J. Willemson, and B. Zhang. Round-Efficient Oblivious Database Manipulation. In X. Lai, J. Zhou, and H. Li, editors, *Information Security, 14th International Conference, ISC 2011. Proceedings*, volume 7001 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2011.

[76] M. Li, Y. Yang, G. Chen, M. Yan, and Y. Zhang. SoK: Understanding Design Choices and Pitfalls of Trusted Execution Environments. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, ASIA CCS 2024*. ACM, 2024.

[77] U. Maheshwari, R. Vingralek, and W. Shapiro. How to Build a Trusted Database System on Untrusted Storage. In M. B. Jones and M. F. Kaashoek, editors, *4th Symposium on Operating System Design and Implementation (OSDI 2000)*, pages 135–150. USENIX Association, 2000.

[78] E. Margolin, K. Newatia, T. Luo, E. Roth, and A. Haeberlen. Arboretum: A planner for large-scale federated analytics with differential privacy. In J. Flinn, M. I. Seltzer, P. Druschel, A. Kaufmann, and J. Mace, editors, *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023*, pages 451–465. ACM, 2023.

[79] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. M. Sommer, A. Gervais, A. Juels, and D. Capkun. ROTE: rollback protection for trusted execution. In E. Kirda and T. Ristenpart, editors, *26th USENIX Security Symposium, USENIX Security 2017*, pages 1289–1306. USENIX Association, 2017.

[80] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa. Oblix: An Efficient Oblivious Search Index. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 279–296. IEEE Computer Society, 2018.

[81] P. Mohassel and Y. Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 19–38. IEEE Computer Society, 2017.

[82] K. Nayak, X. Shaun Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel Secure Computation Made Easy. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 377–394. IEEE Computer Society, 2015.

[83] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma. Observing and Preventing Leakage in MapReduce. In I. Ray, N. Li, and C. Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015*, pages 1570–1581. ACM, 2015.

[84] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In T. Holz and S. Savage, editors, *25th USENIX Security Symposium, USENIX Security 2016*, pages 619–636. USENIX Association, 2016.

[85] F. G. Olumofin and I. Goldberg. Privacy-Preserving Queries over Relational Databases. In M. J. Atallah and N. J. Hopper, editors, *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010. Proceedings*, volume 6205 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2010.

[86] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big Data Analytics over Encrypted Datasets with Seabed. In K. Keeton and T. Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, pages 587–602. USENIX Association, 2016.

[87] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. D. Keromytis, and S. M. Bellovin. Blind Seer: A Scalable Private DBMS. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 359–374. IEEE Computer Society, 2014.

[88] B. Parno, J. R. Lorch, J. R. Douceur, J. W. Mickens, and J. M. McCune. Memoir: Practical State Continuity for Protected Modules. In *32nd IEEE Symposium on Security and Privacy, SP 2011*, pages 379–394. IEEE Computer Society, 2011.

[89] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure Two-Party Computation Is Practical. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[90] R. Poddar, S. Kalra, A. Yanai, R. Deng, R. A. Popa, and J. M. Hellerstein. Senate: A maliciously-secure MPC platform for collaborative analytics. In M. D. Bailey and R. Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021*, pages 2129–2146. USENIX Association, 2021.

[91] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In T. Wobber and P. Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011*, pages 85–100. ACM, 2011.

[92] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings*, pages 264–278. IEEE Computer Society, 2018.

[93] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In D. S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, 1989*, pages 73–85. ACM, 1989.

[94] M. Rathee, C. Shen, S. Wagh, and R. A. Popa. ELSA: secure aggregation for federated learning with malicious actors. In *44th IEEE Symposium on Security and Privacy, SP 2023*, pages 1961–1979. IEEE, 2023.

[95] Q. Ren, Y. Li, Y. Wu, Y. Wu, H. Lei, L. Wang, and B. Chen. DeCloak: Enable Secure and Cheap Multi-Party Transactions on Legacy Blockchains by a Minimally Trusted TEE Network. *IEEE Trans. Inf. Forensics Secur.*, 19:88–103, 2024.

[96] F. Schuster, M. Costa, C., C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 38–54. IEEE Computer Society, 2015.

[97] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, H. Liu, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter rising: a decade of clos topologies and centralized control in Google's datacenter network. *Commun. ACM*, 59(9):88–97, 2016.

[98] TOKI. Multi-prover for IBC Connection: Enhancing Security with a Combination of SGX, MPC and ZKP. https://medium.com/@tokifinance/multi-prover-for-ibc-connection-169862263739. Accessed 2025-01-20.

[99] A. Viand, P. Jattke, and A. Hithnawi. SoK: Fully Homomorphic Encryption Compilers. In *42nd IEEE Symposium on Security and Privacy, SP 2021*, pages 1092–1108. IEEE, 2021.

[100] D. Vinayagamurthy, A. Gribov, and S. Gorbunov. StealthDB: a Scalable Encrypted Database with Full SQL Query Support. *Proc. Priv. Enhancing Technol.*, 2019(3):370–388, 2019.

[101] R. Vingralek. GnatDb: A Small-Footprint, Secure Database System. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002*, pages 884–893. Morgan Kaufmann, 2002.

[102] R. De Viti, I. Sheff, N. Glaeser, B. Dinis, R. Rodrigues, B. Bhattacharjee, A. Hithnawi, D. Garg, and P. Druschel. CoVault: Secure, Scalable Analytics of Personal Data. *CoRR*, abs/2208.03784, 2022. Last revised in 2025.

[103] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. Conclave: secure multi-party computation on big data. In G. Candea, R. van Renesse, and C. Fetzer, editors, *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 3:1–3:18. ACM, 2019.

[104] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical Private Queries on Public Data. In A. Akella and J. Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pages 299–313. USENIX Association, 2017.

[105] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016. Accessed: 2025-01-20.

[106] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 21–37. ACM, 2017.

[107] X. Wang, S. Ranellucci, and J. Katz. Global-Scale Secure Multiparty Computation. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*, pages 39–56. ACM, 2017.

[108] P. Wu, J. Ning, J. Shen, H. Wang, and E. Chang. Hybrid Trust Multi-party Computation with Trusted Execution Environment. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society, 2022.

[109] A. C. Yao. Protocols for Secure Computations. In *23rd Annual Symposium on Foundations of Computer Science, 1982*, pages 160–164. IEEE Computer Society, 1982.

[110] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In A. Akella and J. Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, pages 283–298. USENIX Association, 2017.