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 Tecnico (ULisboa), INESC-ID
[4]University of Maryland
[5]ETH Zürich

Abstract

Analytics on personal data, such as individuals' mobility, financial, and health data can be of significant benefit to society. Such data is already collected by smartphones, apps and services today, but liberal societies have so far refrained from making it available for large-scale analytics. Arguably, this is due at least in part to the lack of an analytics platform that can secure data through transparent, technical means (ideally with decentralized trust), enforce source policies, handle millions of distinct data sources, and run queries on billions of records with acceptable query latencies. To bridge this gap, we present an analytics platform called CoVault, which combines secure multi-party computation (MPC) with trusted execution environment (TEE)-based delegation of trust to be able execute approved queries on encrypted data contributed by individuals within a datacenter to achieve the above properties. We show that CoVault scales well despite the high cost of MPC. For example, CoVault can process data relevant to epidemic analytics for a country of 80M people (about 11.85B data records/day) on a continuous basis using a core pair for every 20,000 people. Compared to a state-of-the-art MPC-based platform, CoVault can process queries between 7 to over 100 times faster, as well as scale to many sources and big data.

1 Introduction

Personal data about individuals' health, nutrition, activity, mobility, social contacts, and finances are being captured at high resolution by individuals' smart devices and apps, by online services, and by offline services that involve digital record-keeping such as hospitals and banks. Large-scale *analytics* 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 [27,28]. To this end, the EU Parliament recently approved the Data Governance Act (DGA) [3], 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.

Requirements. A fundamental requirement for analytics of sensitive data is *security*. Secure analytics has specific security requirements like *selective consent* by data owners as to who gets to use their data for what purpose and for how long, *confidentiality* of contributed data, and *integrity* of query results.[1] The DGA relies on *data intermediaries* who act as trusted data clearinghouses that ensure security. We believe that the security requirements should be enforced by *strong technical means* that do not place trust in individual intermediaries, in order to justify the public's trust and encourage contributions of sensitive data. An additional requirement for secure analytics is *scalability*, ideally to data sources numbering in the hundreds of millions, and data records numbering in the hundreds of billions, without restricting the class of computable queries.

[*] Work done at MPI-SWS (post-doc); currently affiliated with Heliax.

[1] A related requirement is the *privacy* of individuals in query results. However, privacy is a property of queries that is enforced by restricting allowed queries to those that satisfy a strong criterion like differential privacy. CoVault supports restrictions on allowed queries through its query class mechanism.

1.1 State-of-the-art in secure analytics

Considerable progress has been made in recent years towards platforms that enable analytics with strong security and distributed trust. Many of them rely on *secure multi-party computation (MPC)* [43] as a building block. We focus on *actively* (or *maliciously*) secure MPC, which protects against compromised parties that can arbitrarily deviate from the protocol.

Cooperative analytics platforms like Senate [78] perform MPC among a set of large data aggregators to support analytics queries and can tolerate up to $m-1$ of m malicious data aggegators. The data aggregators act as the parties in the MPC, and can therefore naturally enforce selective consent, confidentiality, and integrity of the data they contribute. Cooperative analytics matches perfectly scenarios where individuals already share their data with large organizations such as health providers, banks, or online services they trust and can play the role of the MPC parties. A remaining challenge, however, is supporting contributions of sensitive data by *individuals* without requiring them to trust an aggregator with their sensitive data. It remains a challenge because MPC among data sources does not scale to more than a few dozen sources.

Federated analytics platforms like Mycelium and Arboretum [65, 82] rely on MPC, homomorphic encryption (HE), and zero-knowledge proofs (ZKP) to support secure queries over data stored on billions of smart devices. This is a good match for data contributions by individuals who participate in queries with their own devices. Here, a remaining challenge is that the query results depend on the set of devices that happen to be available at query execution time, making the approach unsuitable for analytics where data sampling is insufficient, like the analysis of rare phenomena.

An additional remaining challenge is scale-out to big data and complex queries. In both cooperative and federated analytics, scale-out is ultimately limited by the latency and bandwidth of wide-area network (WAN) communication, which results from the need to separate MPC parties physically in order to justify their independence. MPC protocols, particularly those that tolerate WAN delays, require high bandwidth, and we will show that scaling out queries quickly exceeds the available bandwidth in a WAN.

Other existing platforms either operate in the weaker *passively secure* (or *semi-honest*) threat model (which protects against partially compromised parties that follow the protocol correctly but try to infer secrets from data they obtain) [14, 25, 38, 48, 68, 69, 71, 74, 91], do not distribute trust [8, 9, 11, 16, 36, 42, 45, 55, 59, 60, 63, 80, 85, 90, 98], support only time-series data [18, 19, 35], or are limited to small data [50].

In this paper, we present *CoVault*, a secure analytics platform that (1) supports complete (not sampled) query results (like cooperative analytics); (2) can operate on data from large numbers of individuals (like federated analytics); and (3) scales to big data sets by leveraging the resources and scalability of a single data center, while offering distributed trust and malicious security. Table 1 compares CoVault with the closest state-of-the-art systems.

Table 1: CoVault versus state-of-the-art.

System	Federated analytics ([65, 82])	Cooperative analytics ([78])	CoVault
Technology	HE+MPC+ZKP	MPC	MPC+TEEs
Malicious parties	$<3\%$ user devices	$m-1$ of m aggregators	1 out of 2 TEEs
Data sources (offline)	Many individuals No	Few large aggregators No	Many individuals Yes
Scaling ultimately limited by	Compute/network of user devices	Wide-area network bandwidth/delay	Resources of a datacenter

1.2 Key insights

Any MPC system assumes the independence and non-collusion of its parties for data security, up to the MPC protocol's fault threshold. In cooperative analytics systems, this independence is not a concern because the data sources assume the role of the MPC parties. But because the overhead of MPC protocols is super-linear in the number of parties, the approach can support only a modest number of data sources. Federated analytics platforms select a small random subset of user devices to serve as MPC parties. This approach decouples the number of MPC parties from the number of data sources; as a consequence, however, the system can tolerate only a small proportion of malicious devices, low enough to ensure that any small random subset has a diminishing probability of including more than the MPC protocol's threshold of malicious parties. Moreover, in both types of systems the geo-distribution of parties places latency and bandwidth limitations on query (MPC) execution and ultimately limits the size of databases and the complexity of queries that can be executed in practice. **Our key insight is that one can overcome these challenges by performing MPC among a small set of parties colocated in a single datacenter, where each party is isolated in a trusted execution environment (TEE) from a different, independent vendor**. This has two consequences:

1) MPC on multiple independent TEEs obviates the need for physical separation of parties. MPC assumes that parties are independent in the sense that their failures/compromise are not correlated. Usually, one uses administrative and geographic separation of the parties as the basis for this independence. We observe that the required independence can also be achieved by running the parties inside attested TEEs independent vendors. Thus, an attacker would have to simultaneously compromise a threshold number of different TEEs to compromise the MPC. As a result, MPC security no longer relies on physical and administrative separation of parties, enabling them to run in a single datacenter. The high bandwidth and low latency within the datacenter along with its computational resources in turn enables MPC to perform more complex queries on larger data than previously possible.[2]

2) TEEs enable data sources to safely entrust their data to a separate set of MPC parties, thereby decoupling the two. In § 1.3, we discuss how data sources can safely entrust their data to a separate set of TEE-isolated parties. As a result, the number of MPC parties can be chosen to meet the needs of the MPC protocol independent of the number of data sources. Moreover, the data sources no longer need to take part in query processing.

1.3 Delegating trust

Compared to a platform where the data sources act as the MPC parties, entrusting data processing to a set of TEE-isolated parties poses several technical challenges:

Authenticating parties. Since the data sources do not themselves act as parties, they need to ascertain that the (separate) parties are legitimate, i.e., they each execute in an authentic TEE of a different type and run the expected code and initial configuration. This can be accomplished using the remote attestation capability of TEEs [31, 67].

Input integrity. When data sources act as MPC parties, each source provides its contributed data directly as input to the multi-party computation, which is sufficient to ensure the data's integrity and confidentiality. With trust delegation, on the other hand, data must be shared in such a way that selective consent, confidentiality and integrity of a data source's contributed data does not depend on any one TEE. We use an authenticated secret-sharing scheme, where each data source secret-shares its contributed data, and sends a different share to each of the parties.

Selective *forward* consent (SFC). When each data source acts as an MPC party, it can prevent the misuse of its data by refusing to participate in queries that it does not approve of. With trust delegation, on the other hand, each data source must be able to ensure *upfront* that the MPC parties will perform only approved queries on behalf of approved analysts, and before their data expires. Towards this end, CoVault's parties include code to enforce the conditions data sources have consented to. The sources encrypt their data in such a way that only legitimate TEEs that execute the expected code can decrypt their data.

Who attests TEEs and verifies their code? In an ideal world, every data source would verify and attest the code and initial configuration of TEEs before contributing its data. However, individuals mostly lack the technical expertise and resources to verify that the code run by the MPC parties implements only the queries they approve and ensures the integrity and confidentiality of their contributed data. We envision a transparent *community process* where a group of *community-appointed experts* verify and attest the TEEs' code and initial configurations on behalf of data sources, and publish the outcomes. Individual data sources who lack the ability to verify and attest TEEs rely on the attestations of subsets of experts they trust. In the rest of this paper, we tacitly assume the existence of such a community process.

In summary, the combination of MPC, authenticated secret sharing, and a community review process allows individual data sources to delegate trust to a small set of TEE-encapsulated parties within a single datacenter. In § 5, we combine this secure core functionality with an oblivious data retrieval scheme and oblivious MapReduce to yield a scalable analytics platform.

1.4 CoVault prototype

While CoVault's design is not tied to any particular malicious-secure MPC protocol or number of parties, our prototype uses two parties, of which one may be compromised. Even with just two parties, having to compromise two types of TEEs presents a formidable challenge to an adversary. For MPC, it uses Boolean garbled circuits evaluated with the DualEx protocol [54], which, to our knowledge, is the fastest known actively secure, boolean 2PC protocol. We modify the protocol slightly to output the query result to the analyst rather than to the parties, and prove our modification secure (§ 4.3). Configurations with more than two parties are currently not of interest to CoVault due to the limited availability of different TEEs on the market and the lack of maliciously secure MPC protocols as efficient as DualEx for more than two parties.

1.5 Contributions

Our contributions include (a) an MPC configuration where each party executes in a different, independent TEE, which obviates the need for administrative and physical separation of parties and enables MPC within a single datacenter with high bandwidth links; (b) a set of techniques that allow the delegation of trust to a set of TEE-encapsulated parties while ensuring authentication, attestation, input integrity, confidentiality, and selective forward consent; (c) the design of CoVault, a secure analytics platform where data sources upload encrypted data to a small set of diverse TEEs, which compute analytics queries using MPC. The design enables secure analytics on encrypted data contributed by individuals at a scale that is limited only by the resources of a datacenter; (d) two technical components of possibly independent interest: A provably secure authenticated secret-sharing scheme (§ 4.2) and an actively secure 2PC protocol that reveals information to a third entity, not the computing parties (§ 4.3); (e) an experimental evaluation with epidemic analytics as an example

[2] We note that colocation centers can provide physical separation without geo-distribution. However, without a system like CoVault, hosting MPC in a colocation center still requires administrative independence of the tenants that host the parties.

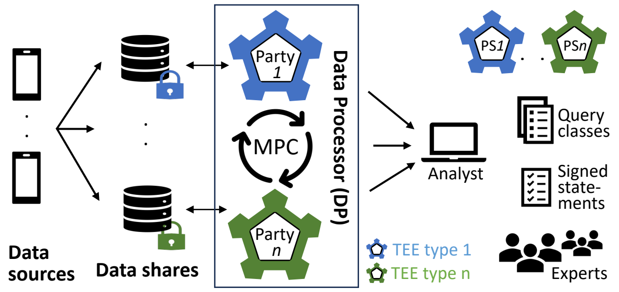

Figure 1: CoVault core functionality. *Conceptually*, data sources encrypt their data in such a way that only queries within a given query class can be executed on their data, by analysts authorized by the query class, and for a period defined by the query class. *Concretely*, (a) Every data source secret-shares its data into n authenticated shares, encrypts each share for a different party, checks (see step (c)) that the parties have been verified, and uploads the shares to an untrusted DB in a datacenter. (b) Queries are processed by a data processor (DP) hosted in the same datacenter. The DP consists of n parties, each encapsulated by a different TEE type, that access their corresponding shares of the data, jointly perform MPC and deliver their respective shares of the query result to an authorized analyst. (c) Provisioning servers (PSs), one per party and implemented in a TEE of the party's type, attest and provision all DPs of a party with their keys. Community experts attest the PSs and DPs, verify their code and configuration, and publish signed statements.

scenario, which shows that CoVault can handle data for a country of 80M using a core pair per 20,000 people. Moreover, we show that CoVault is 7 to over 100 times faster on than existing MPC-based platforms and not limited by WAN bandwidth.

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 be obtained only if a sufficient fraction of shares (possibly all) are combined. *Authenticated* secret sharing additionally adds auxiliary information that allows parties to verify that a value they have reconstituted from their shares is authentic.

Secure multi-party computation (MPC) allows several parties to jointly compute a function without revealing their respective inputs. MPC protocols are either passively secure (corrupt parties are semi-honest [53, 97]) or actively secure (corrupt parties may be malicious [94]). In t-of-n active security, up to t of the n parties may be malicious. The value of t varies from $n/3$ to $n-1$ depending on the protocol. Garbled circuits (GCs) are a specific passively secure 2PC technique ($n = 2, t = 1$) [77]. GCs are inherently data oblivious as circuits have no control flow.

Trusted execution environments (TEEs) are supported by many recent general-purpose CPUs, e.g., Intel SGX, AMD SEV, and the upcoming Intel TDX and ARM CCA [5, 31, 86, 87]. TEEs provide confidentiality and integrity of data and computation under a threat model that tolerates compromised operating systems, hypervisors, and even some physical attacks [21, 31]. Unlike the first-generation SGX, which was designed for client-side applications, the newer SEV-SNP, TDX, and CCA encapsulate an entire VM, providing easy migration of software. 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). Based on the 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 core functionality through a series of strawman designs in § 3 and present the detailed design in § 4. Figure 1 summarizes how CoVault's core functionality is provided. In § 5, we show how to scale out CoVault using oblivious MapReduce [37], where each mapper and reducer is an instance of the DP shown in Figure 1. In § 6, we evaluate CoVault in the context of epidemic analytics as an example application scenario, and compare its performance to the state-of-the-art. The appendix offers more details on CoVault's community approval process (§ A), oblivious map-reduce technique (§ B), ingress processing (§ C), and epidemic analytics scenario (§ D), with additional evaluation material. Furthermore, the appendix includes our algorithm to estimate end-to-end query latency (§ E) and proofs of security of the primitives we use in CoVault (§ F).

Threat model. CoVault relies on MPC with t-of-n active security in a static corruption model. The code of the n parties runs 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 including hardware backdoors, exploits and side channels.

Additionally, CoVault relies on community experts to check and attest the code and initial states of the parties' TEEs. These experts are trusted to a limited extent. Specifically, a rogue (or mistaken) community expert who attests incorrect code may compromise the data confidentiality and consent of data sources who rely on that expert's attestations. Confidentiality and consent of sources who did not rely on rogue experts stays intact. Dually, if an analyst depends on the attestations of a rogue expert, then the integrity of query results provided to that analyst may not hold. The integrity of query results returned to other analysts is unaffected.

We assume that data sources follow the data contribution protocol correctly 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.

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

3 CoVault core design

In this section, we present the design of CoVault's core functionality, beginning with the API and then incrementally refining a strawman design. The subsequent § 4 presents the details of a 2PC implementation of the CoVault 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 instance'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 wish to realize this API satisfying three properties:

SFC/confidentiality. Contributed data remains confidential. SFC: Only authorized analysts (from the set D) may execute authorized queries (from the set Q) on the contributed data, that too only before the expiration time (t_e). No other entity learns anything.

Integrity. Any modifications to the data shares or the query result will be detected.

Colocation. All data processing can be done in a single datacenter without weakening our threat model.

Next, we describe a strawman design, S1, for the core functionality that attains SFC/confidentiality and integrity, but not colocation. In § 3.2, we modify this strawman to attain our full construction, S2.

3.1 Strawman (S1): *n*-party construction

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 SFC/confidentiality and integrity but *not colocation* since the parties need to be physically separated to ensure their independence (colocation is obtained later using TEEs).

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 a circuit-based MPC scheme.

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 function q, and 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 perform MPC, which first checks the authenticity of the shares (using the authenticated secret-sharing scheme), and then computes the query result encrypted for the analyst and signed by a private key that exists only in shared form. If any check fails, the result is $\perp$.

Properties. S1 has the desired SFC/confidentiality and integrity assuming that at least $n - t$ parties are honest. Realizing this assumption requires the parties to have independent roots of trust (for compromise), i.e, their compromisability should not be correlated. Colocating the parties in a single datacenter breaks this independence, so S1 does not satisfy the colocation property.

$MPK[Q,D,t_e] \leftarrow$ **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 $[Q,D,t_e]$. $MPK[Q,D,t_e]$ published for data sources, $MSK[Q,D,t_e]$ retained by the n parties. **Inputs:** κ: security parameter, Q: set of allowed queries, D: set of authorized decryptors, 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:** $MSK[Q,D,t_e]$: 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: decryptor, 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.)

3.2 Full construction (S2): S1 + TEEs

Our full construction, S2, modifies S1 to additionally provide the colocation property.

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 by community experts as part of a review process.

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

Properties. S2 improves S1 by providing colocation in addition to the SFC/confidentiality and integrity of S1. Unlike in 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 irrespective of their physical locations and, in particular, when the parties are colocated in the same datacenter.

4 CoVault: A 2PC implementation

Next, we describe the details of a specific implementation of the CoVault core API (Figure 2) for $n = 2$ parties (of which $t = 1$ may be malicious), using garbled circuits and the DualEx protocol [54] for 2PC. These choices coincide with those in our prototype implementation. We emphasize that the constructions described here generalize to any number of parties and any actively-secure MPC protocol in obvious ways.

General setting. The two parties execute in different, independent TEE implementations. Each party consists of two kinds of components: a single provisioning service (PS) and one or more data processors (DPs). The PSs perform actions on behalf of their party, and attest and provision the DPs in their pipeline with the key pair necessary to decrypt data shares.

The DPs of the two parties are the entities that perform 2PC: each party's DPs *together* constitute one *party* in the sense of MPC. A corresponding pair of DPs (one from each party) is specific to a query class $[Q,D,t_e]$.

Each party's PS holds information on the query classes that have been defined. For each triple, this information consists of the measurement hash of the binary code implementing Q in MPC, the public keys of the authorized analysts in D, the data expiration time t_e, and the public keys of both DPs that implement that query class $[Q,D,t_e]$.

PSs and DPs can be safely shut down and re-started from their sealed state [29] without re-attestation. We discuss how to prevent roll-back of the database in § 5.1.

System initialization. Each party instantiates its PS; then, each PS generates its party's private-public key pair. Community experts attest the PSs and signs their public keys.

4.1 API call Setup$(1^\kappa, Q, D, t_e)$

A new query class can be initiated by interested analysts using the **Setup** call. Each party spawns a fresh DP for the query class, and configures it with the class parameters $[Q,D,t_e]$.

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

In a final step, community experts attest the DPs of the query class and publish the results.

4.2 API call Contribute$(MPK[Q,D,t_e], m)$

Data sources contact the PSs to retrieve information on the available query classes $[Q,D,t_e]$, as well as the public keys of the DP pairs that implement them. To contribute data to a query class $[Q,D,t_e]$, a data source first checks that the two DPs implementing the class have been attested by community experts the source trusts.

Next, the data source uses the authenticated secret-sharing scheme described below to create two *shares* of its data, and encrypts each share with the public key of one of the two DPs. (This pair of encrypted shares is denoted C in the **Contribute** API.) It provides each share to its respective DP. 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.

Authenticated secret sharing. Data sources share data using an *authenticated secret-sharing scheme*, which allows DP pairs to verify that the shares have not been modified by either party before using the sharing (input integrity). Our authenticated secret-sharing scheme is based on a MAC followed by secret-sharing, so we call it *MAC-then-share* or MtS.[3]

The scheme assumes a MAC function M that provides key- and message-non-malleability and message privacy. An example of such a function M is KMAC256 [57], which our prototype uses. 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:

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

The two MtS authenticated shares of the data are (m_1, k_1, t) and (m_2, k_2, t). Each share looks looks random, but anyone possessing both 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.

4.3 API call Query$(q, d, C_1, \ldots, C_K)$

To execute a query q over encrypted shares $C_1, \ldots, C_k$ of a query class $[Q, D, t_e]$, an analyst d sends the two DPs of that class their shares from $C_1, \ldots, C_k$ with the query q. The two DPs independently authenticate d, and check that $q \in Q, d \in D$ and current time $< t_e$. If both DPs are satisfied, they locally decrypt their shares, and run a 2PC that verifies the shares, reconstructs the plaintexts $m_1, \ldots, m_k$ from the shares, and computes $q(m_1, \ldots, m_k)$, which is revealed only to the analyst.

For 2PC, the DPs use garbled circuits (GCs). Most actively-secure GC protocols are orders of magnitude more costly than their passively-secure counterparts. Consequently, we rely on DualEx [54], an actively secure GC 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. We admit this 1-bit leak in exchange for DualEx's high efficiency relative to other actively-secure protocols. Briefly, DualEx runs two instances of a standard passively-secure GC protocol concurrently on separate core pairs, with the roles of the two parties, conventionally called *generator* and *evaluator*, reversed. Afterwards, the results of the two runs are compared for equality using any actively secure GC protocol.

Our extension to DualEx. Like most 2PC protocols, DualEx reveals the computation's result to the two parties. However, we need a protocol that reveals the result to the analyst but not to the two parties. For this, we combine DualEx and MtS (§4.2): We run DualEx to first compute $r = q(m_1, \ldots, m_k)$ and then share r two ways using MtS, resulting in two authenticated shares r_1, r_2, each of which is output to one party (by DualEx). Each party passes its share to the analyst, which verifies the shares and then reconstructs the query result.

Security analysis and proofs. Our assumptions (§2.2) that at least one TEE implementation is uncompromised and that data sources and analysts interact only with attested DPs together imply that at least one DP of each query class is fully honest. Hence, all GCs run securely. To argue end-to-end security, it only remains to show that our MtS scheme and modified DualEx protocol provide confidentiality and integrity, which we do in §F.

5 CoVault scalable analytics

In this section, we discuss how we store data and compute queries scalably in CoVault. (We defer a description of data ingress processing to the appendix, §C.)

5.1 CoVault Databases

So far, we have not discussed how the encrypted data shares, denoted C in § 3–§ 4, are managed. In our prototype, the encrypted data shares C returned by the **Contribute** API are forwarded to the respective DPs by the data sources. *The DPs store these shares in local databases*. In the **Query** call, the encrypted data shares $C_1, \ldots, C_k$, on which the query runs, are read directly from the databases; the querying analyst does not provide them.

Shares stored in a DP's database are encrypted and MAC'ed with keys known only inside the DP's TEE, so the databases are not a part of the trusted computing base. How the databases are organized and what if any processing on source data is performed by the DPs during data ingress depends on the database schema and queries in the query class. We sketch an example as part of our epidemic analytics 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 #1: Efficient oblivious DB random access. CoVault's DPs access the parties' database (DB) of shares in order to compute query results (specifically, in the API call **Query**, the inputs $C_1, \ldots, C_k$ are read by the DPs from their

[3] CoVault's design is not tied to this particular authenticated secret-sharing scheme. We could also have used other schemes [33, 41].

databases). 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, we could implement ORAM [46] within 2PC to solve this problem; however, our experiments showed that even the state-of-the-art Floram [39] is orders of magnitude more expensive than our solution described below. The properties of private information retrieval (PIR) protocols are closer to our requirements, but still have substantial overhead [30, 32, 73, 92]. Another option is to *randomly permute secret-shared tables* [23, 51, 61], but these techniques either have substantial overhead or work only with semi-honest adversaries. CoVault instead relies on an oblivious data retrieval (ODR) scheme, which achieves constant-time lookup at the expense of off-line work. (DB accesses where the accessed locations are independent of secret data need not use ODR.)

Oblivious data retrieval (ODR). Our ODR scheme uses preprocessing inside 2PC followed by pseudorandom table shuffling *outside 2PC*. The scheme works as follows.

Preprocessing. During data ingress, which runs in 2PC, we preprocess every table that requires ODR access. We encrypt-then-MAC (EtM) each row, and separately EtM the row's primary key. Secrets used to generate the EtMs can be recovered in 2PC only.

Shuffling. A preprocessed table is shuffled by a pair of DPs, DP_1 and DP_2, of different parties. The shuffling by the DPs is performed *outside 2PC* for efficiency. First, DP_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 DP_1. Next, DP_2 re-shuffles the already shuffled table, by re-sorting the table along a keyed hash over DP_1's hashes using a fresh key. This doubly-shuffled table is stored in the DB indexed by DP_2's hashes.

Row lookup. To access a row with a given primary key in 2PC, the position of the row in the stored table is computed in constant time by applying the EtM and the two keyed hashes to the primary key. The position is revealed to the two 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 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 of both parties, which only 2PC has. Second, neither party learns any row individually since all data is encrypted by preprocessing using 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 PIR, our ODR scheme does not hide whether two lookups access the same row. Therefore, to avoid frequency attacks, a shuffled table is used for one query only, and a query never fetches the same row twice. Reshuffling can be done ahead of time, 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 § D.2). Otherwise, data-oblivious joins can be very expensive [98], even more so in 2PC. In § 6.3, we show queries that exploit public indexes.

Problem #2: Detecting database roll-back. Malicious platform operators or parties may roll back the database to an earlier version. In general, known techniques can be used to detect rollback [6, 66, 76], e.g., a secure, persistent, monotonic counter within a TEE or TPM can be tied to the most recent version of the database. 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 can be detected trivially. To detect truncation, CoVault additionally provides a warning to the querier whenever records within the index range specified in the query are missing.

5.2 Executing analytics queries at scale

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

```
SELECT aggregate ([DISTINCT] column1), ...
FROM T WHERE condition GROUP BY column2
```

Here, `aggregate` is an aggregation operator like SUM or COUNT. The query can be executed as follows: (i) filter (select) from table `T` the rows that satisfy `condition`, (ii) group the selected rows by `column2`, and (iii) compute the required `aggregate` in each group.

Problem #3: Horizontal scaling. A straightforward implementation of a FGA query would be a single garbled circuit that takes as input the two shares of the *entire* database of the query class and implements steps (i)—(iii). However, this approach does not take advantage of core parallelism to reduce query latency. Moreover, the size of this circuit grows super-linearly with the size of `T` and may become too large to fit in the memory available on any one machine. Instead, inspired by MapReduce [37], CoVault converts FGA queries into a set of small map and reduce circuits, each of which fits one machine. Since queries tend to be highly data parallel, most map and reduce circuits are data independent, and can be executed in parallel 2PCs using all available cores and servers in a datacenter.

Our map and reduce circuits use known oblivious algorithms: bitonic sort [13]), bitonic merge of sorted lists [13]) and a butterfly circuit for list compaction [47, §3], which moves marked records to the end of a list. Each of these 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.

Problem #4: Integrity of intermediate results. When there is a data dependency between two circuits, integrity of the data passed between the circuits has to be ensured. While we could reuse our authenticated secret-sharing scheme, MtS, for this, MAC computation in 2PC is expensive and we want to minimize it. We address this problem by passing data between 2PC circuits in their *in-circuit* representation. While this increases the data transmitted or stored by a factor of 256x in our prototype (128x for the garbled coding of each bit and 2x for DualEx), we have empirically found this to be faster than verifying and creating a MAC in each mapper/reducer. Our design reduces in-2PC MACs to the minimum possible 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. Database accesses using ODR need additional MAC and hash operations as explained in § 5.1.

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 pseudorandom 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 compare to Senate?

6.1 Prototype and experimental setup

We implemented CoVault using EMP-toolkit [93], using emp-sh2pc and emp-ag2pc [93] to implement DualEx and our extension. We also implement circuits for SHA3-256 and AES-128-CTR (directly in emp-tool), primitives for filtering and compaction, and the MtS of § 4.2 (based on KMAC-256, which we implement on top of a pre-existing circuit for Keccak). We compose these to implement our ODR scheme, microbenchmarks and queries. We use Redis (v5.0.3, non-persistent mode) as the DB with the optimizations mentioned in § 5.1. For the ODR scheme, we implement the 2PC shuffle operation from § 5.1; we hold shuffled views in memory on a DP (rather than in Redis) for efficient re-shuffling.

Unless stated differently, the two parties execute on Intel and AMD CPUs, respectively. We use 7 machines with Intel®Xeon®Gold 6244 3.60GHz 16-core processors and 495GB RAM each, and 7 machines with AMD EPYC 7543 2.8GHz 32-core processors and 525 GB RAM each. All machines are connected via two 1/10GB Broadcom NICs to a Cisco Nexus 7000 series switch. In addition, each pair of Intel and AMD machines is directly connected via two 25Gbps links over Mellanox NICs; these are used for 2PC.[4]

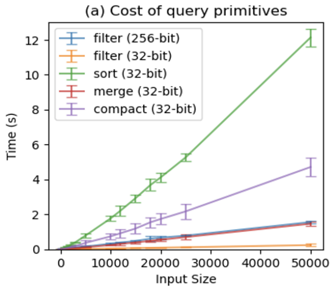

Figure 3: 2PC processing time vs. input size for basic oblivious algorithms and mappers/reducers.

On both types of machines, we use Linux Debian 10, hosted inside VMs managed using libvirt 5.0 libraries, the QEMU API v5.0.0, and the QEMU hypervisor v3.1.0. On the AMD machines we run the VMs in SEV-SNP TEEs provided in the sev-snp-devel branch of the AMDSEV repo [1], along with their patched Qemu fork. We use at most 8 cores on each of the AMD and Intel machines (the same number of cores are used on each type of machine). On the Intel machines, we run the VMs without a TEE, because we have not been able to get access to TDX hardware with an appropriate configuration yet. However, we are confident that our measured performance of 2PC is conservative, for several reasons: (1) Intel TDX was designed to compete with ADM SEV-SNP in the server/Cloud market [10, 26] and has similar capabilities, so one would expect comparable performance; (2) indeed, published results for overhead over conventional VMs for both SEV-SNP and TDX are comparable for both CPU and IO-bound workloads [4, 62] (The measured overhead of AMD SEV-SNP TEEs is 1.6% in our setup); (3) the performance in our setup is limited by the AMD CPUs, which have a 28% slower clock rate than the Intel CPUS (2.80GHz vs 3.60GHzh) and do use TEEs. Therefore, our results remain conservative even if TDX hardware were up to 28% slower than SEV-SNP on our workload.

6.2 Microbenchmarks

We first report the costs of basic oblivious algorithms (§ 5.2), mappers and reducers for a generic FGA query (§ 5.2), and pseudorandom shuffling (§ 5.1). These primitive costs can be used to estimate the total cost of arbitrary FGA queries, beyond the specific epidemic analytics queries we evaluate in § 6.3, assuming sufficient internal network bandwidth, which is normally available in a datacenter.

Query primitives. Figure 3a shows the time taken to execute the basic oblivious query primitives from § 5.2—linear

[4] Such network connectivity is common in datacenters.

scans on 32-bit and 256-bit records, sort, merging two sorted lists, and compact on 32-bit records—as a function of the list length (input size). Each reported number is an average of 100 measurements, with std. dev. shown as error bars.

The trends are as expected: The cost of linear scans grows linearly in the input size, while the costs of sort, sorted merge and compact are slightly super-linear. As expected, the cost of sorting is significantly higher than that of compaction, which is why our reduce trees sort only in the first stage and then use compact only.

DualEx versus AGMPC. To validate our choice of DualEx, we compare its performance against AGMPC [95], a state-of-the-art malicious-secure Boolean MPC protocol. We sort 1000 32-bit inputs using two parties, in AGMPC (executing on the faster Intel machines) versus DualEx (two symmetric runs plus equality check) on two cores (one Intel, one AMD). DualEx is 7 times faster ($0.52s$ versus $5.9s$), confirming that DualEx is the best choice for malicious-secure 2PC on our workload (Figure 4). We also executed the sort in AGMPC with 3 and 4 parties, and the runtime increased, in spite of using proportionally more cores. This indicates that with current malicious-secure boolean MPC protocols, 2PC is the best choice for performance, independent of the number of different TEE types available.

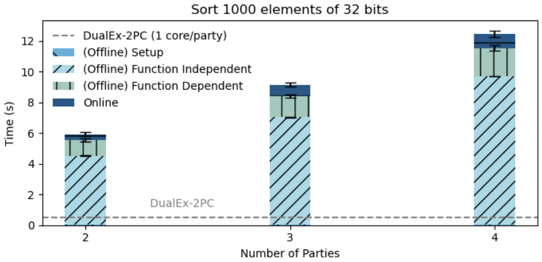

Figure 4: DualEx vs. AGMPC

MapReduce. Figure 3b shows the average time taken to execute, on a single core pair, typical mappers and reducers. (The specific operations are those of query q2 in Figure 5.) The results are expected: Map costs are linear in the input size, while reduce costs are slightly super-linear with the 1st-stage of reduction being more expensive.

Shuffling. Each shuffle requires two sequential oblivious sorts in TEEs of different types, outside 2PC. Shuffling a view of 600M records takes about 56min on a single core pair. Shuffling one-tenth the number, 60M records, takes 4min. The variances are negligible. The scaling is super-linear because oblivious sort (even outside 2PC) runs in $O(N(\log(N))^2)$ time.

A shuffle is used for only one query, but shuffles can be generated ahead of time in parallel. Because a shuffle sorts on one machine at a time, a single pair of cores can, in less than 1h, produce 2 different shuffles of 600M records—a conservative upper bound on the pairwise encounters generated by a country of 80M people in 1h in our epidemic analytics scenario (§6.3). Hence, for a country of this size, we can prepare shuffles for q queries using $q/2$ pairs of cores continuously.

Bandwidth. Garbled circuits are streamed from the generator to the evaluator, and the generator also transmits the garbling of the evaluator's inputs. During a series of sort and linear scan operations, the average bandwidth from the generator to the evaluator, measured with NetHogs [2], is ∼2.9Gbps. The bandwidth from the evaluator to the generator is negligible in comparison. With DualEx, the average bandwidth would be 2.9Gbps in each direction. Thus, 8 active cores on each machine need a bandwidth of 11.6Gbps in each direction, which our 2x25Gbps links can support easily.

6.3 End-to-end scenario: Epidemic analytics

We evaluate CoVault at scale in the context of epidemic analytics as an example scenario. We note that query processing (MPC) overhead is largely independent of the data semantics. Hence, the evaluation generalizes to any scenario and queries with similar data sizes and sequence of FGA operations.

The ingress processing for the epidemic analytics scenario collects location and BLE radio encounter data from smartphones, and is detailed in §C, §D.2. In our evaluation, we use synthetic data. Ingress results in two materialized views, T_E and T_P, whose schemas are shown in Figure 5. Each view has a *public*, coarse-grained index, which is shown in *italics* font. T_E is a list of pairwise encounters with an encounter id (eid) and anonymous ids of the two devices (did1, did2). Its public index is 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). A record also contains pointers to the previous and the next encounters, used to traverse the timeline of a given device. The public index is a coarse-grained *epoch* (∼1h) in which the encounter occurred. T_P is accessed through the private index by data-dependent queries, so T_P is shuffled per our ODR scheme (§5.1).

Our evaluation uses the queries q1–q3 shown in Figure 5, developed in consultation with an epidemiologist. Such queries can be used to understand the impact of contact restrictions (such as the closure of large events) on the frequency of contacts between people during epidemics. Query 3 can be used to determine if two outbreaks of an epidemic (corresponding to the sets of devices A and B) are directly connected. A related query, discussed in §D.4, extends q3 to indirect encounters between A and B via a third device.

We measured end-to-end query latency, as a function of input size and core count, for the queries q1–q3 of Figure 5.

Queries q1 and q2. Queries q1 and q2 run on the view T_E. The queries fetch only the records that are in the space-time region R, using the public index of T_E, which significantly reduces the size of the input table and the query latency. In both queries, we iterate over devices in the given set A. For

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
```

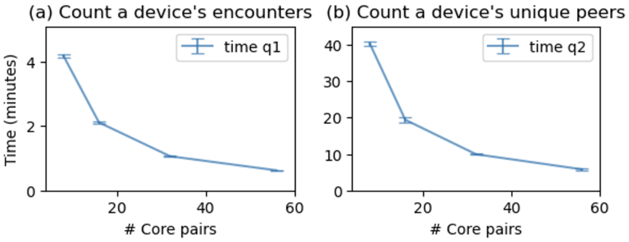

Figure 6: Query latency vs. total number of core pairs (including the parallel DualEx execution) for the FGA query of (q1) and (q2) from Figure 5. The measurements are the average of 20 runs, with std. dev. shown as error bars.

each device *a* in `A` we issue a FGA query. For q1, this FGA query counts the encounters device *a* had in `R`. For q2, the FGA query counts the number of unique devices that *a* met. FGA queries for all devices in `A` can execute in parallel.

Figure 6(a) shows the latency of q1 against the number of available core pairs. As input, we use a table with 28M encounter records (corresponding to 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 shards of size 10k, which minimizes the latency empirically. Mappers filter input encounters to those involving the specific user *a*, and reducers aggregate the number of encounters. As Figure 6(a) shows, the latency is almost inversely proportional to the number of core pairs available, showing near-linear horizontal scaling.

Figure 6(b) shows the latency of q2 in the same setup. The map phase is unchanged, but each reduce combines and removes duplicates from two lists of encountered devices. Again, query latency varies almost inversely with the number of cores, but is higher than for q1 since first-stage reducers do more work in q2.

Overheads: Cost breakdown for q2. For q2, we measured the latency contributions of different design elements. A single (semi-honest) GC execution accounts for 92.2% of the reported latency, adding the parallel DualEx execution on a separate pair of cores plus the DualEx equality check adds 6.2% to the latency, and executing inside a TEE adds a mere 1.6%. A modified query that includes differentially private noise in the result runs with negligible additional cost, on the order of milliseconds.

Scaling to many cores. From measurements of individual mappers and reducers in queries, we can extrapolate the number of machines needed to attain a certain query latency with a given number of input records. For example, if the input was 10 times larger (280M records), answering q2's basic query in 10min needs 736 core pairs (e.g., 46 16-CPU machines). The details of how we extrapolate are provided in § E.

Given our measured bandwidth for 2PC from § 6.2, executing this query with 736 cores requires a total bisection bandwidth of over 1Tbps between pairs of servers hosting the parties. A datacenter can easily meet this demand among pairs of servers in the same rack, but a distributed MPC system would require similar bandwidth across datacenters! Thus, 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, we estimate that continuously ingesting all incoming encounter records (11.85B records/-day) requires 1,660 core pairs on a continuous basis (see § D.3), and running q2 on 14 days of such records (165.9B records) within 24h requires an additional 2,320 core pairs engaged for those 24h. Looked at differently, a core pair is required for roughly every 20,000 citizens at this scale.

Query q3. Query q3 runs on T_P. A naive implementation of q3 in 2PC requires a linear scan of all encounter records during the period of interest (`[start,end]`) to find those between two peers in sets `A` and `B`. Based on our earlier estimate, for a table of 80M people over a 14-day period, this would require loading and filtering 165.9B records *in 2PC*. To make this more efficient, we rely on T_P's private index on one of the encounter peers, and use our shuffled ODR to securely load only those encounters that involve a peer from set `B`. For `A` and `B` of size 10 each, this reduces the number of records read and processed in 2PC by 5 orders of magnitude to ~100K. The total query latency on 2 pairs of cores in our setup with these parameters is 2.24 hours. We refer to § D.4 for additional details of the view T_P and q3's implementation.

6.4 CoVault versus Senate

Senate. Since Senate's source code is not available, we implemented its *m*-Sort primitive (merge of sorted lists) in CoVault and also in AGMPC, which Senate uses as its baseline and is equivalent to Senate for two parties. In CoVault, we run m-Sort in a single DP with DualEx, but using only 2 cores to match the number of cores available to AGMPC with two parties. On an input of 600 32-bit integers, CoVault executes m-Sort more than 7 times faster than AGMPC (0.19s versus 1.36s).

Next, we consider Senate configurations with more than two parties, since that is where Senate improves over AGMPC and because it is required for Senate to support more than two data sources. We take Senate's published results in a LAN setting with 4 and 16 parties, 600 inputs/party and 16 parties, 1400 inputs/party [78, Fig. 5]. On the same sized inputs, CoVault with 2 cores completes the query 129, 238, and 239 times faster (.31s vs. 40s, 1.93 vs. 460s, 4.8s vs. 1150s), respectively. Also, the overhead of Senate strongly increases with the number of parties (equal to the number of sources in Senate), while CoVault benefits from the fact that it executes with two parties regardless of the number of sources. This shows the benefits of decoupling data sources from MPC parties in CoVault, even in the range of up to a modest 16 sources.

Scaling limits of distributed MPC. Based on published runtime and network usage results for Senate's m-Sort circuit with 16 parties [78, Figs. 5a, 7b], we can estimate its average bandwidth during the execution as 4.8Gbps (480GB/800s). We can use this result to estimate its total average bandwidth if one tried to perform parallel subqueries on a large dataset (for which it was not designed). For instance, when executing query q2 on 28M records (see §6.3) and assuming sufficient cores, Senate would have to perform 1750 (28M/16k) parallel sorts in the first stage, resulting in a total average WAN bandwidth of 8.4Tbps. We mention this not as a critique of Senate, which was not designed for big data analytics, but to show that even when using a relatively bandwidth efficient MPC protocol like AGMPC, distributed MPC bottlenecks on WAN bandwidth. In contrast, due to CoVault's single-datacenter colocation, its scale-out to big data is not limited by bandwidth.

7 Related work

CoVault is the first MPC platform that uses multiple TEE types to colocate the computing parties and to decouple these parties from the data sources.

Encrypted databases like CryptDB [79] encrypt data at rest. Early work uses weak encryption like deterministic or order-preserving encryption, later shown to be insecure [17, 49]. Blind Seer [75] uses strong encryption and 2PC to traverse a specialized index, but leaks information about its search tree traversal. It also restricts the set of queries.

TEE-backed platforms [8, 11, 16, 45, 55, 59, 60, 63, 80, 85, 90] protect data at rest and in use by decrypting data only inside a TEE. However, these systems rely on a single TEE, which is fully trusted (unlike CoVault) and they do not mitigate well-known side-channel leaks in TEEs [70].

TEE-backed data-oblivious platforms use oblivious algorithms in addition to TEEs to mitigate side-channel leaks [9, 36, 38, 42, 48, 71, 72, 98]. In almost all cases, the papers propose optimized oblivious algorithms for one or more computational problems. Again, these systems use a single TEE and do not decentralize trust. CoVault's threat model is stronger, as it decentralizes trust among multiple TEEs and a community attestation process.

Secret-shared data analytics aim to decentralize trust. Obscure [50] 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 [25] and GraphSC [69] outsource computation to two untrusted non-colluding servers, but assume only semi-honest adversaries. Waldo [35] 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 non-technical means to mitigate correlated attacks.

Collaborative or cooperative analytics refers to MPC-based analytics where the parties are also the data sources. SMCQL [14] and Conclave [91] consider only passive security. Senate [78] provides active security; we have compared it to CoVault extensively in earlier sections. Senate automatically partitions query plans to run each subquery on exactly the subset of parties whose data the subquery accesses. In deployments of CoVault with more than two parties, similar techniques could be used to reduce query latency. Generally, CoVault goes beyond cooperative analytics in its use of multiple TEEs to decouple the number of data sources from the number of MPC parties and to colocate the MPC parties in a datacenter.

A combination of MPC and multiple TEEs has been proposed in a recent paper [34], which sketches how multiple TEEs can bootstrap decentralized trust and states that, ideally, the TEEs use hardware from different vendors. DeCloak [81] use a combination of MPC and TEEs to obtain a protocol for fair multi-party transactions in blockchains. CryptFlow [58], an MPC framework for secure inference with TensorFlow, runs the parties in fully-trusted TEEs to get active security out of passively-secure protocols. A recent paper [96] describes a database combining MPC and TEEs; however, the system assumes trusted clients (analysts) and was not shown to support group-by aggregates and scale to large databases. None of these papers consider the party-colocation and the source-party decoupling benefits of TEEs, which are our focus.

All work mentioned so far in this section assumes that data is owned by the computing parties and, hence, does not consider selective forward consent (unlike CoVault).

Homomorphic encryption (HE) can also be used for decentralized analytics. Full HE is prohibitively expensive [89]. Partial HE (PHE) restricts query expressiveness significantly, and works efficiently only in weak threat models. For instance, Seabed [74] works in a semi-honest setting, extending at most to frequency attacks. TimeCrypt [18] and Zeph [19] allow data sources to specify access control preferences (similar to selective forward consent); however, they specifically target *time-series* data with a restricted set of operations (additions but not multiplications). Also, Zeph operates in a semi-honest threat model. Neither type of HE by itself provides protection from side-channel leaks.

Federated analytics (FA) leaves data in the hands of the data sources, thus ensuring confidentiality and selective control. ShrinkWrap [15] only considers semi-honest adversaries, while Roth et al. [82–84] and Arboretum [65] extend FA to malicious adversaries. As explained in § 1, these systems support many data sources but run each query on the data from a subset of sources that happen to be online at the time. Some FA frameworks support only restricted classes of queries (unlike MPC, which is general).

Automatic optimization of MPC has been considered in Arboretum and several MPC frameworks – [14, 20, 22, 24, 44, 56, 91] to name a few. Heuristics automatically parallelize and 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.

8 Conclusion

CoVault enables individual data sources to safely delegate control over their encrypted data to a set of colocated MPC parties, each executing in a TEE of a different type. Because CoVault is not impeded by WAN network 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 is 7 to over 100 times faster than existing MPC-based systems on few data sources and small data, while also scaling out to epidemic analytics queries for a country of 80M people and billions of records.

9 Acknowledgments

We would like to thank Gilles Barthe, Jonathan Katz, Matthew Lentz, Chang Liu, and Elaine Shi for their helpful suggestions and contributions to the initial version of this work; Xiao Wang and Chenkai Weng, for their precious insights on emp-toolkit; Lorenzo Alvisi, Natacha Crooks, Aastha Mehta, Lillian Tsai, Vaastav Anand, and the anonymous reviewers, for their insightful feedback on prior versions of this work. This work was supported in part by the European Research Council (ERC Synergy imPACT 610150) and the German Science Foundation (DFG CRC 1223).

References

[1] GitHub: AMDSEV repo. https://github.com/AMDESE/AMDSEV/tree/sev-snp-devel. Accessed: 2023-12-07.

[2] GitHub: Nethogs repo. https://github.com/raboof/nethogs. Accessed: 2023-12-07.

[3] Council approves Data Governance Act, 2022. https://tinyurl.com/consilium-europa-dga.

[4] Performance Considerations of Intel(R) Trust Domain Extensions on 4th Generation Intel(R) Xeon(R) Scalable Processors, 2023. https://www.intel.com/content/www/us/en/developer/articles/technical/trust-domain-extensions-on-4th-gen-xeon-processors.html.

[5] AMD. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. White paper at https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020. Accessed: 2023-12-07.

[6] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath T. V. Setty, and Sudheesh Singanamalla. Nimble: Rollback protection for confidential cloud services. In Roxana Geambasu and Ed Nightingale, editors, *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, pages 193–208. USENIX Association, 2023.

[7] Apple and Google. Privacy-Preserving Contact Tracing. https://www.apple.com/covid19/contacttracing. Accessed: 2021-12-14.

[8] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security With Cipherbase. In *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*, January 2013.

[9] A. Arasu and R. Kaushik. Oblivious Query Processing. *CoRR*, abs/1312.4012, 2013.

[10] Azure. Azure and AMD announce landmark in confidential computing evolution . https://azure.microsoft.com/en-us/blog/azure-and-amd-enable-lift-and-shift-confidential-computing/.

[11] S. Bajaj and R. Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD

'11, pages 205–216, New York, NY, USA, 2011. Association for Computing Machinery.

[12] G. Barthe, R. De Viti, P. Druschel, D. Garg, M. Gomez-Rodriguez, P. Ingo, M. Lentz, A. Mehta, and B. Schölkopf. PanCast: Listening to Bluetooth Beacons for Epidemic Risk Mitigation. *arXiv preprint arXiv:2011.08069*, 2020.

[13] K. E. Batcher. Sorting Networks and Their Applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS 1968 (Spring), pages 307–314, New York, NY, USA, 1968. Association for Computing Machinery.

[14] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. SMCQL: Secure Querying for Federated Databases. *Proceedings of the VLDB Endowment*, 10(6):673–684, February 2017.

[15] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers. ShrinkWrap: Efficient SQL Query Processing in Differentially Private Data Federations. *Proc. VLDB Endow.*, 12(3):307–320, 2018.

[16] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, 33(3), August 2015.

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

[18] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy. TimeCrypt: Encrypted data stream processing at scale with cryptographic access control. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 835–850, 2020.

[19] L. Burkhalter, N. Küchler, A. Viand, H. Shafagh, and A. Hithnawi. Zeph: Cryptographic enforcement of end-to-end data privacy. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, pages 387–404, 2021.

[20] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. Hycc: Compilation of hybrid protocols for practical secure computation. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 847–861. ACM, 2018.

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

[22] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. Ezpc: Programmable and efficient secure two-party computation for machine learning. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 496–511. IEEE, 2019.

[23] M. Chase, E. Ghosh, and O. Poburinnaya. Secret shared shuffle. Cryptology ePrint Archive, Paper 2019/1340, 2019. https://eprint.iacr.org/2019/1340.

[24] Edward Chen, Jinhao Zhu, Alex Ozdemir, Riad S. Wahby, Fraser Brown, and Wenting Zheng. Silph: A framework for scalable and accurate generation of hybrid MPC protocols. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 848–863. IEEE, 2023.

[25] 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.

[26] Google Cloud. Security through collaboration: Building a more secure future with Confidential Computing . https://cloud.google.com/blog/products/identity-security/google-amd-partner-to-build-a-more-secure-future-with-confidential-computing.

[27] European Commission. European data governance act. https://digital-strategy.ec.europa.eu/en/policies/data-governance-act.

[28] European Commission. European data strategy. https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-data-strategy_en.

[29] Intel Corp. Introduction to Intel SGX Sealing. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html.

[30] H. Corrigan-Gibbs and D. Kogan. Private Information Retrieval with Sublinear Online Time. *IACR Cryptol. ePrint Arch.*, 2019:1075, 2019.

[31] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

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

[33] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[34] 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*, pages 38–45, 2022.

[35] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica. Waldo: A private time-series database from function secret sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468, 2022.

[36] A. Dave, C. Leung, R. A. Popa, J. E. Gonzalez, and I. Stoica. Oblivious Coopetitive Analytics Using Hardware Enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[37] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. 2004.

[38] T. T. A. Dinh, P. Saxena, E. Chang, B. C. Ooi, and C. Zhang. M2R: Enabling Stronger Privacy in MapReduce Computation. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 447–462, Washington, D.C., August 2015. USENIX Association.

[39] J. Doerner and A. Shelat. Scaling oram for secure computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 523–535, New York, NY, USA, 2017. Association for Computing Machinery.

[40] M. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015-08-04 2015.

[41] S. Eskandarian and D. Boneh. Clarion: Anonymous communication from multiparty shuffling protocols. *Cryptology ePrint Archive*, 2021.

[42] S. Eskandarian and M. Zaharia. ObliDB: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[43] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2-3):70–246, 2018.

[44] Vivian Fang, Lloyd Brown, William Lin, Wenting Zheng, Aurojit Panda, and Raluca Ada Popa. Costco: An automatic cost modeling framework for secure multi-party computation. In *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*, pages 140–153. IEEE, 2022.

[45] B. Fuhry, R. Bahmani, F. Brasser, F. Hahn, F. Kerschbaum, and A. R. Sadeghi. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 386–408. Springer, 2017.

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

[47] M. T. Goodrich. Data-Oblivious External-Memory Algorithms for the Compaction, Selection, and Sorting of Outsourced Data. In *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA 2011, pages 379–388, New York, NY, USA, 2011. Association for Computing Machinery.

[48] A. Gribov, D. Vinayagamurthy, and S. Gorbunov. StealthDB: a scalable encrypted database with full SQL query support. *CoRR*, abs/1711.02279, 2017.

[49] P. Grubbs, T. Ristenpart, and V. Shmatikov. Why Your Encrypted Database Is Not Secure. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 162–168, New York, NY, USA, 2017. Association for Computing Machinery.

[50] P. Gupta, Y. Li, S. Mehrotra, N. Panwar, S. Sharma, and S. Almanee. Obscure: Information-Theoretic Oblivious and Verifiable Aggregation Queries. *Proceedings of the VLDB Endowment*, 12(9):1030–1043, May 2019.

[51] W. L. Holland, O. Ohrimenko, and A. Wirth. Efficient oblivious permutation via the waksman network. In *Proc. ASIA CCS*. ACM, 2022.

[52] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols?

[53] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, number 1, pages 331–335, 2011.

[54] Y. Huang, J. Katz, and D. Evans. Quid-Pro-Quo-tocols: Strengthening Semi-honest Protocols with Dual Execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284, San Francisco, CA, USA, May 2012. IEEE.

[55] T. Hun, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 533–549, Savannah, GA, November 2016. USENIX Association.

[56] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. *IACR Cryptol. ePrint Arch.*, page 521, 2020.

[57] J. Kelsey, S. Chang, and R. Perlner. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, 2016. NIST Special Publication 800-185. https://doi.org/10.6028/NIST.SP.800-185.

[58] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 336–353. IEEE, 2020.

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

[60] 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.

[61] S. Laur, J. Willemson, and B. Zhang. Round-efficient oblivious database manipulation. In *Proc. ISC*. Springer, 2011.

[62] Dingji Li, Zeyu Mi, Chenhui Ji, Yifan Tan, Binyu Zang, Haibing Guan, and Haibo Chen. Bifrost: Analysis and optimization of network {I/O} tax in confidential virtual machines. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 1–15, 2023.

[63] U. Maheshwari, R. Vingralek, and W. Shapiro. How to Build a Trusted Database System on Untrusted Storage. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI 2000, USA, 2000. USENIX Association.

[64] J. Manweiler, R. Scudellari, and L. P. Cox. SMILE: Encounter-Based Trust for Mobile Social Services. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 246–255, New York, NY, USA, 2009. Association for Computing Machinery.

[65] Elizabeth Margolin, Karan Newatia, Tao Luo, Edo Roth, and Andreas Haeberlen. Arboretum: A planner for large-scale federated analytics with differential privacy. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 451–465, New York, NY, USA, 2023. Association for Computing Machinery.

[66] S. Matetic, M. Ahmed, K. Kostiainen, A. Dhar, D. Sommer, A. Gervais, A. Juels, and S. Capkun. ROTE: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1289–1306, Vancouver, BC, August 2017. USENIX Association.

[67] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. Attestation mechanisms for trusted execution environments demystified. In David Eyers and Spyros Voulgaris, editors, *Distributed Applications and Interoperable Systems*, pages 95–113, Cham, 2022. Springer International Publishing.

[68] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.

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

[70] A. Nilsson, P. N. Bideh, and J. Brorsson. A Survey of Published Attacks on Intel SGX, 2020.

[71] 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, Denver, CO, USA, October 12-16, 2015*, pages 1570–1581. ACM, 2015.

[72] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, Austin, TX, August 2016. USENIX Association.

[73] 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, Berlin, Germany, July 21-23, 2010. Proceedings*, volume 6205 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2010.

[74] 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 *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pages 587–602, 2016.

[75] 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, Berkeley, CA, USA, May 18-21, 2014*, pages 359–374. IEEE Computer Society, 2014.

[76] B. Parno, J. Lorch, J. Douceur, J. Mickens, and J.M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2011.

[77] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*, pages 250–267. Springer, 2009.

[78] 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 *30th USENIX Security Symposium (USENIX Security 21)*, pages 2129–2146. USENIX Association, August 2021.

[79] R. A. Popa, C. MS. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[80] C. Priebe, K. Vaswani, and M. Costa. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 264–278. IEEE Computer Society, 2018.

[81] Qian Ren, Yue Li, Yingjun Wu, Yuchen Wu, Hong Lei, Lei Wang, and Bangdao Chen. Decloak: Enable secure and cheap multi-party transactions on legacy blockchains by a minimally trusted tee network, 2023.

[82] E. Roth, K. Newatia, Y. Ma, K. Zhong, S. Angel, and A. Haeberlen. *Mycelium: Large-Scale Distributed Graph Queries with Differential Privacy*, page 327–343. Association for Computing Machinery, New York, NY, USA, 2021.

[83] E. Roth, D. Noble, B. H. Falk, and A. Haeberlen. Honeycrisp: Large-scale Differentially Private Aggregation Without a Trusted Core. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*, October 2019.

[84] E. Roth, H. Zhang, A. Haeberlen, and B. C. Pierce. Orchard: Differentially Private Analytics at Scale. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, November 2020.

[85] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 38–54, USA, 2015. IEEE Computer Society.

[86] ARM Developer Site. ARM confidential compute architecture (cca). https://developer.arm.com/architectures/architecture-security-features/confidential-computing.

[87] Intel Developer Site. Intel Trust Domain Extensions (Intel TDX). https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.

[88] L. Tsai, R. De Viti, M. Lentz, S. Saroiu, B. Bhattacharjee, and P. Druschel. EnClosure: Group Communication via Encounter Closures. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 353–365, New York, NY, USA, 2019. Association for Computing Machinery.

[89] A. Viand, P. Jattke, and A. Hithnawi. SoK: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1092–1108, 2021.

[90] R. Vingralek. GnatDb: A Small-Footprint, Secure Database System. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*, pages 884–893. Morgan Kaufmann, 2002.

[91] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. Conclave: Secure Multi-Party Computation on Big Data. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[92] 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, Boston, MA, USA, March 27-29, 2017*, pages 299–313. USENIX Association, 2017.

[93] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016. Accessed: 2023-12-07.

[94] X. Wang, S. Ranellucci, and J. Katz. Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation. Cryptology ePrint Archive, Report 2017/030, 2017. https://eprint.iacr.org/2017/030.

[95] X. Wang, S. Ranellucci, and J. Katz. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 39–56, New York, NY, USA, 2017. Association for Computing Machinery. Implementation at https://github.com/emp-toolkit/emp-agmpc.

[96] 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, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022.

[97] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.

[98] 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 *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 283–298, Boston, MA, March 2017. USENIX Association.

A Community approval process

As discussed in §5, CoVault relies on a community approval process to justify trust in the system. Specifically, CoVault relies on community approval for two purposes: (i) to justify the trust in the PSs, which form CoVault's root of trust by attesting and provisioning DPs; (ii) to justify data sources' trust that the query classes to which they contribute their data do what their specification says it does. In both cases, data sources can rely on experts they trust who have attested the PSs and reviewed the implementation of PSs and DPs.

Any interested community member can inspect the source code of each system component, verify their measurement hashes, remotely attest the PSs, and publish a signed statement of their opinion. To do so, a witness performs the following operations:

1. Inspect the source code of the PSs and the build chain used to compile all system components, make sure they correctly implement the system's specification, and verify that the measurement hash recorded in the system configuration matches the source code, and remotely attest the PSs accordingly;
2. Obtain the current system configuration from the PSs;
3. Review the specification and source code of each query in a given class, and verify the measurement hash of the class's DPs recorded in the configuration;
4. Publish a signed statement of their assessment.

We note that attackers could remove or suppress witness statements, and false witnesses could post false statements about PS attestations or PS and DP (query class) reviews. However, this amounts at most to denial-of-service (DoS) as long as data sources are not distracted by reviews (positive or negative) from witnesses they do not fully trust. Furthermore, we note that the state of the PSs could be rolled back by deleting their sealed states or replacing the sealed states with an earlier version. However, this also amounts to DoS, as it would merely have the effect of making inaccessible some recently defined query classes and their data.

B Further details on scalable analytics queries

The unit of querying in CoVault is a SQL filter-groupby-aggregate (FGA) query. In general, a querier may make a series of *data-dependent* FGA queries, where query parameters of subsequent queries may depend on the results of earlier queries. In the following, we first describe how CoVault executes basic FGA queries, and then how it handles data-dependent series of FGA queries.

FGA queries. A FGA query has the form:

```
SELECT aggregate([DISTINCT] column1), ...
FROM T WHERE condition GROUP BY column2
```

Here, `aggregate` is an aggregation operator like SUM or COUNT. The query can be executed as follows: (i) filter (select) from table `T` the rows that satisfy `condition`, (ii) group the selected rows by `column2`, and (iii) compute the required `aggregate` in each group. A straightforward way of implementing a FGA query is to build a *single* garbled circuit that takes as input the two shares of the entire table `T` and implements steps (i)—(iii). However, this approach does not take advantage of *core parallelism* to reduce query latency. Moreover, the size of this circuit grows super-linearly with the size of `T` and the circuit may become too large to fit in the memory available on any one machine. To exploit core parallelism and to make circuit size manageable, CoVault relies on the observation that FGA queries can be implemented using MapReduce [37]. We first explain how this works in general (without 2PC) and then explain how CoVault does this in 2PC.

Background: FGA queries with MapReduce. Suppose we have m cores available. The records of table `T` are split evenly among the m cores.

Step (i): Each core splits its allocated records into more manageable *chunks* and applies a **map** operation to each chunk; this operation linearly scans the chunk and filters only the records that satisfy the WHERE `condition`.

Steps (ii) and (iii): These steps are implemented using a tree-shaped **reduce** phase. The 1st stage of this phase uses half the number of reducers as the map phase. Each **1st-stage reducer** consumes the (filtered) records output by two mappers, sorts the records by the grouping column `column2`, and then performs a linear scan to compute an aggregate for each value of `column2`. The output is a sorted list of `column2` values with corresponding aggregates. Each subsequent stage of reduce uses half the number of reducers of the previous stage: Every **subsequent-stage reducer** merges the sorted lists output by two previous-stage reducers adding their corresponding aggregates and producing another sorted list. The last stage, which is a single reducer, produces a single list of `column2` values with their aggregates.

CoVault: FGA queries with MapReduce in 2PC. CoVault executes FGA queries by implementing the mappers and reducers described above as *separate* garbled circuits and evaluating the circuits in 2PC. Thus, CoVault inherits scaling with cores from the MapReduce paradigm.

The *challenge* here is that circuits can implement only a limited class of algorithms. In particular, a circuit is data-oblivious—it lacks control flow—and the length of the output of a circuit cannot depend on its inputs. However, common algorithms for sorting rely on control flow (they branch based on the result of integer comparison), and standard algorithms for filtering and merging lists produce outputs whose lengths are dependent on the values in the input lists. Hence, to implement mappers and reducers in circuits, we have to use specific algorithms that are data oblivious and pad output to a size that is independent of the inputs (this padded output size, denoted d below, is an additional parameter of the algorithm; it should be an upper bound on the possible output sizes). In the following, we explain basic data-oblivious algorithms that CoVault relies on, and then explain how it combines them with padding when needed to implement mappers and reducers in circuits.

CoVault relies on the following standard data-oblivious algorithms implemented in circuits:

- A *linear scan* passes once over a list performing some operation (e.g., marking) on each element, or computing a running total.
- *Oblivious sort* on a list. While oblivious sort has a theoretical complexity $O(n\log(n))$, all practical algorithms are in $O(n(\log(n))^2)$. CoVault uses bitonic sort [13].
- *Oblivious sorted merge* merges two sorted lists into a longer sorted list. It retains duplicates. CoVault uses bitonic merge, which is in $O(n\log(n))$ [13].
- *Oblivious compact* moves marked records to the end of a list, compacting the remaining records at the beginning of the list in order. For this, CoVault uses an $O(n\log(n))$ butterfly circuit algorithm [47, §3].

CoVault uses these algorithms to implement mappers and reducers as follows. A CoVault **mapper** uses a *linear scan* to only mark records that do *not* satisfy the WHERE `condition` with a discard bit; it does not actually drop them, else the size of the output list might leak secrets. A **1st-stage reducer** *obliviously sorts* the outputs of two mappers, ordering first by the discard bit, and then by the GROUP BY criterion, `column2`. This pushes all records marked as discard by the mappers to the end of the list, and groups the rest in sorted order of `column2`. Records with the same value of `column2` belong to the same group, so the reducer must consider them just once when computing the aggregate. To this end, it performs a *linear scan* to (a) compute a running aggregate for each unique group, and (b) mark all but one record in each group to be discarded. Finally, it does an *oblivious compact* to push all records marked as discard to the end of the list. The unmarked records contain *unique*, sorted values of `column2` paired with corresponding aggregates. This output is truncated or padded to a fixed length d, which, as mentioned earlier, is an additional query parameter that should be an upper bound on the possible number of unique groups in `column2`. **Subsequent-stage reducers** are similar to the 1st-stage reducers, except that they receive two already sorted lists as input, so they use *oblivious sorted merge* instead of the more expensive oblivious sort.

The theoretical complexities of a mapper, 1st-stage reducer, and subsequent-stage reducer are $O(c)$ where c is the chunk size, $O(c(\log(c))^2)$ and $O(d\log(d))$, respectively. The chunk size c has a non-trivial effect on query latency: Larger chunks result in more expensive mappers and 1st-stage reducers, but fewer total number of mappers and reducers. In practice, we determine the chunk size empirically to minimize query latency; our experiments use $c = 10k$.

Data-dependent FGA queries. If a query accesses a table via a private index or if the data rows accessed by a FGA query *depend* on the output of an earlier query, information about database content may leak via the DB access pattern. To avoid such leaks, data-dependent FGA queries use previously shuffled tables (§5.1). Also, the number of records read must not depend on previous query results; to this end, CoVault adds dummy record reads.

C Ingress processing with public attributes

Many analytics applications like epidemic or financial transaction analytics rely on (spatio-)temporal data. In these applications, it is likely that the queries analyse data in a given (space-)time region. If the (spatio-)time attribute reveals no sensitive information, we can exploit data locality: grouping and storing together data according to *public* (spatio-)temporal information speeds up the queries fetching a whole group for processing. To exploit data locality, CoVault creates a materialized view(s) indexed by a public attribute(s), e.g., coarse-grained (space-)time information. Thus, CoVault is a read-only platform but supports DB appends in order of public attributes: CoVault's ingress processing pipeline allows incremental data upload from several data sources and produces materialized views securely and efficiently. In this section, we explain CoVault's ingress processing, which is implemented in 2PC by two dedicated DPs called the two Ingress Processors (IPs).

Batch append. Data sources upload new data in batches, which are padded to obfuscate the exact amount of data being uploaded. Data sources may pre-partition data according to public attributes, or locally pre-join or pre-select data prior to upload. Once a batch is uploaded, the batch becomes immutable and queryable. Before appending that batch to CoVault's DB, the IPs may perform pre-processing operations in 2PC; for instance, the IPs can buffer different batches and run group or sort operations (in 2PC) in order to produce or append to materialized views. The actual operations and materialized views are application-specific; in § D, we discuss the example scenario of epidemics analytics. Note that data sources can contribute data to one or more query classes. CoVault's IP pairs are query-class specific.

Ingress security and integrity. Data sources upload batches to IPs using session keys, thus are not identifiable by the IPs. The batches are padded, and the padding is "revealed" only within 2PC: no single IP party can determine the actual amount of data in any batch. For added security, data sources can split a single batch into multiple batches randomly and upload them with different session keys. They may also use VPN/Tor to obfuscate their Internet addresses during upload.

Storing data in garbled form. As discussed in §5.1, computing MACs in 2PC is expensive. So, we eliminate the MACs between IPs and DPs as well, by storing values in CoVault's DB directly in garbled form. Doing so significantly increases the size of stored data: garbling codes each bit in a 128-bit space. However, this choice eliminates the need for the IPs to compute MACs in 2PC, and for the DPs to verify them. (However, recall that IPs need to *verify* in 2PC the MACs added to their shares by the data sources, and this is unavoidable). This optimization of storing garbled values can be generalized to different query classes by using a separate secret to garble circuits for each query class: an IP pair garbles every datum once for the query class it manages, and each DP pair gets access to the secret of its class only.

D Further details on the epidemic scenario

D.1 Database

Epidemic analytics operates on a time series of individual locations and pairwise contacts among data sources' devices. Such data may originate, for instance, from smartphone apps that record GPS coordinates and pairwise Bluetooth encounters [64, 88], or from a combination of personal devices and Bluetooth beacons installed in strategic locations [7, 12]. We assume that data sources consent to consider as *public* coarse-grained spatio-temporal information related to the data they upload, as well as the inputs and the results of statistical epidemiological queries.

Here, we describe the records and materialized views used in our epidemic analytics scenario. Each view consists of a variable number of records of the form described in Figure 7. There is a separate view for every query class. The views include encounters identified by an encounter id (*eid*). If two data sources share an encounter, they report the same *eid* for that encounter, along with their own device id *did*. Only mutually confirmed encounters—encounters reported by both peers with consistent information on the encounter—are used for analytics. For this purpose, the IPs perform a join of the individual encounter reports, and add a *validity* attribute to potentially mark an encounter as confirmed (see §D.2). Additionally, the records may report t, a fine-grained temporal information indicating when an encounter begins, *dur*, the encounter duration, and *aux*, any additional information not currently used (e.g., fine-grained location or signal strength). Finally, note that the records in T_P are organized by device trajectories: these records are indexed by $did1$ and t, and each entry contains the indexes of the previous (*prev*) and next (*next*) encounters of $did1$. This view is used by queries that search trajectories in an encounter graph (q3 in Figure 5 and q4 in Figure 9).

View	Record fields							
T_E	*s.-t.-region*	eid	did1	did2	t	dur	aux	validity
T_S	*s.-t.-region*	did1	aux	period of contagion				
T_P	*epoch*	did1, t	did2	dur	prev	next	aux	validity

Figure 7: Materialized views used for epidemic analytics. The first column is a public index. (*s.t.* = *space-time*).

Space-time views. These materialized views speed up queries that have no data-dependent flow control. The views are grouped in space-time regions, indexed by a *public*, coarse-grained space-time index. Space-time views contain either encounter records (T_E) or information related to sick people (T_S). When a data source uploads a batch of encounters, it pre-selects the coarse-grained space-time index that batch maps to. The IPs collect batches from different data sources, and generate space-time views grouping data in the same space-time region. Using space-time views accelerates queries whose input parameters specify an arbitrary space-time region in input: prior to run the query in 2PC, the DPs can locally fetch all records whose coarse-grained index falls within the space-time region in input.

The resolution of the space-time regions depends on publicly known population density information and mobility patterns at a given location, day of the week, and time of the day, and is chosen so as to achieve an approximately even number of records per region. Each region is padded to its nominal size to hide the actual number of records it contains. (Note that many shards corresponding to locations at sea, in the wilderness, or night time have a predicted size of zero and therefore do not exist in the views.)

Shuffled encounter views. As discussed in §5.1, space-time views can be used only if the queries do not require any secret-dependent data access. Otherwise, we use shuffled encounter views, which support our ODR scheme (§5.1). These views allow running a sequence of FGA queries on T_P; however, CoVault cannot reveal the *sequence* of space-time regions analysed, as such sequence might reveal data sources' movements in time. Thus, the only public attribute is a conservative coarse-grained *time* information, which we call *epoch*. The primary key is encrypted and *the records within each epoch are randomly shuffled*. The mapping between a record, its encrypted key, and its position within the view can be reconstructed only in 2PC. The views are produced by the IPs and the records in an epoch are re-shuffled by the DPs after each use in a query.

Risk encounter view. This materialized view contains encounters that involve a diagnosed patient and took place during the patient's contagion period. Conceptually, it is the result of a join of T_E and T_S, computed incrementally during ingress processing and in cooperation with data sources. The view supports efficient queries that focus on potential and actual infections.

D.2 Ingress Processing

Next, we discuss CoVault's ingress processing (§C) specific to epidemic analytics. As mentioned in §D.1, the IPs check whether an encounter is confirmed, and potentially mark it as valid. Here, we give more details about this computation. Before uploading data to a view, the data source partitions its encounters into *per-space-time-region* batches, sorts each batch by *eid*, and *randomly pads* each batch to hide the actual number of encounters in the batch. Then, the data source secret-shares and MACs the batches following the protocol of §4.2, and uploads the batches to the two IPs using session keys.

Throughout a day, each IP pair receives batches from data sources and stores them locally (outside 2PC). Periodically, e.g., once per day, each IP pair runs a 2PC, which:

1. consumes all per-device batches for the space-time region
2. reconstructs the batches from the shares
3. verifies the per-batch MACs
4. merges the (sorted) batches into a space-time region buffer using oblivious sorted merge [52]
5. and truncates or pads the sorted list to the expected space-time region size.

The sorting puts padding uploaded by the data sources at the end of the buffer, so truncating the buffer deletes padding first. Next, the IPs confirm encounters within 2PC. For each *eid*, they check if both peers have uploaded the encounter with consistent locations and times, and set the validity bit of each encounter accordingly. This requires a linear scan of the space-time region. Finally, each IP in the pair appends the space-time region (in garbled form, see §C) to the appropriate tables and materialized views, performing random shuffling when needed.

D.3 Evaluation of Ingress Processing

Ingress processing converts batches of records uploaded by data sources to tables/views used by queries. Again, we report the costs of only one of the two circuits of DualEx because ingress does not perform DualEx's equality check and runs the two circuits completely independently in parallel (the tables/views created by the IPs are stored in their in-circuit, garbled form).

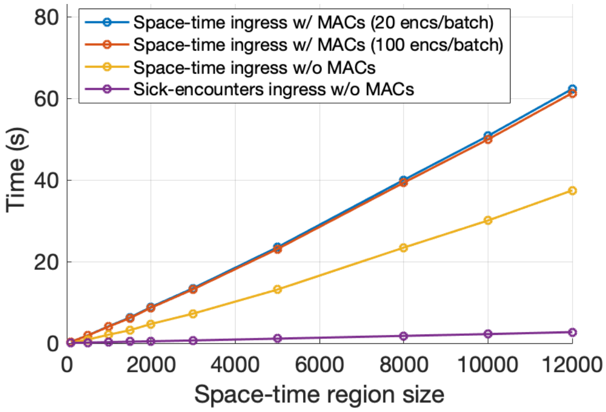

Figure 8: Cost of ingress processing on a single space-time region as a function of the region size.

In Figure 8, the two highest, nearly overlapping lines show the average time for ingress processing to generate a single space-time region in T_E, as a function of the region size (x-axis) and the upload batch size (20 encounters/batch or 100 encounters/batch) on a pair of cores (i.e., a pair of IPs). The IPs performs all the (1–5) steps in §D.2. The costs are slightly super-linear because the IPs sort each batch and merge the sorted batches. A significant part of this cost (between 40% and 65% depending on the region size) is the verification of per-batch MACs (*cf.* the 3rd line, which shows the cost without it). This high cost of MAC verification in 2PC is why we store tables/views in garbled form and avoid verifying MACs again in query processing. Note that the cost of ingress processing without MAC verification depends on the region size but not the batch size; in fact, the batch size impacts only the number of batches and the time to upload each batch to the IPs via a different, secure connection. This cost is negligible compared to the costs of 2PC operations. The lowest line of Figure 8 is the cost of ingress operations to populate the table of data sources diagnosed sick (T_S). This cost is much lower than that of generating a space-time region since T_S is not sorted and each record in T_S has fewer bits, which reduces MAC verification time.

Scaling. The relevant performance metric for ingress processing is throughput: We want to determine how many core pairs we need to keep up with the data rate generated by a given administrative entity (e.g., country, city). Thus, we run a 5h experiment where m core pairs generate space-time regions of 100 records each from uploads in batches of size 20. We measured the throughput of ingress processing (in encounters processed per hour), varying m from 1 to 8.

As expected, the results show perfect linear scaling with the number of available core pairs, from $584k$ to $4.75M$ encounters processed per hour for 1 and 8 core pairs, respectively. From this, we can extrapolate the number of core pairs needed to keep up with all encounters generated in a given world region. For example, conservatively assuming 200 encounters/person/day in urban areas and 100 encounters/person/day in rural areas, we estimate that, every 24h:

- a mid-sized country with population 80M would generate 11.85B encounters;
- a big metropolitan area with population 8M would generate 1.6B encounters;
- a urban city with population 3M would generate 600M encounters;
- a urban town with population 200k would generate 40M encounters.

Extrapolating from the linear scaling above, we estimate that ingress processing needs a total of 1660, 226, 86, and 6 core pairs, respectively. According to these estimates, the ingress processing of a whole country is well within the means of a small-scale data center, while that of a town would require a single pair of machines.

D.4 Evaluation of data-dependent queries

Details of the implementation of query q3. This query asks for the number of devices in `B` that directly encountered a device in `A` in the time interval `[start,end]`. To implement q3, we traverse the trajectories of devices in `B` backwards in time, by iterating over *epochs*, from the epoch that contains `end` to the epoch that contains `start`. This iteration over epochs is done outside 2PC since epochs are public. The data for each epoch is successively loaded into a temporary table, called `TT` in the query, and this table is then processed via a query in 2PC. The 2PC query traverses the trajectory of each device in `B` from the device's last encounter in the epoch to their earliest encounter, by following encounter pointers. For this traversal, 2PC makes data-dependent queries to the epoch's records in `TT` but since the records in each epoch are randomly shuffled per our ODR scheme, this does not reveal any secrets. To ensure that no secrets leak via the *number* of lookups in an epoch, we use a fixed, conservative number of lookups per device (in `B`) per epoch, fetching dummies when needed (the size of `B` is part of the query, hence, public). Every other device encountered by a device (in `B`) during the traversal is checked for membership in `A`, via a Bloom filter initialized with devices in `A` (the Bloom filter is implemented in 2PC). Whenever the membership test succeeds, we mark the device in `B` as having encountered someone in `A`. At the end, we simply count the number of marked devices.

In our experiments, it took (in 2PC) a constant 52ms to initialize the Bloom filter for $A = 10$, 27ms (std. dev. 2ms) to fetch an encounter from a shuffled view, 43ms (std. dev. 5ms) to check membership in the Bloom filter, and 10ms (std. dev. 2ms) to check that an encounter's time is between `start` and `end`. Assuming $B = 10$, a total of $e = 336$ epochs (corresponding to a period of 14 days with 1h epochs), and at most $n = 30$ encounters/person/h, the total query latency comes to $52 + (27 + 10 + 43) \cdot e \cdot n \cdot |\texttt{B}|$ = 2.24 hours.

Note that the ODR scheme improves the latency of q3 significantly by enabling secure data-dependent accesses. Without ODR, we would have to scan *all* encounters in the time interval `[start,end]`. Assuming, as before, that 11.85B encounters are generated in a country every day, and the interval `[start,end]` is 14 days long, this data-*in*dependent approach would have to process nearly $11.85\text{B}\cdot 14 = 165.9\text{B}$ encounter records in 2PC. By contrast, the data-dependent approach above processes a total of $336 \cdot 30 \cdot 10 = 100,800$ records in 2PC, which is over 6 orders of magnitude fewer records fetched and processed in 2PC.

An additional query: q4. Here, we describe another data dependent query q4 (Figure 9), which we also evaluate for latency. This query asks how many devices from set `B` encountered a device from the set `A` indirectly through an intermediate device (all within a given time interval). This query can be used to determine if two outbreaks of an epidemic (corresponding to the sets `A` and `B`) are indirectly connected over 1-hop.

We implemented and evaluated the latency of q4 in the same setting as q3 (§6.3). Unlike q3, which makes one traversal over the trajectories of devices in `B`, q4 uses *two* traversals. One traversal collects all encounters of devices in `A` moving forwards in time; the second traversal collects all encounters of devices in `B` moving backwards in time. We then sort the

(q4) Count #devices in `B` that encountered a device which *previously* encountered a device in `A`, with both encounters in the time interval `[start,end]`

```
WITH TT AS
 (SELECT * FROM T_P
  WHERE start<epoch<end)
SELECT COUNT(DISTINCT(T2.did2))
FROM (TT AS T1) JOIN (TT AS T2)
 ON T1.did2 == T2.did1
WHERE T1.did1∈A AND T2.did2∈B AND
 start<T1.time<T2.time<end
```

Figure 9: Query q4. The selection on the public attribute `epoch` is done outside 2PC using T_P's public index.

collected encounters of `A`∪`B` ascending by time, and make a linear pass over them maintaining two Bloom filters, one of all devices that have encountered a device in `A` and the other of devices in `B` who have encountered a device *already* in the first Bloom filter. The result of the query is the size of the second filter at the end.

In our experiments, Bloom filter operations take 709ms per encounter and there is a one-time setup cost of 129ms. Sorting the encounter list of `A`∪`B` has negligible cost in comparison. Assuming $e = 336$ epochs, $n = 30$ encounters/person/h, $|\texttt{A}| = 10$ and $|\texttt{B}| = 10$ (the same values that we assumed for q3), this encounter list has length $m = e \cdot n \cdot (|\texttt{A}| + |\texttt{B}|) = 201,600$. The total query latency is $129 + 27 \cdot e \cdot n \cdot (|\texttt{A}| + |\texttt{B}|) + (10 + 709) \cdot m$ = 1.74 days. In this case, the savings over a naive approach of scanning all encounter records are even more pronounced than those in the case of q3, since the naive approach would have to first sort 165.9B records in 2PC and then scan them maintaining the same two Bloom filters.

E Estimation of end-to-end query latency

We describe how we estimate the end-to-end latency of a query executed in 2PC using our MapReduce approach, as a function of the number of available machine pairs. By reversing the estimation function, we can also easily determine the number of machines needed to attain a given query latency.

We start with two basic mathematical facts that we need for our estimates.

Lemma 1. *Given n i.i.d. random variables $X_1, \ldots, X_n$ with normal distributions of mean μ and standard deviation σ, let $X = \max(X_1, \ldots, X_n)$. Then, $\mathbb{E}[X] \leq \mu + \sigma\sqrt{2\ln(n)}$.*

Proof. This is a folklore result. We provide a simple proof here. Let $t > 0$ be a parameter. Since e^y is a convex function of y, by Jensen's inequality, $e^{t\mathbb{E}[X]} \leq \mathbb{E}[e^{tX}] = \mathbb{E}[\max_i e^{tX_i}] \leq \mathbb{E}[\sum_{i=1}^n e^{tX_i}] = \sum_{i=1}^n \mathbb{E}[e^{tX_i}]$. From the moment generating function of the normal distribution, $\mathbb{E}[e^{tX_i}] = t\mu + \frac{1}{2}\sigma^2 t^2$. Hence, $e^{t\mathbb{E}[X]} \leq ne^{(t\mu + \frac{1}{2}\sigma^2 t^2)}$, and $\mathbb{E}[X] \leq \frac{\ln(n)}{t} + \mu + \frac{t\sigma^2}{2}$. The function on the right is minimized for $t = \frac{\sqrt{2\ln(n)}}{\sigma}$, and its minimum value is $\mu + \sigma\sqrt{2\ln(n)}$, as required. □

In the sequel, we let $\mathbb{E}[D;n]$ denote the expected value of the maximum of n i.i.d. random variables, each drawn from the normal distribution D. Lemma 1 says that

$$\mathbb{E}[D;n] \leq \mu + \sigma\sqrt{2\ln(n)}$$

where μ and σ are the mean and standard deviation of D.

Lemma 2. *For any natural number $K \geq 0$,*

$$\sqrt{K} + \sqrt{K-1} + \ldots + \sqrt{1} \leq \frac{2}{3}((K+1)^{3/2} - 1)$$

Proof.

$$\begin{array}{ll} & \sqrt{K} + \sqrt{K-1} + \ldots + \sqrt{1} \\ = & \int_K^{K+1} \sqrt{K}\, \mathrm{d}x + \ldots + \int_1^2 \sqrt{1}\, \mathrm{d}x \\ \leq & \int_K^{K+1} \sqrt{x}\, \mathrm{d}x + \ldots + \int_1^2 \sqrt{x}\, \mathrm{d}x \\ = & \int_1^{K+1} \sqrt{x}\, \mathrm{d}x \\ = & \frac{2}{3}((K+1)^{3/2} - 1) \end{array}$$

□

Now we explain our estimation of end-to-end latency. Suppose we have M available machines pairs and each machine has $2C$ available cores. This gives us a total of C *units of DualEx execution* per machine pair and a total of MC units of DualEx execution. In the sequel, we use the term *unit* to mean "a unit of DualEx execution".

For a given query, let the queried table have N records. We divide the records of the table into *chunks* of t records each, and then divide the chunks evenly among the MC units. So, each unit starts with N_c chunks, where

$$N_c = \frac{N}{tMC}$$

The best value of t is determined empirically: If t is too small, each mapping and initial reducing circuit does very little work (and setup costs dominate). If t is too high, the state of all parallel cores on a machine may not fit in memory. For tables in our evaluation, we usually pick t to be 10,000 records.

The query actually executes in 4 fine-grained stages. All but the last stage do *not* perform the final equality check of DualEx, but each stage does run the two symmetric DualEx 2PC computations.

1. (Unit-level) Each unit (2+2 cores on a machine pair) maps two chunks out of its N_c chunks and then reduces them to an intermediate result. Each unit then alternates mapping a new chunk and reducing the mapped chunk

with the intermediate result previously available on the unit, producing a new intermediate result. This second step is repeated $N_c - 2$ times on each unit till all chunks are consumed and one intermediate result is obtained on each unit. All MC units do these computations in parallel. Let D_{mmr} be the latency distribution of the first two maps followed by reduce on a single unit, and let D_{mr} be the latency distribution of each of one subsequent map+reduce on a single unit.

2. (Machine-pair-level) All C units on a single machine pair reduce their results to a single result using a balanced reduction tree. All M machine pairs do this independently in parallel. Let D_{mach} be the latency distribution of this on a single machine.

3. (Cross-machine) All intermediate results across all machine pairs are reduced using a balanced reduce tree to two final results, which are on a single machine. Only one unit per machine pair is used. At the end, one machine ends up with two results. Let D_{cmr} be the latency distribution of the following cross-machine computation: Two pairs of machines reduce in parallel and one pair sends its result to the other pair at the end. We call this a cross-machine reduction unit (*cmru*).

4. (Final reduce) The machine pair getting the last two intermediate results performs one final reduce with the DualEx equality check at the end. Let the latency distribution of this final step be D_{fr}.

We empirically estimate D_{mmr}, D_{mr}, D_{mach}, D_{cmr}, and D_{fr} by observing the means and standard deviations of the corresponding latencies while running a query on a small number of machines. Let μ_{mmr} and σ_{mmr} denote the mean and standard deviations of D_{mmr}, and similarly for the remaining four distributions. For our end-to-end latency estimate we model each of these distributions as a *normal* distribution with the measured mean and standard deviation.

There is no synchronization at the end of each stage in our implementation, e.g., if all units on a machine pair have finished stage 1, that machine pair goes ahead with stage 2 without waiting for other machine pairs to finish stage 1. However, we estimate the end-to-end query latency *conservatively* by instead calculating the latency of a hypothetical execution model where there is a full, instantaneous synchronization at the end of each of the first three stages. This latter latency is definitely a conservative upper bound on the actual latency, and is much easier to calculate.

Let cost$_1$–cost$_4$ be the expected costs of the four stages above in the conservative (synchronizing) model. We upper-bound each of these separately.

Estimating cost$_1$ The latency of stage 1 on each unit has a normal distribution given by $D_1 = D_{mmr} + (N_c - 2)D_{mr}$. From standard properties of normal distributions, D_1 has mean and standard deviation μ_1 and σ_1 where

$$\begin{aligned} \mu_1 &= \mu_{mmr} + (N_c - 2)\mu_{mr} \\ \sigma_1 &= \sqrt{\sigma_{mmr}^2 + (N_c - 2)\sigma_{mr}^2} \end{aligned}$$

Then, since stage 1 consists of MC parallel units, we have:

$$\text{cost}_1 = \mathbb{E}[D_1; MC] \le \mu_1 + \sigma_1\sqrt{2\ln(MC)}$$

Estimating cost$_2$ In stage 2, M machines operate in parallel, each with latency D_{mach}. Thus, we have:

$$\text{cost}_2 = \mathbb{E}[D_{mach}; M] \le \mu_{mach} + \sigma_{mach}\sqrt{2\ln(M)}$$

Estimating cost$_3$ Stage 3 is a tree-shaped cross-machine reduce. The number of levels in this tree is K where:

$$K = \lceil \log_2(M) \rceil$$

At the first level, we have $\lceil M/4 \rceil$ parallel cmrus. At the second level, we have $\lceil M/8 \rceil$ parallel cmrus, and so on, till we have only 1 cmru. Hence, we get:

$$\begin{aligned} \text{cost}_3 &\le \sum_{i=2}^{K} \mathbb{E}[D_{cmr}; \lceil M/(2^i) \rceil] \\ &\le \sum_{i=2}^{K} \left(\mu_{cmr} + \sigma_{cmr}\sqrt{2\ln\left(\lceil M/(2^i) \rceil\right)} \right) \\ &= (K-1)\mu_{cmr} + \sigma_{cmr}\sqrt{2\ln 2}\sum_{i=2}^{K}\sqrt{\left(\lceil M/(2^i) \rceil\right)} \\ &\le (K-1)\mu_{cmr} + \sigma_{cmr}\sqrt{2\ln 2}\sum_{i=1}^{K-2}\sqrt{i} \\ &\le (K-1)\mu_{cmr} + \tfrac{2}{3}\sigma_{cmr}\sqrt{2\ln 2}\left((K-1)^{3/2} - 1\right) \end{aligned}$$

Estimating cost$_4$ This cost is immediate:

$$\text{cost}_4 = \mu_{fr}$$

Our computed upper-bound on the query latency is then cost$_1$ + cost$_2$ + cost$_3$ + cost$_4$.

F Proofs of Security

In this section, we prove the security of CoVault's MtS and modified DualEx protocol (which we call *asymmetric DualEx* here).

In §F.1, we introduce notation and the preliminaries necessary for our proofs of security: message authentication codes and their properties (§F.2) and secure secret sharing (§F.2.1). In §F.3, we combine MACs and secret sharing to give a construction of a secure secret sharing scheme and a proof of its security. In §F.4, we modify DualEx, a symmetric 2PC protocol with one-bit leakage, to enable asymmetric outputs, then prove that this modified protocol (asymmetric DualEx) is secure.

F.1 Notation and preliminaries

For any deterministic functionality $\mathcal{H}$, let $\mathcal{H}^-$ be the functionality that computes $\mathcal{H}$ with one-bit leakage. That is, for a malicious party P_i, the two-party functionality $\mathcal{H}^-$ returns the same output as $\mathcal{H}$ along with some one-bit leakage function ℓ of P_i's choice applied to the other party's input x ($|\ell(x)| = 1$). For a randomized functionality, ℓ can depend on the other party's input and the randomness used by the functionality.

Throughout this appendix, we assume that some algorithms can output a distinguished symbol $\perp$. Our security proofs consider PPT (probabilistic polynomial-time) adversaries $\mathcal{A}$. As is standard in cryptography, the security properties of the schemes we introduce are proven by showing that the *advantage* (defined separately for every property) of a PPT adversary $\mathcal{A}$ is upper-bounded by a *negligible function*:

Definition 1 (negligible function)**.** *A function f is negligible if for all $c \in \mathbb{N}$ there exists an $N \in \mathbb{N}$ such that $f(n) < n^{-c}$ for all $n > N$.*

F.2 Message authentication codes

We use MACs to provide integrity for the data sent by a device to the system. When a device generates data, it secret-shares it before sending one share to each of the two TEEs. Because standard, additive secret sharing schemes are malleable, we utilize message authentication codes (MACs) to add integrity.

Definition 2 (message authentication code (MAC))**.** *A message authentication code (MAC) is a triple of polynomial-time algorithms $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ such that*

- KeyGen *takes as input a security parameter 1^κ and outputs a random key $k \leftarrow_\$ \mathcal{D}_\kappa$*
- Mac *takes as input a key k in some domain $\mathcal{D}_\kappa$ associated with a security parameter 1^κ and a message m in some domain $\mathcal{D}_m$ and outputs a tag t.*
- Verify *takes as input a key $k \in \mathcal{D}_\kappa$, a message $m \in \mathcal{D}_m$, and a tag t and outputs one of two distinguished symbols $\top, \perp$.*

For correctness, we require that for all $m \in \mathcal{D}_m$ and $k \in \mathcal{D}_\kappa$, $\mathsf{Verify}(k, m, \mathsf{Mac}(k, m)) = \top$.

Notation. Define $\mathsf{Mac}_k(m) := \mathsf{Mac}(k, m)$ and $\mathsf{Verify}_k(m, t) := \mathsf{Verify}(k, m, t)$.

The security guarantees of MACs span a wide range. We define the relevant notions below.

One-time strong unforgeability. Informally, one-time strong unforgeability says that it is infeasible to forge a tag on a message without knowing the key (including a new tag on a message for which the attacker already knows a tag).

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to M as $\mathsf{MAC1\text{-}sforge\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ (m', t') \leftarrow \mathcal{A}(t) \end{array} : \begin{array}{c} (m', t') \neq (m, t) \ \wedge \\ \mathsf{Verify}_k(m', t') = \top \end{array}\right].$$

Definition 3 (one-time strong unforgeability)**.** *We say a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ is* one-time strongly unforgeable *(alternatively,* one-time strongly secure*) if, for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that $\mathsf{MAC1\text{-}sforge\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$.*

One-time key authenticity. Standard notions of MAC security deal only with the consequences of an attacker viewing a message-tag pair. In our setting, we introduce a new notion of security for MACs which considers the case in which $\mathcal{A}$ has access to a message-key pair. Informally, a MAC is key authentic if it is difficult to find a new key which still authenticates a given message-tag pair.

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to M as $\mathsf{MAC1\text{-}kauth\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ k' \leftarrow \mathcal{A}(k) \end{array} : \begin{array}{c} k' \neq k \ \wedge \\ \mathsf{Verify}_{k'}(m, t) = \top \end{array}\right].$$

Definition 4 (one-time key authenticity)**.** *We say a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ is* one-time key authentic *if, for all PPT adversaries $\mathcal{A}$, there exists a nonnegligible function negl such that $\mathsf{MAC1\text{-}kauth\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$.*

One-time non-malleability. In this case, we require that an adversary cannot cause a fixed, known tag to verify a message even if it can modify the message and key by some additive shift.

Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to M as $\mathsf{MAC1\text{-}nmall\text{-}adv}[\mathcal{A}, M] :=$

$$\Pr\left[\begin{array}{c} m \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ t \leftarrow \mathsf{Mac}_k(m); \\ (\Delta_m, \Delta_k) \leftarrow \mathcal{A}(t) \end{array} : \begin{array}{c} (\Delta_m, \Delta_k) \neq (0, 0) \ \wedge \\ \mathsf{Verify}_{k+\Delta_k}(m + \Delta_m, t) = \top \end{array}\right].$$

Definition 5 (one-time non-malleability)**.** *We say a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ is* one-time non-malleable *if, for all PPT adversaries $\mathcal{A}$, there exists a nonnegligible function negl such that $\mathsf{MAC1\text{-}nmall\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$.*

Privacy. Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to M as $\mathsf{MAC\text{-}priv\text{-}adv}[\mathcal{A}, M] :=$

$$\left| \Pr\left[\begin{array}{c} (m_0, m_1) \leftarrow \mathcal{A}(1^\kappa); \\ k \leftarrow \mathsf{KeyGen}(1^\kappa); \\ b \leftarrow_{\$} \{0,1\}; \\ t_b \leftarrow \mathsf{Mac}_k(m_b) \end{array} : \mathcal{A}(t_b) = b \right] - \frac{1}{2} \right|.$$

Definition 6 (privacy)**.** *We say a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ is* private *if, for all* PPT *adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathsf{MAC\text{-}priv\text{-}adv}[\mathcal{A}, M] \leq negl(\kappa)$.*

F.2.1 Secure secret-sharing

As mentioned previously, encryption secret-shares data between two TEEs. Basic secret-sharing schemes only deal with privacy of the shared data and do not consider accuracy of reconstruction. In this section, we define (in addition to the notion of privacy) a notion of authenticity which captures the property that the holder of a share of the data cannot, without being detected, change its share in a way that causes the reconstructed data to be altered. A secret sharing scheme with this additional property is called "secure" and will ultimately be achieved by leveraging MACs (§F.3).

Definition 7 (secret-sharing scheme)**.** *A pair of polynomial-time algorithms $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is a (two-party) secret-sharing scheme if*

- Share *takes as input a security parameter 1^κ and a value x in the domain $\mathcal{D}_\kappa$ associated with κ (e.g., $\mathcal{D}_\kappa = \{0,1\}^\kappa$) and outputs two shares $\mathsf{sh}_1, \mathsf{sh}_2$. We assume κ is implicit in each share.*
- Rec *takes as input two shares and outputs either a value $y \in \mathcal{D}_\kappa$ or $\perp$.*

For correctness, we require that for all κ and $x \in \mathcal{D}_\kappa$, $\mathsf{Rec}(\mathsf{Share}(1^\kappa, x)) = x$.

Privacy. Informally, privacy says that, given a share of one of two values x_0, x_1, an adversary $\mathcal{A}$ cannot distinguish whether x_0 or x_1 was shared.

Let $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ be a secret-sharing scheme. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s *advantage* with respect to Σ as $\mathsf{SS\text{-}priv\text{-}adv}[\mathcal{A}, \Sigma] :=$

$$\left| \Pr\left[\begin{array}{c} (x_0, x_1, i) \leftarrow \mathcal{A}(1^\kappa); \\ b \leftarrow_{\$} \{0,1\}; \\ (\mathsf{sh}_{b,1}, \mathsf{sh}_{b,2}) \leftarrow \mathsf{Share}(1^\kappa, x_b) \end{array} : \mathcal{A}(\mathsf{sh}_{b,i}) = b \right] - \frac{1}{2} \right|.$$

Definition 8 (privacy)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* private *if, for all* PPT *adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathsf{SS\text{-}priv\text{-}adv}[\mathcal{A}, \Sigma] \leq negl(\kappa)$.*

Authenticity. Informally, authenticity guarantees that any modification to a share will result in Rec returning $\perp$ with high probability.

Let $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ be a secret-sharing scheme. For a given PPT adversary $\mathcal{A}$, we define $\mathcal{A}$'s advantage with respect to Σ as $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A}, \Sigma] :=$

$$\Pr\left[\begin{array}{c} (x, i) \leftarrow \mathcal{A}(1^\kappa); \\ (\mathsf{sh}_1, \mathsf{sh}_2) \leftarrow \mathsf{Share}(1^\kappa, x); \\ \mathsf{sh}'_i \leftarrow \mathcal{A}(\mathsf{sh}_i) \end{array} : \begin{array}{c} \mathsf{sh}'_i \neq \mathsf{sh}_i \land \\ \mathsf{Rec}(\mathsf{sh}'_i, \mathsf{sh}_{3-i}) \neq \perp \end{array} \right].$$

Definition 9 (authenticity)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* authenticated *if, for all* PPT*adversaries $\mathcal{A}$, there is a negligible function negl such that $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A}, \Sigma] \leq negl(\kappa)$.*

Definition 10 (security)**.** *We say a secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ is* secure *if it is both private and authenticated.*

F.3 Secure secret sharing construction

There are several ways to construct a secure secret-sharing scheme $\Sigma_M = (\mathsf{Share}_M, \mathsf{Rec}_M)$ using a MAC $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ and base (non-authenticated) secret-sharing scheme $\Sigma = (\mathsf{Share}, \mathsf{Rec})$. We introduce MAC-then-Share, the construction we use, and analyze its security.

Definition 11 (MAC-then-share (MtS))**.** *Let $M = (\mathsf{KeyGen}, \mathsf{Mac}, \mathsf{Verify})$ be a MAC and $\Sigma = (\mathsf{Share}, \mathsf{Rec})$ a secret-sharing scheme. Define $\Sigma_M = (\mathsf{Share}_M, \mathsf{Rec}_M)$, the MtS secret-sharing scheme based on M and Σ, as follows:*

- Share_M *takes as input a security parameter 1^κ and a value x in the domain $\mathcal{D}_\kappa$ associated with κ. It generates a single key k using* KeyGen *and uses it to computes one tag t on the secret value x as $t := \mathsf{Mac}_k(x)$. Now it shares x, k by computing $(x_1, x_2) \leftarrow \mathsf{Share}(x)$ and $(k_1, k_2) \leftarrow \mathsf{Share}(k)$, and outputs two shares sh_1, sh_2 with $\mathsf{sh}_i := (x_i, k_i, t)$.*
- Rec_M *takes as input two shares. If the tags match, it reconstructs $y := \mathsf{Rec}(x_1, x_2)$ and $k' := \mathsf{Rec}(k_1, k_2)$. Then, if $\mathsf{Verify}_{k'}(y, t) = \top$, it returns y; otherwise it returns $\perp$.*

What properties are required of the MAC and secret-sharing scheme in order for their composition to be a secure secret-sharing scheme? In Theorem 1, we show that the MAC must meet all four properties presented in Section F.2 in order for the MtS construction to be secure.

Theorem 1. *If M is a one-time strongly unforgeable, key authentic, non-malleable, and private MAC and Σ an additive secret-sharing scheme, then the MtS secret-sharing scheme Σ_M constructed from M and Σ is secure.*

Proof. The privacy of Σ_M follows directly from privacy of M (since $\mathsf{Mac}_k(x)$ reveals nothing about x) and Σ (x_i also reveals nothing about x).

Authenticity of Σ_M is a bit more unwieldy. We prove the contrapositive that if Σ_M is not authenticated, then either (1) M is not strongly secure, (2) M lacks key authenticity, or (3) M is malleable. As before, we do this by reduction of an adversary $\mathcal{A}$ with $\mathsf{SS\text{-}auth\text{-}adv}[\mathcal{A},\Sigma_M]$ nonnegligible to three adversaries for each of the three corresponding games, and show that at least one of them has nonnegligible advantage in its game.

First, we construct an adversary $\mathcal{A}_{\mathsf{sforge}}$ for the game $\mathsf{MAC1\text{-}sforge}_{\mathcal{A}_{\mathsf{sforge}},M}$. $\mathcal{A}_{\mathsf{sforge}}$ receives x,i from $\mathcal{A}$ and constructs a MtS triple $\mathsf{sh}_i := (x_i,k_i,t_i)$ as follows: it sends x to its game to get $t := \mathsf{Mac}_k(x)$, picks a random $k_i \in \mathcal{D}_\mathsf{K}$, and runs Share on x,t to get x_1,x_2,t_1,t_2. Now $\mathcal{A}_{\mathsf{sforge}}$ runs $\mathcal{A}$ on sh_i to get $\mathsf{sh}'_i := (x'_i,k'_i,t'_i)$ and returns (x',t'), where $x' \leftarrow \mathsf{Rec}(x'_i,x_{3-i})$ and $t' \leftarrow \mathsf{Rec}(t'_i,t_{3-i})$.

Next, we construct an adversary $\mathcal{A}_{\mathsf{kauth}}$ for the game $\mathsf{MAC\text{-}kauth}_{\mathcal{A}_{\mathsf{kauth}},M}$. $\mathcal{A}_{\mathsf{kauth}}$ is given x,i by $\mathcal{A}$ and sends x to its game, which sends back a key k. It uses this key to construct a MtS triple $\mathsf{sh}_i := (x_i,k_i,t)$, computing $t := \mathsf{Mac}_{k_i}(x)$ and running Share on x,k to get x_1,x_2,k_1,k_2. Now $\mathcal{A}_{\mathsf{kauth}}$ runs $\mathcal{A}$ on sh_i to get $\mathsf{sh}'_i := (x'_i,k'_i,t')$ and returns $k' \leftarrow \mathsf{Rec}(k'_i,k_{3-i})$.

Third, we construct an adversary $\mathcal{A}_{\mathsf{nmall}_M}$ for the MAC non-malleability game.$\mathcal{A}_{\mathsf{nmall}_M}$ is given x,i by $\mathcal{A}$ and sends x to its game to receive t. It constructs anMtS triple $\mathsf{sh}_i := (x_i,k_i,t_i)$ by choosing a random $k \in \mathcal{D}_\mathsf{K}$ and running Share on x,k,t. Now $\mathcal{A}_{\mathsf{nmall}_M}$ runs $\mathcal{A}$ on sh_i to get $\mathsf{sh}'_i := (x'_i,k'_i,t')$. It computes $x' \leftarrow \mathsf{Rec}(x'_i,x_{3-i}), k' \leftarrow \mathsf{Rec}(k'_i,k_{3-i})$, and returns (x',k').

We now analyze the winning probability of each of the three adversaries. By our assumption, $\mathcal{A}$ wins its game with nonnegligible probability, implying $t' = t$. If $\mathcal{A}$ returns sh'_i such that $k'_i = k_i$ with nonnegligible probability, then $\mathcal{A}_{\mathsf{sforge}}$ has nonnegligible advantage and we are done. If not, then with nonnegligible probability, $\mathcal{A}$ returns sh'_i with $k'_i \neq k_i$. If $x'_i = x_i$ a nonnegligible fraction of the time, $\mathcal{A}_{\mathsf{kauth}}$ has nonnegligible advantage in its game, and we are done. Otherwise, $x'_i \neq x_i$ and $\mathcal{A}_{\mathsf{nmall}_M}$ has nonnegligible advantage in the non-malleability game for M: it has a pair $(\Delta_m := x'_i - x_i, \Delta_k := k'_i - k_i)$ such that $\mathsf{Verify}_{k'_i+k_{3-i}}(x'_i + x_{3-i},t) = \mathsf{Verify}_{(k_i+k_{3-i})+\Delta_k}((x_i+x_{3-i})+\Delta_m,t) = \top$.

Thus, if M is authenticated, it is not strongly secure, lacks key authenticity, or is malleable, all of which contradict our assumptions. □

Note that the proof also holds for the XOR secret sharing scheme by substituting $+,-$ for $\oplus$.

Concrete Construction. CoVault uses KMAC256 [57], a NIST-standardized SHA3-based MAC. It can be abstracted as follows, where $||$ indicates concatenation. (We omit some additional parameters which are constant and public in CoVault; see Appendix A in [57] and Section 6.1 in [40].)

KeyGen: Use $\mathsf{SHA3}$'s key generation algorithm.

$\mathsf{Mac}_k(m)$: Compute $h := \mathsf{SHA3}(k)$ and announce it publicly. Output $t := \mathsf{SHA3}(k||m)$.

$\mathsf{Verify}_k(m,t)$: Output $\top$ iff $\mathsf{SHA3}(k) = h \wedge \mathsf{SHA3}(k||m) = t$.

This MAC meets the conditions of Theorem 1 in the random oracle model (ROM):

- **one-time strong unforgeability:** Due to randomness of the output $\mathsf{SHA3}(k||m')$.
- **one-time key authenticity:** Due to collision-resistance, which guarantees that for $k \neq k'$ we have with overwhelming probability that $\mathsf{SHA3}(k||m) \neq \mathsf{SHA3}(k'||m)$.
- **one-time non-malleability:** Again due to collision-resistance, since for $(k,m) \neq (k',m')$ we have with overwhelming probability that $\mathsf{SHA3}(k||m) \neq \mathsf{SHA3}(k'||m')$.
- **privacy:** Due to randomness of the output of SHA3.

Hence, CoVault's MtS construction of Σ_M with Σ as the additive secret sharing scheme and M as KMAC256 is secure.

F.4 DualEx functionality with asymmetric outputs

Before turning to the use of our secure secret sharing construction within a secure multi-party (two-party) computation protocol, we must discuss the exact protocol we use and its non-standard security guarantees.

We use the DualEx protocol [54] as a building block. DualEx guarantees security against malicious adversaries at almost the cost of of semi-honest protocols, but with the following caveats: it can compute any *symmetric* two-party functionality $\mathcal{F}_{\text{sym}}$, and it does so with one-bit leakage. By symmetric we mean that both parties are required to receive the same output.

In this section, we will address how to modify DualEx to allow for the computation of *asymmetric* functionalities while maintaining the same security guarantees (malicious with one-bit leakage). This modified DualEx, which we dub "asymmetric DualEx", will be used to implement the core decryption functionality, and the one-bit leakage will carry through the remainder of the proofs.

Let $f : \mathcal{X} \times \mathcal{X} \to \mathcal{Y}$ be some function. In the our setting, we want the output of f to be shared between the parties so that neither learns the output. More specifically, we want to compute the functionality $\mathcal{F}_{\text{asym}}$ that takes as input x_1 from one party and x_2 from the other, and returns a uniformly random r to the first party and $f(x_1,x_2) \oplus r$ to the second party. When P_1 is honest, r is chosen by $\mathcal{F}_{\text{asym}}$; when P_1 is malicious, P_1 chooses r.

To do this, we construct a two-party protocol Π that computes $\mathcal{F}_{\text{asym}}$ via access to the symmetric functionality $\mathcal{F}_{\text{sym}}$ that takes inputs $(x_1,r_1),(x_2,r_2) \in \mathcal{X} \times \mathcal{Y}$ from the parties, computes $y := f(x_1,x_2) \oplus r_1 \oplus r_2$, and returns y to both parties. The protocol proceeds as follows:

- Each party P_i holds an input $x_i \in \mathcal{X}$. It additionally samples a uniform blinding value $r_i \in \mathcal{Y}$. The parties then provide their inputs (x_1, r_1) and (x_2, r_2), respectively, to $\mathcal{F}_{\text{sym}}$.
- Both parties receive in return $y := f(x_1, x_2) \oplus r_1 \oplus r_2$.
- P_1 computes its output as $y_1 := r_1$, while P_2 computes its output as $y_2 := y \oplus r_2$.

Notice that $y \oplus r_2 = f(x_1, x_2) \oplus r_1$, so the protocol outputs $f(x_1, x_2) \oplus r_1$ to the second party and the parties now hold shares of $f(x_1, x_2)$. Below we show that this modified protocol maintains the same security guarantees as the base DualEx.

Theorem 2. Π *securely computes $\mathcal{F}^-_{asym}$ against malicious adversaries in the $\mathcal{F}^-_{sym}$-hybrid model.*

Proof. Let $\mathcal{A}$ be a PPT adversary corrupting party P_i. To prove the security of Π against malicious adversaries in the $\mathcal{F}^-_{\text{sym}}$-hybrid world, we give an adversary $\mathcal{S}_i$ in the ideal world that simulates an execution of Π in the hybrid world. We first consider the case of a corrupted P_1.

Simulator $\mathcal{S}_1$: $\mathcal{S}_1$ has access to the ideal functionality $\mathcal{F}^-_{\text{asym}}$ computing $f(x_1, x_2) \oplus r$ with one-bit leakage. Given f, $\mathcal{S}_1$ works as follows:

- Receive inputs x'_1, r'_1, and a leakage function $\ell : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ from P_1.
- Sample $y^* \leftarrow_{\$} \mathcal{Y}$. Convert ℓ into a function $\ell^* : \mathcal{X} \to \{0,1\}$ by letting $\ell^*(x) = \ell(x, y^* \oplus r'_1 \oplus f(x'_1, x))$ for all $x \in \mathcal{X}$.
- Send (x'_1, r'_1) and ℓ^* to the ideal functionality $\mathcal{F}^-_{\text{asym}}$; the honest party sends x_2 to $\mathcal{F}^-_{\text{asym}}$.
- Receive in return from $\mathcal{F}^-_{\text{asym}}$ r'_1 and some one-bit leakage $b^* := \ell^*(x_2)$. The honest party receives $y^*_2 := f(x'_1, x_2) \oplus r'_1$.
- Send y^*, b^* to P_1.

We prove indistinguishability of the following distribution ensembles.

Ideal experiment. This is defined by the interaction of $\mathcal{S}_1$ with the ideal functionality $\mathcal{F}^-_{\text{asym}}$. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs what it received from the experiment, namely y^*_2. Let $\text{IDEAL}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the adversary $\mathcal{S}_1$ and the output of the honest party. Concretely,

$$\text{IDEAL}_{\mathcal{S}_1}(1^\kappa, x_1, x_2, \ell, f) = ((y^*, b^*), y^*_2).$$

Hybrid experiment. let $\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the view of the adversary $\mathcal{A}$ and the output of the honest party in the $\mathcal{F}^-_{\text{sym}}$-hybrid world. Concretely,

$$\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f) = ((y, b), y_2).$$

$\mathcal{S}_1$ perfectly simulates the view of a malicious P_1 in the hybrid world. Because r_2 is chosen uniformly at random, both y^* and y are distributed uniformly at random and are thus perfectly indistinguishable. By the definition of ℓ^*, $b^* = \ell(x_2, y^* \oplus f(x'_1, x_2) \oplus r'_1$. Furthermore, by the definition of y, $r_2 = y \oplus f(x'_1, x_2) \oplus r'_1$, so $b = \ell(x_2, r_2)$ is perfectly indistinguishable from b^*. Hence the joint distributions of (y^*, b^*) and (y, b) are perfectly indistinguishable. Finally, y^*_2 and y_2 are identical by the definition of y_2 and thus perfectly indistinguishable.

Next, we give a simulator $\mathcal{S}_2$ for the case of a corrupted P_2.

Simulator $\mathcal{S}_2$: $\mathcal{S}_2$ has access to the ideal functionality $\mathcal{F}^-_{\text{asym}}$ computing $f(x_1, x_2) \oplus r$ with one-bit leakage. Given f, $\mathcal{S}_2$ works as follows:

- Receive inputs x'_2, r'_2, and a leakage function ℓ from P_2.
- Forward x'_2 and ℓ to the ideal functionality $\mathcal{F}^-_{\text{asym}}$; the honest party sends x_1 to $\mathcal{F}^-_{\text{asym}}$.
- Receive in return from $\mathcal{F}^-_{\text{asym}}$ the output $z := f(x_1, x'_2) \oplus r_1$ for uniformly random r_1 and some one-bit leakage $b := \ell(x_1)$. The honest party receives r_1.
- Compute $y^* := z \oplus r'_2$. Send y^*, b to P_2.

We prove indistinguishability of the following distribution ensembles.

Ideal experiment. This is defined by the interaction of $\mathcal{S}_2$ with the ideal functionality $\mathcal{F}^-_{\text{asym}}$. $\mathcal{A}$ outputs an arbitrary function of its view; the honest party outputs its honestly computed value of y_1, namely r_1. Let $\text{IDEAL}_{\mathcal{S}_2}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the output of the honest party and the output of the adversary $\mathcal{S}_2$. Concretely,

$$\text{IDEAL}_{\mathcal{S}_2}(\kappa, x_1, x_2, \ell, f) = (r_1, (y^*, b)).$$

Hybrid experiment. let $\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f)$ be the joint random variable containing the the output of the honest party and the view of the adversary $\mathcal{A}$ in the $\mathcal{F}^-_{\text{sym}}$-hybrid world. Concretely,

$$\text{H}_{\mathcal{A}}(1^\kappa, x_1, x_2, \ell, f) = (r_1, (y, b)).$$

$\mathcal{S}_2$ perfectly simulates the view of a malicious P_2 in the hybrid world, since $y^* = z \oplus r'_2 = f(x_1, x'_2) \oplus r_1 \oplus r'_2 = y$.

Therefore, Π is secure (up to 1 bit of leakage) against malicious adversaries in the $\mathcal{F}^-_{\text{sym}}$-hybrid model. □

Notice that the DualEx protocol is an instantiation of $\mathcal{F}_{\text{sym}}^{-}$, so our modified protocol is a real-world protocol with the same security guarantees against malicious parties as the original DualEx: privacy up to one bit of leakage and full correctness of the output.