Flock: A Framework for *Deploying* On-Demand Distributed Trust

Darya Kaviani[1]* Sijun Tan[1]* Pravein Govindan Kannan[2] Raluca Ada Popa[1]

[1]*UC Berkeley* [2]*IBM Research*

Abstract

Recent years have exhibited an increase in applications that *distribute trust* across n servers to protect user data from a central point of attack. However, these deployments remain limited due to a core obstacle: establishing n distinct trust domains. An application provider, a *single* trust domain, cannot directly deploy *multiple* trust domains. As a result, application providers forge business relationships to enlist third-parties as trust domains, which is a manual, lengthy, and expensive process, inaccessible to many application developers.

We introduce the *on-demand distributed-trust architecture* that enables an application provider to deploy distributed trust automatically and immediately without controlling the other trust domains. The insight lies in *reversing* the deployment method such that each user's client drives deployment instead of the application provider. While at a first glance, this approach appears infeasible due to cost, performance, and resource abuse concerns, our system Flock resolves these challenges. We implement and evaluate Flock on 3 major cloud providers and 8 distributed-trust applications. On average, Flock achieves 1.05x the latency and 0.68-2.27x the cloud cost of a traditional distributed-trust deployment, without reliance on third-party relationships.

1 Introduction

Existing systems typically suffer from a central point of attack: an application provider holding many users' private data becomes the target of data breaches [117]. As a result, an increasing number of applications are using *distributed trust* [35, 38, 46, 47, 66, 67, 72, 77, 108, 161, 177–179, 195, 198]. This powerful paradigm avoids a central point of attack by distributing the users' sensitive data among n parties to protect its confidentiality or integrity. A typical requirement is that these n parties are in different *trust domains*, each of which corresponds to a distinct organization to ensure that they are controlled by different entities. Fig. 1 illustrates the stakeholders of this setting: the application provider, its users, and $n-1$ other trust domains. Even if $n-1$ out of n parties are compromised, the sensitive data remains secure: an attacker would have to breach all n parties to compromise the sensitive data. Various cryptographic tools rely on distributed trust, such as secure multi-party computation (MPC) [112, 128, 147, 163, 164, 202, 213, 219] and two-party private information retrieval (PIR) [105, 116, 124, 146].

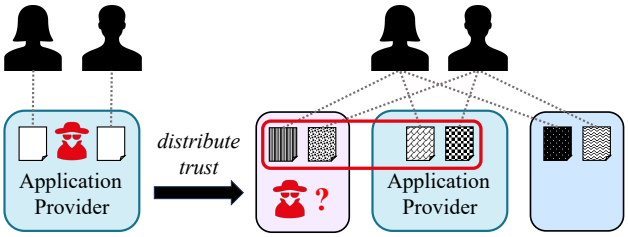


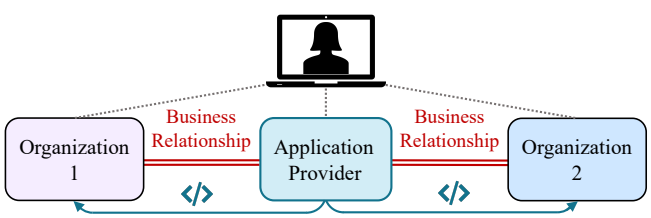
Figure 1: Secret-sharing data over 3 trust domains: breaching 2 trust domains reveals nothing.

Recent years have exhibited an increased adoption of distributed trust by application providers who aim to *protect their users' data* [161] (§6), including Signal [178, 179], Coinbase [35, 177], Fireblocks [46], Google [108], Apple [108], Meta [198], and J.P. Morgan [195]. For example, Signal's secure value recovery project aims to enable users to securely back up their private keys through distributed trust [178, 179]. Likewise, MPC wallets [35, 38, 42, 46, 47, 66, 72, 77, 97, 134, 177], including Coinbase [35, 177] and Fireblocks [46], secure billions of dollars by distributing their users' private keys and using MPC for signing [196].

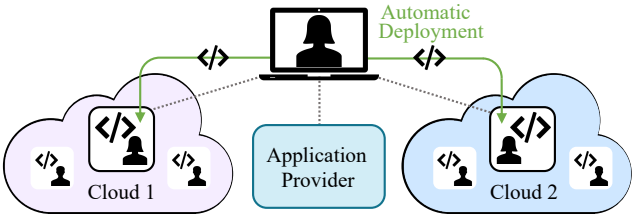
The deployment challenge. Despite this interest, the adoption of distributed-trust applications remains limited. Recent works [130, 178, 198] discuss a core challenge: the *difficulty of deploying n distinct trust domains*. Indeed, the application provider must find $n-1$ organizations in different trust domains, who are willing to run the provider's workload while restricting access to anyone, including the provider itself. These organizations must offer sufficient availability, security, fault tolerance, logging, swift recovery, and must have a credible reputation in the user community—criteria that have empirically been challenging to satisfy [178]. Moreover, such business relationships are costly and require both time and manual effort to set up. While large corporations were able to forge such partnerships [108, 198], this is a barrier to entry for application developers [178] who lack the same resources.

For clarity, consider the **running example** of digital asset

*Equal contribution



(a) *Traditional Distributed Trust*: The application provider forms manual, costly relationships with third-parties, who deploy code.



(b) *On-Demand Distributed Trust*: The user's client automatically deploys code to $n-1$ clouds, which also have other user deployments.

Figure 2: Deployment workflow in traditional vs. on-demand distributed trust.

custody (although our work applies to a wide range of applications, as discussed in §4). Cryptocurrency [119, 189] users exchange digital assets by signing transactions with their private keys. A compromised private key can be used to steal any assets in the user's wallet [101, 118, 142, 184, 203]. This is why wallets like Coinbase [35, 177] and Fireblocks [46] secret-share the private key among n different parties. When Alice initiates a transaction, the n parties engage in MPC, inside which they reconstruct her key and sign the transaction. Although their desired design is to secret-share among multiple entities, many wallets only secret-share between the application provider and the client due to the difficulty of setting up other trust domains [178].

More generally, many academic papers on distributed trust simply assume the presence of n servers in different trust domains [126, 127, 131, 132, 138, 149, 165, 211], but do not offer guidance on how to establish such deployments in practice. In this paper, we address the following systems challenge with deploying distributed trust:

> How can an application provider, which is inherently a *single* trust domain, deploy a *multi-trust-domain* system?

To address this challenge, we propose the **on-demand distributed-trust** architecture, which enables an application provider to offer distributed-trust services to its users *automatically*, *immediately*, and *without* requiring third-parties. This is the first architecture that removes the burden of setting up cumbersome, manual business relationships with $n-1$ parties. We provide an intuition and overview in §1.1. Our second contribution is the design and implementation of *Flock*,[1] a system that realizes our on-demand distributed-trust architecture across major cloud providers. Flock enables an application provider to setup $n-1$ trust domains on $n-1$ cloud providers without the application provider being able to control the deployment. A straightforward instantiation of the on-demand architecture suffers from significant scalability and security issues. Flock overcomes these challenges with two additional technical contributions that are reusable beyond Flock: the *Flock Relay* and a *three-tier authentication protocol*, both overviewed in §1.2.

[1] Multi-species bird flocks may not always trust each other, but flock together among the clouds to increase the likelihood of detecting predators [62].

1.1 On-Demand Distributed Trust

To understand our approach and its challenges, first consider a natural strawman of using n reputable cloud providers as trust domains. Recent years have exhibited a spike in multi-cloud services [4, 11, 63, 94] and multi-cloud applications [12, 135, 153, 156, 193, 204, 218]. While *accounts* within a *single* cloud can be accessed by the cloud's administrators, *distinct* cloud providers have their own data centers, storage, compute resources, networking solutions, and, crucially, administrators. While the clouds are indeed distinct trust domains, this approach suffers from a central point of attack: the application provider that deploys VMs to each cloud controls them all, reducing the system to a single trust domain. As we discuss in §6, some proposals attempt to approximate trust domains with hardware enclaves [130, 179], but enclaves are vulnerable to side-channel attacks [115, 175, 188, 209] that allow the application provider to once again fully control the deployment. Hence, we seek an approach to deploying $n-1$ trust domains that the application provider cannot control, without depending on trusted hardware.

The primary insight of on-demand distributed trust is a paradigm shift in the deployment approach: Instead of the application provider deploying the n parties to distinct clouds, *each user* drives the deployment for *their own data*. At a first glance, this approach appears infeasible with respect to ease-of-use and cost because every user has a *separate* deployment. From a security standpoint, though, it is fitting: a user Alice is the trusted owner of her *own* sensitive data. Hence, she can deploy VMs across $n-1$ major clouds, with the application provider as the n-th party. Alice can secret-share her sensitive data—for example, her private key for digital asset custody—across these n VMs because no other party can control all of them, not even the application provider. Fig. 2 compares the deployment workflow of traditional and on-demand distributed trust.

The on-demand architecture enables a wide range of distributed-trust applications, but not all of them:

> Flock can support all applications where every distributed-trust computation takes as input only data that is owned by *exactly one user* or is public.

We show that Flock can support 8 types of distributed-trust applications, including secret key recovery for end-to-end encrypted systems (as in Signal [179]), password managers, digital asset custody (as in Coinbase [35], Fireblocks [46] and other MPC wallets [38, 42, 47, 66, 72, 77, 97, 134, 177]), certificate authority signing, code signing, two-server private information retrieval, and data rollback protection. These applications demonstrate different aspects of Flock's expressivity: they enable 5 major cryptographic modules, which showcase protection for data confidentiality, data integrity, data-sharing, or query privacy on a public database. An example of an application that Flock does not support is privately training a machine learning model over *all* users' private data because the input to the training process is the combined data of all users that no *individual* user owns and is not public.

1.2 Summary of Techniques

We now summarize Flock's techniques. A careful reader may be concerned about the cost of the on-demand approach. Continuously running one VM *per* user *per* cloud would be prohibitively expensive and shutting them off intermittently would introduce significant startup time.

We identify *serverless computing* [18, 24, 52, 102] as the most fitting paradigm (§3.1) for Flock. Their "pay-as-you-go" model means that we can invoke a serverless instance exclusively when the user runs an operation, and incur no cost when the user is idle. For example, in the digital asset custody application, Alice's serverless instances only run when Alice wants to perform a transaction. Further, executing a serverless instance on-demand is fast, unlike typical VM boot times. Thus, Flock offers a **cross-cloud serverless system for secure computation (§3.1)**, which runs sophisticated cryptographic libraries in serverless across clouds. A challenge is that serverless offerings cannot innately form peer-to-peer connections [150, 214] because they are publicly inaccessible and ephemeral. To enable end-to-end secure cross-cloud serverless networking, we design a relay that leverages the application provider to connect the serverless instances without trusting the provider with the contents of the communication. To achieve this, we introduce a two-phase TLS establishment protocol that only requires a *single* TCP connection per serverless instance, which is used for both authentication and end-to-end TLS. By reducing secure message-forwarding to copying bytes across sockets, the relay achieves 27x higher throughput (Table 4) than the current best potential approach. We expect the relay to have independent utility in any application that requires end-to-end secure cross-domain (e.g. cross-cloud, cross-region) serverless communication.

Flock's **automatic deployment mechanism (§3.4)** empowers *regular* users to *automatically* setup cross-cloud accounts and serverless deployments without being exposed to underlying cloud-level intricacies. First, each user's Flock client automates multi-cloud account creation by filling in the corresponding forms for the user through the webpage automation framework Playwright [65]. Second, the Flock client conducts programmatic deployment through cloud-provided APIs. The user experience of a Flock application is comparable to that of a regular application. Users will not have to conduct manual cloud registration or serverless deployment per cloud, and the only difference is that users may need to complete $n-1$ authentication steps for cloud registration (e.g. SMS, email).

To secure this deployment, Flock contributes a **three-tier authentication protocol (§3.2)**, which safeguards against the impersonation of a user Alice, her deployments, or the application provider. Our new setting of user-driven distributed-trust deployment introduced new attack vectors, requiring a novel design for authentication: *How can an application provider secure a deployment they do not control?* We first identified the required security "checkpoints" across three tiers—cloud, network, and application—leading us to design a unified protocol spanning these layers. At the cloud level, fine-grained access keys prevent unauthorized users from invoking Alice's serverless instances. At the network level, a secure deployment protocol guards the communication amongst Alice and her "flock." At the application level, Alice's "flock" must authenticate her before conducting operations on her sensitive data. However, it is onerous for each user to authenticate n times, once *per* party. To avoid this, we identify MPCAuth [206] as particularly well-suited in this scenario. MPCAuth enables a user to perform the usual work of authenticating to a *single* "logical" server—which is an MPC of the n servers—with the same security as authenticating to n servers independently.

User-centered deployment introduces a new axis of challenges: Flock should allow the provider to manage **billing without controlling users' cloud instances (§3.3)** or exposing the provider to resource abuse. We use cloud billing infrastructure to prevent malicious users from draining application provider funds and cloud access keys to prevent an attacker from wasting serverless compute resources before they are detected by application-level authentication.

1.3 Evaluation Summary

We implement and evaluate Flock (§5) across three major cloud providers: Amazon Web Services, Azure, and Google Cloud Platform. We have also successfully deployed Flock to IBM Code Engine. When compared to the traditional distributed-trust setup (Fig. 2a), Flock has 1.05x latency and 0.68-2.27x the cloud cost, averaged over all 5 modules. This value does not account for the traditional method's additional cost of business relationships with the $n-1$ third-party organizations (e.g. employee salaries, operational costs)—expenses that do not exist in Flock. Moreover, Flock achieves this without the manual and time-consuming process of identifying and setting up other organizations as trust domains. By removing this deployment barrier, we believe that Flock can foster a new wave of adoption for distributed trust.

2 Threat Model & Security Guarantees

System model. An application provider seeking to offer distributed-trust security to its users invokes the Flock API in its client and server code. Users install an *application client* on their device. The application provider runs the *application server*, which we consider to be a logical server (even if it comprises of multiple physical servers). We refer to the n distinct trust domains that execute the distributed-trust modules as *parties*. The application provider constitutes one party, and the $n-1$ clouds constitute the other $n-1$ parties.

Security guarantees. Flock is not a specific cryptographic scheme or application, but a *system for deploying* distributed trust across the clouds for a variety of applications with different threat models. The guarantee of the on-demand distributed-trust architecture is that each of the n parties are deployed independently of each other, without any one party being able to control the others. Hence, none of these parties are a central point of attack, and crucially, the application provider cannot control the deployments in the $n-1$ clouds. To provide this guarantee, Flock relies on the security mechanisms of each cloud in a black-box manner. As long as cloud i upholds its guarantees, party i stands as an uncompromised trust domain in the Flock deployment.

When running a distributed-trust application App using Flock, the resulting security guarantees are a *combination* of the guarantees of App and Flock, and often, the weaker of the two. The Flock system and modules provide the strong guarantee of malicious security against $n-1$ out of n compromised parties. In particular, an attacker cannot see any secret data distributed across the parties or tamper with the integrity of sensitive operations. Hence, if App *also* provides malicious security, so does the overall App-Flock deployment. If, on the other hand, App provides the weaker semi-honest security, so does the overall App-Flock deployment.[2]

Availability. While the application provider is not trusted with confidentiality and integrity, it is trusted for availability because it is the entity that wishes to provide this service. We also assume that the clouds are available given their service-level agreements [20, 51, 78]. In each cloud, Flock uses cloud services that are fault-tolerant. If, despite this, the provider or a cloud are not available, Flock does not offer availability.

Application code. Like prominent end-to-end encrypted applications [69, 79, 86, 95, 98] and blockchains [111, 119, 189], Flock assumes that the application client code is not compromised and that it is open source and community-scrutinized. Likewise, Flock is open source (§5.1). Flock's focus is to protect against attacks to the application servers. Application servers are a prolific target of attack because they aggregate data across all users. They can also read user data and alter server execution unchecked, which is more difficult to perform with openly scrutinized client code.

[2]The on-demand distributed-trust architecture also supports applications with t out of n security for a threshold $t < n$, but our current implementation only supports $t = n$.

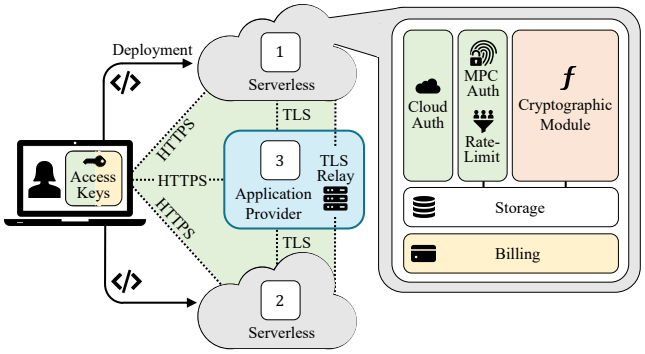


Figure 3: The Flock system architecture. *Client:* Programmatic deployment. *Authentication (green):* Cloud-level via access keys, network-level via TLS, and application-level via MPCAuth. *Resource Protection (yellow):* Unauthorized client abuse is prevented by access keys, while user abuse is prevented through billing. *Cryptographic modules (orange).*

Compromised client devices of a user do not affect the security of *another* user in Flock. For the same user, If Alice's device is compromised *during an active Flock session*, Flock does not provide security guarantees for Alice. This is not specific to Flock, and is the case for many distributed-trust applications. For example, Signal's SVR [79, 179] saves Alice's private keys on her device and distributes them among parties for the purpose of backup. However, if Alice is logged out (and thus not in an active session) during the compromise of her device, Flock's security guarantees remain. Many apps, like password managers and digital wallets, log users out after sessions to bolster security upon a device compromise. Flock implements sessions and removes each user's sensitive key material from the client when the user is logged out.

Authentication (§3.2). Users in Flock authenticate through multiple factors, e.g. with email and SMS on cloud accounts, or PIN and U2F for MPCAuth. Naturally, Flock only protects the data of users with uncompromised authentication factors.

Resource protection (§3.3) in Flock prevents malicious users from draining compute resources and cloud funds from the application provider or denying its service.

3 Flock's System Design

Architecting an on-demand distributed-trust system poses several challenges along the dimensions of cost-efficiency, networking, authentication, resource protection, deployment, and registration. In this section, we describe how we address each challenge in building Flock. Fig. 4 illustrates the Flock system architecture.

3.1 Serverless Architecture

As discussed in §1.1, the straightforward method of implementing on-demand distributed trust is having *each user* deploy one always-on VM on each of $n-1$ clouds. Despite the infrequency of operations like secret recovery due to device loss, the clouds will charge constantly for each VM. A natural strawman is to have the user turn off each VM upon operation completion and reactivate them as needed. Unfortunately, the user would incur *minutes* of additional latency for VMs to boot *per* operation. This delay is prohibitive for applications like password managers, where users frequently invoke Flock.

We observe that the *serverless* computing paradigm [102, 176, 180] alleviates this problem by charging only for active use with a "pay-as-you-go" model [176]. Developers upload code that exclusively consumes resources at execution [102, 180]. Serverless instances are *event-driven*, so they can be triggered with minimal start-up time through a programmable HTTPS interface [176]. The most common serverless compute offerings are serverless functions [18, 24, 50], where a client triggers an API query, it is validated by the cloud provider, and a previously deployed function is invoked in an isolated environment [73–75, 176, 182]. However, basic serverless functions [24, 50] do not naturally support multi-language codebases or runtimes for programming languages that are commonly used to implement cryptographic tools, such as C++. This makes them inconvenient for porting existing cryptographic frameworks and codebases in Flock. Instead, we turn to serverless *containers*, which are light-weight, standalone executable software packages that include the code, runtime, and system libraries. Containers can support multi-language codebases and any runtime, and are offered by AWS Lambda [18] and Google Cloud Run [52].

MPC requires high interactivity, often with one party awaiting another's response. Unfortunately, serverless instances commonly communicate via services like cloud storage, which is prohibitively slower and pricier than direct networking [107, 150]. The challenge is that serverless instances possess private IPs under unique network address translations (NATs), so they cannot accept incoming network connections. Serverless offerings that expose public IP addresses [16, 22] are intended for long-running workloads, and therefore suffer significant coldstart delays [3], and even charge for a minute-long minimum runtime for AWS Fargate [16].

Several works employ NAT traversal and hole-punching to facilitate serverless communication [137, 139, 186, 214], but this method is not robust since it relies on a cloud provider's NAT configurations, which are prone to change. Both Lambda and Google Cloud Run only support cross-cloud hole-punching through NAT gateways and virtual private clouds [29] with impractical per-user cost. Instead, prior systems have facilitated serverless communication through a central relay [140, 141, 210], but do not consider security. A secure relay-based approach for connecting publicly inaccessible endpoints is Wireguard [96] over Tailscale DERP relay servers [37]. As we show in §5.4, this setup incurs significant overhead since Wireguard conducts per-packet encryption and must redundantly layer TLS over Wireguard to supplement it with mutual authentication. Also, Wireguard does not use a federally approved encryption protocol [5], unlike TLS.

3.1.1 Secure Cross-Cloud Serverless Networking

To resolve this challenge, we architect a NAT-independent Flock Relay protocol for secure serverless networking at the transport layer (L4). To deploy the relay, we employ the help of the application provider for availability without trusting it otherwise. The application provider runs a multi-user relay that connects serverless instances by accepting their *incoming* connections, then securely routes messages between them with authentication and end-to-end encryption. The relay observes only message lengths, which do not reveal the private user inputs because of the oblivious nature of MPC. Hence, the provider, though capable of barring the *availability* of the relay, cannot compromise data confidentiality or integrity.

To facilitate secure serverless-to-serverless communication, the Flock Relay must authenticate each serverless instance to ensure that the correct endpoints connect to one another, and facilitate their communication without serving as a central point of attack. While standard TLS connections are established between two endpoints directly, the Flock Relay needs to facilitate end-to-end TLS establishment between two authenticated endpoints.

An insecure strawman for establishing serverless-to-serverless TLS is the following: (1) After a serverless instance initiates a serverless-to-relay connection, the relay verifies the serverless instance and provides it with an authentication token. In future connections, the token will inform the relay that it has previously authorized the serverless endpoint. (2) The serverless instance sends the token to the relay to authorize the end-to-end serverless establishment. Crucially, however, the second step would require the token to be sent in *plaintext*. A passive network eavesdropper could use the token to impersonate valid connections in the future.

Two-phase connection establishment. To ameliorate this issue, we architect our relay to connect serverless instances with the *same* sockets that were already authorized. The Flock Relay executes two phases of TLS establishment to form a *single* end-to-end TLS session, as we show in Fig. 4.
(1) *Serverless-to-relay* (S2R): Every pair of serverless instances s_1 and s_2 each initiate an independent TLS connection with the relay. The relay needs to maintain access control to ensure that only the instances within a single user's "flock" can connect to one other. Thus, the TLS handshake in this phase authenticates the serverless instances. Over the S2R TLS session, each instance notifies the relay of the ID of the serverless instances it wishes to connect to, which hides these IDs from the public Internet. Next, the relay downgrades its

TLS connection with s_1 and s_2 to TCP, and begins forwarding messages between the two TCP sockets.

(2) *Serverless-to-serverless* (S2S): Every pair of serverless instances in a user's "flock" performs a TLS handshake over the TCP connection obtained after the S2R phase, setting up the S2S TLS session between them. To send a message to s_2, s_1 sends the relay a TLS-protected message under the S2S session, which the relay forwards to s_2.

For both S2R and S2S, we use mTLS (Mutual TLS), which enables *mutual* authentication. We also use the relay to connect serverless instances to the application provider (as if it were another serverless instance), for ease of implementation. We assume the user's application client obtains the relay's destination from the application provider.

3.1.2 Flock Relay Certificate Issuance

We now discuss how Flock sets up TLS certificates to ensure all communication occurs with intended parties. The client maintains hardcoded public keys for the application server and relay. For the serverless instances to verify each other in S2S sessions, an observation is that a user Alice is trusted within *her* "flock" and can therefore serve as its certificate authority. Alice's client creates a public-private keypair for each party in her deployment (serverless instances and application provider) and signs a certificate for each of their public keys. Each party stores its certificate and Alice's public key, allowing parties to mutually verify one another.

S2R sessions enlist the application provider as a certificate authority. Indeed, this phase of TLS only exchanges information about message recipients, which must be hidden from the public Internet, but visible to the relay. Overall, the Flock Relay is responsible for managing a public-private keypair for signing, engaging in a setup protocol for certificate issuance for each user, verifying these certificates per-invocation, and facilitating message-forwarding between serverless instances.

For client-to-serverless connections, upon invocation, Alice's client contacts each Flock instance and the application provider via HTTPS. Alice provides parameters including the IP address and port of the relay, as well as query-specific input, so the provider can redeploy or load-balance the relay without requiring Alice to redeploy her instances. Google Cloud Run [52] and AWS Lambda [18] URLs offer HTTPS with trusted CAs, so Alice knows that she is contacting the intended instances.

Reissuance. To prevent an attacker that steals Alice's device from issuing certificates, Alice deletes her certificate issuance secret key post-deployment. Because reissuance is uncommon, she can reauthenticate to each cloud, regenerate keypairs (including her own), and reissue. If the relay updates its public-private keypair, it must reissue a certificate for each user. The application client will be updated with this new relay public key and certificate.

The convenient aspect is that Alice can update her parties' certificates and public keys (both her own and the relay's) without redeploying the entire codebase because they are stored in cloud-provided secret managers [34, 91]. This process is more lightweight than a full-fledged application software update, which necessitates serverless redeployment. For a consistent certificate keypair, one can use Flock's secret recovery (§4.1) or signing (§4.2) to store Alice's secret key.

3.1.3 Flock Relay Protocol

We now detail the Flock Relay protocol, shown in Fig. 4. For simplicity, we only list parameters in certificates or message tuples that are specific to our protocol, but a deployment must contain all the other standard parameters and defenses.

Per-Relay Setup:

1: The application provider generates a keypair $(\mathsf{PK}_r, \mathsf{SK}_r)$, which is used to self-sign a certificate for the relay $\text{RelayCert}_r = \text{GenerateCert}(\mathsf{SK}_r; r, \mathsf{PK}_r)$.
2: The application provider deploys the relay with $(\mathsf{PK}_r, \mathsf{SK}_r, \text{RelayCert}_r)$.
3: Relay listens for new users at RelayUserTarget and for serverless connections at RelayTarget.

Per-User Setup:

1: Alice generates $(\mathsf{PK}_u, \mathsf{SK}_u)$.
2: For each party $i \in \{1, \ldots, n\}$, Alice generates $(\mathsf{PK}_i, \mathsf{SK}_i)$ and certificate $\text{E2ECert}_i = \text{GenerateCert}(\mathsf{SK}_u; i, \mathsf{PK}_i)$.
3: Alice locally deletes SK_u.
4: Alice sends each PK_i to the relay at RelayUserTarget.
5: Relay generates the relay-specific user certificate $\text{RelayCert}_i = \text{GenerateCert}(\mathsf{SK}_r; \text{"Alice"}.i, \mathsf{PK}_i)$.
6: Relay sends $(\mathsf{PK}_r, \text{RelayCert}_i)$ to Alice for each party i.
7: In each serverless deployment for party i, Alice embeds: $(\mathsf{PK}_u, \mathsf{PK}_r, \text{E2ECert}_i, \text{RelayCert}_i, \mathsf{PK}_i, \mathsf{SK}_i, i)$.

Per-Invocation Protocol (Fig. 4)**:**

1: Alice invokes each s_i using HTTPS, with the parameters $\{\text{PartyID} : i, \text{RelayTarget} : (\text{IP}, \text{Port})\}$.
2: Each s_i loads $(\mathsf{PK}_i, \mathsf{SK}_i, \mathsf{PK}_r, \text{RelayCert}_i, \mathsf{PK}_u, \text{E2ECert}_i)$.
3: To establish an S2R session, each s_i performs an mTLS handshake with the relay, in which PK_r is used to verify RelayCert_r is from the intended relay and RelayCert_i is from Alice's i-th party.
4: Over S2R, s_1 sends the relay "2" as its intended destination and s_2 sends the relay "1".
5: Over S2R, the relay arbitrarily assigns s_1 as "TLS Server" and s_2 as "TLS Client," declaring the assignment to both.
6: Over S2R, the relay sends s_1 (TLS Server) an SSL_shutdown OpenSSL message to close the SSL connection. s_1 confirms completion and starts listening on the same socket for a future TLS connection request.

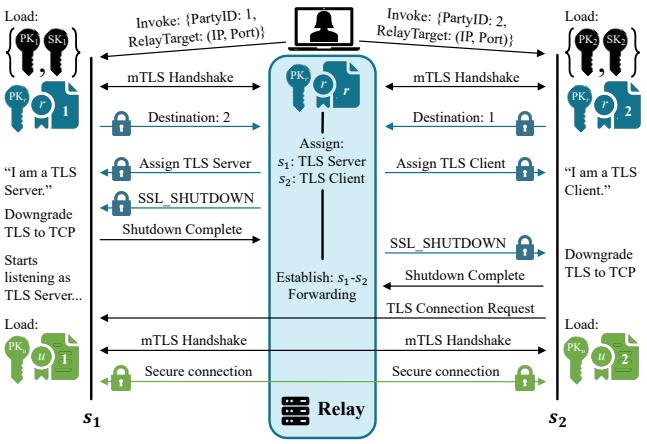


Figure 4: Flock Relay Per-Invocation Protocol. Certificates denote the signing entity in the circle. Teal denotes serverless-to-relay (S2R); green denotes serverless-to-serverless (S2S).

7: Over S2R, the relay sends s_2 (TLS Client) an SSL_shutdown message. When s_2 confirms, the relay establishes s_1-s_2 forwarding over the pair of sockets.

8: To establish S2S, s_2 sends s_1 a TLS connection request through the relay. s_1 and s_2 engage in an mTLS handshake, in which they verify the other's E2ECert$_i$ with PK$_u$. s_1 and s_2 now share an end-to-end TLS connection.

Hence, the Flock Relay only requires a *single* TCP connection per serverless instance, which is used for *both* authentication (S2R) and end-to-end TLS (S2S). By reducing secure message-forwarding to merely copying bytes across sockets, the relay achieves high throughput, as we show in §5.

3.2 Three-Tier Authentication

Authentication in Flock differs from a traditional system because of the three-layered nature of Flock's design: Alice must authenticate to her cloud deployments, their network sessions, and their running application to be able to execute sensitive operations. We have already described Flock's network-level authentication via certificates in §3.1.2. We now present the application-level and cloud-level authentication in Flock.

3.2.1 MPCAuth for Application-level Authentication

To perform a sensitive operation, Alice needs to authenticate to her serverless instances and the application provider. The natural approach is for her to run multi-factor authentication with each one of these parties. In an application with m authentication factors, Alice must authenticate m times for each party, which is burdensome. For example, Alice must input her password $n = 3$ times, perform 3 U2F authentications or lookup 3 emails with security codes, amounting to $n \times m$ total authentications. For applications like cryptocurrency transaction signing (§4.2) or password retrieval during web browsing (§4.1), such repetitive tasks are on the critical path.

Instead, we employ a recent cryptographic protocol, MPCAuth [206], as a black box. MPCAuth enables users to authenticate once, achieving the security of n distinct authentications. At a high level, MPCAuth performs an MPC computation between the n parties to simulate a "trusted server inside MPC" to which the user authenticates. This imaginary server, an amalgamation of the n parties, ensures that as long as one server remains honest, authentication proceeds correctly. A user seeking to trigger an operation will authenticate to the n serverless containers using their m pre-configured authentication factors. If the user successfully authenticates, the sensitive task is executed. The user's experience is unchanged: Alice authenticates to one logical server, when in fact, she is authenticating to all n servers via MPCAuth.

Any factors supported by MPCAuth can be integrated into Flock, including PIN, passcode, U2F, email, SMS, server-side biometrics, and security questions. Distributed-trust applications like Signal's SVR employ PIN because it does not rely on an outside provider like email or SMS. U2F keys are also a common choice because the authentication secret resides on separate hardware. Flock currently integrates U2F, PIN, and passcode. MPCAuth does *not* require cloud support because it is embedded directly in the Flock deployment.

Supplementing MPCAuth. We remark that using MPCAuth alone is insufficient: it does not offer protections for the cloud tier or our more complex network tier because in MPCAuth, the n servers are fixed and known. Supporting n ephemeral, cross-cloud, user-owned instances introduces attack vectors at the cloud and network layers. To secure these layers, Flock ensures serverless instances are invoked by authorized users (§3.2.2) and have authorized network connections (§3.1.2) per invocation.

Rate-limiting is essential for thwarting brute force attacks in distributed-trust applications, particularly those using low-entropy PINs. Flock supplements MPCAuth with a rate-limiting protocol that tracks two parameters at each party: a counter for remaining attempts and a timestamp for the last failed attempt. Upon a failed authentication, the counter decreases. If it hits zero and the time since the last attempt is less than a set lockout period, further attempts are halted. Successful authentication resets this counter. Even if $n-1$ instances are malicious, a single honest instance preserves the rate-limit's integrity by locking out malicious users eventually. While this framework provides a solid rate-limiting foundation, it is also flexible, allowing integration of sophisticated mechanisms tailored to application needs, including serverless product offerings [17, 70, 71].

3.2.2 Access Keys for Cloud-level Authentication.

An attacker might continually invoke other users' serverless instances, depleting application provider funds. Upon invocation, the serverless instances run MPCAuth, preventing

the attacker from authenticating and executing a sensitive operation. However, executing MPCAuth incurred a charge. Further, if rate-limiting is in place, the attacker can even lock the legitimate user out of their account.

To address this, Flock ensures that only pre-approved users can activate their serverless containers. In each cloud, we leverage specialized cloud access keys for local device storage, configured with fine-grained IAM permissions [9, 25, 32]. Without these, an attacker cannot invoke a user's instances. The owner and authorized users of the instances (see data-sharing applications in §4) store access keys locally. Even if an attacker compromises a user's device, the keys do not grant access to the user's secrets, but only to the invocation of the serverless instances. In the unlikely scenario that a user loses their device with the serverless URLs and cloud access keys, the user can manually authenticate with the cloud providers to retrieve them for a new client installation.

3.3 Resource Protection & Billing

Billing is challenging because the provider must pay for user storage and compute *without* being able to access them. At the same time, even though the provider relinquishes its deployment control to the users, malicious users should not be able to deplete the provider's funds. As we outline in §3.2.2, *invocation access keys* prevent an attacker from invoking other users' parties and draining the provider's funds. Now, we must ensure that a user cannot abuse provider resources through its own deployment, especially since attackers can also create user accounts. Hence, in Flock, application providers set a maximum spending limit per user.

To enforce this spending limit, we first discuss what cloud providers offer in this direction, and then describe a solution based on virtual cards. Prominent cloud tools like AWS Organizations [19, 215] and GCP Projects [53] allow a billing account to pay for other accounts' resources without having access to them. AWS Budgets [14] and GCP Budgets & Alerts [48] grant providers policies which trigger alerts and halt spending [30, 45] if a user's spending surpasses a limit. While these services provide what we need, they are not foolproof because cloud providers have not previously operated in the model of strictly preventing a billing account from accessing the accounts it funds. Hence, it is likely that the billing account can gain access in a case-by-case basis to the paid-for account, e.g. by calling a cloud admin for support. However, the fact that these mechanisms already exist suggest that it would be a small change for these clouds to turn this into a strict enforcement, which we advocate for.

Meanwhile, Flock can use virtual cards, which render Flock fully functional today with existing tools. *Virtual card services* such as Karta offer credit cards with set monthly limits for Azure, AWS, and GCP [159], and AWS also accepts pre-paid cards [93]. These providers can issue capped digital cards to users, replenishing funds as needed. Payment platforms like Stripe offer APIs to issue standard cards with spending limits [84], which only incur a few cents per user.

`create_acc(user_info, module, auth_policy)` `(cloud_auth)` Create user account: programmatic registration & deployment.
`deploy(user_info, module, auth_policy)` Called in `create_acc` & for application & Flock updates.
`setup_module(module, auth_policy, new_state)` Authenticates the client & sets sensitive data for the parties.
`execute(module_inputs, auth_inputs)` Authenticates the client & executes a sensitive module.

Table 1: Flock API

3.4 Programmatic Registration & Deployment

Table 1 presents the Flock API used by application developers. A key feature of Flock is that we automate the user experience for clients, so that it is similar to a regular application. The main difference from the standard experience is that the user's application client may surface $n-1$ interactive authentication steps, one for each cloud. However, the Flock API ensures that the user does not bear the burden of manually registering for $n-1$ cloud accounts or interfacing directly with the clouds to deploy the serverless instances.

3.4.1 Registration

When registering for an application, the user's client needs to create $n-1$ cloud accounts. To ensure that the *user* does not manually perform this work, Flock utilizes Playwright [65], a webpage automation framework [65, 76]. Users input their details (name, password, email) in the UI of the application only once at account setup, as they typically do. Flock then automatically populates these details across the $n-1$ cloud registration forms, deriving unique passcodes from the user's provided passcode. Flock inputs application-provided data without user intervention.

Cloud registration requires multi-step user interactions like SMS, email, or CAPTCHAs. To handle these, the `create_acc` function (Table 1) takes the initial `user_info` input that can be automatically populated. Next, the function returns the interactive `cloud_auth` object, which surfaces the steps that necessitate user intervention during registration. For example, AWS registration [109] (1) sends an email with a code to input, (2) asks for contact information, (3) requests billing data, and (4) sends an SMS with a code to input. Thus, Flock anticipates `user_info` to initially collect an email, contact information, billing data, and phone number. This data is held in-memory and fed programmatically into each cloud form as registration proceeds, while `cloud_auth` surfaces mid-registration interactive user input requests (e.g.

SMS codes or CAPTCHAs) to the application UI, transferring the resulting input to the cloud frontends.

3.4.2 Deployment

Once all cloud accounts are instantiated, `create_acc` calls `deploy` (Table 1), which deploys the serverless containers corresponding to the `module`. This deletes any previous deployments, and performs the setup for the cryptographic module and `auth_policy` authentication factors by calling `setup_module` (Table 2). Upon deployment, the client device locally stores cloud access keys (§3.2.2).

4 Applications & Cryptographic Modules

Flock enables a diversity of applications where every distributed-trust computation only takes input data from a single owner or from public sources. Some applications naturally meet this criteria because sensitive data often equates to user-owned secret values. Other applications might initially appear as if they do not fit the on-demand distributed-trust model (e.g. data-sharing, private information retrieval). By *reframing* these applications to the Flock setting, we demonstrate how they, too, can benefit from on-demand deployment. We show how Flock can enable 8 types of distributed-trust applications, based on 5 fundamental cryptographic modules. Across these applications, Flock enables data confidentiality and integrity, private queries on public data, and data-sharing. Table 2 summarizes how each module is encapsulated by the Flock API functions `setup_module` and `execute`.

4.1 Secret Recovery

Secret recovery applications allows a user to back up her secret key k in the form of n secret-shares $\{k_1, \ldots, k_n\}$ [202], each one stored at each party. Even if an attacker compromises $n-1$ parties, it cannot reconstruct the key k without the n-th share. The user retrieves all shares to reconstruct k. For integrity, the client initially stores a salted hash of k at each party and confirms the hash of the reconstructed key matches.

Secure key recovery for end-to-end encryption (E2EE) applications [56, 69, 79, 86, 95, 98] has been a long-standing issue. If a user loses their private key k (e.g. by losing their device), they lose access to their data. However, backing up the key on the application server breaks the guarantees of E2EE by introducing a central point of attack. To avoid this, application providers like Signal secret-share user keys [179]. The user can authenticate to the n servers using a PIN [200] to retrieve the shares of k. One honest party prevents brute force attacks through PIN rate-limiting and allows the client to detect if the recovered key is incorrect.

Password managers [6, 7, 26, 59, 60, 83, 205] store encrypted or hashed versions of user passwords in the cloud. When a single cloud or cloud account is compromised as in the recent high-profile LastPass hack [133], an attacker can brute-force passwords [190]. Flock secret-shares passwords across n parties, and allows the user to reconstruct the passwords locally upon use. If up to $n-1$ parties are compromised, the attacker cannot brute-force the passwords.

4.2 Signing

Signing applications secret-share a signing key k among n parties. Later, the client authenticates to the parties, who run MPC to sign the client's message m with the secret key k. The secret key is never materialized, and the MPC produces the signature. Flock uses a maliciously-secure multi-party signature generation protocol [143].[3]

Digital asset custody is offered by MPC wallets [35, 38, 42, 46, 47, 66, 72, 77, 97, 134, 177], who secure billions [101, 196]. While these wallets typically secret-share between the the client and the application server, Flock enables them to achieve their roadmapped objective of increasing the number of trust domains to $n > 2$ [178]. To send assets to Bob, Alice's client formulates a message tx and invokes the parties, who reconstruct her key within MPC to sign tx.

Certificate authorities [39, 43, 49, 61] routinely sign certificates that bind digital identities to cryptographic keys. Breached signing keys have led to fraudulent certificates that green-light malware and impersonate trusted websites [216]. Flock enables certificate issuance without ever materializing the signing key on one server, providing a more cost-effective alternative to hardware security modules [15, 23].

Code signing services [2, 33, 39, 49, 57, 68, 87, 92] allow organizations who provide critical software to sign code updates. With Flock, the attacker cannot endorse malicious software unless all n parties are breached.

4.3 Decryption

Using Flock, Alice can secret-share an encryption key k_{Alice} among her n parties. An authorized user, say Bob, can provide a ciphertext c to the parties, who will reconstruct k_{Alice} within MPC and decrypt c for Bob. Unlike secret recovery, Bob cannot obtain k_{Alice}, but can use it to perform decryptions that satisfy a certain access policy, as exemplified below. Flock uses a maliciously-secure AES-in-MPC protocol [41, 129], guaranteeing the decryption's integrity.

4.4 Data-sharing in Hierarchical File Systems

End-to-end encryption systems [21, 28, 36, 56, 67, 81, 85, 88] typically operate hierarchically: a user or group key encrypts a directory key, which then encrypts a group of file keys, each of which encrypts a file. If Alice, for instance, is unavailable, and Bob urgently needs a file F encrypted with k_F under

[3] While an attack for the GG18 [143] and GG20 [144] protocols was recently discovered [181], the patch was integrated into the library we use [89].

	`setup_module`	`execute`
Secret Recovery	Shard k as $k_1 \dots k_n$, $\texttt{cloud}_i$ stores k_i.	$\texttt{cloud}_i$ sends k_i to client, who reconstructs k from $k_1 \dots k_n$.
Signing	Clouds gen. $k_1 \dots k_n$ in MPC, each store k_i.	Client sends m. Clouds retrieve k_i, sign m in MPC.
Decryption	Shard AES key as $k_1 \dots k_n$, $\texttt{cloud}_i$ stores k_i.	Clouds retrieve k_i, AES decrypt ciphertext in MPC.
PIR	No client setup.	Send clouds DPF requests, reconstruct $d[i]$ via responses.
Freshness	Store file k in $\texttt{cloud}_1$, $h = H(k)$ in $\texttt{cloud}_2$.	Client retrieves k and h, and checks $h = H(k)$.

Table 2: Setup & execution specification for Flock modules.

Alice's key k_{Alice}, he should be able to access the file based on an access policy (e.g. a period of inactivity, signatures from users with authority), but without learning the key k_{Alice}, which would grant him excessive access. During setup, Alice shares her invocation access keys with Bob and configures her parties with the access policy and the authentication factors to verify from Bob (e.g. Bob's U2F) (§3.2.2). Bob can then authenticate, supply the encrypted k_F to Alice's n parties, which, after policy verification, allow Bob to decrypt k_F.

4.5 Private Information Retrieval

Private information retrieval (PIR) [113, 116, 124, 125, 146, 152, 169] enables users to query a public database at index i, without the servers learning i, and has many use cases [27, 105, 106, 110, 148, 149, 157, 158, 166, 174, 191, 197, 201, 211, 212].

The integration of two-party PIR in Flock showcases a different type of sensitive data access compared to aforementioned modules. Instead of storing a user-specific secret at the parties, we have a public dataset accessed by all the users and the sensitive data of each user is their *query*. In a traditional deployment, each PIR server stores the database. In Flock, we observe that the user's parties can serve as PIR parties since the data is public. However, the cost of storing the entire database in *each* user's cloud would compound. Instead, the database owner can place a public database copy in each of the n cloud providers, accessible by any Flock user deployment. For instance, if Alice queries index i from her Flock deployment in AWS, she would only need to access the AWS database copy, eliminating cross-cloud latency and egress. We port an existing PIR implementation [10] to Flock.

To introduce malicious security, a trusted database owner can store each entry with a signature of the entry, which the client can verify upon reconstruction. If one party tampers with the signature and the other is honest, all queries will fail. In contrast, adding malicious security to single-server PIR exhibits significant costs [125]. Public databases are community-scrutinized to prevent the database owner from tampering with the database. However, a known approach of encoding MAC key into DPF keys can remove this trust assumption from the database owner [125, 132].

4.6 Data Freshness

Data freshness applications often power rollback protection and file integrity, which are long-standing obstacles in systems where the application provider has control over stored user data [104, 155, 160, 167, 183]. Flock utilizes a hash-based freshness module based on Verena [160]. This application demonstrates that Flock's sensitive data does not need to be *secret* data; rather, the sensitive data is the *integrity* of the file system. In Flock, the application provider acts as the file storage server, while a hash server is deployed by the user via Flock. Users safeguard against tampered or outdated file versions by storing a hash of their latest file in the hash server, allowing users to guarantee *their own* file integrity. When a file is stored, its latest hash is saved and signed by the hash server for client verification. During retrieval, the hash server sends the client a signed hash, confirming the file's latest version. Flock's freshness module also incorporates access control. The deploying user retains ownership, granting `read` and `write` permissions so other users can view or update the latest file hash. Signatures from the hash server guarantee the integrity of the file.

5 Evaluation

In this section, we answer: *How do the performance and cost of Flock compare to traditional distributed trust?*

5.1 Implementation

We implemented Flock using ∼2,000 lines of Go (signature protocol, relay), ∼2,000 lines of Python (freshness protocol, client, deployment, storage, server-side "frontend"), and ∼2,000 lines of C++ (passcode, decryption, PIR, relay client). We used cloud SDKs and black-boxed foundational cryptographic libraries: `tss-lib` [89] for the Multi-Party Threshold Signature Scheme [143], `emp-agmpc` [41] for Global-Scale Secure Multi-Party Computation [213], and Google's implementation [10] of incremental distributed point functions [113]. For the relay, we used OpenSSL [64] and its Go bindings[4] for the SSL_shutdown [82] procedure (§3.1).

[4] We fork and adapt github.com/spacemonkeygo/openssl.

Module	System	Breakdown (ms) *Client*	Breakdown (ms) *Server*	End-to-End (ms) Mean (μ)	End-to-End (ms) SD (σ)
Secret Recovery: `setup_module`	*Baseline*	80.50	252.74	356.05	20.03
	Flock	84.51	260.21	409.09	35.73
Secret Recovery: `execute`	*Baseline*	77.45	201.91	302.33	18.13
	Flock	77.52	208.82	340.48	19.75
Signing: `setup_module`	*Baseline*	2.49	4,537.43	4,574.12	19.55
	Flock	2.75	4,718.79	4,776.36	21.84
Signing: `execute`	*Baseline*	2.60	1,031.31	1,053.44	8.55
	Flock	2.81	1,322.75	1,360.35	23.72
Decryption: `setup_module`	*Baseline*	1.33	200.11	213.82	7.74
	Flock	3.05	171.67	231.65	7.76
Decryption: `execute`	*Baseline*	3.36	21,767.18	21,786.63	489.57
	Flock	3.33	21,926.81	21,974.29	626.51
PIR: `execute`	*Baseline*	12.25	202.54	227.20	6.19
	In-memory	12.25	10.62	35.28	3.29
	Flock	12.24	131.02	170.22	8.85
Freshness: `setup_module`	*Baseline*	14.87	218.55	252.47	23.65
	Flock	5.05	209.79	244.63	10.80
Freshness: `execute`	*Baseline*	4.77	189.27	205.90	14.04
	Flock	4.99	188.48	222.76	8.10
Authentication Factor					
U2F	*Baseline*	7.12	527.28	595.88	32.85
	Flock	8.74	515.16	646.85	18.77
PIN	*Baseline*	13.40	909.48	988.60	28.44
	Flock	12.96	1,268.09	1,341.47	35.45

Table 3: Latency of modules & authentication factors. We fix 2^{10} as the input size for brevity.

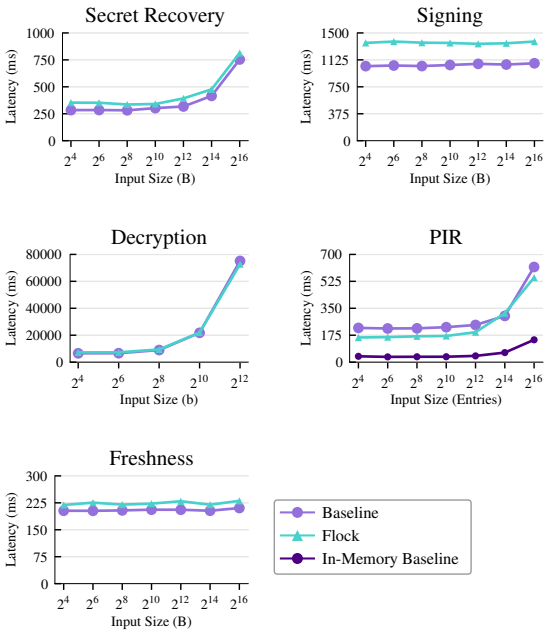


Figure 5: End-to-end latency of cryptographic modules across varying input sizes: number of database entries (Entries) for PIR, bits (b) for decryption, bytes (B) for all other modules.

Our implementation is open-sourced at github.com/flock-org/flock.

5.2 Experiment Setup

The **baseline** is a traditional distributed-trust setup consisting of three VMs in the three major clouds. We use 2 vCPU, 8 GB memory servers in California for AWS (m5.large, $80.64/month), Azure (Standard_D2as_v4, $80.64/month), and GCP (n2-standard-2, $85.16/month). Selecting servers in close proximity minimizes network delays, in-line with typical MPC deployments. The client is an AWS m5.large. This setup is comparable to that of traditional distributed-trust systems in prior literature (§6).

In **Flock**, application providers typically run one party and users deploy $n-1$ parties (§1.1). It is more cost-efficient for an application provider to run their party on a single VM because they can amortize costs among many users without halting the instance [154]. The application server is in Azure and has the same configuration as the baseline's application server in Azure. The client and the regions for the parties are also like in the baseline. We use an Azure Standard_B2s VM (2 vCPU, 4 GB, $36/month) for the Flock Relay. For the serverless containers of signing, decryption, and PIR, we used 2 vCPU AWS Lambda [18] (3,538 MB memory, $0.0000575/s) and Google Cloud Run [52] (512 MB, $0.00006895/s) instances. For secret recovery and freshness which primarily conduct cloud storage operations rather than server-side compute, we use smaller serverless instances: 0.5 vCPU AWS Lambda (895 MB memory, $0.0000144/s) and 0.75 vCPU Google Cloud Run (512 MB, $0.00002695/s). We selected the smallest available memory size for serverless instances.

5.3 Latency

We evaluate the latency of the baseline and Flock for each cryptographic module (described in §4), scaling with their respective input sizes: the secret size for secret recovery, message size for signing, plaintext size for decryption, number of database entries for PIR, and file size for freshness checks. We chose sizes reflective of typical workloads (e.g. size of a cryptocurrency transaction, certificate, or encryption key). For PIR, input size is the number of *database entries* of size 128 B. For decryption, input size is the number of *bits* since our suggested application only requires efficient file key encryption. For all others, input size is the *byte count*. Latency benchmarks were averaged over 10 runs. For the two-party modules (PIR and freshness), we use the application provider server and the Lambda.

Fig. 5 depicts the latency-input size relationship for both

baseline and Flock. Table 3 breaks down latency results for an input size of 2^{10} into client, server, and end-to-end times, with the latter also accounting for client-server network time. We also break down the latency of MPCAuth authentication factors PIN (standard 4-digit format) and U2F. The PIN implementation also supports long passcodes. The function-independent phase in the decryption and PIN circuits can be executed offline to further reduce latency. Utilizing Flock for PIR necessitates streaming and deserializing the database from cloud storage for each request; in a traditional two-server PIR, databases can be held in-memory so we also evaluate a version of our baseline with an in-memory database.

As evidenced in Table 3 and Fig. 5, Flock does not significantly impact the latency of the 5 major cryptographic modules and their MPCAuth factors. More modules exhibit slightly higher standard deviation in Flock, which is expected due to the burstiness of serverless computing. The relay is used in decryption, signing, and PIN: While decryption is 1.01x the latency of the baseline since it is more compute-heavy, signing is 1.28x since it is communication-heavy and the relay slightly impacts performance, as we will concretize in §5.4. As anticipated, Flock exhibits considerably higher latency that the PIR in-memory baseline variant. In exchange, Flock PIR deployments reap the benefits of automatic trust domains and practical malicious security (§4.5). For consistency between Flock and the baseline, as well as between the modules, we use the S3 storage PIR baseline for the remainder of experiments and calculations. Flock-enabled PIR can be enhanced by employing latency-optimized cloud storage services at an extra cost [8] or by parallelizing computation while streaming the remainder of the database.

Averaged over all modules, Flock has a 1.05x latency overhead compared to the S3 baselines. As expected, most cryptographic modules exhibit higher latency with greater input size. Signing remains constant since the protocol [143] signs a *hash* of the message. Freshness is also constant since it is bottlenecked by reads and writes to cloud storage, which are fast at this file size.

Serverless coldstart & deployment latency are factors in Flock, unlike traditional distributed trust. We opted for serverless containers with low coldstart [18, 52, 58] over those with high coldstart [3, 16, 22] and designed lean Docker containers (634 MB pre-compression, 225 MB post-compression). Containers are stored in the application provider's Elastic Container Registry on each cloud, so that users need not build containers. Deployment latency averaged 16.13s for AWS Lambda and 15.88s for Google Cloud Run across 10 tests. For coldstart measurements, we used AWS CloudWatch's X-Ray and invoked Google Cloud Run after idle periods, resulting in 1.02s (AWS Lambda) and 2.10s (Google Cloud Run), averaged over 10 runs. Providers can minimize (or eliminate) coldstart times by keeping containers warm through periodic polling [1]. Applications can also deploy smaller containers with only necessary module dependencies, not all 5.

	Per-Conn. Gb/s	Setup Latency (ms) *S2R*	Setup Latency (ms) *Total*	Concurrent Users *Sign*	Concurrent Users *Decrypt*
Baseline	1.94	–	24.48	–	–
Flock	1.72	20.66	49.72	11,700	1,900
Wireguard	0.063	25.5	78.43	–	–

Table 4: Single connection throughput & establishment latency, and number of concurrent users supported by the relay. S2R includes steps 1-7 of the per-invocation protocol (§3.1.3).

5.4 Relay Evaluation

Per-connection latency & throughput. Table 4 compares the setup latency of the Per-Invocation protocol (§3.1.3) and the throughput[5] of a Flock Relay TLS connection to the baseline's direct TLS connection. We use our Azure and GCP VMs as endpoints, with the relay hosted in the AWS VM. Averaged over 50 runs, a Flock connection's throughput is 0.89x that of a direct TLS connection since all traffic is forwarded through the relay. The setup latency is 2x that of a direct connection due to the additional S2R handshake.

We also benchmark the method used by Tailscale DERP [37], which connects Wireguard [96] endpoints with a relay that re-encrypts Wireguard packets into TLS messages. The Wireguard setup is significantly less performant than the Flock Relay at 0.03x the per-connection throughput of the baseline. As we explain in §3.1, the Wireguard setup incurs significant overhead from encrypting packets at the more granular IP layer, decrypting and re-encrypting all traffic using TLS at the relay, and redundantly TLS-encrypting the Wireguard packets at the endpoints. Setting up Wireguard interfaces and iptable routes introduces 3.2x the setup latency of the baseline. Therefore, Flock Relay connections outperform prior work and nearly match the efficiency of direct TLS.

Cross-user throughput. To evaluate the maximum capacity of the relay, we measure its concurrent user throughput, independent of external factors like application server performance and the compute of cryptographic modules. By emulating traffic patterns for the signing and decryption modules from multiple threads, we saturated the CPU utilization of the relay VM. Results in Table 4 show it can handle 11,700 concurrent signing or 1,900 decryption requests, using 1.9 GB memory. The application provider can further scale the relay deployment based on user demand. We remark that these values represent a worst-case scenario where *all* users invoke Flock, generating traffic patterns in a burst without compute-induced delay. Typically, the Flock Relay can support additional users when they spend intermittent time on compute tasks.

[5] We use `github.com/udhos/goben/` for throughput measurements.

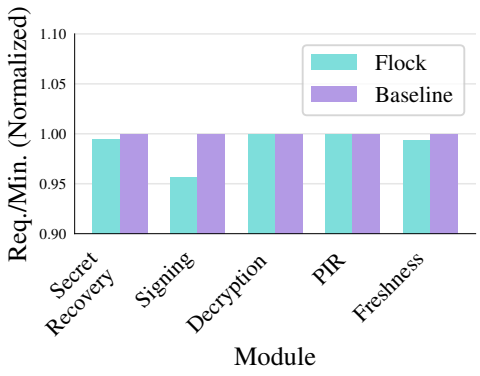


Figure 6: Maximum requests/min. of Flock (F), normalized over the baseline (B). (Secret Recovery: F-1376, B-1384; Signing: F-66, B-69; Decryption: F-5, B-5; PIR: F-1195, B-1196; Freshness: F-1162, B-1169)

5.5 Throughput & Cost

System throughput. Flock achieves throughput comparable to the baseline, as illustrated in Fig. 6. We demonstrate this by fixing an input size of 2^{10} and invoking as many Flock requests as possible.We verify that the CPU utilization of the VM(s) in both Flock and the baseline is 100% at the maximum load, which is the threshold at which additional threads cannot further increase the number of successfully completed requests per minute.

Cost. We use this experiment to calculate the cloud cost estimates for the baseline and Flock in the worst case (Table 5) and across varying server utilization rates (Fig. 7). For each module, we use the baseline's request-per-minute from Fig. 6 to calculate the cost-per-request by dividing the monthly server cost by the number of monthly requests. We then measure Flock's cost-per-request using the per-second vCPU and memory costs, and the same method as the baseline to calculate the application provider's server cost. Finally, we aggregate all cloud expenses to get the total computational cost per operation. We measure the bytes of network traffic transferred by each protocol. AWS, GCP, and Azure charge a network egress fee of \$0.09/GB, \$0.085/GB, and \$0.087/GB, respectively, which we use to calculate the cross-cloud and cloud-to-client data transfer fees. Each module also includes a persistently stored state. For one month, AWS, GCP, and Azure charge \$0.026/GB, \$0.023/GB, and \$0.021/GB, respectively. We include the resulting bandwidth, storage, network, and compute cost per invocation in Table 5.

Averaging across modules, Flock is 2.27x the worst-case cost of the baseline. Table 5 assumes server load is saturated, yielding the lowest possible cost-per-user. However, application providers rarely operate at full server capacity and often provision excessive resources to handle spikes in usage. Fig. 7 shows Flock and the baseline's module average of the per-invocation cost, varying the server utilization from 5-100% of the maximum requests-per-minute. For operating at 50% utilization, the cost of Flock is only 23% more. Flock is actually less expensive on average than the baseline when the monthly requests completed are up to 20% of the baseline's maximum capacity, because of the serverless compute model.

Finally, we emphasize that the baseline has *additional costs* beyond the cloud cost, which do not exist in Flock (§1.1). First, the application provider must compensate its business partners (who have employees or seek profit). Second, the hidden price of the traditional setup is the manual, time-consuming, and difficult challenges of finding suitable business relationships to setup distributed trust.

6 Related Work

Traditional distributed-trust deployments have exhibited a host of obstacles [178] for industry-leading teams, including Signal [79], ISRG [54], and Coinbase [177]. Prio has been employed for private analytics in COVID-19 exposure notifications and Firefox telemetry [100, 108, 126, 136], but ISRG encountered difficulties with cross-organizational inconsistencies in testing and debugging [99, 145, 178]. Signal struggled to deploy traditional distributed trust for its secret recovery application [179], citing reliance on third-parties for security, constant up-time, and user trust [79, 178]. Meta's Private Lift leverages MPC for private advertising, yet advertiser onboarding is time-consuming [187, 198]. Astran [12] has attempted secret-sharing user data across clouds, but their servers see the plaintext data and are therefore a central point of attack [13]. Thus, while the cryptographic guarantees of distributed trust have been instrumental in securing several impactful applications, *deployment* has been a central challenge.

MPC [112, 128, 147, 162–164, 213, 219] **and PIR** [40, 105, 113, 116, 125, 146, 152, 169] **applications** are growing in relevance. MPC applications include private analytics [114, 126] and MPC wallets [35, 38, 42, 46, 47, 66, 72, 77, 97, 101, 134, 177]. PIR applications include private contact discovery [158], credential reporting [174, 191, 207, 212], blocklist lookups [166], and media delivery [149]. Both primitives have been used for private search [131, 132, 151, 194, 197, 211], private advertising [110, 148, 157, 187, 201], and anonymous messaging [105, 106, 123, 127, 138, 170, 171, 217]. **Data freshness** is important for preventing rollback attacks, e.g. in trusted execution environments [104, 167, 183]. Prior work introduces hash servers for file integrity [155, 160]. Flock's contribution is orthogonal and focuses on *deploying* such systems. Many of the systems that Flock supports can be mapped to our baseline setup in §5, and use an underlying cryptographic module that we benchmark. Another line of work [172] aids in the deployment of non-cryptographic distributed trust by offering different privileges to each trust domain; our work instead focuses on offering a deployment mechanism for distributed trust based on strong *cryptographic* guarantees.

Module	*Bandwidth (KB)*	*Storage*	*Network*	*Compute*	***Total***	*Compute*	***Total***
Secret Recovery	25.83	0.0000	0.0002	0.0004	0.0006	0.0016	0.0018
Signing	67.38	0.0000	0.0006	0.0083	0.0089	0.0213	0.0219
Decryption	59,763	0.0000	0.5219	0.1141	0.6360	0.3340	0.8559
PIR	1.38	0.0009	0.0000	0.0005	0.0014	0.0024	0.0033
Freshness	2.89	0.0000	0.0000	0.0005	0.0005	0.0011	0.0011

Table 5: Worst-case cost per one user invocation (USD *cents*). We show the maximum number of requests per minute that the baseline can handle, bandwidth (KB), storage cost, networking cost, and the compute cost for each a single invocation in the baseline and Flock.

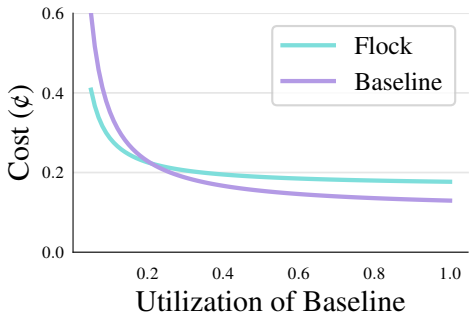


Figure 7: Average per-invocation cost (¢) across all modules for Flock and the baseline, between 5-100% utilization of the maximum baseline capacity.

Serverless networking is a longstanding limitation [107, 150] of serverless computing. While service meshes [31, 55] and proxies [44] can connect services by abstracting network connections, they do not handle *private* endpoints. As we discuss in §3.1, a line of work employs NAT traversal and hole-punching for serverless communication [137, 139, 186, 214], but requires costly per-user services like private clouds or NAT gateways. Recent systems use a relay to enable serverless networking [140, 141, 210], but do not consider security. Some works conduct TLS over multi-step network connections [80, 208], but cannot handle publicly inaccessible endpoints. Wireguard [96] and Tailscale DERP relays [37] securely connect private endpoints, but are unsuitable for serverless as we explained in §3.1. We build upon the the relay-based technique in the literature to architect the first *end-to-end encrypted* relay which has negligible detriment to performance.

Hardware enclaves have been proposed [130, 179] as a replacement to deploying $n-1$ trust domains to safeguard secrets and execution from the application provider. However, enclaves are vulnerable to side-channel attacks that compromise remote attestation, including leaks through SGAxe [209], Plundervolt [188], AEPic Leak [115], and CIPHERLEAKS [175]. With root access to deployment servers, application providers can exploit such side-channels to access secrets. Hence, while enclaves are often utilized as a *supplementary* defense alongside cryptography, applications often opt for cryptography as the primary security measure [178]. In contrast, Flock sets up distributed trust on n major clouds without relying on trusted hardware.

User-centric deployment has been validated in traditional systems work [90, 103, 120–122, 168, 173, 185, 192, 199] in which users deploy components of the applications to retain privacy from a provider. Users sandbox and isolate components of their application to enforce user control. Unlike Flock, these methods do not utilize distributed trust, and thus position a cloud, device, or server as a central point of attack. Flock draws from the underlying principles of user-centric deployment by applying this framework to distributed trust.

7 Conclusion

This work introduces the on-demand distributed-trust architecture, which enables application providers to automatically deploy distributed-trust applications, thus surpassing the cumbersome, manual, and time-consuming process of setting up business relationships. To reverse the deployment from provider to users, our platform Flock consists of a cost-effective cross-cloud serverless framework supporting a variety of distributed-trust applications. We hope that Flock catalyzes an increase in the deployment of distributed trust.

Acknowledgements. We thank the anonymous reviewers, our shepherd Luis Rodrigues, as well as Sam Kumar, Emma Dauterman, Sebastian Angel, Henry Corrian-Gibbs, and the students in the Sky security group for their feedback. This work is supported by NSF CAREER 1943347 and gifts from Accenture, AMD, Anyscale, Google, IBM, Intel, Microsoft, Mohamed Bin Zayed University of Artificial Intelligence, Samsung SDS, SAP, Uber, and VMware.

References

[1] 3 solutions to mitigate the cold-starts on Cloud Run. https://medium.com/google-cloud/3-solutions-to-mitigate-the-cold-starts-on-cloud-run-8c60f0ae7894, Nov 2020.

[2] Authenticode Digital Signatures. https://learn.microsoft.com/en-us/windows-hardware/drivers/install/authenticode, Dec 2021.

[3] AWS Fargate vs. Lambda: Performance. https://www.simform.com/blog/aws-fargate-vs-lambda/, June 2022.

[4] Connect multicloud microservices using Oracle API Gateway. https://docs.oracle.com/en/solutions/oci-multicloud-api-gateway/, Dec 2022.

[5] Should businesses consider WireGuard? https://www.twingate.com/blog/what-is-wireguard-vpn, Aug 2022.

[6] 1Password. https://1password.com/, April 2023.

[7] About the 1Password security model. https://support.1password.com/1password-security, April 2023.

[8] Amazon Elastic Block Store (Amazon EBS). https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS, Oct 2023.

[9] AmazonECSTaskExecutionRolePolicy. https://docs.aws.amazon.com/aws-managed-policy/latest/reference/AmazonECSTaskExecutionRolePolicy.html, October 2023.

[10] An Implementation of Incremental Distributed Point Functions in C++. https://github.com/google/distributed_point_functions, September 2023.

[11] Anthos Multi-Cloud API. https://cloud.google.com/anthos/clusters/docs/multi-cloud/reference/rest, Oct 2023.

[12] Astran. https://astran.io/, Nov 2023.

[13] Astran API Tutorial. https://docs.astran.io/docs/api/getting-started/, Nov 2023.

[14] AWS Budgets. https://aws.amazon.com/aws-cost-management/aws-budgets/, October 2023.

[15] AWS CloudHSM Pricing . https://aws.amazon.com/cloudhsm/pricing/, Oct 2023.

[16] AWS Fargate. https://aws.amazon.com/fargate/, September 2023.

[17] AWS Fargate throttling quotas. https://docs.aws.amazon.com/AmazonECS/latest/userguide/throttling.html, Oct 2023.

[18] AWS Lambda. https://aws.amazon.com/lambda/, April 2023.

[19] AWS Organizations. https://aws.amazon.com/organizations/, October 2023.

[20] AWS Service Level Agreements (SLAs). https://aws.amazon.com/legal/service-level-agreements/, April 2023.

[21] AxCrypt. https://axcrypt.net/, April 2023.

[22] Azure Container Instances. https://azure.microsoft.com/en-us/products/container-instances, September 2023.

[23] Azure Dedicated HSM pricing. https://azure.microsoft.com/en-gb/pricing/details/azure-dedicated-hsm, Oct 2023.

[24] Azure Functions. https://azure.microsoft.com/en-us/products/functions/, April 2023.

[25] Azure resource provider operations. https://learn.microsoft.com/en-us/azure/role-based-access-control/resource-provider-operations#microsoftcontainerinstance, October 2023.

[26] Bitwarden. https://bitwarden.com/, April 2023.

[27] Blyss. https://blyss.dev/, October 2023.

[28] Boxcryptor. https://www.boxcryptor.com/, April 2023.

[29] Building a secure webhook forwarder using an AWS Lambda extension and Tailscale. https://aws.amazon.com/blogs/compute/building-a-secure-webhook-forwarder-using-an-aws-lambda-extension-and-tailscale/, Sep 2023.

[30] Capping API usage. https://cloud.google.com/apis/docs/capping-api-usage, October 2023.

[31] Cilium. https://cilium.io, Nov 2023.

[32] Cloud Run IAM roles. https://cloud.google.com/run/docs/reference/iam/roles, October 2023.

[33] Comodo. https://www.comodo.com/business-security/code-signing-certificates/code-signing.php, April 2023.

[34] Configure secrets. https://cloud.google.com/run/docs/configuring/services/secrets, Nov 2023.

[35] Cryptography and MPC in Coinbase Wallet as a Service (WaaS). https://coinbase.bynder.com/m/687ea39fd77aa80e/original/CB-MPC-Whitepaper.pdf, June 2023.

[36] Cryptomator. https://cryptomator.org/, April 2023.

[37] DERP Servers. https://tailscale.com/kb/1232/derp-servers/, Oct 2023.

[38] Dfns. https://www.dfns.co/, September 2023.

[39] DigiCert. https://www.digicert.com/, April 2023.

[40] DoTS: Berkeley Distributed Trust Stack. https://sky.cs.berkeley.edu/dots/, May 2023.

[41] emp-agmpc. https://github.com/emp-toolkit/emp-agmpc, April 2023.

[42] Entropy. https://entropy.xyz/, April 2023.

[43] Entrust. https://www.entrust.com/, April 2023.

[44] Envoy. https://www.envoyproxy.io, Nov 2023.

[45] Examples of automated cost control responses. https://cloud.google.com/billing/docs/how-to/notify#cap_disable_billing_to_stop_usage, October 2023.

[46] Fireblocks. https://www.fireblocks.com/, April 2023.

[47] Fordefi. https://www.fordefi.com/, September 2023.

[48] GCP Budgets. https://cloud.google.com/billing/docs/how-to/budgets, October 2023.

[49] GlobalSign. https://www.globalsign.com/, April 2023.

[50] Google Cloud Functions. https://cloud.google.com/functions, April 2023.

[51] Google Cloud Platform Service Level Agreements. https://cloud.google.com/terms/sla, April 2023.

[52] Google Cloud Run. https://cloud.google.com/run, September 2023.

[53] IAM roles for billing-related job functions. https://cloud.google.com/iam/docs/job-functions/billing, April 2023.

[54] Internet Security Research Group. https://www.abetterinternet.org/, September 2023.

[55] Istio. https://istio.io, Nov 2023.

[56] Keybase. https://keybase.io/, April 2023.

[57] Keyfactor. https://www.keyfactor.com/platform/enterprise-code-signing/, April 2023.

[58] Lambda execution environments. https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environments, Oct 2023.

[59] LastPass. https://www.lastpass.com/, April 2023.

[60] LastPass Technical Whitepaper. https://support.lastpass.com/help/lastpass-technical-whitepaper, April 2023.

[61] Let's Encrypt. https://letsencrypt.org/, April 2023.

[62] Mixed-Species Flocking. https://web.stanford.edu/group/stanfordbirds/text/essays/Mixed-Species_Flocking, Oct 2023.

[63] Multi-cloud provisioning. https://www.terraform.io/use-cases/multi-cloud-deployment, Oct 2023.

[64] OpenSSL. https://www.openssl.org/, Oct 2023.

[65] Playwright. https://playwright.dev/, September 2023.

[66] Portal. https://www.portalhq.io/, September 2023.

[67] PreVeil. https://www.preveil.com/, April 2023.

[68] PrimeKey. https://www.primekey.com/solutions/code-signing/, April 2023.

[69] ProtonMail. https://proton.me/, April 2023.

[70] Rate limiting overview. https://cloud.google.com/armor/docs/rate-limiting-overview, Oct 2023.

[71] Resource availability & quota limits for ACI. https://learn.microsoft.com/en-us/azure/container-instances/container-instances-resource-and-quota-limits, Oct 2023.

[72] Safeheron. https://www.safeheron.com/, September 2023.

[73] Securing Azure Functions. https://learn.microsoft.com/en-us/azure/azure-functions/security-concepts, April 2023.

[74] Securing Cloud Functions. https://cloud.google.com/functions/docs/securing, April 2023.

[75] Security in AWS Lambda. https://docs.aws.amazon.com/lambda/latest/dg/lambda-security.html, April 2023.

[76] Selenium. https://www.selenium.dev/, April 2023.

[77] Sepior. https://sepior.com/, April 2023.

[78] Service Level Agreements (SLA) for Online Services. https://www.microsoft.com/licensing/docs/view/Service-Level-Agreements-SLA-for-Online-Services, April 2023.

[79] Signal. https://signal.org/, April 2023.

[80] Signal TLS Proxy. https://github.com/signalapp/Signal-TLS-Proxy, Oct 2023.

[81] SpiderOak. https://spideroak.com/, April 2023.

[82] SSL_shutdown. https://www.openssl.org/docs/manmaster/man3/SSL_shutdown.html, Nov 2023.

[83] Storage: Bitwarden Help Center. https://bitwarden.com/help/data-storage/, April 2023.

[84] Stripe Card Issuance. https://stripe.com/issuing, October 2023.

[85] Sync. https://sync.com/, April 2023.

[86] Telegram. https://telegram.org/, April 2023.

[87] Thawte. https://www.thawte.com/code-signing/, April 2023.

[88] Tresorit. https://tresorit.com/, April 2023.

[89] tss-lib. https://github.com/bnb-chain/tss-lib, April 2023.

[90] Use Advanced Data Protection for your iCloud data . https://support.apple.com/en-gb/guide/iphone/iph584ea27f5/ios, Oct 2023.

[91] Use AWS Secrets Manager secrets in AWS Lambda functions. https://docs.aws.amazon.com/secretsmanager/latest/userguide/retrieving-secrets_lambda.html, Nov 2023.

[92] Venafi. https://venafi.com/codesign-protect/, April 2023.

[93] Visa Prepaid Cards. https://usa.visa.com/pay-with-visa/cards/prepaid-cards.html, October 2023.

[94] VMware Cross-Cloud Services. https://www.vmware.com/cross-cloud-services, Oct 2023.

[95] WhatsApp. https://www.whatsapp.com/, April 2023.

[96] Wireguard. https://www.wireguard.com/, Nov 2023.

[97] ZenGo. https://zengo.com/, April 2023.

[98] Zoom. https://zoom.us/, April 2023.

[99] John Aas. Project Update and New Name for ISRG Prio Services: Introducing Divvi Up. https://divviup.org/blog/prio-services-update/, Dec 2021.

[100] Josh Aas and Tim Geoghegan. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/, Nov 2020.

[101] Svetlana Abramova and Rainer Böhme. Anatomy of a High-Profile Data Breach: Dissecting the Aftermath of a Crypto-Wallet Case. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 715–732. USENIX Association, August 2023.

[102] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight Virtualization for Serverless Applications . In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 419–434. USENIX Association, February 2020.

[103] Muneeb Ali, Ryan Shea, Jude Nelson, and Michael J Freedman. Blockstack Technical Whitepaper. *Blockstack PBC, October*, 12, 2017.

[104] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: Rollback Protection for Confidential Cloud Services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 193–208, 2023.

[105] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*, pages 962–979. IEEE, 2018.

[106] Sebastian Angel and Srinath Setty. Unobservable Communication over Fully Untrusted Infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569. USENIX Association, November 2016.

[107] Lixiang Ao, Liz Izhikevich, Geoffrey M Voelker, and George Porter. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 263–274, 2018.

[108] Apple and Google. Exposure Notification Privacy-preserving Analytics (ENPA) White Paper. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf, Apr 2021.

[109] AWS. Creating an AWS account. https://docs.aws.amazon.com/accounts/latest/reference/manage-acct-creating.html, Apr 2023.

[110] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. ObliviAd: Provably Secure and Practical Online Behavioral Advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271. IEEE, 2012.

[111] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State Machine Replication in the Libra Blockchain. 2019.

[112] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, page 1–10. Association for Computing Machinery, 1988.

[113] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776, 2021.

[114] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776, 2021.

[115] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022.

[116] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 1292–1303. Association for Computing Machinery, 2016.

[117] Chuck Brooks. Alarming Cyber Statistics For Mid-Year 2022 That You Need To Know, 2022.

[118] Jon Buck. Coincheck: Stolen $534 Mln NEM Were Stored On Low Security Hot Wallet. https://cointelegraph.com/news/coincheck-stolen-534-mln-nem-were-stored-on-low-security-hot-wallet, Jan 2018.

[119] Vitalik Buterin. Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform. 2013.

[120] Tej Chajed, Jon Gjengset, M Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. Oort: User-Centric Cloud Storage with Global Queries. 2016.

[121] Tej Chajed, Jon Gjengset, Jelle Van Den Hooff, M Frans Kaashoek, James Mickens, Robert Morris, and Nickolai Zeldovich. Amber: Decoupling User Data from Web Applications. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, 2015.

[122] Ramesh Chandra, Priya Gupta, and Nickolai Zeldovich. Separating Web Applications from User Data Storage with BSTORE. In *USENIX Conference on Web Application Development (WebApps 10)*, 2010.

[123] David Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *J. Cryptology*, 1:65–75, 1988.

[124] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50, 1995.

[125] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J Wu, and Bryan Ford. Authenticated private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3835–3851, 2023.

[126] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282. USENIX Association, March 2017.

[127] Henry Corrigan-Gibbs, Dan Boneh, and David Mazieres. Riposte: An Anonymous Messaging System Handling Millions of Users. *2015 IEEE Symposium on Security and Privacy*, page 321–338, 2015.

[128] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPDZ2k: Efficient MPC mod 2^k for Dishonest Majority. Cryptology ePrint Archive, Paper 2018/482, 2018. https://eprint.iacr.org/2018/482.

[129] Ivan Damgård and Marcel Keller. Secure Multiparty AES. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, page 367–374. Springer-Verlag, 2010.

[130] Emma Dauterman, Vivian Fang, Natacha Crooks, and Raluca Ada Popa. Reflections on Trusting Distributed Trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, page 38–45. Association for Computing Machinery, 2022.

[131] Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An Encrypted Search System with Distributed Trust. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1101–1119. USENIX Association, November 2020.

[132] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. Waldo: A Private Time-Series Database from Function Secret Sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2450–2468, 2022.

[133] Ben Demers. Struggling LastPass Suffers New Data Breach. Is Your Account at Risk? https://www.kiplinger.com/personal-finance/lastpass-hack, Jan 2023.

[134] Juergen Eckel. Hardware Security Modules vs. Secure Multi-Party Computation in Digital Asset Custody: The Drawback of Choosing Just One and What Happens When You Combine Them. https://blog.riddleandcode.com/hardware-security-modules-vs-107729d6d3ea, Oct 2022.

[135] Yehia Elkhatib. Mapping Cross-Cloud Systems: Challenges and Opportunities. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[136] Steven Englehardt. Next steps in privacy-preserving Telemetry with Prio. https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/, Jun 2019.

[137] Jeffrey Eppinger. TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem. 01 2005.

[138] Saba Eskandarian, Henry Corrigan-Gibbs, Matei Zaharia, and Dan Boneh. Express: Lowering the Cost of Metadata-hiding Communication with Cryptographic Privacy. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1775–1792. USENIX Association, August 2021.

[139] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-Peer Communication Across Network Address Translators. In *2005 USENIX Annual Technical Conference (USENIX ATC 05)*. USENIX Association, April 2005.

[140] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 475–488. USENIX Association, July 2019.

[141] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 363–376. USENIX Association, March 2017.

[142] Jake Frankenfield. Private key: What it is, how it works, best ways to store. https://www.investopedia.com/terms/p/private-key.asp, Feb 2023.

[143] Rosario Gennaro and Steven Goldfeder. Fast Multiparty Threshold ECDSA with Fast Trustless Setup. Cryptology ePrint Archive, Paper 2019/114, 2019. https://eprint.iacr.org/2019/114.

[144] Rosario Gennaro and Steven Goldfeder. One Round Threshold ECDSA with Identifiable Abort. Cryptology ePrint Archive, Paper 2020/540, 2020. https://eprint.iacr.org/2020/540.

[145] Tim Geoghegan. Exposure Notifications Private Analytics: Lessons Learned From Running Secure MPC at Scale. https://divviup.org/blog/lessons-from-running-mpc-at-scale/, May 2022.

[146] Niv Gilboa and Yuval Ishai. Distributed Point Functions and Their Applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 640–658. Springer Berlin Heidelberg, 2014.

[147] O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, page 218–229. Association for Computing Machinery, 1987.

[148] Matthew Green, Watson Ladd, and Ian Miers. A Protocol for Privately Reporting Ad Impressions at Scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1591–1601, 2016.

[149] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and Private Media Consumption with Popcorn. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 91–107. USENIX Association, March 2016.

[150] Joseph M. Hellerstein, Jose M. Faleiro, Joseph E. Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. Serverless Computing: One Step Forward, Two Steps Back. *CoRR*, abs/1812.03651, 2018.

[151] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private Web Search with Tiptoe. *Cryptology ePrint Archive*, 2023.

[152] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905. USENIX Association, August 2023.

[153] Jiangshui Hong, Thomas Dreibholz, Joseph Adam Schenkel, and Jiaxi Alessia Hu. An Overview of Multi-cloud Computing. In *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33*, pages 1055–1068. Springer, 2019.

[154] Shay Horovitz, Roei Amos, Ohad Baruch, Tomer Cohen, Tal Oyar, and Afik Deri. FaaStest - Machine Learning based Cost and Performance FaaS Optimization. In *Economics of Grids, Clouds, Systems, and Services: 15th International Conference, GECON 2018, Pisa, Italy, September 18–20, 2018, Proceedings 15*, pages 171–186. Springer, 2019.

[155] Yuncong Hu, Sam Kumar, and Raluca Ada Popa. Ghostor: Toward a Secure Data-Sharing System from Decentralized Trust. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 851–877. USENIX Association, February 2020.

[156] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G Patil, Joseph E Gonzalez, and Ion Stoica. Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1375–1389, 2023.

[157] Ari Juels. Targeted Advertising... And Privacy Too. In *Cryptographers' Track at the RSA Conference*, pages 408–424. Springer, 2001.

[158] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile Private Contact Discovery at Scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, 2019.

[159] Stacy Kanevskaia. 4 ways Azure, AWS, and Google Cloud virtual cards simplify subscription management. https://karta.io/blog/13-4-ways-azure-aws-and-google-cloud-virtual-cards-simplify-subscription-management, Dec 2022.

[160] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. Verena: End-to-End Integrity Protection for Web Applications. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 895–913, 2016.

[161] Darya Kaviani and Raluca Ada Popa. https://mpc.cs.berkeley.edu.

[162] Marcel Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, page 1575–1590. Association for Computing Machinery, 2020.

[163] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct 2016.

[164] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ Great Again. *Advances in Cryptology – EUROCRYPT 2018*, page 158–189, Mar 2018.

[165] Dmitry Kogan and Henry Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, August 2021.

[166] Dmitry Kogan and Henry Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892, 2021.

[167] Kari Kostiainen, N Asokan, and Jan-Erik Ekberg. Credential Disabling from Trusted Execution Environments. In *Information Security Technology for Applications: 15th Nordic Conference on Secure IT Systems, NordSec 2010, Espoo, Finland, October 27-29, 2010, Revised Selected Papers 15*, pages 171–186. Springer, 2012.

[168] Maxwell Krohn, Alex Yip, Micah Brodsky, Robert Morris, Michael Walfish, et al. World Wide Web Without Walls. In *6th ACM Workshop on Hot Topics in Networking (Hotnets)*, 2007.

[169] E. Kushilevitz and R. Ostrovsky. Replication Is Not Needed: Single Database, Computationally-Private Information Retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[170] Albert Kwon, Henry Corrigan-Gibbs, Srinivas Devadas, and Bryan Ford. Atom: Horizontally Scaling Strong Anonymity. In *Proceedings of the 26th Symposium on Operating Systems Principles*, page 406–422. Association for Computing Machinery, 2017.

[171] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An Efficient Communication System With Strong Anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016, 08 2015.

[172] Brent Lagesse, Mohan Kumar, Justin Mazzola Paluska, and Matthew Wright. DTT: A Distributed Trust Toolkit for pervasive systems. In *2009 IEEE International Conference on Pervasive Computing and Communications*, pages 1–8. IEEE, 2009.

[173] Sangmin Lee, Edmund L. Wong, Deepak Goel, Mike Dahlin, and Vitaly Shmatikov. πBox: A Platform for Privacy-Preserving Apps. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 501–514. USENIX Association, April 2013.

[174] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for Checking Compromised Credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1387–1403, 2019.

[175] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 717–732. USENIX Association, August 2021.

[176] Zijun Li, Linsong Guo, Jiagan Cheng, Quan Chen, Bingsheng He, and Minyi Guo. The Serverless Computing Survey: A Technical Primer for Design Architecture. *ACM Comput. Surv.*, 54(10s), Sep 2022.

[177] Yehuda Lindell. How Smart Cryptography Makes Coinbase More Secure. https://www.coinbase.com/blog/how-smart-cryptography-makes-coinbase-more-secure, Oct 2022.

[178] Yehuda Lindell, David Cook, Tim Geoghegan, Sarah Gran, Rolfe Schmidt, Ehren Kret, Darya Kaviani, and Raluca Ada Popa. The Deployment Dilemma: Merits and Challenges of Deploying MPC. https://mpc.cs.berkeley.edu/blog/deployment-dilemma, Sep 2023.

[179] Joshua Lund. Technology Preview for Secure Value Recovery. https://signal.org/blog/secure-value-recovery/, 2019.

[180] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 162–169, 2017.

[181] Nikolaos Makriyannis and Oren Yomtov. Practical Key-Extraction Attacks in Leading MPC Wallets. *Cryptology ePrint Archive*, 2023.

[182] Eduard Marin, Diego Perino, and Roberto Di Pietro. Serverless Computing: A Security Perspective - Journal of Cloud Computing, Oct 2022.

[183] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback Protection for Trusted Execution. In *Proceedings of the 26th USENIX Conference on Security Symposium*, page 1289–1306. USENIX Association, 2017.

[184] David Z. Morris. 4 Unanswered Questions About the Bitfinex Hack. https://www.coindesk.com/layer2/2022/02/09/4-unanswered-questions-about-the-bitfinex-hack/, Feb 2022.

[185] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, Andy Crabtree, James Colley, Tom Lodge, et al. Personal Data Management with the Databox: What's Inside the Box? In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, pages 49–54, 2016.

[186] Daniel William Moyer. Punching Holes in the Cloud: Direct Communication between Serverless Functions Using NAT Traversal. Master's thesis, Virginia Tech, Jun 2021.

[187] Graham Mudd. Privacy-Enhancing Technologies and Building for the Future. https://www.facebook.com/business/news/building-for-the-future, Aug 2021.

[188] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against

Intel SGX. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[189] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, Dec 2008.

[190] Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[191] Bijeeta Pal, Mazharul Islam, Thomas Ristenpart, and Rahul Chatterjee. Might I Get Pwned: A Second Generation Compromised Credential Checking Service. In *USENIX Security*, 2022.

[192] Shoumik Palkar and Matei Zaharia. DIY Hosting for Online Privacy. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 1–7, 2017.

[193] Dana Petcu. Multi-Cloud: Expectations and Current Approaches. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 1–6, 2013.

[194] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph 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.

[195] Antigoni Polychroniadou, Gilad Asharov, Benjamin Diamond, Tucker Balch, Hans Buehler, Richard Hua, Suwen Gu, Greg Gimler, and Manuela Veloso. Prime Match: A Privacy-Preserving Inventory Matching System. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6417–6434. USENIX Association, August 2023.

[196] PRNewsfoto/Fireblocks. Fireblocks Surpasses $600 Billion in Digital Assets Transferred. https://www.prnewswire.com/news-releases/fireblocks-surpasses-600-billion-in-digital-assets-transferred-301293822.html, 2021.

[197] Joel Reardon, Jeffrey Pound, and Ian Goldberg. Relational-Complete Private Information Retrieval. *University of Waterloo, Tech. Rep. CACR*, 34(2007), 2007.

[198] James Reyes. Building the next generation of digital advertising with MPC. *Real World Crypto*, 2022.

[199] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. Solid: A Platform for Decentralized Social Applications Based on Linked Data. *MIT CSAIL & Qatar Computing Research Institute, Tech. Rep.*, 2016.

[200] Randall Sarafa. Introducing Signal PINs. https://signal.org/blog/signal-pins/, May 2020.

[201] Sacha Servan-Schreiber, Kyle Hogan, and Srinivas Devadas. AdVeil: A Private Targeted Advertising Ecosystem. *Cryptology ePrint Archive*, 2021.

[202] Adi Shamir. How to Share a Secret. In *Programming Techniques R. Rivest Editor*, 1979.

[203] Jimmy Song. Mt. Gox hack technical explanation. https://jimmysong.medium.com/mt-gox-hack-technical-explanation-37ea5549f715, Aug 2017.

[204] Ion Stoica and Scott Shenker. From Cloud Computing to Sky Computing. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, page 26–32. Association for Computing Machinery, 2021.

[205] Nick Summers. Why you can trust 1Password's cloud-based storage and syncing. https://www.kiplinger.com/personal-finance/lastpass-hack, February 2023.

[206] Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. MPCAuth: Multi-Factor Authentication for Distributed-Trust Systems. In *IEEE Symposium on Security and Privacy*, 2023.

[207] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, 2019.

[208] Kfir Toledo, Pravein Govindan Kannan, Michal Malka, Etai Lev-Ran, Katherine Barabash, and Vita Bortnikov. ClusterLink: A Multi-Cluster Application Interconnect. In *Proceedings of the 16th ACM International Conference on Systems and Storage*, page 138. Association for Computing Machinery, 2023.

[209] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX Fails in Practice. https://sgaxeattack.com/, 2020.

[210] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov,

Feng Yan, and Yue Cheng. InfiniCache: Exploiting Ephemeral Serverless Functions to Build a Cost-Effective Memory Cache. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 267–281. USENIX Association, February 2020.

[211] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical Private Queries on Public Data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313. USENIX Association, March 2017.

[212] Ke Coby Wang and Michael K Reiter. Detecting Stuffing of a User's Credentials at Her Own Accounts. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2201–2218, 2020.

[213] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-Scale Secure Multiparty Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, page 39–56. Association for Computing Machinery, 2017.

[214] Michal Wawrzoniak, Ingo Muller, Rodrigo Bruno, and Gustavo Alonso. Boxer: Data Analytics on Network-enabled Serverless Platforms. In *11th Annual Conference on Innovative Data Systems Research (CIDR'21)*, January 2021.

[215] Iam Hazel Virginia Whitehouse-Grant-Christ. IAM tutorial: Delegate access to the billing console. https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_billing.html, 2011.

[216] Josephine Wolff. How a 2011 Hack You've Never Heard of Changed the Internet's Infrastructure. https://slate.com/technology/2016/12/how-the-2011-hack-of-diginotar-changed-the-internets-infrastructure.html, Dec 2016.

[217] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in Numbers: Making Strong Anonymity Scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182. USENIX Association, October 2012.

[218] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, et al. SkyPilot: An Intercloud Broker for Sky Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 437–455, 2023.

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