Secure Account Recovery for a Privacy-Preserving Web Service

Ryan Little, *Boston University;* Lucy Qin, *Georgetown University;* Mayank Varia, *Boston University*

https://www.usenix.org/conference/usenixsecurity24/presentation/little

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Secure Account Recovery for a Privacy-Preserving Web Service

Ryan Little
Boston University

Lucy Qin
Georgetown University

Mayank Varia
Boston University

Abstract

If a web service is so secure that it does not even know—and does not want to know—the identity and contact info of its users, can it still offer account recovery if a user forgets their password? This paper is the culmination of the authors' work to design a cryptographic protocol for account recovery for use by a prominent secure matching system: a web-based service that allows survivors of sexual misconduct to become aware of other survivors harmed by the same perpetrator. In such a system, the list of account-holders must be safeguarded, even against the service provider itself.

In this work, we design an account recovery system that, on the surface, appears to follow the typical workflow: the user types in their email address, receives an email containing a one-time link, and answers some security questions. Behind the scenes, the defining feature of our recovery system is that the service provider can perform email-based account validation without knowing, or being able to learn, a list of users' email addresses. Our construction uses standardized cryptography for most components, and it has been deployed in production at the secure matching system.

As a building block toward our main construction, we design a new cryptographic primitive that may be of independent interest: an oblivious pseudorandom function that can either have a fully-private input or a partially-public input, and that reaches the same output either way. This primitive allows us to perform online rate limiting for account recovery attempts, without imposing a bound on the creation of new accounts. We provide an open-source implementation of this primitive and provide evaluation results showing that the end-to-end interaction time takes 8.4-60.4 ms in fully-private input mode and 3.1-41.2 ms in partially-public input mode.

Content Warning: *This paper discusses, at a high level, the issue of sexual assault on college campuses, particularly in Section 1. From Section 3 onward, the paper is more focused on the technical design of the account recovery protocol.*

1 Introduction

Account recovery—the ability for users to regain access to their accounts after losing their password—is a near-ubiquitous feature of account-based web services. However, the typical account recovery system is not private, and relies on the web service to retain some information about the existence of an account under some identity (e.g., a username or email address). In this work, we designed an account recovery process to be compatible with a secure matching system operated by a nonprofit organization, Callisto, that uses conventional account recovery workflows but does not retain any plaintext information about a user's email address, username, security questions, or security answers. As of October 2023, this protocol has been deployed by Callisto for their matching system, which is currently available on college campuses across the United States.

We discuss the application space of secure matching systems to provide context for this work, and then elaborate on the challenge of account recovery in this setting and provide details about our protocol. While this protocol was designed to support Callisto's secure matching system, our cryptographic techniques are generic and may also be used independently by other web services that want to enable account recovery without collecting personal information about users.

Overview of secure matching systems. Secure matching systems are cryptographic tools that intend to support survivors of sexual assault by connecting them with one another. Although statistics may be difficult to capture, there is evidence that sexual assault is highly prevalent on college campuses, in particular. A 2019 study across 33 universities found that 13% of students experienced sexual assault while enrolled [21]. Due to mandatory reporting policies that limit survivors' control over their own narratives, victim-blaming, fear of retaliation, and other barriers, survivors rarely formally report incidents of sexual assault [32, 37, 55]. For those who do, survivors may experience retraumatization through interactions with the police and/or the legal system and may ultimately feel failed by formal avenues of pursuing justice [54]. Inspired by the #MeToo movement, secure matching systems [6, 39, 48, 49, 61] allow survivors of sexual assault to document their experiences and securely connect with other survivors who have been harmed by the same perpetrator to seek mutual support and explore different pathways either collectively or individually, since healing and seeking justice may look different for each survivor.

Callisto is a nonprofit organization that currently deploys a free secure matching system across college campuses in the United States [14]. Callisto allows survivors of sexual assault to document details of their assault in an encrypted record (for potential future use) and participate in matching with other survivors. If two or more survivors match by providing identifying information about the same perpetrator, contact information for each survivor is revealed to a legal options counselor (a lawyer) so that rather than being cryptographically protected, the same information is then protected by attorney-client privilege [15]. Prior to a match, information about both a survivor and perpetrator is not accessible to Callisto. Once connected to a legal options counselor, survivors are then offered the opportunity to connect with one another. Upon connecting, they may choose to collectively pursue legal justice, restorative justice, or nothing at all. Matching enables survivors to seek support from a legal options counselor and one another to explore options (individually or collectively) for action, ideally providing each survivor more support when decision-making about their options for pursuing further action, should they want to.

The challenge of account recovery. In standard account recovery processes, the service provider typically stores an email address that it uses for validation and to share a link that allows a user to regain access to their account. Since contact information about a survivor may reveal sensitive information (in our setting, whether an individual is a survivor of sexual assault), privacy-respecting service providers like Callisto cannot store any contact information about its users. Maintaining a list of account-holders creates a sensitive and valuable asset that may be targeted by bad actors. It may also create vulnerability toward legal threats, through subpoenas for information about account holders. Hence, a secure matching system must protect both *data* and *metadata*—even against the service provider itself, or anyone who might attempt to compromise it. In more detail:

1. All data submitted by survivors must be encrypted with a key derived from the user's password, so that the service cannot access user-submitted information (e.g. identifying information about a user, details of their assault) until they have reached the matching threshold. This cryptographic problem has been addressed in prior works [6, 39, 48, 49, 61] with general-purpose secure multi-party computation (MPC) and specific primitives like oblivious pseudorandom functions, group signatures, and verifiable secret sharing.
2. Additionally, the service provider must not have (or even have a simple way to recover) a list of all account-holders, since even the fact of a survivor using the system is extremely sensitive. On its own, this problem can be addressed using cryptographically oblivious login mechanisms (e.g., [45]), along with network and systems security precautions supporting anonymous communications and not keeping long-term logs that could later be exfiltrated by a hacker or compelled by an authority.

When addressing these two requirements together, one runs into a practical challenge: *what happens if an account-holder forgets their password*? This issue is inevitable in a system where strong, unique passwords are encouraged but logins are infrequent, and by default it would make all of the user's encrypted data irrecoverable.

In a typical website, account recovery involves the service provider sending an email to the email address on file; however, this is incompatible with our no-metadata requirement. Even federating the email address list and the delivery of emails among multiple servers using secure multi-party computation [1, 38, 64] is insufficient for our needs because it is slow, difficult to implement, and (most importantly) is still vulnerable to the risk of all servers being compelled to disclose their secret shares of the email list.

1.1 This work

In this work, we describe the design of an account recovery system such that a web service provider (like Callisto) *does not store email addresses, and cannot easily provide a list of all email addresses even if compelled to do so (e.g., through a subpoena).*

Protocol design. At a high level, our cryptographic protocol involves a client who (if honest) wishes to regain access to their account, along with a collection of recovery servers. The protocol uses email validation and a set of security questions/answers to determine whether the client should be given access to the account. Only the client recovers the cryptographic key that protects their account data; the recovery servers never see the security answers or key material.

In more detail, our protocol has two phases: some work is performed during *account creation* when the client knows their password and cryptographic key material, and subsequently the client can run an *account recovery* procedure if they forget the password and associated key. The recovery procedure itself contains three parts: request, verification, and restoration. First, the client requests recovery by (obliviously) providing their email address and responses to any generic questions (e.g., a phone number). Second, the servers verify that the client has control of this email address. Third, the servers fetch the client-specific security questions; the client's answers to these questions can be used to restore access to their account.

Design requirements. Our protocol is grounded in the framework of trauma-informed design that acknowledges the impact trauma may have on the user experience and seeks to avoid retraumatization [15, 25, 63]. When designing an account recovery process that is trauma-informed in this setting, the user's interactions with the process must be familiar and easy to use: in particular, it must adhere to an expected account recovery workflow (e.g., "click on a link"), run seamlessly in a web browser, and not impose any restrictions or rate-limiting of legitimate uses.

On the back-end, our protocol had to accommodate the resource constraints of Callisto. For ease of deployment and maintenance on the web, the recovery servers must run on commodity hardware and the protocol must only use standard public and symmetric key cryptography in widely used and available software libraries.

Our protocol has been implemented and tested, and as of October 2023 is running live in production within Callisto's secure matching system. It uses only the cryptographic primitives that were already in place for Callisto's existing matching process, for ease of development and maintenance. However, we emphasize that our account recovery protocol is independent of Callisto's services; it is generic and can be used by any privacy-respecting website that desires not to keep personally-identifiable logs about their users.

Security guarantees. Our protocol provides security against three types of attackers. As with account recovery in other web services, email validation is our main security protection against external attackers. If an attacker has additionally compromised some of our recovery servers, they must still perform an online dictionary attack to obtain any email address, which we rate-limit through honest recovery servers. Finally, even if all of the recovery servers are compromised then an offline dictionary attack is still needed—which is not impossible, but onerous enough to allow Callisto to defend itself against legal compulsion of the email list.

The challenge of rate-limiting. Beyond standard symmetric key crypto primitives and a slow hash function, the only crypto primitive that we require is an *oblivious pseudorandom function* (OPRF). The client independently interacts with each server to compute an OPRF for two distinct reasons: (a) to hide the client's email address, contact information, and security answers from the recovery servers, and (b) as part of our rate-limiting mechanism, whereby an honest recovery server can detect and stop a brute-force dictionary attack of contact information or security answers.

The idea of using an OPRF for rate-limiting is not new. Several prior works have designed *partially* oblivious pseudorandom functions (pOPRFs) where the server publicly sees some information about which account is being accessed. This approach can aid in rate-limiting (e.g., [33, 67]).

However, rate-limiting in our context is challenging. This is because we need to hide the contact info and security answers during *both* account creation and recovery, but we want rate-limiting *only* in account recovery. Account creation is a delicate moment for a survivor, and we do not want to risk re-traumatizing them by denying signups due to rate-limiting.

K-pop: a new type of OPRF. As a result, in this work we design a new kind of primitive that can operate either as a pOPRF or as an OPRF. Identical outputs are produced in either mode. During account creation, we desire a pOPRF that uses a fresh public nonce as input, so that the servers are assured that this PRF query is for a new account and therefore do not need to impose rate-limiting. During account recovery, we desire a standard OPRF to hide which account is being accessed; hence, only here is rate-limiting needed.

We call this new primitive a *kaleidoscopic partially oblivious PRF*, or K-pop. In this work, we provide a formal definition of a K-pop along with a construction from any group where the Discrete Diffie-Hellman assumption is hard (e.g., elliptic curve groups), in the random oracle model.

1.2 Our contributions

In summary, this work provides three contributions.

First, we contribute a new primitive called a *kaleidoscoping partially oblivious PRF*, or K-pop. This function can be calculated either as an OPRF or pOPRF, and reaches the same result either way. We construct a K-pop without bilinear maps and provide an open-source implementation of this construction.

Second, we use the K-pop in order to construct an interactive protocol for account recovery where even the service provider does not require, and does not learn, the email addresses of its account-holders. This construction uses only standard public and symmetric key primitives that are available in many crypto libraries. This construction has been developed, tested, integrated, and deployed within Callisto's matching system.

Third, we provide two proofs of security of our account recovery protocol: a game-based security analysis against static adversaries, and a universally composable (UC) security [18] analysis against adaptive adversaries. In both analyses, a key challenge is to ensure security up to abort against malicious adversaries, but while avoiding the use of zero knowledge proofs (and therefore, also avoiding verifiable OPRFs).

1.3 Ethics and limitations

The question of whether a new technology is even needed and appropriate must be determined in consultation with subject-matter experts and in partnership with relevant communities.

Since these systems are intended to support survivors of sexual assault, designing any cryptosystem (new or otherwise) should only be done if deemed necessary and useful to survivors. *Development of this account recovery protocol was specifically requested by Callisto*, based on their users' needs.

We wish to stress the importance of developing technology through a trauma-informed lens that minimizes the risk of retraumatization. Lack of account recovery may exacerbate existing trauma as it prevents survivors from accessing prior information they had submitted. This may produce feelings of loss of control as personal details about sexual violence they had experienced are rendered inaccessible. Reactions to trauma, such as heightened anxiety and hypervigilance, can also impact survivors' interactions with technology [25]. It is therefore critical that an account recovery process is familiar and easy to use so as to not increase survivors' cognitive burden. When creating this account recovery process, we worked extensively with Callisto to make sure that the design would meet their technical constraints and also enable them to use a common and familiar user interface on the front end. As part of our ongoing work together, we have been in regular communication with Callisto about the publication of this work, which they are supportive of.

Both our problem formulation and our proposed solution have important limitations. First, our threat models are tailored to the specific needs of Callisto. Assumptions made in some of our threat models, such as honest server behavior during account creation and a non-colluding set of servers, may not be reasonable for other web services. Second, implementing our protocol necessarily increases the attack surface of the web service. A security bug in implementation could expose user information and put vulnerable users at risk. Third, providing stronger anonymity on the Internet may not always be a desirable goal, and we stress the importance of working with domain experts to determine the appropriate balance between any privacy technology and other social principles, policy objectives, or legal requirements in the context of a particular web service.

Lastly, our work does not address the structural barriers that make it difficult for survivors of sexual assault to find support and seek their own pathways toward accountability and healing. We note the significant barriers survivors face in addressing harms due to victim-blaming and institutional failures. Our protocol, and secure matching systems in general, are not solutions to this root problem, but nevertheless may be helpful to some survivors when navigating their options.

2 Technical Overview

In this section, we provide an informal summary of the technical contributions in this work. We begin by detailing the system setup, threats considered, and rationale for our design choices. Then, we describe our K-pop and account recovery constructions and explain how they build upon prior work.

2.1 Design principles and threat model

In this section, we take a deeper dive into the security threats and design considerations that influence our protocol design.

System setup. The account recovery system contains two types of actors: one or more clients who possess user accounts in the system, and a collection of N servers who participate in account recovery. The design follows the 'anytrust' paradigm in which only one server needs to be honest to provide the strongest security, and it also provides some meaningful security guarantees even if all N servers are compromised. Here, an honest server has two responsibilities: not sharing its OPRF secret key with the other servers, and properly enforcing rate limits on requests from clients and the other servers. All network communications are protected using TLS, and the adversary is presumed to have some amount of network control but not a fully global view—for instance, we presume that the adversary does not have full visibility or control of the client's email service provider.

Design principles. In this section, we expand upon the design requirements discussed in §1.1. Then, we describe some design accommodations that were deemed acceptable in our setting by our team and Callisto. While admittedly some of these principles are specific to our specific tech transition, nevertheless we believe that many of our design principles—and the motivations behind them—may generalize to other web services that want private account recovery.

As discussed in §1.3, our account recovery system is designed for survivors of sexual assault who may be using the system while having just experienced trauma. It is essential that any technology that is built is trauma-informed and is designed to account for these experiences to avoid retraumatization. Since user accounts for Callisto's matching system contain personal information that may include details about sexual assault, it is imperative that survivors are able to access their accounts quickly and seamlessly. Therefore, an account recovery process must be easy and familiar for survivors to use. As such, we designed a protocol to be compatible with a front-end user experience that is common to account recovery: a user enters their email along with additional contact information and receives security answers they must respond to. Survivors should not be burdened with learning new mechanisms that are complicated or unfamiliar. Trauma-informed design principles such as *safety* and *trust* [63] are pursued by creating an account recovery process that operates in a predictable manner while respecting the privacy of the user's personal information.

We impose three server-side requirements. First, the recovery servers must run on commodity machines without making any specific assumptions about the hardware. These constraints are common in many web applications on the cloud, and in particular we rejected the use of trusted hardware due to concerns about cost, challenges with upgrades and maintenance, and security vulnerabilities (e.g., [13,23,34,51,52,68]). Second, to simplify software development and maintenance, the recovery servers must only use common crypto primitives with widely-used software libraries. In particular, we avoided the use of pairing-based cryptography since these libraries are less widely available in deployment settings and less understood by the general software engineering community. Third, for ease of deployment, the recovery servers must be able to communicate independently with the client (e.g., to execute separate instances of the OPRF protocol) rather than with each other; that said, they can maintain joint state like a counter acting as a nonce.

On the flip side, a non-collusion assumption in the anytrust setting was deemed acceptable if it provides defense-in-depth such that no collection of all-but-one recovery servers (or the engineers who administer them) can recover email addresses. Moreover, we deemed it to be acceptable if the recovery servers learn the client's email address at the moment of account creation for two reasons: (a) the website's onboarding process already involves sending an email at the moment of account creation, and (b) the main threat was deemed to be bulk, retrospective extraction of already-registered accounts rather than the smaller set of accounts created or recovered during the interval of server compromise.

We remark that it is possible to protect addresses even during the act of email delivery by using MPC for TLS [1,64], but even that would not prevent against a different threat: one of compulsion. Due to our focus on trauma-centered design and ensuring that the secure matching system itself could not be used to harm survivors, it was important to consider the possibility that all servers become under control of a single adversary through technical or legal means, and even then to avoid a large-scale breach of users' identity and data.

Threats and security guarantees. We require that the account recovery system provide correctness (up to abort) and privacy against three types of malicious adversaries. We emphasize upfront that this work focuses on threats and mitigations at the cryptographic level. As stated in §1.3, this work does not impose any new barriers to an adversary's ability to exploit a web service; instead, we seek to mitigate the damage of an adversary who controls some or all recovery servers.

1. An *external adversary* who interacts with the recovery system protocols (but lacks control of any recovery server) should be unable even to determine whether any email address they do not control corresponds to an account in the system. Looking ahead, our account recovery protocol achieves this goal by not providing any information to the client to determine whether an email address matches an account in the system—at least, not until the moment that an honest client receives an email in her inbox.
2. An *internal adversary* who has compromised some of the recovery servers must be restricted from performing a brute-force attack of email addresses. The threat here is that an internal adversary might conduct a probing attack in which it pretends to be a client, submits a variety of different email addresses, and uses its insider access to determine whether an account exists by observing whether an email is sent out to the victim. We address this threat in two ways: first to require some additional information beyond the email address even at the first step of recovery request, and second to have the honest server(s) perform online rate limiting of the attacker. In particular, we intentionally avoid providing any deterministic function of the client's data to an individual server, because that could be used to perform an offline dictionary attack.
3. A *legal adversary* that subpoenas all recovery servers must still need to execute a (now inevitable) offline brute-force attack to learn any account information, and furthermore they must be required to perform a brute-force attack after the last server has been compromised. Looking ahead, our recovery request and account restoration protocols provide this guarantee through their use of a slow, password-based hash function to make offline dictionary attacks slower and more costly. Moreover, the slow hash function requires the outputs of the K-pop, which prevents so-called "pre-computation attacks" [45] and ensures that the expensive offline attack must occur after the servers are corrupted.

Finally, all of these security properties must hold even against honest clients who have performed account recovery or changed their security questions, potentially several times. Looking ahead, our account recovery protocol uses client and server nonces to ensure domain separation of the space that must be brute-forced before vs. after an account reset.

2.2 Our model of account recovery

Our account recovery protocol has 4 procedures: account creation and the 3 parts of account recovery (plus a simple initialization step for the servers to generate their cryptographic keys). We presume the client (who we call Alice) already has a web account with a key k_u (e.g., derived from her password) that is used for encryption of her account data at rest.

Account creation. The client Alice begins this procedure with her user account key k_u, and the goal is for her to prepare and upload some cryptographic material that facilitates later recovery of k_u. Specifically, she (obliviously) provides her account's email address E, responses to additional generic questions like her phone number x, a set of personalized security questions Q together with their corresponding answers A and a recovery email address e where she would like to be contacted if recovery is needed (this can, but does not have to, be the same as the account email address E). Anyone who later proves ownership of the recovery email account e and knows the answers x and A to the generic and personalized security questions will be able to recover the account key k_u.

Recovery request. After Alice forgets her password, she (obliviously) submits her email address E and her responses x to any additional questions asked of everyone. Importantly, Alice herself receives no output from this protocol; she does not yet know whether E and x matched any previously-created accounts. If a match exists, then the servers receive (secret shares of) Alice's recovery email address e and security questions Q. This stage has two purposes: involving the servers for online rate-limiting of any dictionary attack, and providing the servers with the information they need to perform verification and ask personalized security questions to the client.

Account verification. The servers send an email to Alice's recovery address e containing a one-time link. In this work, we model the email delivery as an instance of secure message transmission $\mathcal{F}_{\mathsf{SMT}}$. This functionality can be instantiated by the servers in the clear (in which case they temporarily learn Alice's email address and must then delete it afterward) or using secure multi-party computation (so that they can jointly send the email while individually not learning e) [1, 38, 64].

Account restoration. This stage begins when the client clicks on the link from the email, which (among other things) contains her security questions Q. The client obliviously provides her corresponding security answers A, which—if correct—can be used to recover her key k_u. The servers receive no output from restoration. Upon restoration, Alice should immediately re-run account creation to choose a new password for her account and select (possibly new, or possibly the same) security questions and answers to protect her keys.

2.3 Related work

In this section, we describe some prior work that informs our choice of a protocol design.

Secure matching systems. Secure matching systems[1] are cryptographic protocols that aim to augment informal "whisper networks" that exist in many communities such as college campuses. They permit a collection of survivors of sexual assault to determine whether they have been harmed by the same individual—even though the survivors have never met each other, never communicate directly with each other, and may never be online at the same time as each other. To accomplish this goal, secure matching systems use special-purpose secure computation among a non-colluding set of servers.

The work of Rajan et al. [61] was the first to construct a cryptographically secure matching system; their work relies on non-collusion of two servers, and it provides confidential matching when two survivors provide identifying information about the same individual. Subsequent works achieved stronger functionality and integrity guarantees, at the expense of requiring a public key infrastructure and an honest majority among 3 or more servers. WhoToo [49] allows for each survivor to choose their own matching threshold (potentially greater than two), and adds traceability for false accusations. Arun et al. [6] contribute a constant-time matching protocol to determine, after each submission, whether any collection of reports has exceeded the threshold. WhoToo+ [39] fixes some ambiguities in WhoToo and extends it also to support matching with a constant number of online operations. Finally, Shield [48] improves upon the work of Arun et al. in two ways: hiding the threshold chosen by each survivor, adding integrity checks against fake and duplicative submissions.

MPC and TEEs. Secure multi-party computation (MPC) is a cryptographic technique for a collection of servers to perform a joint computation while not learning any of the underlying data. While the fundamental concepts of MPC have been known for four decades [10, 11, 24, 35, 70], MPC on its own is typically incapable of withstanding the threat of legal adversaries, as we describe in more detail in §2.5.

Trusted execution environments (TEEs) offer an alternative method to protect sensitive data using hardware isolation (rather than cryptography). They have been proposed in commercial processors (e.g., [4,5,26,46,66]), academic prototypes (e.g., [27, 40, 62, 69]), and cloud-based services [3, 36, 57]. However, nearly all of these systems are subject to both physical and remote attacks (e.g., [13, 23, 34, 51, 52, 68]), and it remains an open question as to what extent the vision of trusted hardware can be realized. For these reasons and due to our design requirement to operate on general-purpose hardware, we avoid use of TEEs in this work.

Oblivious PRFs. A pseudorandom function (PRF) is a deterministic but "random-looking" function $y = f_k(x)$ in which the mapping between the input x and output y is unpredictable without knowledge of the secret key k. An oblivious PRF is an interactive protocol to evaluate $f_k(x)$ that hides the server

[1]While some other works in this area refer to these systems as "secure allegation escrows," we have selected an alternative name based on Callisto's terminology for their service [16], which they describe as a "matching system." The use of the term "allegation" may *unintentionally* imply a lack of belief in survivors' experiences. As advocates have noted in a guide on language use for sexual assault [47], "Many people say they use the word 'alleged' to refer to sexual assault cases, because they have not reached a final resolution within the criminal justice system ... However, it is important to keep in mind that only a miniscule percentage of sexual assaults ever make their way through the entire criminal justice process. ... [A]lmost all sexual assaults remain "unresolved" by the legal system, and it would be inappropriate to refer to all such reports (or even disclosures) of sexual assault as 'alleged.' "

key k and the client input x from each other; it also allows the server to perform service-wide rate-limiting by refusing to participate after some number of queries within a time period. A *partially* oblivious pseudorandom function (pOPRF) has two inputs $f_k(x_{\mathsf{kal}}, x_{\mathsf{priv}})$, where the server provides k and x_{kal}, and the client provides x_{priv}. This additional 'nonce' or 'salt' input x_{kal} can be used to perform per-account rate-limiting while still hiding the client's input x_{priv}.

There are a wide variety of OPRF works in the literature, starting with the work of Naor and Reingold [58]. Modern constructions tend to be based on a Hashed-Diffie-Hellman PRF $f_k(x) = H(x)^k$ that is secure in the random oracle model, or the Dodis-Yampolskiy PRF $f_k(x) = g^{1/(k+x)}$ [31]; we will use some ideas from both of these constructions in this work. Both styles of OPRFs also have verifiable counterparts that add integrity checks [33,42], often via zero-knowledge proofs (which we purposely avoid in this work as discussed in §2.1). We refer readers to the SoK by Casacuberta et al. [22] for additional OPRF and pOPRF constructions.

Among their many uses, OPRFs and pOPRFs have found value in several applications involving oblivious interactions on the web with passwords or other low-entropy secrets, such as (threshold) password-protected secret sharing [8,42,44], password hardening services [33, 50], compromised password checkers [60,65], password-authenticated key exchange [42, 45, 67], and single sign-on [9]. Many of these works also provide universally composable (UC) security models of OPRFs, optionally with a threshold or verifiability requirement. Looking ahead, we use the UC modeling of Jarecki et al. [44] in this work, which is intentionally designed to avoid the verifiability requirement. We also desire security against pre-computation attacks, as defined within OPAQUE [45].

2.4 Overview of our K-pop protocol

In this section, we provide an informal overview of our construction of a *kaleidoscoping partially oblivious PRF*, or K-pop. As a reminder, this is a function that can be calculated as an OPRF or pOPRF, and reaches the same result either way. In other words, the input x_{kal} is *kaleidoscopic* in that it can be rapidly changed between being client- or server-provided.

We observe that if bilinear maps are permissible, then it is straightforward to build a K-pop. Concretely, the Pythia pOPRF of Everspaugh et al. [33] is computed as $f_k(x_{\mathsf{kal}}, x_{\mathsf{priv}}) = e(H_1(x_{\mathsf{kal}}), H_2(x_{\mathsf{priv}}))^k$, where H_1 and H_2 are random oracles. Using standard techniques for oblivious exponentiation (shown in §3), this function f can be easily computed either as an ordinary OPRF (where the client computes the bilinear map and the server obliviously exponentiates by k) or as a pOPRF (where the client blinds the left input before the server computes the bilinear map).

However, introducing a bilinear map violates our design principle only to use widely available crypto libraries. As a result, we have designed a K-pop that can be built purely from group and finite field operations, in the random oracle model. Constructing this is non-trivial, as there are no OPRFs and pOPRFs in the literature [22] that already compute the same outputs as each other.

Our starting point for this work is the recent pOPRF of Tyagi et al. [67]. It uses the pseudorandom function family $f_k(x_{\mathsf{priv}}, x_{\mathsf{kal}}) = H_2(x_{\mathsf{kal}}, x_{\mathsf{priv}}, H_1(x_{\mathsf{priv}})^{1/(k+H_3(x_{\mathsf{kal}}))})$, which combines characteristics of the Hashed-DH and Dodis-Yampolskiy OPRFs described in §2.3. We extend this construction to a K-pop by providing an OPRF protocol as well, which uses additively homomorphic encryption between the server (who holds k) and the client (who holds x_{kal}) to compute the exponent $\frac{1}{k+H_3(x_{\mathsf{kal}})}$ that is used in oblivious exponentiation. While the OPRF mode requires a larger number of public-key operations, this mode of our K-pop only needs to be executed in the (less frequent) account recovery phase.

2.5 Overview of our account recovery protocol

This section provides a high-level description of our account recovery protocol. To provide some intuition about the challenges involved in this construction, we iteratively build it up by describing several ideas that look promising but ultimately fall short of meeting our objectives. We then propose changes until arriving at our final construction.

MPC-based approaches. A natural approach here is to use MPC, either generally (e.g., run account recovery as a large circuit) or using private information retrieval, encrypted database search, or other special-purpose primitives. Within our 3-step recovery request, verification, and restoration procedure, it might seem at first glance that the hardest approach to satisfy with MPC is the email verification step. But actually this part is mostly a solved problem: Abram et al. [1] showed how a coalition of MPC servers can collectively emulate TLS 1.3 communications, and MPCAuth [64] applied this technique to email delivery in such a way that the servers never learn the client's email address or contents of the email.

Instead, the primary challenge is to withstand the threats of an internal and legal adversary during the recovery request and restoration steps, since MPC-based approaches typically do not (a) provide any form of rate-limiting of queries or (b) stop a colluding coalition of all servers from instantly reconstructing all data. Here the recent TLS-OPAQUE construction of Hesse et al. [38] is more promising: it uses secure computation of TLS for email delivery along with an OPRF that could be adapted to ensure that the client's data (e.g., email address and security questions/answers) is protected by online rate limiting if even a single server is honest and requires an offline dictionary attack even after all servers are corrupted. In this work, we purposely choose not to use any form of MPC-for-TLS since (a) it is slow and non-standard to implement, (b) it is not necessary to satisfy our design principles (§2.1), and (c) even with MPC-for-TLS, it remains unclear how to protect the client's data using an OPRF. We focus on this latter question next.

OPRF-based approaches. When using an OPRF in account recovery, an initial thought might be for recovery request to involve a standard OPRF on the email address E: that is, to have the client and servers calculate $y = f_k(E)$ and to protect the client's user account key using y. However, this approach would allow an external adversary to conduct a brute-force search of others' email addresses. A better approach is to reveal y only after verifying that the client controls the email address. Even so, an internal adversary (who maliciously also acts as a client) could perform a brute-force search of email addresses and recover user data; ultimately, email addresses do not have enough entropy on their own.

Hence, we wish to add additional questions, but here we face a chicken-and-egg issue. Ideally, we want each client to be able to choose their own custom security questions, and then construct some method where the OPRF output y is used by the servers to retrieve each client's custom questions. But since a retrieving client has forgotten all passwords and crypto keys, any brute-force attacker would also be able to read the same security questions. Hence, an internal adversary could use the mere fact that security questions have been successfully retrieved to conclude that an email address E must be registered in the system.[2] We resolve this issue by allowing for generic questions for additional information x (e.g., a phone number) and computing the OPRF as $y = f_k(E,x)$ where y is used as a key to unlock a second layer of custom security questions. In this way: the generic questions provide increased resistance against brute-force attack by an internal adversary, and the custom questions provide stronger (and industry standard) protection against external adversaries.

The next challenge is that if the same OPRF $y = f_k(E,x)$ is used during account creation and recovery, then both need to be rate-limited or else an internal adversary could use account creation requests to brute-force y. We resolve this issue by using a K-pop togther with a server-chosen nonce n that is guaranteed to be unique for all account creation requests. Then, we can run a pOPRF $y = f_k(n,(E,x))$ during account creation and the corresponding OPRF during recovery. With a K-pop, we can enforce rate-limiting of account recovery requests in such a way that we do not know which account is being recovered and need not rate-limit account creation.

Finally, any OPRF-based approach (on its own) does nothing against a legal adversary in control of all servers, since a PRF provides no security against the key-holder. We resolve this issue by feeding y into a slow cryptographic hash function H that can slow down the rate of offline brute-force attempts. Importantly, the OPRFs of all servers are performed first and only the results are fed into H, so that this offline brute-force attack must occur *after* the moment that all server keys are compromised. In this way, our construction is secure against pre-computation attacks [45]. This finally yields a secure construction, which we describe in detail in §4.

[2]We remark that this leakage is not inherent from a cryptographic perspective; indeed, it is possible to calculate the security questions pseudorandomly from the email address E so that it can be retrieved whether or not E was ever registered in the system. But we rejected this approach because it went against our goal of trauma-informed care: if a survivor accidentally mistyped their email address, we did not want to subject them to try in vain to respond to an incorrect set of security questions.

Symbol	Description
$\mathbf{C}$	Client
$\mathbf{S}_i$	Recovery server i
N	Number of recovery servers (default is $N = 2$)
λ	Cryptographic security parameter, e.g., 256 bits
k_u	Client's user account key (which decrypts account data)
H	Cryptographic hash function, modeled as a random oracle
E	Client's email address associated with the account
x	Client's answers to any generic questions (e.g., contact info)
e	Client's recovery email address (can be the same as E)
Q	Security questions chosen by the client
A	Answers to the client-chosen security questions
n	Nonce chosen publicly by the recovery servers
m	Nonce chosen privately by the client
r	Recovery string, set as $r = e \parallel Q \parallel m \parallel \mathit{padding}$
ℓ	fixed length of the recovery string after padding is applied
D	Database held by both recovery servers (this stores all data provided by all clients during account creation)
$\hat{E}_i$	Result of the K-pop run in recovery request with server i
$\hat{A}_i$	Result of the K-pop run in recovery restoration with server i
id	Client's identifier for her record in the database D
$\mathsf{ct}_r, \mathsf{ct}_u$	Ciphertexts produced by the client to store in the server database; they are a one-time pad of r and k_u, respectively
k_E, k_A	One-time pad keys for the ciphertexts above
k	Server's secret key for an OPRF
x_{priv}	Client's private input to an OPRF or pOPRF
x_{kal}	Input provided by the server in a pOPRF, or by the client in an OPRF

Table 1: Notation used in this paper.

3 Kaleidoscopic Partially Oblivious PRF

In this section, we define and construct a new primitive that we call a *kaleidoscopic partially oblivious PRF*, or K-pop for short. This primitive is named for the fact that it can be quickly changed between an OPRF and a pOPRF at will.

3.1 Preliminaries

In this section, we briefly introduce the notation and OPRFs that we use in this work. We describe other cryptographic primitives (hash functions, Diffie-Hellman groups, and additively homomorphic encryption) in the full version of this work [53].

Notation. In this work, we often use upper-case letters like S to denote finite sets, and calligraphic letters $\mathcal{D}$ to denote distributions. The notation $x \leftarrow \mathcal{D}$ means to sample x from the distribution $\mathcal{D}$, and the notation $x \leftarrow S$ means to select

Functionality $\mathcal{F}_{\mathsf{SMT}}$

- Upon invocation, with input $(\mathsf{send}, \mathsf{sessionid}, R, m)$ from a sender party S that is intended for a receiver party R, send a message $(\mathsf{sent}, \mathsf{sessionid}, S, R, |m|)$ to the adversary $\mathcal{A}^*$.
- Upon receiving message $(\mathsf{delivered}, \mathsf{sessionid})$ from $\mathcal{A}^*$: If not yet generated output, then output $(\mathsf{sent}, \mathsf{sessionid}, S, R, m)$ to R.
- Upon receiving message $(\mathsf{corrupt}, \mathsf{sessionid}, m', R')$ from the environment **Env**: record being corrupted. Additionally, if no output has been generated yet, then output $(\mathsf{sent}, \mathsf{sessionid}, S, R', m')$ to R'.

Figure 1: Secure message transmission functionality $\mathcal{F}_{\mathsf{SMT}}$, adapted from [18]. In this work, we model both network communication via TLS and the act of sending an email from the servers to the client's email provider as instantiations of $\mathcal{F}_{\mathsf{SMT}}$.

Functionality $\mathcal{F}_{\mathsf{OPRF}}$

An instance of this functionality is uniquely defined by a session id $\mathsf{sessionid} = (\mathsf{OPRF}, \mathsf{sid})$ containing the party ID of the server (sid). The functionality interacts with both the server $\mathbf{S}_{\mathsf{sid}}$ and anyone who acts in the role of a client $\mathbf{C}$, including possibly the adversary $\mathcal{A}^*$.

- Upon receiving $(\mathsf{Init}, \mathsf{sessionid})$ from the server $\mathbf{S}_{\mathsf{sid}}$, uniformly sample $\mathsf{kid} \leftarrow\$ \{0,1\}^\lambda$, and initialize an empty table $T(\mathsf{kid}, x)$.
- Upon receiving $(\mathsf{Eval}, \mathsf{sessionid}, \mathsf{qid}, x)$ from a client $\mathbf{C}$ (which can be adversary $\mathcal{A}^*$) where qid is a unique identifier for this Eval query: end this invocation if called before Init. Otherwise, record $\langle \mathsf{qid}, \mathbf{C}, x \rangle$ and send $(\mathsf{EvalContinue}, \mathsf{sessionid}, \mathsf{qid})$ to $\mathcal{A}^*$.
- Upon receiving $(\mathsf{EvalContinue}, \mathsf{sessionid}, \mathsf{qid}, \mathsf{kid}^*)$ from $\mathcal{A}^*$: end this invocation if there is no record with qid. Otherwise, find and delete record $\langle \mathsf{qid}, \mathbf{C}, x \rangle$. If $\mathbf{S}_{\mathsf{sid}}$ is honest, set $\mathsf{kid}^* \leftarrow \mathsf{kid}$ (i.e., use the correct key). Next, fetch $\rho \leftarrow T(\mathsf{kid}^*, x)$ as follows:
  - If $T(\mathsf{kid}^*, x)$ is defined, then look up $\rho \leftarrow T(\mathsf{kid}^*, x)$ and send $(\mathsf{EvalComplete}, \mathsf{sessionid}, \mathsf{qid}, \rho)$ to $\mathbf{C}$.
  - Otherwise pick ρ at random from $\{0,1\}^\ell$, assign $T(\mathsf{kid}^*, x) := \rho$, and send $(\mathsf{EvalComplete}, \mathsf{sessionid}, \mathsf{qid}, \rho)$ to $\mathbf{C}$.
- Upon receiving $(\mathsf{OfflineQuery}, \mathsf{sessionid}, x, \mathsf{kid}^*)$ from $\mathbf{C}$: fetch $\rho \leftarrow T(\mathsf{kid}^*, x)$ as above. Send $(\mathsf{OfflineQuery}, \mathsf{sessionid}, \rho)$ to $\mathbf{C}$.
- Upon receiving $(\mathsf{corrupt}, \mathsf{pid})$ from the environment **Env**: mark the party corresponding to pid as corrupted. Additionally:
  - For corruption of the server, send kid to the adversary.
  - For corruption of a client $\mathbf{C}$, send all corresponding records $\langle \mathsf{qid}, \mathbf{C}, x \rangle$ to the adversary.

Figure 2: Functionality $\mathcal{F}_{\mathsf{OPRF}}$, based on Jarecki et al. [43] modified to use a unique query identifier qid like Das et al. [28] to track commands sent to $\mathcal{F}_{\mathsf{OPRF}}$ about the same evaluation query. $\mathcal{F}_{\mathsf{OPRF}}$ does not guarantee honest behavior by the server; the key identifier kid corresponds to the server's choice of key to use when responding to each query. $\mathsf{OfflineQuery}$ allows anyone to evaluate the OPRF on input x and key id kid^* of their choice—which is unlikely to match the honest kid unless $\mathbf{S}$ is corrupted.

x uniformly at random from the set S. We also use $[n] = \{1, \ldots, n\}$ to denote the set of integers from 1 to n, inclusive.

We denote the protocol participants using bold letters. A client is denoted as $\mathbf{C}$, and a server is denoted as $\mathbf{S}$. For protocols that contain multiple servers, we use subscripts to distinguish them: server 1 is denoted as $\mathbf{S}_1$, server 2 is $\mathbf{S}_2$, and so on. We also use **Adv** to denote a real-world adversary who attempts to attack our protocol—either maliciously or semi-honestly, as stated in the respective theorem statements—and **Sim** to denote the corresponding ideal-world simulator. See Table 1 for a detailed list of all variables used in this work.

Oblivious pseudorandom function. As described in §2.3, an OPRF is an interactive two-party protocol in which a client $\mathbf{C}$ has input x, a server $\mathbf{S}$ has input k, and they jointly compute $y = f_k(x)$ in such a way that neither party learns anything about the other input. However, the server can deviate from the protocol, and correctness is not guaranteed in this case. Formally, we model an OPRF as an instantiation of the UC functionality $\mathcal{F}_{\mathsf{OPRF}}$ shown in Figure 2, which we adapt from Jarecki et al. [43] with a small tweak to add support for server-side rate-limiting. If a client submits an input x and the server is honest, then the client receives the correct output—potentially after some adversarially-chosen delay since this is an interactive protocol. If the server is malicious, then it could act as though it used a different key or produce a random output, which the functionality also accounts for.

3.2 Defining K-pop

Like an OPRF or pOPRF, a K-pop is an interactive protocol between two participants who we call the *client* $\mathbf{C}$ and the *server* $\mathbf{S}$. Concretely, a K-pop is a keyed family of pseudorandom functions with two inputs $f_k^{\mathsf{K\text{-}pop}}(x_{\mathsf{kal}}, x_{\mathsf{priv}})$. Looking ahead to our account recovery protocol: the client will execute independent instances of K-pop with each recovery server.

The K-pop can be computed either as a pOPRF or OPRF. Concretely, the input x_{kal} is *kaleidoscopic* in that it can be rapidly changed between being public to the server (in pOPRF mode) or private to the client (in OPRF mode). Importantly, the K-pop returns the same answer in either mode.

We formalize these requirements through the UC functionality $\mathcal{F}_{\mathsf{K\text{-}pop}}$ provided in Figure 3, which codifies both the functionality and security requirements of K-pop. It calls the OPRF functionality $\mathcal{F}_{\mathsf{OPRF}}$ (Fig. 2) as a subroutine. We stress that $\mathcal{F}_{\mathsf{K\text{-}pop}}$ makes the *same* call to $\mathcal{F}_{\mathsf{OPRF}}$ whether the input

Functionality $\mathcal{F}_{\textsf{K-pop}}$

An instance of this functionality is identified by a session id $\textsf{sessionid} = (\textsf{K-pop}, \textsf{sid})$ containing the party ID $\textsf{sid}$ of its server. It interacts with the server $\mathbf{S}$ and any client $\mathbf{C}$ (possibly the adversary $\mathcal{A}^*$), it has a subroutine $\mathcal{F}_{\textsf{OPRF}}$ and a query limit L, and it operates as follows.

- Upon receiving $(\textsf{Init}, \textsf{kid})$ from server $\mathbf{S}_{\textsf{sid}}$: set $\textsf{ctr} \leftarrow 0$ and send $(\textsf{Init}, \textsf{kid})$ to $\mathcal{F}_{\textsf{OPRF}}$. (Other methods abort if called before Init.)
- Upon receiving $(\textsf{Reset})$ from the server $\mathbf{S}_{\textsf{sid}}$: set $\textsf{ctr} \leftarrow 0$ and send the message $(\textsf{ResetComplete})$ to $\mathcal{A}^*$.
- Upon receiving $(\textsf{Eval}, \textsf{sessionid}, \texttt{OPRF-mode}, \textsf{qid}, x_{\textsf{kal}}, x_{\textsf{priv}})$ or $\boxed{(\textsf{Eval}, \textsf{sessionid}, \texttt{pOPRF-mode}, \textsf{qid}, x_{\textsf{kal}}, x_{\textsf{priv}})}$ from a client $\mathbf{C}$:
  - Send an error $(\textsf{Eval}, \textsf{sessionid}, \textsf{qid}, \perp)$ to $\mathcal{A}^*$ if there was a prior message with $\textsf{qid}$, or if $\mathbf{S}$ is honest and $\textsf{ctr} > L$.
  - $\boxed{\text{If in pOPRF mode: send } (\textsf{Eval}, \textsf{sessionid}, \textsf{qid}, x_{\textsf{kal}}) \text{ to } \mathcal{A}^*\text{, and wait for a response } (\textsf{EvalContinue}, \textsf{sessionid}, \textsf{qid}).}$
  - Increment query counter $\textsf{ctr} \leftarrow \textsf{ctr} + 1$, and send $(\textsf{Eval}, \textsf{sessionid}, \textsf{qid}, (x_{\textsf{kal}}, x_{\textsf{priv}}))$ to $\mathcal{F}_{\textsf{OPRF}}$ on behalf of $\mathbf{C}$.
- Upon receiving $(\textsf{EvalComplete}, \textsf{sessionid}, \textsf{qid}, \rho)$ from $\mathcal{F}_{\textsf{OPRF}}$: output $(\textsf{EvalComplete}, \textsf{sessionid}, \textsf{qid}, (x_{\textsf{kal}}, x_{\textsf{priv}}), \rho)$ to client $\mathbf{C}$.
- Upon receiving $(\textsf{OfflineQuery}, \textsf{sessionid}, x_{\textsf{kal}}, x_{\textsf{priv}}, \textsf{kid}^*)$ from $\mathbf{C}$: send this message to $\mathcal{F}_{\textsf{OPRF}}$, and send its response to client $\mathbf{C}$.
- Upon receiving $(\textsf{corrupt}, \textsf{pid})$ from the environment $\mathbf{Env}$: mark the party with pid as corrupted, and send $(\textsf{corrupt}, \textsf{pid})$ to $\mathcal{F}_{\textsf{OPRF}}$.

Figure 3: Functionality $\mathcal{F}_{\textsf{K-pop}}$ with subroutine $\mathcal{F}_{\textsf{OPRF}}$. Differences between OPRF and pOPRF modes are shown in a $\boxed{\text{box}}$. $\mathcal{F}_{\textsf{K-pop}}$ maintains a counter ctr of the number of Eval queries, and it enforces a limit of L evaluations between Reset commands.

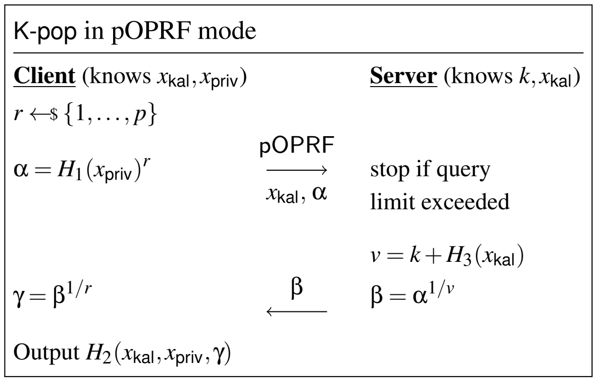

K-pop in pOPRF mode
Client (knows $x_{\textsf{kal}}, x_{\textsf{priv}}$)
Server (knows $k, x_{\textsf{kal}}$)
$r \leftarrow\!\$ \{1, \ldots, p\}$
$\alpha = H_1(x_{\textsf{priv}})^r$
pOPRF
$x_{\textsf{kal}}, \alpha$
stop if query limit exceeded
$v = k + H_3(x_{\textsf{kal}})$
$\beta = \alpha^{1/v}$
β
$\gamma = \beta^{1/r}$
Output $H_2(x_{\textsf{kal}}, x_{\textsf{priv}}, \gamma)$

Figure 4: K-pop in pOPRF mode, where both the client and server know $x_{\textsf{kal}}$. The server need not be honest; it can use incorrect k^* or $x^*_{\textsf{kal}}$ in its oblivious exponentiation. All communications use secure message transmission $\mathcal{F}_{\textsf{SMT}}$ (Fig. 1).

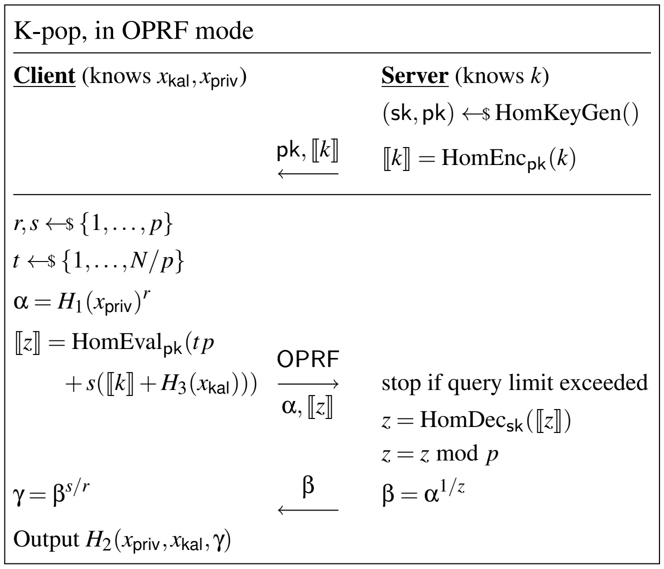

K-pop, in OPRF mode
Client (knows $x_{\textsf{kal}}, x_{\textsf{priv}}$)
Server (knows k)
$(\textsf{sk}, \textsf{pk}) \leftarrow\!\$ \text{HomKeyGen}()$
$[\![k]\!] = \text{HomEnc}_{\textsf{pk}}(k)$
$\textsf{pk}, [\![k]\!]$
$r, s \leftarrow\!\$ \{1, \ldots, p\}$
$t \leftarrow\!\$ \{1, \ldots, N/p\}$
$\alpha = H_1(x_{\textsf{priv}})^r$
$[\![z]\!] = \text{HomEval}_{\textsf{pk}}(tp + s([\![k]\!] + H_3(x_{\textsf{kal}})))$
OPRF
$\alpha, [\![z]\!]$
stop if query limit exceeded
$z = \text{HomDec}_{\textsf{sk}}([\![z]\!])$
$z = z \bmod p$
$\beta = \alpha^{1/z}$
β
$\gamma = \beta^{s/r}$
Output $H_2(x_{\textsf{priv}}, x_{\textsf{kal}}, \gamma)$

Figure 5: K-pop in OPRF mode, where $x_{\textsf{kal}}$ and $x_{\textsf{priv}}$ are both private inputs supplied by the client. All communications use secure message transmission $\mathcal{F}_{\textsf{SMT}}$ (Fig. 1). The first message can be sent in a preprocessing step or using a PKI.

$x_{\textsf{kal}}$ is provided by the server (in pOPRF mode) or by the client (in OPRF mode). Since $\mathcal{F}_{\textsf{OPRF}}$ does not know whether it was called in OPRF or pOPRF mode, its response must be the same in both cases; in other words, $\mathcal{F}_{\textsf{K-pop}}$ ensures identical outputs in the OPRF and pOPRF modes by design. Also, note that if the parties are honest, then $\mathcal{F}_{\textsf{K-pop}}$ is guaranteed to be deterministic and repeatable because $\mathcal{F}_{\textsf{OPRF}}$ is.

However, if the server $\mathbf{S}$ is malicious, then in pOPRF mode it *can* ignore the agreed-upon $x_{\textsf{kal}}$ and choose a different $x^*_{\textsf{kal}}$. While one could incorporate a verifiability guarantee into a K-pop, we choose not to do so because it is not needed in our account recovery protocol in §4 and this way we do not need to use zero knowledge proofs. Instead, we leave it up to any protocol that uses K-pop to detect and abort in case of malicious server behavior.

3.3 Constructing K-pop

In this section, we construct a K-pop by combining and extending techniques from Tyagi et al. [67] and Miao et al. [56]. Specifically, our K-pop construction computes the same function as the pOPRF of Tyagi et al. [67]:

$$f_k^{\textsf{K-pop}}(x_{\textsf{kal}}, x_{\textsf{priv}}) = H_2\left(x_{\textsf{kal}}, x_{\textsf{priv}}, H_1(x_{\textsf{priv}})^{1/(k+H_3(x_{\textsf{kal}}))}\right).$$

Here, H_1 is a cryptographic hash function whose output is a point on an elliptic curve group, and H_2 and H_3 are random oracles that return finite field elements. We use $x_{\textsf{priv}}$ and $x_{\textsf{kal}}$ to denote the inputs, where x_{priv} is always a secret input supplied by the client, and x_{kal} is the kaleidoscopic input. In the OPRF mode, x_{kal} is chosen by the client and kept secret from the server; by contrast, in the pOPRF mode, x_{kal} is known and agreed upon by the client and server.

This construction by Tyagi et al. [67] combines ideas from two prior PRF families: the Dodis-Yampolskiy [31] PRF $f_k(x) = g^{1/(k+x)}$ and the Hashed-DH PRF $f_k(x) = H(x)^k$. The outer-most hash function H_2 is useful to provide extraction for UC security, in the random oracle model [42]. Below, we reproduce the pOPRF design of Tyagi et al. [67], and then we construct an OPRF that computes the same result. In both cases, we model all network communications with a secure message transmission functionality $\mathcal{F}_{\mathsf{SMT}}$ in Fig. 1; that said, it is sufficient to use an authenticated communication channel as defined in several prior works (e.g., [7, 18–20]).

pOPRF Mode. We begin by describing the K-pop in pOPRF mode. Because the server knows x_{kal}, it can compute $(k + H_3(x_{\mathsf{kal}}))^{-1}$ and use this as the exponent in a two-message oblivious exponentiation protocol with the client, as illustrated in Figure 4. Finally, the client computes the outer hash function H_2. This mode exactly follows the work of Tyagi et al. [67] (but without a zero knowledge proof), and it is the more efficient of the two modes: it requires 3 group exponentiations along with some hash function evaluations.

OPRF Mode. It remains to compute the same function in OPRF mode, in which x_{kal} (like x_{priv}) is a secret known only to the client that the server is not supposed to learn. Rather than having the server compute $v = k + H_3(x_{\mathsf{kal}})$ directly, in this mode we have the client and server compute v together, using additively homomorphic encryption so that neither party learns any intermediate state along the way. Let p denote the order of the elliptic curve group, and let the notation $[\![x]\!]$ denote a homomorphic encryption of the plaintext value x.

First, we describe the main technique under the simplifying assumption that the plaintext space of the homomorphic encryption algorithm is also a group of order p. With this simplifying assumption, we can perform a secure computation of the required addition and multiplicative inversion as follows.

1. The server chooses an ephemeral key pair for homomorphic encryption, computes $[\![k]\!] = \mathrm{HomEnc}(k)$, and sends this ciphertext to the client.
2. The client chooses two blinding factors $r, s \leftarrow \{1, \ldots, p\}$. The client computes the group element $\alpha = H_1(x_{\mathsf{priv}})^r$ in the same way as in the pOPRF mode, and also uses homomorphic addition and scalar multiplication to compute the ciphertext $[\![z]\!]$ corresponding to the plaintext $z = s(k + H_3(x_{\mathsf{kal}}))$. The client sends α and $[\![z]\!]$ to the server.
3. The server decrypts z, inverts it, and sends $\beta = \alpha^{1/z} = H_1(x_{\mathsf{priv}})^{\frac{r}{s(k+H_3(x_{\mathsf{kal}}))}}$ to the client.
4. The client computes $\gamma = \beta^{s/r} = H_1(x_{\mathsf{priv}})^{1/(k+H_3(x_{\mathsf{kal}}))}$.

Correctness is clear by inspection, and privacy against the server follows from the fact that z is blinded using s so that the server cannot learn the client's x_{kal}.

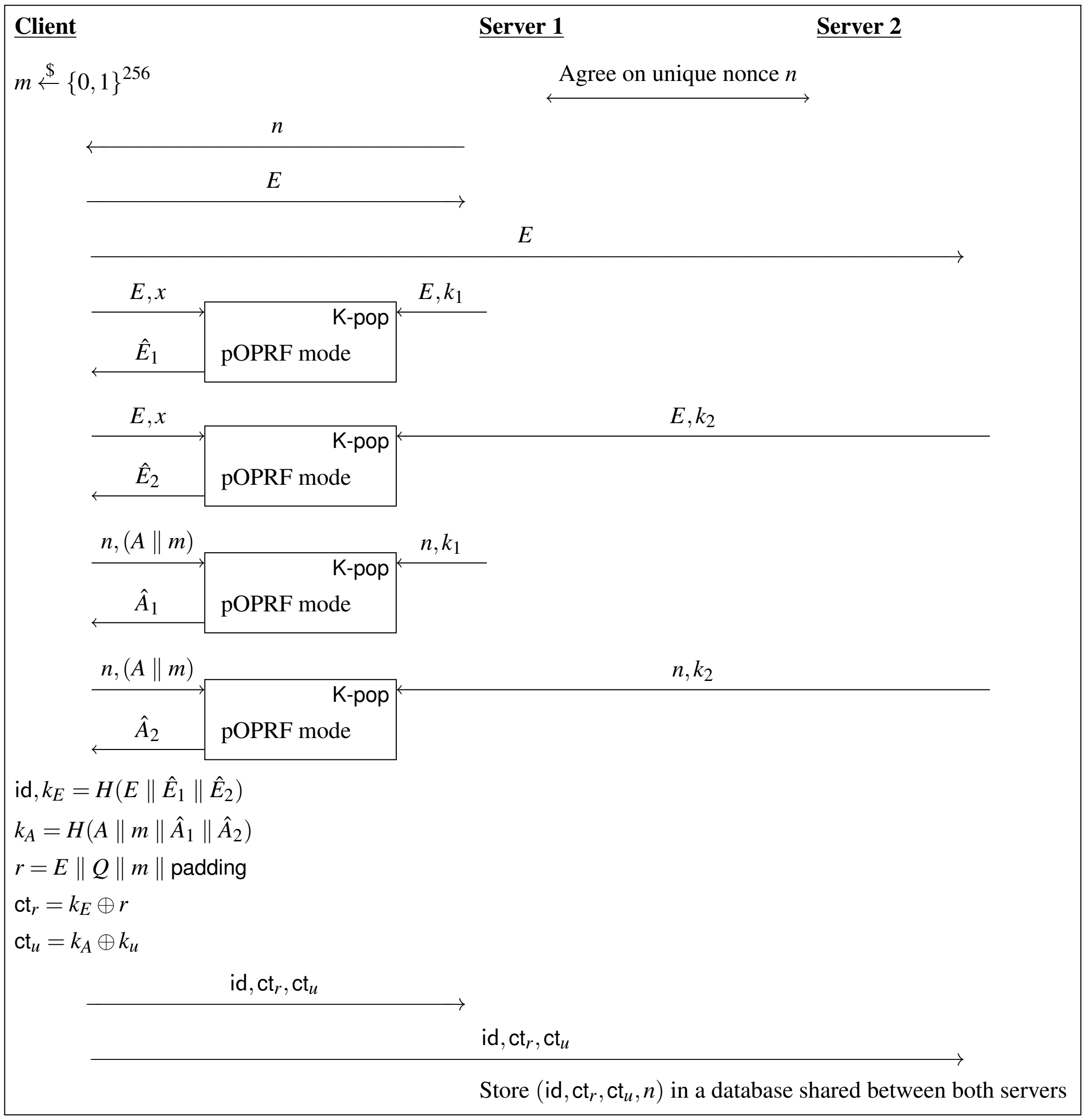
The remaining challenge is to address the fact that the exponents and HomEnc plaintexts are *not* from the same group, which we handle using a technique from Miao et al. [56]. Concretely, let $N \gg p$ denote the order of the plaintext space within an additively homomorphic encryption scheme, like Paillier [59] or Camenisch-Shoup [17] encryption (for the latter, we omit the integrity check). The issue here is that we wanted the homomorphic evaluation to produce $z = s(k + H_3(x_{\mathsf{kal}})) \bmod p$ because the server plans to use z as an exponent, but the homomorphic evaluation performs the calculation mod N instead. If $N \gg p^2$ then the calculation $z = s(k + H_3(x_{\mathsf{kal}})) \bmod N$ does not perform any modular wrapping at all, so the blinding is ineffective and revealing z to the server would allow it to learn information about the size of the client's secrets s and $H_3(x_{\mathsf{kal}})$. Fortunately, there is a simple solution to this problem shown by Miao et al. [56]: add a random multiple of p, so that the resulting plaintext is $z = s(k + H_3(x_{\mathsf{kal}})) + tp \bmod N$, where $t \leftarrow \{1, \ldots, N/p\}$ is another blinding factor chosen by the client. In this way, z still reduces to the correct exponent mod p with overwhelming probability, and its size does not reveal information about the client's secret inputs.

This protocol requires 3 group exponentiations (just like pOPRF mode) along with one homomorphic encryption, evaluation, and decryption. We illustrate the protocol in Figure 5 and prove the following theorem in the full version [53].

Theorem 1. *Assume that group G satisfies the one-more gap strong Diffie-Hellman inversion assumption and that* HomEnc *satisfies indistinguishability under a chosen plaintext attack (IND-CPA). Then, the* K-pop *protocol $\Pi_{\textsf{K-pop}}$ (in Figures 4-5) UC-realizes the ideal functionality $\mathcal{F}_{\textsf{K-pop}}$ in the presence of a programmable random oracle $\mathcal{F}_{\mathsf{pro}}$.*

4 Account Recovery

In this section, we describe our cryptographic protocol for private account recovery. Our protocol consists of the four stages described in §2.2: *account creation*, *recovery request*, *account verification*, and *account restoration*. Below we describe constructions for each protocol, with full details provided in Figures 6-8. For simplicity, our constructions are written here for the setting of $N = 2$ servers, but the protocols immediately generalize to allow for more servers. We provide a security analysis of this protocol in §4.6 and the the full version of this work [53].



Client
Server 1
Server 2
$m \xleftarrow{\$} \{0,1\}^{256}$
Agree on unique nonce n
n
E
E
E, x
E, k_1
K-pop
pOPRF mode
$\hat{E}_1$
E, x
E, k_2
K-pop
pOPRF mode
$\hat{E}_2$
$n, (A \parallel m)$
n, k_1
K-pop
pOPRF mode
$\hat{A}_1$
$n, (A \parallel m)$
n, k_2
K-pop
pOPRF mode
$\hat{A}_2$
$\mathsf{id}, k_E = H(E \parallel \hat{E}_1 \parallel \hat{E}_2)$
$k_A = H(A \parallel m \parallel \hat{A}_1 \parallel \hat{A}_2)$
$r = E \parallel Q \parallel m \parallel \mathsf{padding}$
$\mathsf{ct}_r = k_E \oplus r$
$\mathsf{ct}_u = k_A \oplus k_u$
$\mathsf{id}, \mathsf{ct}_r, \mathsf{ct}_u$
$\mathsf{id}, \mathsf{ct}_r, \mathsf{ct}_u$
Store $(\mathsf{id}, \mathsf{ct}_r, \mathsf{ct}_u, n)$ in a database shared between both servers

Figure 6: Account Creation

4.1 Initialization

Each server independently and secretly chooses a K-pop key. We denote server i's K-pop key as k_i. Additionally, a common database DB is initialized, and each server is granted read/write access.

4.2 Account creation

Account creation allows a client to register a new account in the system. We depict the complete procedure in Figure 6. The client begins by choosing the following information:

- E, the client's official email address for this account,
- e, a personal email address to use in account recovery,
- x, additional personal information provided in response to any generic questions, such as a telephone number,
- k_u, a cryptographic key used to encrypt all account data,
- Q, a set of client-chosen security questions,
- A, the client's answers to the security questions Q,
- m, a randomly generated nonce of length λ, where λ denotes the security parameter (e.g., 256 bits), and
- $r = e \parallel Q \parallel m$, a recovery string padded to fixed length ℓ.

First, the servers jointly agree on a unique nonce n for the client. This nonce is immediately sent to the client, who sends back their email string E to both servers. The servers should verify the client's email at this point by emailing them a link. Also, providing E to the servers serves a cryptographic purpose in our protocol: the client makes a K-pop query to

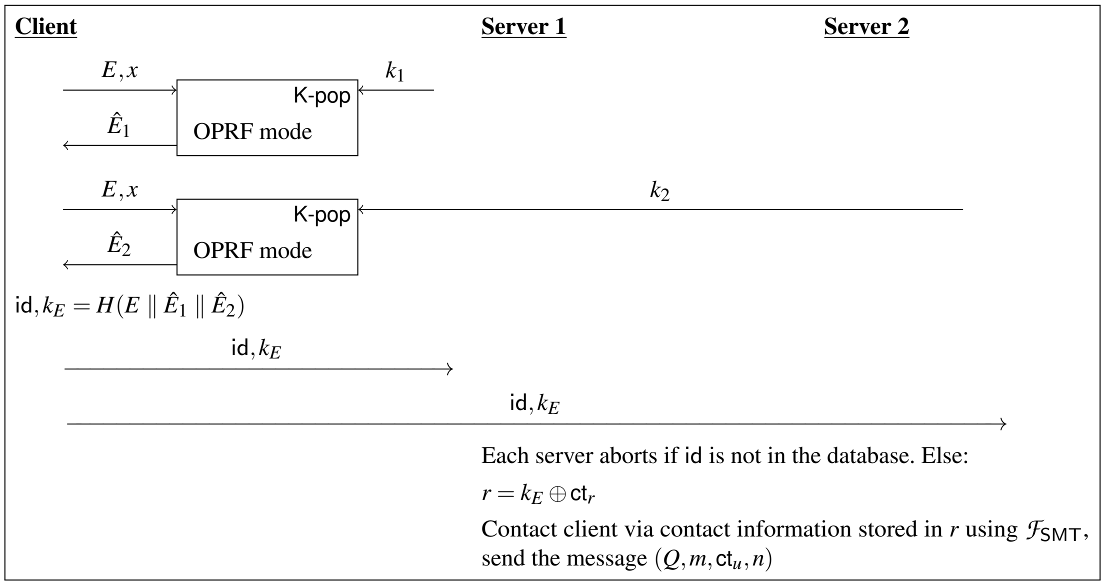

Client
Server 1
Server 2
E, x
k_1
K-pop
OPRF mode
$\hat{E}_1$
E, x
k_2
K-pop
OPRF mode
$\hat{E}_2$
$\mathsf{id}, k_E = H(E \parallel \hat{E}_1 \parallel \hat{E}_2)$
id, k_E
id, k_E
Each server aborts if id is not in the database. Else:
$r = k_E \oplus \mathsf{ct}_r$
Contact client via contact information stored in r using $\mathcal{F}_{\mathsf{SMT}}$,
send the message (Q, m, ct_u, n)

Figure 7: Recovery Request

each server in pOPRF mode, using email E as the public input and personal info x as the private input. Let $\hat{E}_1$ and $\hat{E}_2$ denote the respective K-pop outputs to the client.

Next, the client computes $H(E \parallel \hat{E}_1 \parallel \hat{E}_2)$, where H is a slow password-based key derivation function of variable output length (modeled as a random oracle). The slowness of this function is the reason why our construction provides offline rate limiting against a legal adversary, and the fact that it uses the outputs of the K-pop means that an offline attack can only be performed *after* the adversary compromises all servers. We parse the output of H into two strings: id is the first λ bits of this output, and k_E is the next ℓ bits. This ensures that k_E is the same length as r. The client then computes a one-time-pad ciphertext ct_r of r encrypted under key k_E.

Then, the client makes another K-pop query to each server in pOPRF mode, this time using n as the public input and $A \parallel m$ as the private input, receiving back the respective outputs $\hat{A}_1$ and $\hat{A}_2$. Using the same password-based key derivation function H, they compute $k_A = H(A \parallel \hat{A}_1 \parallel \hat{A}_2)$, where k_A is of length λ. The client then XOR-s k_A with their user key to produce a one-time-pad ciphertext ct_u. Finally, the client sends $\mathsf{id}, \mathsf{ct}_r$, and ct_u to each server. The servers store these values together with the nonce n in a single row of the shared database DB.

4.3 Recovery request

In this stage, the client initiates a request to recover their account after losing their user key. We assume that the client still remembers the same email string E and personal information string x used during account creation. We depict the full protocol in Figure 7.

Recovery requests begin with the client making a K-pop query to each server in OPRF mode, using inputs E and x, and receiving back outputs $\hat{E}_1, \hat{E}_2$. Note that the inputs are the same ones the client provided to the K-pop during account creation, but this time E is a private input instead of a public input. Since both K-pop modes of operation compute the same function, the outputs are the same as in account creation.

The client computes $\mathsf{id}, k_E = H(E \parallel \hat{E}_1 \parallel \hat{E}_2)$, and sends id and k_E to each server. The servers can then look up id in their shared database and retrieve the corresponding ct_r ciphertext. Both servers can decrypt this with k_E to reveal the client's r string, which consists of their recovery email address e, security questions Q, and nonce m.

To impose rate limiting on recovery requests, each server can restrict the number of K-pop queries they respond to in OPRF mode. This limit will not restrict the server's ability to process account creations, since all K-pop operations in account creation are run in pOPRF mode.

4.4 Account verification

In this stage, the recovery servers send an email to the recovery address e. This email contains a customized link that encodes the message (Q, m, ct_u, n). The servers only agree to participate in account restoration if the client clicks on this link within a short time interval.

Our protocol supports two options for account verification. First, the verification step can be done non-cryptographically by having the recovery data r revealed to the servers at the end of the recovery phase, in which case they can send an email in the clear to the recovery address e. This approach has the advantage of being easier to implement, since all other

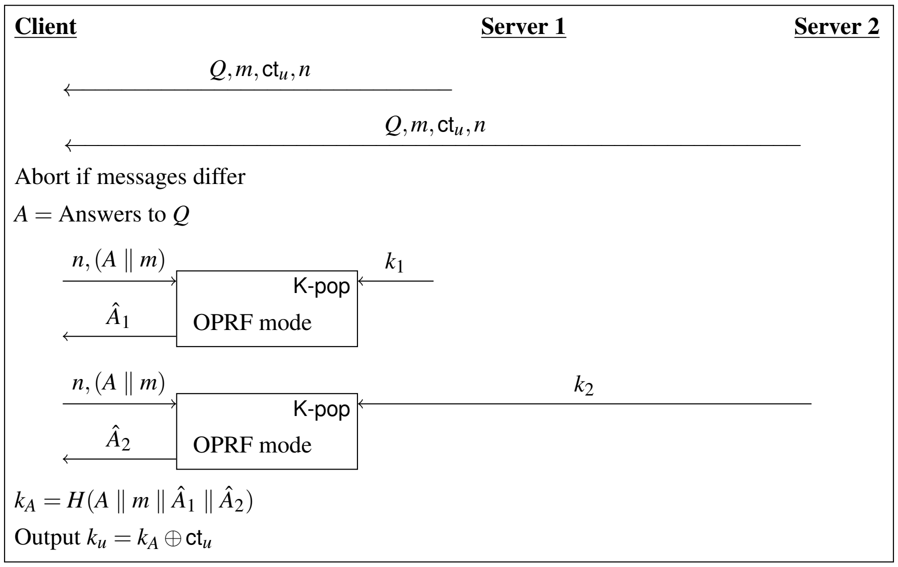


Figure 8: Account Restoration

aspects of our account recovery protocol use standard public and symmetric key primitives, but it has the disadvantage that the servers temporarily learn the contact info of clients during account recovery and must be trusted to delete this information promptly. (Still, we stress the importance here of learning at most the email addresses of clients who perform account recovery, rather than a full list of account-holders.) Second, the servers can jointly perform account verification under secure multi-party computation, in order to prevent any server from learning the client's email address. In this case, either the Oblivious TLS protocol of Abram et al. [1] or the MPCAuth protocol [64] would suffice.

In our modeling, we consider both of these options as instantiations of the UC functionality for secure message transmission $\mathcal{F}_{\mathsf{SMT}}$, as shown in Figure 1.

4.5 Account restoration

This final stage of the protocol is shown in Figure 8. Conceptually, this stage is most similar to recovery request, in that it starts with the client submitting K-pop queries to each of the servers in OPRF mode, this time with inputs $x_{\mathsf{priv}} = (A \| m)$ and $x_{\mathsf{kal}} = n$. Additionally, the client computes a slow hash function on the OPRF inputs and outputs. One difference between the stages is that account restoration is intended to provide an output to the client, not the servers. Ergo, recovering the one-time pad key k_A from the OPRF outputs is nearly the end of the protocol; all that remains is to use this ephemeral key to recover the user account key k_u.

After the client's account has been restored, it is crucial that they immediately reset their account information (e.g., the password that they have forgotten) so they maintain access in the future. While most of the reset process involves properties of the service outside of our modeling, one step that is crucial is to perform another instance of account creation in order to choose a new nonce to protect the account going forward (and optionally new security questions and answers, if desired).

4.6 Security analysis

In this work, we provide game-based and simulation-secure security analyses of our account recovery construction.

First, we provide an indistinguishability game-based security analysis in the full version of this work [53] Our game defines an adversarial model in which the attacker can statically corrupt one of two recovery servers, and adaptively corrupt any number of clients. The adversary can additionally control when honest clients create accounts and run account recovery. The adversary wins if they are able to distinguish between an uncorrupted client's real user key and a randomly sampled key. We prove that our account recovery scheme has the following security.

Theorem 2. *An adversary that sends at most q messages to honest parties has advantage at most $\frac{q}{|A|} + \varepsilon$ in the random oracle model, where ε is negligible in the security parameter, and A is a set of possible security question answers from which real clients' answers are uniformly sampled.*

This result demonstrates the effectiveness of rate-limiting against internal adversaries. Rate-limiting allows honest recovery servers to effectively impose a q value limiting the number of queries that other servers can make. The honest parties can therefore restrict the advantage of internal adversaries via the choice of q.

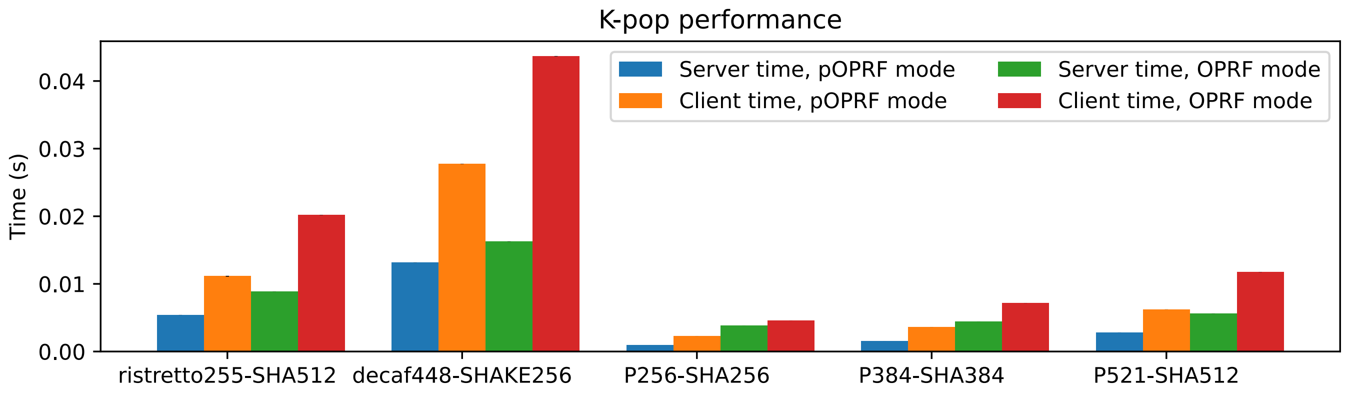


Figure 9: Performance of K-pop implementation over different choices of elliptic curve group and cryptographic hash function. Each bar represents the average of 1000 measurements.

Our game-based analysis additionally shows that our account recovery construction detects adversarial deviations from the protocol during account recovery and restoration and therefore provides security up to abort, even though our K-pop on its own does not provide verifiability. This proof is conceptually the simpler one to understand, but it restricts the adversary to a single server corruption and only protects accounts created prior to the corruption.

Second, in the full version [53], we provide a simulation-based analysis with universally composable (UC) security in the model of Canetti [18]. We write a protocol description Π_{acc} in the UC style, contribute an idealized version of our protocol as the functionality $\mathcal{F}_{\mathsf{acc}}$, and prove the following.

Theorem 3. *Protocol* Π_{acc} *UC-realizes functionality* $\mathcal{F}_{\mathsf{acc}}$ *in the random oracle model.*

This analysis provides a stronger guarantee because it does not restrict the power of the adversary: it can act maliciously from the beginning and in all phases of account creation, recovery, verification, and restoration. To prove Theorem 2, we construct a simulator **Sim** and argue the environment **Env**'s view is identically distributed whether it interacts with (a) Π_{acc} and real-world adversary **Adv** in the $\mathcal{F}_{\mathsf{K\text{-}pop}}$-hybrid world or (b) $\mathcal{F}_{\mathsf{acc}}$ and **Sim** in the ideal world. Our formal theorem statements and proofs are provided in the full version [53].

5 Implementation

We provide an open-source implementation of the K-pop construction (from §3.3) in Sage.[3] We emphasize that this codebase is distinct from Callisto's own implementation in TypeScript for use on the web.

Our code is based on an implementation [41] of RFC 9497 [30], which standardizes the pOPRF construction of Tyagi et al. [67]. Our implementation of the K-pop in pOPRF mode uses their code directly, with the only modification being the removal of zero-knowledge proofs (since our construction does not require verifiability). For the OPRF mode, we instantiate the additively homomorphic encryption scheme using an open-source implementation [29] of Paillier encryption with a key length of 2048 bits.

[3]Our K-pop implementation code is available at https://github.com/ryanjlittle/kpop-oprf.

5.1 Single-threaded experimental results

We evaluated our implementation on a 1.6GHz Intel Core i5-10210U CPU with 16GB RAM. Figure 9 shows the client-side and server-side performance of the K-pop in both modes of operation for five different choices of Diffie-Hellman group and hash function. Excluding network time, the end-to-end time of a K-pop interaction takes 3.1-41.2 ms in pOPRF mode and 8.4-60.3 ms in OPRF mode, depending on the choice of group and hash function.

Our proof-of-concept implementation is not optimized for speed, and we expect that the computation time can be reduced further. Even so, already these benchmarks lend credence to the efficiency of the account recovery protocol, which requires two separate instances of the K-pop together with a small number of hash function operations, XORs, and database operations. Since all of these server-side costs are negligible compared to the K-pop, our benchmarks show that in a 2-server account recovery system, account creation and the entire recovery procedure each run in 12.4-164.8 ms, ignoring network latency. The client-side runtime would be dominated by the (tunable) cost of the slow, memory-hard hash function like argon2 [12] or scrypt [2].

5.2 Multiprocessing experimental results

In a real-world deployment of our account recovery system, the server side work can scale to a larger number of clients

Ciphersuite	Mode	Number of cores		
		1	2	4
ristretto255-SHA512	pOPRF	6.866	3.942	2.534
	OPRF	8.829	5.364	3.671
decaf448-SHAKE256	pOPRF	13.259	8.165	5.566
	OPRF	16.357	10.209	6.779
P256-SHA256	pOPRF	1.027	0.544	0.366
	OPRF	3.823	2.289	1.416
P384-SHA384	pOPRF	1.584	0.889	0.616
	OPRF	4.478	2.648	1.639
P521-SHA512	pOPRF	2.824	1.653	1.018
	OPRF	5.547	3.368	2.326

Figure 10: Amortized evaluation time in milliseconds of K-pop server running on $P \in \{1,2,4\}$ parallel cores. Each data point is the average of 512 measurements.

by running many K-pop instances in parallel. We attempted to measure the performance in this setting by evaluating our K-pop implementation locally running on multiple cores. We simulated 512 (single-process) clients simultaneously interacting with one server that delegates the server response for each client to one of P processes run on separate parallel cores, for $P \in \{1,2,4\}$. The work is evenly split such that each core handles $512/P$ clients.

The performance results for this experiment are given in Figure 10. Experiments were run on an 4-core 1.6 GHz Intel Core i5-10210U CPU with 16GB RAM. On $P = 4$ cores, the server performance is 2-3 times faster than a single-core implementation, depending on the mode of operation and ciphersuite. The run times for $P = 1$ core are slightly higher than the results of the experiment of §5.1 due to overhead from multiprocessing.

6 Conclusion

In summary, this work designs an account recovery protocol that works under stringent functionality and privacy requirements. First, the service provider cannot know or learn the email addresses of all clients. Second, the clients must be able to follow a 'normal' account recovery workflow and have the ability to choose their own security questions—but again, without creating a visible mapping between identities and their choice of security questions. Third, a non-collusion assumption might be broken, and even in this setting the protocol must resist the adversary's ability to recover client data. Finally, our design uses the cryptographic building blocks that our partner organization, Callisto, already understood and knew how to implement and maintain, such as an oblivious pseudorandom function.

Our design is inspired by the application to a secure matching system, and it has been deployed for use in this setting. That said, at least the first three requirements from above are generic and can apply to other account-based privacy-preserving services as well. This work shows that strong security and privacy can be compatible with usability and quality-of-life features like account recovery.

Acknowledgments

R. Little and M. Varia were supported by DARPA under Agreement No. HR00112020021 and by the National Science Foundation under Grants No. 1801564, 1915763, 2209194, 2217770, and 2228610. L. Qin was supported by the NSF Graduate Research Fellowship while at Brown University (where she was during the completion of most of this work) and the Fritz Fellowship through the Initiative for Technology and Society at Georgetown University. We thank Tracy DeTomasi, Scott MacDonald, Ari Adair, and David Archer for lending their expertise in engineering and trauma-informed design through numerous conversations. We also thank the anonymous reviewers and shepherd for their detailed, thoughtful feedback on several iterations of this work.

References

[1] Damiano Abram, Ivan Damgård, Peter Scholl, and Sven Trieflinger. Oblivious TLS via multi-party computation. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 51–74. Springer, Heidelberg, May 2021.

[2] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 33–62. Springer, Heidelberg, April / May 2017.

[3] Amazon Web Services. AWS nitro system. https://aws.amazon.com/ec2/nitro/, 2023.

[4] Apple Platform Security. Secure enclave. https://support.apple.com/guide/security/secure-enclave-sec59b0b31ff/web, 2021.

[5] ARM. TrustZone technology for the ARMv8-M architecture version 2.0. https://developer.arm.com/documentation/100690/0200/ARM-TrustZone-technology, 2019.

[6] Venkat Arun, Aniket Kate, Deepak Garg, Peter Druschel, and Bobby Bhattacharjee. Finding safety in numbers with secure allegation escrows. In *NDSS 2020*. The Internet Society, February 2020.

[7] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 1–30. Springer, Heidelberg, November 2020.

[8] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM CCS 2011*, pages 433–444. ACM Press, October 2011.

[9] Carsten Baum, Tore Kasper Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In *EuroS&P*, pages 587–606. IEEE, 2020.

[10] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[11] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[12] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *EuroS&P*, pages 292–302. IEEE, 2016.

[13] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: SGX cache attacks are practical. In *WOOT*. USENIX Association, 2017.

[14] Callisto. Mission + Vision.

[15] Callisto. Designing trauma-informed and inclusive technology for survivors of sexual assault, November 2017.

[16] Callisto. Callisto Vault, June 2024.

[17] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

[18] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[19] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, page 219. IEEE Computer Society, 2004.

[20] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Heidelberg, March 2016.

[21] David Cantor, Bonnie Fisher, Susan Chibnall, Shauna Harps, Reanne Townsend, Gail Thomas, Hyunshik Lee, Vanessa Kranz, Randy Herbison, and Kristin Madden. Report on the AAU Campus Climate Survey on Sexual Assault and Misconduct, 2020.

[22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. In *EuroS&P*, pages 625–646. IEEE, 2022.

[23] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. SoK: Understanding the prevailing security vulnerabilities in TrustZone-assisted TEE systems. In *2020 IEEE Symposium on Security and Privacy*, pages 1416–1432. IEEE Computer Society Press, May 2020.

[24] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[25] Janet X. Chen, Allison McDonald, Yixin Zou, Emily Tseng, Kevin A Roundy, Acar Tamersoy, Florian Schaub, Thomas Ristenpart, and Nicola Dell. Trauma-informed computing: Towards safer technology experiences for all. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery.

[26] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.

[27] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 857–874. USENIX Association, August 2016.

[28] Poulami Das, Julia Hesse, and Anja Lehmann. DPaSE: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 682–696. ACM Press, May / June 2022.

[29] CSIRO's Data61. Python paillier library. https://github.com/data61/python-paillier, 2013.

[30] Alex Davidson, Armando Faz-Hernandez, Nick Sullivan, and Christopher A. Wood. Oblivious pseudorandom functions (OPRFs) using prime-order groups. Internet Requests for Comments, December 2023.

[31] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

[32] Marla E. Eisenberg, Katherine Lust, Michelle A. Mathiason, and Carolyn M. Porta. Sexual Assault, Sexual Orientation, and Reporting Among College Students. *Journal of Interpersonal Violence*, 36(1-2):62–82, 2021.

[33] Adam Everspaugh, Rahul Chatterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The pythia PRF service. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 547–562. USENIX Association, August 2015.

[34] Shufan Fei, Zheng Yan, Wenxiu Ding, and Haomeng Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.*, 54(6):126:1–126:36, 2022.

[35] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[36] Google Cloud. Confidential computing concepts. https://cloud.google.com/confidential-computing/confidential-vm/docs/about-cvm, 2023.

[37] Christine L. Hackman, Sarah E. Pember, Amanda H. Wilkerson, Wanda Burton, and Stuart L. Usdan. Slut-shaming and victim-blaming: a qualitative investigation of undergraduate students' perceptions of sexual violence. *Sex Education*, 17(6):697–711, November 2017.

[38] Julia Hesse, Stanislaw Jarecki, Hugo Krawczyk, and Christopher Wood. Password-authenticated TLS via OPAQUE and post-handshake authentication. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 98–127. Springer, Heidelberg, April 2023.

[39] Alejandro Hevia and Ilana Mergudich-Thal. Implementing secure reporting of sexual misconduct - revisiting WhoToo. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912 of *LNCS*, pages 341–362. Springer, Heidelberg, October 2021.

[40] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J. Rossbach, and Emmett Witchel. Telekine: Secure computing with cloud GPUs. In *NSDI*, pages 817–833. USENIX Association, 2020.

[41] Internet Engineering Task Force. Oblivious pseudorandom functions (OPRFs) using prime-order groups. https://github.com/cfrg/draft-irtf-cfrg-voprf, 2023.

[42] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014.

[43] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *EuroS&P*, pages 276–291. IEEE, 2016.

[44] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 39–58. Springer, Heidelberg, July 2017.

[45] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 456–486. Springer, Heidelberg, April / May 2018.

[46] David Kaplan, Jeremy Powell, and Tom Woller. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2020.

[47] Kimberly A. Lonsway and Sergeant Joanne Archambault. Suggested Guidelines on Language Use for Sexual Assault. Technical report, End Violence Against Women International, June 2013.

[48] Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, and Bhavish Raj Gopal. Shield: Secure allegation escrow system with stronger guarantees. In *WWW*, pages 2252–2262. ACM, 2023.

[49] Benjamin Kuykendall, Hugo Krawczyk, and Tal Rabin. Cryptography for #MeToo. *PoPETs*, 2019(3):409–429, July 2019.

[50] Russell W. F. Lai, Christoph Egger, Dominique Schröder, and Sherman S. M. Chow. Phoenix: Rebirth of a cryptographic password-hardening service. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 899–916. USENIX Association, August 2017.

[51] Dayeol Lee, Dongha Jung, Ian T. Fang, Chia che Tsai, and Raluca Ada Popa. An off-chip attack on hardware enclaves via the memory bus. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 487–504. USENIX Association, August 2020.

[52] 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 Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 717–732. USENIX Association, August 2021.

[53] Ryan Little, Lucy Qin, and Mayank Varia. Secure account recovery for a privacy-preserving web service. Cryptology ePrint Archive, Report 2024/962, 2024. https://eprint.iacr.org/2024/962.

[54] Katherine Lorenz, Anne Kirkner, and Sarah E. Ullman. A Qualitative Study Of Sexual Assault Survivors' Post-Assault Legal System Experiences. *Journal of Trauma & Dissociation*, 20(3):263–287, May 2019.

[55] Mara Dolan. These Students Are Bringing Transformative Justice to Their Campus. *The Nation*, January 2020.

[56] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2020.

[57] Microsoft Ignite. Azure confidential computing. https://learn.microsoft.com/en-us/azure/confidential-computing/, 2023.

[58] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

[59] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[60] Bijeeta Pal, Mazharul Islam, Marina Sanusi Bohuk, Nick Sullivan, Luke Valenta, Tara Whalen, Christopher A. Wood, Thomas Ristenpart, and Rahul Chatterjee. Might I get pwned: A second generation compromised credential checking service. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 1831–1848. USENIX Association, August 2022.

[61] Anjana Rajan, Lucy Qin, David W. Archer, Dan Boneh, Tancrède Lepoint, and Mayank Varia. Callisto: A cryptographic approach to detecting serial perpetrators of sexual misconduct. In *COMPASS*, pages 49:1–49:4. ACM, 2018.

[62] Ling Ren, Christopher W. Fletcher, Albert Kwon, Marten van Dijk, and Srinivas Devadas. Design and implementation of the ascend secure processor. *IEEE Trans. Dependable Secur. Comput.*, 16(2):204–216, 2019.

[63] SAMHSA's Trauma and Justice Strategic Initiative. SAMHSA's Concept of Trauma and Guidance for a Trauma-Informed Approach. Technical report, Substance Abuse and Mental Health Administration, July 2014.

[64] Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. MPCAuth: Multi-factor authentication for distributed-trust systems. In *2023 IEEE Symposium on Security and Privacy*, pages 829–847. IEEE, 2023.

[65] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1556–1571. USENIX Association, August 2019.

[66] Trusted Computing Group. Architecture overview. *Specification Revision*, 1, 2007.

[67] Nirvan Tyagi, Sofía Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A fast and simple partially oblivious PRF, with applications. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 674–705. Springer, Heidelberg, May / June 2022.

[68] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 991–1008. USENIX Association, August 2018.

[69] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on GPUs. In *OSDI*, pages 681–696. USENIX Association, 2018.

[70] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.