Post-Quantum Access Control with Application to Secure Data Retrieval

Behzad Abdolmaleki[1], Hannes Blümel[2], Giacomo Fenzi[3], Homa Khajeh[4], Stefan Köpsell[2], and Maryam Zarezadeh[2]

[1] University of Sheffield, Sheffield, UK
behzad.abdolmaleki@sheffield.ac.uk
[2] Barkhausen Institute, Dresden, Germany
{hannes.bluemel|stefan.koepsell|maryam.zarezadeh}@barkhauseninstitut.org
[3] EPFL, Lausanne, Switzerland
giacomo.fenzi@epfl.ch
[4] Independent researcher, Isfahan, Iran
homa0khajeh@gmail.com

Abstract. Servan-Schreiber et al. (S&P 2023) presented a new notion called private access control lists (PACL) for function secret sharing (FSS), where the FSS evaluators can ensure that the FSS dealer is authorized to share the given function. Their construction relies on costly non-interactive secret-shared proofs and is not secure in post-quantum setting. We give a construction of PACL from publicly verifiable secret sharing (PVSS) under short integer solution (SIS). Our construction adapts the Gentry et al's scheme (Eurocrypt 2022) for post-quantum setting based on learning with error assumption (LWE). The implementation of our PACL with different files showed that it is feasible even at different sizes, and should remain so even with large secret vectors. This construction has many applications for access control by applying FSS. We show how to apply the proposed PACL construction to secure data retrieval. We also present a scheme for secure data retrieval with access control, which might be of independent interest.

Keywords: Access control · Function secret sharing · LWE · Data retrieval· Post-Quantum security

1 Introduction

In cloud computing, users can remotely store their data in the cloud to utilise the on-demand applications and services from shared configurable computing resources. Outsourcing data to the cloud causes a loss of control over outsourced data and this loss of control is further exacerbated with the lack of the management of users' access to the data retrieval from practical perspectives. We present a novel post-quantum access control for data retrieval. In this method, only users who have permission to access a specific database record can retrieve it. Our constructions make black-box use of the underlying function secret sharing (FSS) schemes. Secret sharing is a way of splitting a value into multiple

shares such that the shares can be recombined to reveal the original value and strict subsets of shares reveal nothing about the secret value. FSS [8] has the similar requirement that the shares of f can be evaluated on a input x to obtain shares of $f(x)$ which can be combined to reveal $f(x)$. FSS has many applications for instance in the domain of cloud computing. Assume that a client wants to run a function on data stored on servers without revealing the function to the servers. The client applies FSS to share the function f with the servers and they evaluate the secret-shared function f and return the secret-shared result $f(x)$ to the client. Finally, the client locally recombines the shares to obtain $f(x)$. FSS can be used in systems with many users who may have different access rights regarding different functions. The challenge is that the servers should be sure that clients with access rights to a function f can secretly share that. On the other hand, the privacy of FSS must be maintained and the function secret shared must be hidden. In other words, the privacy in access control should be preserved.

Our goal is to develop a post-quantum secure access control scheme for FSS. Such a scheme has many applications. For example, private information retrieval (PIR) where access control prevents users from accessing records in the database that they do not have permission to access [20]. Our approach can be utilized in the cloud as an important platform for data storage and processing. The cloud centralizes essentially unlimited resources (e.g., storage capacity) and delivers elastic services to end users. However, some challenges, including concerns about data security and users' privacy, still exist. For example, the user's electronic health records (EHRs) are sensitive data and, if uploaded into the cloud, should not be disclosed to the cloud administrators and any other unauthorized users without data owners' permission. Thus data confidentiality protection against unauthorized parties and data access control are required when storing data in the cloud. As healthcare is distributed in nature, ideally EHR data should be available all the time, provided securely regardless of the application or device that forwards the request. Privacy is a serious consideration and an important challenge in healthcare, so access control is a primary mechanism to be deployed in EHR systems to protect patient privacy. Protecting patient data is of utmost importance, and the basic requirement to achieve it is through well-defined policies and access control mechanisms. Encryption is a commonly used method to preserve data confidentiality. This methodology is extremely unpractical for traditional networks, especially for wireless networks e.g., wireless sensor networks and mobile networks, seriously constrained by resources like energy and computation capability. This paper describes the application of the proposed access control for secure data retrieval in post-quantum setting.

1.1 Our Contributions

We present an innovative post-quantum access control mechanism for data retrieval, ensuring that only authorised users can access specific database records. Our design utilises function secret sharing (FSS) schemes in a black-box manner. Secret sharing involves dividing a value into multiple shares that can be

combined to reveal the original value, while any subset of the shares provides no information about the secret. Similarly, FSS allows the shares of a function f to be evaluated on an input x, producing shares of $f(x)$ that can be combined to reveal $f(x)$. This is especially useful in environments with many users who have different access rights to various functions. The primary challenge is ensuring that servers, acting as function evaluators, can verify that clients with the correct access rights can perform secret sharing. At the same time, it is essential to maintain the privacy of the FSS, ensuring that the function being shared remains hidden. This requires preserving access control privacy. Our goal is to develop a post-quantum secure access control scheme for FSS and apply it to secure data retrieval in a post-quantum context. Servan-Schreiber et al. [29] introduced a new concept called private access control lists (PACL) for FSS. This innovation enables FSS evaluators to confirm that the FSS dealer is authorised to distribute the given function. We propose a new PACL construction utilising publicly verifiable secret sharing (PVSS) based on the short integer solution (SIS) problem. Our method builds on the scheme by Gentry et al. [17] to provide post-quantum security through the learning with errors (LWE) assumption. In comparison with [29], our approach does not rely on expensive secret-shared proofs and guarantees security in a post-quantum setting. Our PACL evaluation with various files has shown its practicality across different sizes, indicating that it is likely to perform well with large secret vectors as well. We also illustrate how our PACL design can be used for secure data retrieval and present a secure data retrieval scheme with access control which may be of independent interest.

1.2 Technical Overview

FSS [8] allows a dealer to share a function f with two or more evaluators. Given secret shares of a function f, the evaluators can locally compute secret shares of $f(x)$ for any input x, without learning information about f. In this paper, we investigate the problem of access control for FSS. Given the shares of f, the evaluators can ensure that the dealer is authorised to learn $f(x)$. For a function family F and an access control list defined over the family, the evaluators receiving the shares of $f \in F$ can efficiently check that the dealer knows the access key for f. The main building block behind our method for access control over FSS is a LWE-based zero-knowledge proof-of-knowledge over secret-shared elements, which may be of independent interest. [33] presents a solution to access control for FSS. They applied FSS for anonymous communication where users privately write messages into mailboxes using a distributed point function (DPF). To prevent malicious users from corrupting mailboxes belonging to honest users, the system requires a form of access control applied over DPFs, which is enforced through a secure computation protocol. This method for access control requires a large output range and leads to large computational overhead for the evaluators and the secure computation protocol imposes extra communication between evaluators and the dealer. Recent work [29] suggests a method for private access control list for FSS. Their construction is for black-box DPF.

However, their scheme relies on costly secret-shared proofs and is not quite efficient in practice. Also the scheme is not secure in the post-quantum setting. Ji et al. [22] proposed more fine-grained policy constraints for DPF. Their scheme allows an attribute-based access control that is secure based on the decisional Bilinear Diffie-Hellman (BDH) assumption.

We present a method for PACL to retrieve data in post-quantum setting. The approach is based on PVSS and DPF to get access to the data to only authorised users. We consider a set of access and verification keys to prove the access rights for the function under the verification keys. The access key is shared between the evaluators by running secret sharing. Then, the user runs PVSS on the shared values. The evaluators audit the verification key and the proofs of the user from PVSS. Our construction adapts the PVSS scheme of Gentry et al. [17] for post-quantum setting based on LWE. We utilize the proposed PACL for secure data retrieval based on FSS. We assume that a proxy server with massive computational power and storage capacity is deployed for evaluating the input on the garbled deterministic finite automata (DFA) matrix of server S_k. Let $|\Sigma|$ denote the size of the alphabet being used, let $|Q|$ denote the number of states of DFA and let n_u denote the size of user's input. The symmetric computation is the pseudorandom generator (PRG) invocations. The asymmetric commutation is for oblivious transfers (OTs). We have taken advantage of the concept of Proxy OT (POT) [14] in our scheme to improve efficiency. The communication overhead of our scheme consists of OT communication and garbled DFA matrix transfer. In our scheme, the proxy $\mathcal{S}$ obtains the keys by running a few 1-out-of-2 OTs and XOR operations. In our construction, the parties should run Proxy OT, which is described in Appendix A.1 such that in offline phase, any server S_k, $k \in [1, n_s]$, where n_s denotes the number of servers, computes the related key for OT and sends the encoded messages to proxy. So, in the online phase any user can only send a message to proxy which is encoded format of his/her input. Also, user computes $O(n)$ PRGs to compute the matched result. For communication, the server transmits $O(|Q|out \cdot inp)$ bits, note that Q is the number of states of DFA, *out* and *inp* denote the maximum number of output and input edges in DFA graph representation, respectively. For each cell the server performs one PRG G invocation. On the other hand, the client invokes one PRG for each row of the garbled matrix.

The rest of the paper is organized as follows. Section 2 and Section 3 present the primitives and the concepts used in our constructions, respectively. Next, the proposed PACL scheme is described in Section 4 and its application for data retrieval is detailed. The evaluation results of PACL scheme are presented in Section 5.

2 Preliminaries

Notation. Throughout the paper, we denote the security parameter by λ. We use bold upper-case letters like $\mathbf{M}$ to denote matrices, and bold lower-case letters like $\mathbf{v}$ to denote vectors. We use $x \xleftarrow{\$} \mathcal{D}$ to denote a random sample from a

distribution $\mathcal{D}$. We denote a set of secret shares of x (function secret shares of f respectively) as $([x]_1, \ldots, [x]_n)$ $(([f]_1, \ldots, [f]_n)$ respectively) and $[x]_i$ (resp. $[f]_i$) as the ith secret share.

Secret sharing. Linear secret sharing [31] is parameterised by a threshold $2 \leq t \leq n$ and consists of the algorithm $\mathsf{Share}_{(\mathbb{F},t,n)}$ which generates shares of a secret value in the field $\mathbb{F}$ such that any subset of at least t shares can be combined by the algorithm Recover to reveal the secret value. It should be mentioned that no subset of less than t shares obtains any information about the secret.

Function secret sharing (FSS). FSS introduced by Boyle et al. [8], provides a way for additively secret-sharing a function f from a given function family F. FSS is a generalisation of secret sharing which targets secret sharing a function rather than secret sharing a value. A $(t, n) - \mathsf{FSS}$ scheme for a function $f : \{0,1\}^n \to \mathbb{G}$ ($\mathbb{G}$ is an Abelian group) consists of the following algorithms:

- $(k_1, \ldots, k_n) \leftarrow \mathsf{Gen}(1^\lambda, f)$: inputs a security parameter λ and function f and outputs the evaluation keys $(k_1, \ldots, k_n)$.
- $[y]_i \leftarrow \mathsf{Eval}(k_i, x)$: inputs an evaluation key k_i and $x \in \{0,1\}^n$ and outputs secret share $[y]_i$.

Distributed point function (DPF). DPF is a special case of FSS where F is the family of point functions, namely functions $f_{\alpha,\beta}$ that evaluate to β on the input α and to 0 on all other inputs.

Short Integer Solution (*SIS*). Let $q = q(\lambda)$, $n = n(\lambda)$ and $m = m(\lambda)$. We say that the SIS assumption holds if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$ the following holds: $Pr[\mathbf{A} \cdot \mathbf{z} = \mathbf{0} \,|\, \mathbf{A} \leftarrow \mathcal{R}_q^{n \times m}, \mathbf{z} \leftarrow \mathcal{A}(\mathbf{A})] \leq \mathsf{negl}(\lambda)$.

Learning with error (LWE) problem. For a prime number q, the LWE problem with secret $\mathbf{s} \in \mathbb{Z}_q^k$ is defined as follows: given polynomially random, noisy linear equations in $\mathbf{s}$, find $\mathbf{s}$. More precisely, given $(\mathbf{a}_i, \mathbf{a}_i \cdot \mathbf{s} + e_i)$, where $\mathbf{a}_i \in \mathbb{Z}_q^k$ is uniformly random, and e_i is drawn from a random distribution, the problem is to find $\mathbf{s}$. We denote m samples from the LWE distribution compactly as $(\mathbf{A}, \mathbf{As} + \mathbf{e})$, where $\mathbf{A} \in \mathbb{Z}_q^{m \times k}$ and $\mathbf{e}$ is sampled from discrete Gaussian noise, or noise sampled uniformly from $[-b, b]$ where $b \ll q$ is a bound on the magnitude.

Proxy OT . The notation of POT is defined by Naor et al. [25] in which there are three parties: a sender that holds inputs $(\mathbf{x}_0, \mathbf{x}_1)$, a chooser that holds input b, and a proxy that, at the end of the protocol, learns $\mathbf{x}_b$ and nothing else. The details of a non-interactive POT [14] are described in Appendix A.1.

Deterministic finite automaton (DFA). In computer science, a DFA is a finite-state machine that accepts or rejects a given string of symbols, by running through a state sequence uniquely determined by the string. A DFA, denoted by the five-tuple $\Gamma = (\Sigma, Q, \sigma, q_0, F_q)$, consists of 1) Σ, a finite set of input

symbols. 2) Q, a finite set of states. 3) σ, a transition function that takes as arguments a state and an input symbol and returns a state. 4) $q_0 \in Q$, a start state. 5) $F_q \subset Q$, a set of final or accepting states [21].

3 Definitions

Definition 1. *A Publicly Verifiable Secret-Sharing Scheme* (PVSS) *enables a dealer to share a secret among a committee of shareholders in a manner that allows everyone, not just the shareholders, to verify that the secret was shared correctly and it can be assured that it is recoverable. A non-interactive* PVSS *scheme allows the sender to broadcast just a single message, enabling shareholders to receive their shares and enabling everyone to verify that the sharing was done correctly [32].* The idea of PVSS is that the parties should be able to verify that the shares they receive are compatible with some shared secret. In the PVSS setting, the parties do not necessarily trust each other. Therefore, we can extend the verifiability of the shares so that any party can verify any of the shares, even if they are assigned to another party. In fact, the verification process should be carried out without any private information from the dealer or the parties. A verifier who has no such information should be capable of checking the shares. *A* PVSS *scheme consists of the following algorithms [12].*

- *Setup*
  - $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{ip})$ *outputs public parameters* pp*. The initial parameters* ip *include information about the number of parties, privacy and reconstruction thresholds and spaces of secrets and shares. The public parameters* pp *contain a description of spaces of private and public keys* SK *and* PK *and the relation* $\mathsf{R}_{Key} \subseteq \mathsf{SK} \times \mathsf{PK}$ *describing valid key pairs.*
  - $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*, generates a pair of private and public key, where* $(\mathsf{sk}, \mathsf{pk}) \in \mathsf{R}_{Key}$*.*
- *Distribution*
  - $((\mathsf{C}_i)_{i\in[n]}, \mathsf{Pf}_{\mathsf{Sh}}) \leftarrow \mathsf{Dist}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[n]}, \mathbf{s})$ *where* $\mathbf{s} \in \mathcal{S}$ *is a secret, outputs encrypted shares and a proof* $\mathsf{Pf}_{\mathsf{Sh}}$ *of sharing correctness.*
- *Distribution Verification*
  - $0/1 \leftarrow \mathsf{VerifySharing}(\mathsf{pp}, (\mathsf{pk}_i, \mathsf{C}_i)_{i\in[n]}, \mathsf{Pf}_{\mathsf{Sh}})$ *determines whether the sharing is valid or not.*
- *Reconstruction*
  - $(\mathsf{d}_i, \mathsf{Pf}_{\mathsf{Dec}_i}) \leftarrow \mathsf{DecShare}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{sk}_i, \mathsf{C}_i)$ *outputs a decrypted share* d_i *and a proof* $\mathsf{Pf}_{\mathsf{Dec}_i}$ *of correct decryption.*
  - $\mathsf{Rec}(\mathsf{pp}, \{\mathsf{d}_i : i \in \mathcal{T}\})$ *for some* $\mathcal{T} \subseteq [n]$ *outputs an element of the secret space* $\mathbf{s}' \in \mathcal{S}$ *or an error symbol* $\perp$*.*
- *Reconstruction Verification*
  - $0/1 \leftarrow \mathsf{VerifyDec}(\mathsf{pp}, \mathsf{pk}_i, \mathsf{C}_i, \mathsf{d}_i, \mathsf{Pf}_{\mathsf{Dec}_i})$ *determines whether* d_i *is a valid decryption of* C_i*.*
  - $\mathsf{Rec}(\mathsf{pp}, \{\mathsf{d}_i : i \in \mathcal{T}\})$ *for some* $\mathcal{T} \subseteq [n]\}$ *outputs an element of the secret space* $s' \in \mathcal{S}$ *or an error symbol* $\perp$*.*

Security properties [12]:

Correctness with t-reconstruction. This requirement guarantees that if all participants act honestly, all proofs will be validated, and any group of at least t participants can reconstruct the secret using their shares. Each participant must first decrypt their share, after which they collectively apply the reconstruction algorithm Rec.

Verifiability. This property ensures that if the verification procedures VerifySharing *and* VerifyDec *are passed, then the key pairs are properly constructed, the set of encrypted shares represents a correct sharing of the secret, and the shares have been accurately decrypted.*

Definition 2. *A private access control list* (PACL) *[29] applied to* FSS *is initiated between a prover and a set of s verifiers. The prover holds an access key* ak *and the function f. The verifiers hold secret-shares* $[f]_i$ *and have the access control* Λ *for the function f. The verifiers check whether or not* CheckAccess *outputs* yes*, without learning f.*

Definition 3. *Let* $f : \{0,1\}^n \to \{0,1\}^*$ *be a function. Set* $2 \leq t \leq s$ *and let* (Gen, Eval) *instantiate a* (t,s)-FSS *scheme for f. A* (t,s)-PACL *scheme consists of the algorithms* KeyGen, Prove, Audit *and* Verify *as follows.*

- $(\mathsf{ak}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, f)$. *Inputs a security parameter* λ *and a function f. The output is a pair of access and verification keys* $(\mathsf{ak}, \mathsf{vk})$.
- $(\pi_1, \ldots, \pi_s) \leftarrow \mathsf{Prove}(f, \mathsf{ak})$. *Inputs a function f and an access key* ak *and outputs proofs of s shares.*
- $\tau_i \leftarrow \mathsf{Audit}(\Lambda, [f]_i, \pi_i)$. *Inputs access control* $\Lambda = \mathsf{vk}$*, function secret share* $[f]_i$ *of f and a proof of share* π_i*. The audit token* τ_i *is the output.*
- $0/1 \leftarrow \mathsf{Verify}(\mathcal{T} = \{\tau_i \mid i \in I\})$. *Inputs a set of at least t audit tokens indexed by the set* $I \subseteq \{1, \ldots, s\}$ *and outputs 0 or 1.*

The above functionality should satisfy these requirements [29]:

- **Completeness.** *A* (t,s)-PACL *scheme is complete if for all subsets* $I \subseteq \{1, \ldots, s\}$ *with* $|I| \geq t$ *and* $\Lambda = (\mathsf{vk})$ *where* vk *is sampled according to* KeyGen *algorithm and for all secret shares* $([f]_1, \ldots, [f]_s)$ *of f sampled according to* $\mathsf{Setup}(1^\lambda, f)$ *algorithm, it holds that*
$$Pr\begin{bmatrix} (\pi_1, \ldots, \pi_s) \leftarrow \mathsf{Prove}(f, \mathsf{ak}); \\ \{\tau_i \leftarrow \mathsf{Audit}(\Lambda, [f]_i, \pi_i) \mid i \in I\} : \\ 0/1 \leftarrow \mathsf{Verify}(\mathcal{T} = \{\tau_i \mid i \in I\} = \\ \mathsf{CheckAccess}(\Lambda, f, \mathsf{ak})) \end{bmatrix} = 1$$
- **Privacy.** *For all subsets* $I \subset \{1, \ldots, s\}$, $|I| < t$, *define* $J = \{1, \ldots, s\} \backslash I$ *and* $D_{I,J}$ *to be the distribution over* $\{(\pi_i, \tau_i^*) \mid i \in I\} \cup \{\tau_j \mid j \in J\}$ *where each* π_i *is sampled according to* $\mathsf{Prove}(f, \mathsf{ak})$*, each* τ_i^* *is sampled arbitrary, and* $\tau_j \leftarrow \mathsf{Audit}(\Lambda, [f]_j, \pi_j)$ *for all* $j \in J$*. A* (t,s)-PACL *is private if there exists an efficient simulator* Sim *such that* $D_{I,J} \approx_c \mathsf{Sim}(1^\lambda, I, \{\tau_i^* \mid i \in I\})$.
- **Soundness.** *There exists a negligible function* negl *such that for all efficient algorithms* $\mathcal{A}$ *and subsets* $I \subseteq \{1, \ldots, s\}$ *where* $|I| \geq t$, $\Pr[\mathsf{soundness}_{(\mathsf{PACL},\mathcal{A},\mathsf{I})}(\lambda) = \mathsf{yes}] \leq \mathsf{negl}(\lambda)$*, where* $\mathsf{soundness}_{(\mathsf{PACL},\mathcal{A},\mathsf{I})}(\lambda)$ *is defined in Fig. 1. In more*

detail, no efficient adversary $\mathcal{A}$ without knowledge of an access key for f_β, can forge a proof π that verifies with non-negligible probability.

- Oracle $\mathsf{GetShareKey}(j)$:
  1. $T = T \cup \{j\}$, return $\mathbf{s}_j$
- Game $\mathsf{soundness}_{(\mathsf{PACL},\mathcal{A},\mathsf{I})}(\lambda)$:
  1. $(\mathsf{ak}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, f_\beta)$
  2. $\Lambda = \mathsf{vk}, T = \{\}$
  3. $([f_\beta]_j, (\pi_\beta)_j) \leftarrow \mathcal{A}^{\mathsf{GetShareKey}}(1^\lambda, \Lambda)$
  4. $f_\beta \leftarrow \mathsf{Recover}([f_\beta]_i)$
  5. for $i \in I : \tau_i \leftarrow \mathsf{Audit}(\Lambda, [f_\beta]_i, (\pi_\beta)_i)$

Fig. 1. PACL soundness game [29]

4 Post-Quantum Access Control scheme

In this section, we propose (the first) post-quantum PACL based on SIS assumption for FSS. Our model and definitions for post-quantum PACL for FSS are derived from [30]. This scheme is a requirement for several existing applications of access control for FSS [16, 26, 30] and demands a stringent set of efficiency requirements that closely align with the goals of FSS. To achieve this goal, we initially built a non-interactive PVSS scheme from the LWE assumption, which is supposed to be the first post-quantum PVSS. Subsequently, using PVSS, we construct our post-quantum PACL scheme. In Section 4.1, we first revisit the PVSS construction by Gentry *et al.* [17], then present our non-interactive post-quantum PVSS, constructed upon Gentry et al.'s work, leveraging the LWE assumption. Then, we introduce our post-quantum PACL in Section 4.2.

4.1 Post-Quantum Non-Interactive PVSS

To build our post-quantum non-interactive PVSS, we initially consider the PVSS scheme by Gentry *et al.* [17], which relies on both the Discrete Logarithm (DL) and LWE assumptions. We then transition it into a construction solely based on the LWE assumption.

Gentry et al. [17] presented a non-interactive PVSS scheme, which we refer to as GHL-PVSS, that leverages the Peikert-Vaikuntanathan-Waters (PVW) encryption scheme in a multi-receiver setting. We will now recall a brief description of the PVW encryption scheme as outlined in [28]. Let $\mathbf{A}$ be a public random matrix. Each party i sets $\mathbf{B}_i = \mathbf{S}_i \cdot \mathbf{A} + \mathbf{e}_i$ as its public key, where the matrix $\mathbf{S}_i$ is secret. The parties' public keys are collected into a matrix $\mathbf{B}$. The public key of the PVSS system is $\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\end{bmatrix}$. The encryption of message $\mathbf{m} \in \mathbb{Z}_q^k$ is $\begin{bmatrix}\mathbf{A}\\ \mathbf{B}\end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix}\mathbf{e}_1\\ \mathbf{e}_2\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{m}\end{bmatrix} = \begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix}$ where $\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2$ are small vectors. A member from the committee will apply PVSS scheme to encrypt k re-shares of this share to the next committee with k members, $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{Z}_q^k, \mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{k\times m}, \mathbf{r} \in \mathbb{Z}_q^m$. GHL-PVSS uses Bulletproofs [11] to get compact proofs of correct encryption/decryption

of shares. In particular, bulletproofs enable efficient exact range proofs, which enable to show that the vectors used within the key generation and encryption process are indeed short. Gentry *et al.* left the construction of a post-quantum PVSS as an open problem, inviting exploration into potential replacements for a variant of bulletproofs based on lattice problems [15].

Theorem 1 (GHL-PVSS). *Let bulletproofs be a ZK argument of knowledge over a DL-group. Let* PVW *an encryption scheme based on the LWE assumption. Then* GHL-PVSS *is a non-interactive* PVSS.

Our Proposed Non-Interactive PVSS from LWE. We adopt the concept of GHL-PVSS while incorporating a post-quantum proof system for verifying the correctness of the encryption and decryption of the associated shares in the PVSS scheme. Specifically, we employ a ZK proof system from the STARK [5] family of protocols, known as the RISC Zero protocol [10] (denoted by RISC0), enabling a shareholder to prove that incoming and outgoing PVW ciphertexts accurately encrypt re-shares associated with its share. For the sake of readability, we present our proposed post-quantum non-interactive PVSS, so-called GHL^+ as follows:

- $\mathsf{GHL}^+.\mathsf{Setup}(1^\lambda, \mathsf{ip})$:
  - $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{ip})$ outputs public parameters pp. Specify a set of pairwise distinct points $\{\alpha_1, \ldots, \alpha_m, \gamma_1, \ldots, \gamma_n\} \subset \mathbb{Z}_q$. Let $\mathsf{pp} = (q, k, t, n, (\alpha_j)_{j\in[m]}, , (\gamma_i)_{i\in[n]})$. The initial parameters ip contain information about the number of parties, privacy and reconstruction thresholds and spaces of secrets and shares.
  - $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{PVW.KeyGen}(\mathsf{pp}, \mathsf{id})$, where sk and pk are the secret and public key of PVW encryption scheme.
  - $\mathsf{pp}_{\mathsf{RISC0}} \leftarrow \mathsf{RISC0.KeyGen}(\mathsf{pp}, \mathsf{id})$, where $\mathsf{pp}_{\mathsf{RISC0}}$ is the public parameter of the RISC zero protocol.
- $\mathsf{GHL}^+.\mathsf{Dist}(\mathsf{sk}_i, \mathbf{s})$:
  - $([\mathbf{s}]_1, \cdots, [\mathbf{s}]_n) \leftarrow \mathsf{Secret.Share}(\mathbf{s})$ where $\mathbf{s} \in \mathcal{S}$ is a secret.
  - $(\mathsf{C}_i)_{i\in[n]} \leftarrow \mathsf{PVW.Enc}(\mathsf{sk}_i, [\mathbf{s}]_i)$ where C_i is a PVW ciphertext of the shares $[\mathbf{s}]_i$.
  - $(\pi_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]} \leftarrow \mathsf{RISC0.Prover}(\mathsf{pp}_{\mathsf{RISC0}}, [\mathbf{s}]_i, \mathbf{s}, \mathsf{C}_i)$ where $\pi_{\mathsf{i},\mathsf{RISC0}}$ is a RISC zero protocol proof of sharing correctness.
- $\mathsf{GHL}^+.\mathsf{VerifySharing}(\mathsf{pp}, \mathsf{pp}_{\mathsf{RISC0}}, (\mathsf{pk}_i, \mathsf{C}_i, \pi_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]})$:
  - $0/1 \leftarrow \mathsf{RISC0.Verify}(\mathsf{pp}, \mathsf{pp}_{\mathsf{RISC0}}, (\mathsf{pk}_i, \mathsf{C}_i, \pi_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]})$ (checks C_i are well-formed and indeed encrypt the shares)
- $\mathsf{GHL}^+.\mathsf{DecShare}(\mathsf{pp}, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{sk}_i, \mathsf{C}_i)$:
  - $\mathsf{d}_i \leftarrow \mathsf{PVW.Dec}(\mathsf{sk}_i, \mathsf{C}_i)$ outputs a decrypted share.
  - $\pi_{\mathsf{Dec_i},\mathsf{RISC0}} \leftarrow \mathsf{RISC0.Prover}(\mathsf{pp}_{\mathsf{RISC0}}, \mathsf{d}_i, \mathsf{C}_i)$: outputs a proof of correct decryption.
  - $\mathsf{Rec}(\mathsf{pp}, \{\mathsf{d}_i : i \in \mathcal{T}\})$: for some $\mathcal{T} \subseteq [n]$ outputs an element of the secret space $\mathbf{s}' \in \mathcal{S}$ or an error $\perp$.
- $\mathsf{GHL}^+.\mathsf{VerifyDec}(\mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{C}_i, \mathsf{d}_i, \pi_{\mathsf{Dec_i},\mathsf{RISC0}})$:
  - $0/1 \leftarrow \mathsf{RISC0.Verfiy}(\mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{C}_i, \mathsf{d}_i, \pi_{\mathsf{Dec_i},\mathsf{RISC0}})$ as a verdict on whether d_i a valid decryption of C_i.

As we noted before, the bulletproof proof system, as used in [17], relies on the hardness of the DL assumptions, and is thus not secure against a quantum adversary. While there exist lattice variants of bulletproofs [7], the consensus is that they do not lead to concretely efficient proof systems. Another line of work that achieves post-quantum security is that initiated by [6], which is secure in the quantum random oracle model [13]. Our proof system of choice was the RISC Zero proof system, which is based on the STARK [5] family of protocols. We did consider recent linear-time proof systems such as Brakedown [19] and Orion [34], but ultimately decided for a proof system with better proof sizes and whose tooling would facilitate our implementation.

Theorem 2. *Let* Π_{RISC0} *be the* RISC *Zero protocol. Let* PVW *be an encryption scheme based on LWE assumption. Then, our proposed* GHL^+-PVSS *is a non-interactive* PVSS.

Proof. Let λ be the security parameter and $q > 2^\lambda$ prime. Let m (size of the secret), t (privacy threshold) and n (number of parties). The scheme consists of the following algorithms.

We show that the GHL^+ scheme guarantees the security properties from Definition 1. We prove that GHL^+ is a correct PVSS with $t+m$-reconstruction. If all parties in $[n]$ honestly create keys, then for all i we have pk_i for some sk_i. If the dealer is honest the values C_i are computed correctly where $p(X)$ is of degree $t+m-1$ and $p(\alpha_j) = s_j$ are the coordinates of the secret, a set of pairwise distinct points $\{\alpha_1, \ldots, \alpha_m\}$. Then clearly DecShare, when honestly applied to C_i, computes d_i. Therefore given a set $\mathcal{T}$ of parties of size at least $t+m$ who correctly decrypted their shares, and a subset $\mathcal{T}'$ of exactly $t+m$ parties, we have that the reconstructed values are $\Pi_{i\in\mathcal{T}'} p(\gamma_i) L_i(\alpha_j) = p(\alpha_j) = s_j$, for each $j \in [m]$, by Lagrange interpolation, where L_i is a polynomial degree at most $n-1$. On the other hand, if RISC0 is a proof with soundness error negligible in λ then GHL^+ has verifiability of sharing distribution. Also, if RISC0 with soundness error negligible in λ then GHL^+ has verifiability of share decryption. The proof is trivial as the statements proved by the ZK proofs exactly guarantee correct sharing distribution and share decryption, respectively.

4.2 The Proposed Post-Quantum PACL

Now, we introduce our construction of post-quantum PACL tailored for DPFs. Our DPF-PACL construction for the matching predicate can be viewed as an extension of the approach employed by Newman *et al.* [26] and Schreiber *et al.* [30], based on the LWE and SIS assumptions, which are presumed to be post-quantum secure.

Construction. In Fig. 2, we present the proposed construction for a DPF-PACL with CheckAccess instantiated for the match predicate described in Section 2. Our construction relies mainly on the following primitives: (i) a post-quantum non-interactive verifiable secret sharing (GHL^+-PVSS construction in Section 4.1) and (ii) a DPFs scheme (e.g., DPFs by Gilboa-Ishai [18]). Roughly speaking,

the construction leverages the DPFs to locally select shares of the i-th verification key in Λ. Two key facts enable this process: (1) all verifiers possess $\Lambda = \mathbf{As}$, and (2) the FSS key k_i, encoding a DPF. The versifiers first evaluate the $[y_x]_i \leftarrow \mathsf{DPF.Eval}(k_{x,i})$, $x \in [1, z], i \in [1, n]$, and then compute the product as $\mathbf{B}_i \leftarrow \Sigma_{x=1}^{z}(\mathbf{As}_x) \cdot [y_x]_i$ (Audit phase). To verify knowledge of $\mathbf{s}$, utilizing the GHL$^+$-PVSS, the prover distributes additive secret shares of $\mathsf{ak} = -\mathbf{s}$ to the verifiers (described in the Prove phase). In our scheme, the secret value $\mathbf{s}$ is determined based on the number of files to which the prover has access. In more details, if we consider z files and the prove has only access to files f_2, f_7, f_{12}, he/she gets the sum of $\mathbf{s}_2$, $\mathbf{s}_7$ and $\mathbf{s}_{12}$, i.e., $\mathsf{ak} = -(\mathbf{s}_2 + \mathbf{s}_7 + \mathbf{s}_{12})$ but the verification key is vk. Each verifier computes $\tau_i := \mathbf{b}_i + \mathbf{A}[\mathbf{s}]_i$ using the Audit algorithm and reveals τ_i in a verifiable manner (using the GHL$^+$.DecShare algorithm of the PVSS) to all other verifiers. All verifiers proceed to check that $|\Sigma_i \tau_i| \stackrel{?}{\approx} \mathbf{0}$ (described in the Verify phase).

Lemma 1. *If adversary $\mathcal{A}$ can win the* PACL *soundness game described in Fig. 1 for our post-quantum* PACL *scheme with non-negligible probability $\gamma(\lambda)$ for a function f^*_β which is not sampled from $\mathcal{F}_{\mathsf{DPF}}$ and proof π^*, then there exists an adversary $\mathcal{A}'$ that wins the* PACL *soundness game with probability $\gamma(\lambda)$, such that $\mathcal{A}'$ outputs $f_\beta \in \mathcal{F}_{\mathsf{DPF}}$ and π.*

Proof. A property of our DPF-PACL scheme is the ability to aggregate proofs across different DPFs and access control lists. In more details, for any integer k that is polynomial in the security parameter λ and family of point functions F it holds the following property. Let Λ be an ACL for the family F and let $f_1, \ldots, f_k \in F$ have associated access keys $\mathbf{s}_1, \ldots, \mathbf{s}_k \in \Lambda$, then $\mathbf{s}' := \sum_{i=1}^{k} \mathsf{ak}_i$ is an access key for $f'(x) := \sum_{i=1}^{k} f_i(x)$. Consider an efficient $\mathcal{A}$ that outputs a not point function f^*_β and π^*. So, it holds that $f^*_\beta = \sum_{j \in S} c_j f_j$ where $Y \subseteq \{0, 1, \ldots, z\}$, each f_j is a point function, and c_j's are arbitrary non-zero scales in $\mathbb{Z}_p$. We construct an adversary $\mathcal{A}'$ that breaks the PACL soundness game with a point function f_β as follows. First, run $\mathcal{A}$ to get function f^*_β. Then compute $f_\beta := f^*_\beta - \sum_{j \in Y, j \neq \gamma} c_j \alpha_j$, note that $\mathcal{A}'$ is allowed to query for all α_j provided $j \neq \gamma$. Finally, output f_β and π. It must hold that $\gamma \in Y$, otherwise $\mathcal{A}$ does not succeed since it queried all the necessary access keys ak_j for $j \in Y$. Based on the aggregation property of our construction, it follows that f_β is a point function and π is a valid access proof for f_β. Hence, $\mathcal{A}'$ succeeds with the same probability.

Theorem 3. *Assume* GHL$^+$-PVSS *is a non-interactive publicly verifiable secret sharing from LWE assumption (in Section 4.1) . Let* DPF *be a distributed point function, let $\mathbb{Z}_q$ denote the ring of integers modulo q in which the SIS problem is assumed to be computationally hard. Then, the proposed* PACL *in Fig. 2 satisfies the completeness, efficiency, soundness, and privacy properties of Definition 3 as defined in Section 2.*

Proof. The proof is as follows:

- **Completeness**: Suppose that we have z files. If the requirements are satisfied PACL can be passed as follows. In Audit step, $\mathbf{B}_i$ is computed as $\mathbf{B}_i \leftarrow \Sigma_{x=1}^{z}(\mathbf{A}\mathbf{s}_x) \cdot [y_x]_i = \mathbf{A}\Sigma_{x=1}^{z}\mathbf{s}_x \cdot [y_x]_i$. Also, the versifiers compute $\tau_i = \mathbf{B}_i + \mathbf{A}[s_j]_i$. Verify phase will be done by computing $\Sigma\tau_i = \Sigma_{i=1}\mathbf{A}\Sigma_{x=1}^{z}\mathbf{s}_x \cdot [y_x]_i + \Sigma_{i=1}\mathbf{A}[s_j]_i = \mathbf{A}\Sigma_{i=1}\Sigma_{x=1}^{z}\mathbf{s}_x \cdot [y_x]_i + \Sigma_{i=1}\mathbf{A}[s_j]_i = \mathbf{A}\Sigma_{x=1}^{z}\mathbf{s}_x\Sigma_{i=1} \cdot [y_x]_i + \Sigma_{i=1}\mathbf{A}[s_j] = \mathbf{A}\Sigma_{x=1}^{z}\mathbf{s}_x + \Sigma_{i=1}\mathbf{A}[s_j]_i = \mathbf{A}\mathbf{s} - \mathbf{A}\mathbf{s}$.
- **Privacy**: The simulator Sim on input $(1^\lambda, I, \{\tau_i^* | i \in I\})$ for any subset of $t-1$ verifiers acts as follows. Sim sets $J = \{1, \ldots, s\} \backslash I$. Next, Sim shares the vector $\mathbf{0}$,$([\mathbf{0}]_1, \ldots, [\mathbf{0}]_s) \leftarrow \mathsf{Share}_{\mathbb{F}_p,t,s}(\mathbf{0})$, and computes the proofs of the shares by $(\mathsf{C}_i, \mathsf{Pf}_{\mathsf{Sh}_i})_{i\in[n]} \leftarrow \mathsf{GHL}^+.\mathsf{Dist}(\mathsf{pp}, (\mathsf{pk}_i)_{i\in[n]}, ([\mathbf{0}]_i)_{i\in[n]})$. Then, Sim sets $\tau_k = \mathbf{A}[0]_k$ for all $k \in I \cup J$ and outputs $\{(\mathsf{Pf}_{\mathsf{Sh}_i}, \tau_i) \mid i \in I\} \cup \{\tau_j \mid j \in J\}$. In the real world, the audit tokens are computed using the output of the DPF and they are computationally-hiding additive secret shares of $\mathbf{A}[\mathbf{0}]$ while in the ideal execution Sim outputs information-theoretically hiding additive shares. So, the difference of two real and ideal executions is this point. If an efficient distinguisher can distinguish two views, that means that the FSS scheme is not secure which is a contradiction.
- **Soundness**: Suppose there exists an efficient prover $\mathcal{A}$ and a non-negligible function γ so that for all $I \subseteq \{1, \ldots, s\}$, $|I| \geq t$, $\Pr\big[\mathsf{soundness}_{(\mathsf{PACL},\mathcal{A},\mathsf{I})}(\lambda)\big] = \mathsf{yes} \geq \gamma(\lambda)$. Based on Lemma 1, suppose that f_β is output by $\mathcal{A}$ in Fig. 1. Then adversary $\mathcal{A}'$ proceeds as follows. On input $\mathbf{y} = \mathbf{A}\mathbf{s}$, selects $\beta' \xleftarrow{\$} \{1, \ldots, n\}$ and $(\alpha_1, \ldots, \alpha_n) \xleftarrow{\$} (\mathbb{Z}_q^m)^n$ as shares and computes $\mathbf{A}\alpha_1, \ldots, \mathbf{A}\alpha_n$ but replace $\mathbf{A}\alpha_{\beta'}$ with $\mathbf{y}$. In the soundness game, the adversary $\mathcal{A}'$ runs $\mathcal{A}^{\mathsf{GetShareKey}}(1^\lambda, \Lambda)$ and responses to each query with α_j such that if $j \neq \beta'$ aborts, and adds j to T. If $\beta = \beta'$ outputs $\mathbf{s}$ and sets $\mathsf{ak} = -\mathbf{s}$, otherwise outputs fail. The probability that $\beta = \beta'$ is $\frac{1}{n}$ and so $\mathcal{A}'$ succeeds with probability at least $\frac{1}{n} \cdot \gamma(\lambda)$ which is non-negligible. Hence $\mathcal{A}'$ successfully can solve the SIS in $\mathbb{Z}_q$ which is a contradiction.

4.3 Secure Data Retrieval with Access Control

We apply our proposed $\mathsf{DPF\text{-}PACL}$ for secure data retrieval based on FSS with access control. The scheme is described in Fig. 3 and the related protocol is denoted by $\pi_{\mathsf{DR-ACL}}$. The scheme $\pi_{\mathsf{DR-ACL}}$ is described in Fig. 4. Suppose that user owns keyword w_u. n_s servers $\mathsf{S}_1, \ldots, \mathsf{S}_{n_s}$ and data base $\mathsf{DB} = \{(\mathsf{w}_1, \mathsf{d}_1), (\mathsf{w}_2, \mathsf{d}_2), \ldots, (\mathsf{w}_{n_s}, \mathsf{d}_{n_s})\}$, $\mathsf{d}_i \in \{0,1\}^m$, $i \in [1, n_s]$.

Figure 5 describes the garbled DFA and the steps are as follows.

- *Constructing the matrix.* The server $\mathsf{S_k}$ computes a DFA using its keyword $\mathsf{w_k}$ and constructs a matrix $\mathbf{M}$. For a state q_i we use the notion of a character group [27] to denote the set of characters in Σ that for them there exists a transition from q_i to the same destination state q_j. In other words, c and d are in the same character group if $\sigma(q_i, c) = \sigma(q_i, d) = q_j$. The set of character groups of a state q_i is denoted by $\mathbf{ch}_i$ and ch_{max} is the number of unique character groups.

– Setup:
  - $(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pp})_{i\in[n]} \leftarrow \mathsf{GHL}^+.\mathsf{Setup}(1^\lambda, \mathsf{ip})$
  - Return $(k_{x,1} \ldots, k_{x,n}, (\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pp})_{i\in[n]})$

– $\mathsf{KeyGen}(1^\lambda, f_x) : x \in [1, z]$
  - $\mathbf{s}_x \xleftarrow{\$} \mathbb{Z}_q^m$, $\mathbf{A} \in \mathbb{Z}_q^{k\times m}$
  - $\mathsf{ak} = -(\mathbf{s}_1, \ldots, \mathbf{s}_z)$, access to z files $f_1, \ldots, f_z$, $\mathsf{vk}_x = \mathbf{A}\mathbf{s}_x$
  - Return $(\mathsf{ak}, \mathsf{vk}_1, \ldots, \mathsf{vk}_z)$

– $\mathsf{Prove}(f, \mathsf{ak})$:
  - $([s_j]_1, \ldots, [s_j]_n) \leftarrow \mathsf{Share}_{(\mathbb{F}_p,t,n)}(-s_j)$, $j \in [1, m]$, $\mathbf{s} = \Sigma_{x\in[z]}\mathbf{s}_x = (s_1, \ldots, s_m)$
  - $(\mathsf{C}_i^j, \pi^j_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]} \leftarrow \mathsf{GHL}^+.\mathsf{Dist}(\mathsf{pp}, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{ak}, \mathsf{pk}_i, [s_j]_i)$
  - Return $(\mathsf{C}_i^j, \pi^j_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]}$

– $\mathsf{Audit}(\Lambda, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pp}, ([f_x]_i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{C}_i^j, \pi^j_{\mathsf{i},\mathsf{RISC0}})_{i\in[n]})$:
  - $0/1 \leftarrow \mathsf{GHL}^+.\mathsf{VerifySharing}(\mathsf{pp}, \mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{C}_i^j, \pi^j_{\mathsf{i},\mathsf{RISC0}})$
  - parse $\Lambda = (\mathbf{A}\mathbf{s}_1, \ldots, \mathbf{A}\mathbf{s}_z)$ and $[f_x]_i = k_{x,i}, x \in [1, z], i \in [1, n]$
  - $[y_x]_i \leftarrow \mathsf{DPF}.\mathsf{Eval}(k_{x,i})$, $x \in [1, z], i \in [1, n]$
  - $\mathbf{B}_i \leftarrow \Sigma_{x=1}^z(\mathbf{A}\mathbf{s}_x) \cdot [y_x]_i$
  - $([s_j]_i, \pi^j_{\mathsf{Dec_i},\mathsf{RISC0}}) \leftarrow \mathsf{GHL}^+.\mathsf{DecShare}(\mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{sk}_i, \mathsf{C}_i^j)$
  - $\tau_i = \mathbf{B}_i + \mathbf{A}[s_j]_i$
  - Return $(\tau_i, \mathsf{d}_i^j := [s_j]_i, \pi^j_{\mathsf{Dec_i},\mathsf{RISC0}})$

– $\mathsf{Verify}(\mathcal{T}, \mathsf{pp}_{\mathsf{RISC0}}, (\mathsf{pk}_i, \mathsf{C}_i^j, \mathsf{d}_i^j, \pi^j_{\mathsf{Dec_i},\mathsf{RISC0}})_{i\in[n]})$:
  - $0/1 \leftarrow \mathsf{GHL}^+.\mathsf{VerifyDec}(\mathsf{pp}_{\mathsf{RISC0}}, \mathsf{pk}_i, \mathsf{C}_i^j, \mathsf{d}_i^j, \pi^j_{\mathsf{Dec_i},\mathsf{RISC0}})$
  - parse $\mathcal{T} := \{\tau_1, \ldots, \tau_t\}$
  - $\Sigma_i\tau_i \overset{?}{\approx} 0$
  - Return 1 if the both checking output 1. Otherwise 0.

Fig. 2. Post-Quantum DPF-PACL for Match Predicates.

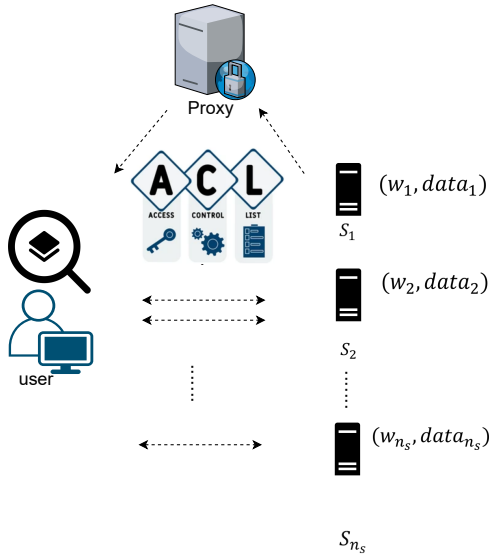

Fig. 3. Scheme overview

- Any server S_k, $k \in [1, n_s]$ constructs a DFA on its keyword w_k, the algorithm is described in Fig. 5, and sends a garbled matrix resulted from DFA protocol to the proxy
- The servers collectively enforce access control by running the proposed DPF-PACL described in Fig. 2. If this step is successful:
  - the proxy evaluates the garbled on the input of the user and sends the result to user, the protocol is described in Fig. 6
  - Next, user using the scores obtained from the result can retrieve top relevant data the servers $\mathsf{S}_1, \ldots, \mathsf{S}_{n_s}$, the algorithm is described in Fig. 7

Fig. 4. Data Retrieval with Access Control, $\pi_{\mathsf{DR-ACL}}$

- *Generating random pads.* The server $\mathsf{S_k}$ generates random pads $\mathbf{r_0}^j$, $j \in [1, |\mathbf{ch_0}|]$ for the start state q_0. For other state q_j, inp random pads are generated, denoted by $\mathbf{pad_i}^v$, $v = 1, \ldots, |\mathbf{ps}_i|$. $\mathbf{ps}_i$ denotes the states for which there exists a transition from them to the state q_i. Suppose that inp is the maximum number of previous states for any state.
- *Generating random keys.* The parties agree on a Proxy $\mathsf{OT}_1^{|\Sigma|}$ denoted by $\mathcal{F}_{\mathsf{POT}}(\mathsf{K}'_1, \ldots, \mathsf{K}'_{|\Sigma|})$ where $\mathsf{K}'_1, \ldots, \mathsf{K}'_{|\Sigma|}$ are the keys which are constructed by each server. The server $\mathsf{S_k}$ assigns the keys to each character group. Then, for any character $y_i \in \Sigma$, $\mathsf{S_k}$ concatenates the keys of character groups of which y_i is a member of them.
- *Computing the garbled matrix* The server $\mathsf{S_k}$ constructs the garbled DFA matrix through the following steps.
  - $\mathsf{S_k}$ generates a random vector for the start state and a matrix for $|Q|$ states from randomly chosen pads where $\xleftarrow{\$}$ denotes the assignment of a randomly selected element. Then $\mathsf{S_k}$ generates a permuted states for a set $\{0, \ldots, |Q|\}$ using the permutation function $\mathsf{PER}(s) \rightarrow s'$ and inputs a state $s \in \{1, \ldots, |Q|\}$ outputs an unique state $s' \in \{1, \ldots, |Q|\}$.
  - Next $\mathsf{S_k}$ garbles the DFA matrix according to the following algorithm. K_j's are the random keys which are used in OTs. Note that $\mathsf{PRG} : \{0,1\}^{\kappa} \rightarrow \{0,1\}^{\kappa'}$ is a secure pseudorandom generator (PRG); note that $\kappa' = 2\kappa + 2\log|Q|$ and out is the maximum number of outgoing edges for a state of DFA. For any state, the next state is denoted by s and the permuted next state is obtained by PER. $\mathsf{perm}(i, j) \rightarrow s'$ takes the state i, j as the index of j-th character group as inputs and outputs a vector which is a random permutation of values $(1, \ldots, inp)$. Note that, PER and perm are used to hide the structure of the DFA on the keyword. Any element of the garbled matrix is the resultant from xoring the concatenation of t', $\mathsf{PER}(s)$, random pad and κ zeros and the corresponding keys.
- *Evaluating the matrix.* In this phase, server $\mathsf{S_k}$ sends the random value $r^k = \Sigma\, r_i^k$ to the user and the garbled matrix to the proxy server $\mathcal{S}$. Also n Proxy OTs are run between $\mathcal{S}$, $\mathsf{S_i}$, and user to evaluate the user's input $\mathsf{w_u}$, on the garbled matrix. This phase is described in Fig. 6. Finally, the server $\mathcal{S}$ computes the score_k and send it to the user.

– *Retrieve data.* The user computes the scores for any document. Then, user retrieves the document with the maximum value of scores and sends a request to the servers $\mathsf{S}_1, \ldots, \mathsf{S}_{\mathsf{n_s}}$ as follows. Assume that the client would like to retrieve $\mathsf{data}_{\mathsf{i_c}}$. The client uses a DPF $f : \{0,1\}^{\kappa} \to \{\mathbb{Z}_2\}^{l}$ into $f_1, \ldots, f_{n_s}$ and sends each key to the corresponding server. Server $\mathsf{S_i}$ sends back the $\mathsf{ans_i} = \bigoplus_{j=1}^{n_s} \mathsf{data}_{\mathsf{i_j}} f_i(i_j)$, $i = 1, 2$. So $\bigoplus_{j=1}^{n_s} \mathsf{ans_i}$ outputs $\mathsf{data}_{\mathsf{i_c}}$.

This algorithm is run by the user
- $\mathsf{score}_k = r^k - (\mathbf{score}_k)_{10}$, where $(\mathbf{score}_k)_{10}$ denotes the decimal representation of $\mathbf{score}_k$
- $\mathsf{index} = \mathsf{inx} \mid \mathsf{score}_{\mathsf{inx}} = \min_{i=1}^{n_s}(\mathsf{score}_i)$
- DPF $f_{\mathsf{K}} : \{0,1\}^{\kappa} \to \{\mathbb{Z}_2\}^{l}$
- $[f]_i \leftarrow k_i$, $[y]_i \leftarrow \mathsf{DPF.Eval}(k_i)$
- $\mathsf{ans}_i = \mathsf{data}_i[y]_i$, $i \in [1, n_s]$
- $\mathsf{data}_{\mathsf{index}} \leftarrow \bigoplus_{j=1}^{n_s} \mathsf{ans}_i$

Fig. 7. Data Retrieval

Theorem 4. *In the* OT*-hybrid model, given a computationally secure* PRG*, the scheme* $\pi_{\mathsf{DR-ACL}}$ *is secure in the presence of semi-honest setting.*

The proof of Theorem 4 is presented in Appendix A.2.

5 Evaluation

We have employed a ZK proof system from the STARK (the acronym means scalable transparent argument of knowledge). Scalable means that proving time scales quasilinearly in $|w|$ (the size of witness) and simultaneously, verification times scales poly-logarithmically in $|w|$. In the Prove phase, the computations are related to $O(k^2)$ matrix multiplications to generate the ciphertexts in PVW. Then, in the Audit step $O(nk)$ matrix multiplication is done to decrypt the ciphertexts. The complexity of DPF is linear in the DPF's domain size. The concrete communication overheads on the prover and the verifiers are related to DPF and the exchanged messages in PVW. $n\lambda$ bits will be exchanged in PVW. Independence of the DPF construction, assume that $|r|$ denote the size of a DPF with range r, $n \log(r) \cdot (\lambda + 2)$ bits [9] are communication overhead.

Table 1 and Table 2 describe the evaluation of the PACL scheme described in Section 4.2. Our implementation can be found at [2]. We used C++ for our main protocol. For the ZK proofs we used the Risc0 framework [3] which is implemented in Rust while implementing the ZK-proofs newer frameworks claim to be faster than Risc0 where is released. To improve the times for calculation it could be interesting to substitute the frameworks like [4], which could improve the computation of the proofs. Further improvement proposals can be found in our repository description. Our DPF implementation was integrated from [1] and used as it is. Our application consists of a single program creating 3 threads (user, data server 1, data server 2). They all run on one machine and use network communication to exchange data between each other.

This algorithm is run by the the server $\mathsf{S_k}$

- Input: $\mathsf{w_k}$ as keyword of $\mathsf{S_k}$
- Ouput: the garbled matrix GM^k, the vectors $\mathbf{gm}_{0,k}$, $\mathbf{a}_0^k$ and the vectors $\{\mathbf{ga}_i^k\}_{i\in[|Q|]}$
- *Constructing the matrix*, $i \in [0, |Q|-1]$
  - $\mathbf{m_i} \leftarrow \sigma(i, \mathbf{ch}_i)$, $\mathbf{m_i} = (m_i(1), \ldots, m_i(|\mathbf{ch}_i|))$
  - $m_i(0) \leftarrow 1$ if $i \in F$, else $m_i(0) \leftarrow 0$
  - $\mathbf{m}_{|\mathbf{Q}|} \leftarrow \mathbf{m_0}$
  - $m_i(j) \leftarrow |Q|$ if $m_i(j) = 0$, $j \in [1, |\mathbf{ch}_i|]$, $i \geq 1$
- *Random pads*, $i \in [0, |Q|]$, $j \in [1, |\mathbf{ch}_i|]$
  - *generating random strings*
    - $s \leftarrow m_i(j)$
    - if $s \neq 0$: , $s' \leftarrow \mathsf{PER}(s)$, $\mathbf{r}^{\mathbf{i-s'}} \xleftarrow{\$} \{0,1\}^\kappa$
    - if $s = 0$: $\mathbf{r}^{\mathbf{i-0}} \xleftarrow{\$} \{0,1\}^\kappa$
  - *computing previous random pads*, $i \in [1, |Q|]$, $\mathbf{ps_i} = (ps_i(1), \ldots, ps_i(|\mathbf{ps_i}|))$
    - if $ps_i(v) = 0$: $\mathbf{pd}_i^v \leftarrow \mathbf{r}^{0-i}$, $v = 1$ to $|\mathbf{ps_i}|$
    - if $ps_i(v) \neq 0$: $s'' \leftarrow \mathsf{PER}(ps_i(v))$, $\mathbf{pd}_i^v \leftarrow \mathbf{r}^{s''-i}$
    - $\mathbf{pd}_i^z \xleftarrow{\$} \{0,1\}^\kappa$, $z = |\mathbf{ps_i}| + 1$ to *inp*
- *Generating random keys*
  - $\mathsf{K}_i^j \xleftarrow{\$} \{0,1\}^{\kappa'}, i \in [0, |Q|], j \in [1, |\mathbf{ch}_i|]$
  - $t = 0, m \in [0, |\Sigma|], y_m \in \Sigma$
  - if $y_m \in \mathbf{ch}_i^j, i \in [0, |Q|], j \in |\mathbf{ch}_i|$:
    - $t++, \mathsf{K'}_m^t \leftarrow \mathsf{K}_i^j$
  - $\mathsf{K'}_m \leftarrow \mathsf{K'}_m || \mathsf{K}_m^{z(b)}, b \in [1, t], \mathbf{z} \leftarrow \text{permute}(1, \ldots, t)$
  - $l = \kappa' \times ch_{max}$, if $|\mathsf{K'}_m| < l : \mathbf{k} \xleftarrow{\$} \{0,1\}^{\kappa' \times (ch_{max} - |\mathsf{K'}_m|)}, \mathsf{K'}_m \leftarrow \mathsf{K'}_m || \mathbf{k}$
- *Garbling the matrix*
  - *Start state:* $i = 0$, $j \in [1, |\mathbf{ch}_i|]$
    - $s \leftarrow m_0(j)$,
    - if $s \neq 0$: $s' \leftarrow \mathsf{PER}(s)$, $\mathbf{m'_0}^j \leftarrow (s' || \mathbf{r}^{\mathbf{0-s'}} || 0^{\kappa + \lg |Q|}) \oplus \mathsf{K}_i^j$
    - if $s = 0$: $\mathbf{r}^{\mathbf{0-0}} \xleftarrow{\$} \{0,1\}^\kappa$, $\mathbf{m'_0}^j \leftarrow (s || \mathbf{r}^{\mathbf{0-0}} || 0^{\kappa + \lg |Q|}) \oplus \mathsf{K}_i^j$
    - $\mathbf{m'_0}^z \leftarrow \{0,1\}^{\kappa'}$, $z = |\mathbf{ch}_i| + 1$ to *out*
    - $\mathbf{a}_0^k \leftarrow (r_0^k)_2$ if $m_0(0) = 1$, else $\mathbf{a}_0^k \leftarrow (r_0^k + 1)_2$, where r_0^k is a random value with κ bits in binary format $(r_0^k)_2$
  - *Other states:* $i = 1$ to $|Q|$, $j = 1$ to $|\mathbf{ch}_i|$
    - $s \leftarrow m_i(j)$,
    - $s' \leftarrow \mathsf{PER}(s)$
    - *Input edges for any state:* $v = 1$ to *inp*
      - $\mathbf{t} \leftarrow \mathsf{perm}(i, j)$
      - $\mathbf{mm_i}^{j,v} \leftarrow (t(v) || s' || \mathbf{r}^{\mathbf{i-s'}} || 0^\kappa) \oplus \mathsf{PRG}(\mathbf{pd_i}^{t(v)}) \oplus \mathsf{K}_i^j$
    - $\mathbf{mm}_i^{z,1} \leftarrow \{0,1\}^{\kappa'}, \ldots, \mathbf{mm}_i^{z,inp} \leftarrow \{0,1\}^{\kappa'}$, $z = |\mathbf{ch}_i| + 1$ to *out*
    - $\mathbf{a}_i^k \leftarrow (r_i^k)_2$ if $m_i(0) = 1$, else $\mathbf{a}_i^k \leftarrow (r_i^k + 1)_2$, where r_i^k is a random value with κ bits in binary format $(r_i^k)_2$
- *Permuting and constructing the matrix*, $i \in [1, |Q|]$
  - $\mathbf{gm}_{\mathsf{PER}(i),k}^{1,t} \leftarrow \mathbf{mm}_i^{1,t}, \ldots, \mathbf{gm}_{\mathsf{PER}(i)}^{out,t} \leftarrow \mathbf{mm}_i^{out,t}$, $t = 1$ to *inp*
  - $\mathbf{GM}^k \leftarrow \{\mathbf{gm}_{i,k}\}_{i\in[|Q|]}$, $\mathbf{gm}_{0,k} \leftarrow \mathbf{m}_0^{'}$, $\mathbf{ga}_{\mathsf{PER}(i)}^k \leftarrow \mathbf{a}_i^k$

Fig. 5. Constructing the garbled DFA matrix

This algorithm is run by the proxy server $\mathcal{S}$
- Input: the garbled matrix GM^k, the vectors $\mathbf{gm}_{0,k}$, $\mathbf{a}_0^k$ and $\{\mathbf{ga}_i^k\}_{i\in[|Q|]}$
- Output: random value $\mathbf{score}_k$

Steps for any $z \in [1, |\mathbf{w}_u|]$ where $\mathbf{w}_u$ is input of the user
- parse $\mathbf{GM}^k$ as $\{\mathbf{gm}_{i,k}\}_{i\in[|Q|]}$
- $\mathsf{K}''_1, \ldots, \mathsf{K}''_{ch_{max}} \leftarrow \mathcal{F}_{\mathsf{POT}}(\mathsf{K}'_1, \ldots, \mathsf{K}'_{|\Sigma|})$
- **Start state:** $(i = 0)$, $j \in [1, out]$
  - $state||\mathbf{pad}||\mathbf{taill} \leftarrow \mathbf{gm}_{0,k} \oplus \mathsf{K}''_v$, $v \in [1, ch_{max}]$
    - if $tail = \mathbf{0}^{\kappa+\lg|Q|}$:
      - $t' \leftarrow 0$
      - $\mathbf{score}_k \leftarrow \mathbf{a}_0^k$
      - $i \leftarrow state$
      - $\mathbf{pd_i} \leftarrow \mathbf{pad}$
- **Other states:**
  - if $t' = 0$:
    - $t \in [1, inp]$
    - $t'\|state\|\mathbf{pad}\|\mathbf{tail} \leftarrow {\mathbf{gm}_i}^{j,t} \oplus \mathsf{K}''_v \oplus \mathsf{PRG}(\mathbf{pd_i})$, $v \in [1, ch_{max}]$
    - if $\mathbf{tail} = \mathbf{0}^{\kappa}$:
      - $\mathbf{score}_k \leftarrow \mathbf{score}_k + \mathbf{ga}_i^k$
      - $i \leftarrow state$
      - $\mathbf{pd_i} \leftarrow \mathbf{pad}$
  - if $t' \neq 0$:
    - $t \leftarrow t'$
    - $t'\|state\|\mathbf{pad}\|\mathbf{tail} \leftarrow {\mathbf{gm}_i}^{j,t} \oplus \mathsf{K}''_v \oplus \mathsf{PRG}(\mathbf{pd_i})$, $v \in [1, ch_{max}]$
    - if $\mathbf{tail} = \mathbf{0}^{\kappa}$:
      - $\mathbf{score}_k \leftarrow \mathbf{score}_k + \mathbf{ga}_i^k$
      - $i \leftarrow state$
      - $\mathbf{pd_i} \leftarrow \mathbf{pad}$

Fig. 6. Evaluating the garbled matrix

Table 1. Performance results of DPF-PACL with Enc redundancy=20, LWE redundancy=1 and secret vector size= 128.

	Time(sec)			
Parameter	User	Server		
# of .files	GHL^+.Dist	GHL^+.Setup	GHL^+.VerifySharing	GHL^+.DecShare
32	503	458	0.44	737
1024	510	459	0.39	738
32768	510	459	0.4	736

For our benchmarks, we run our program on a server with an Intel Xeon Gold 5317 with 32 cores @ 3.00 GHz and 256 GB of RAM. To evaluate the numbers it is important to understand that our implementation in C++ uses only one thread and the Risc0 application performs on all 32 cores. Therefore the computation of the DPF-PACL could be improved significantly. The results show that on the client side the setup of DPF and secret sharing is really fast and calculations only need time because of the ZK-proof, DPF-PACL part would be really fast. On the server side, setup of GHL^+ takes time. But it only needs to be executed once on startup. The number of files has a linear impact on the

PACL reconstruction but is compared to the other values negligible fast. The size of the secret vector defines the security of the system but also has a strong impact on the computational times. It should be mentioned that redundancy of PVW encryption to determine the exact secret key of the user is crucial for granting access. Increasing it this encoding linearly scales the times required for the ZK-proofs. Therefore this process needs to be optimised.

Table 2. Performance results of DPF-PACL with 1024 files, Enc redundancy=20 and LWE redundancy=1.

Parameter	Time(sec)			
	User	Server		
size of secret vector	GHL^+.Dist	GHL^+.Setup	GHL^+.VerifySharing	GHL^+.DecShare
64	250	205	0.25	369
128	510	459	0.4	743
256	1018	918	0.78	1476

Acknowledgments. The authors appreciate Geoffroy Couteau's valuable and profound comments. Giacomo Fenzi is partially supported by the Ethereum Foundation. The authors from Barkhausen Institute are supported by the Federal Ministry of Education and Research, Germany (project 03ZU1210AA) and are also financed based on the budget passed by the Saxonian State Parliament in Germany.

References

1. C++ dpf-pir library (2021), https://github.com/dkales/dpf-cpp
2. Post-quantum access control implementation (2024), https://github.com/Barkhausen-Institut/PQACL
3. Risc0 (2024), https://github.com/risc0/risc0
4. Sp1 (2024), https://github.com/succinctlabs/sp1
5. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 701–732. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_23
6. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_2
7. Bootle, J., Lyubashevsky, V., Nguyen, N.K., Seiler, G.: A non-PCP approach to succinct quantum-safe zero-knowledge. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 441–469. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_16
8. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367. Springer (2015)
9. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1292–1303 (2016)
10. Bruestle, J., Gafni, P.: Risc zero zkvm: Scalable, transparent arguments of risc-v integrity. Draft. July **29** (2023)

11. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
12. Cascudo, I., David, B.: Publicly verifiable secret sharing over class groups and applications to dkg and yoso. Cryptology ePrint Archive (2023)
13. Chiesa, A., Manohar, P., Spooner, N.: Succinct arguments in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 1–29. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_1
14. Choi, S.G., Katz, J., Kumaresan, R., Cid, C.: Multi-client non-interactive verifiable computation. In: Theory of Cryptography Conference. pp. 499–518. Springer (2013)
15. Cini, V., Lai, R.W., Malavolta, G.: Lattice-based succinct arguments from vanishing polynomials. In: Annual International Cryptology Conference. pp. 72–105. Springer (2023)
16. Eskandarian, S., Corrigan-Gibbs, H., Zaharia, M., Boneh, D.: Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021. pp. 1775–1792. USENIX Association (Aug 2021)
17. Gentry, C., Halevi, S., Lyubashevsky, V.: Practical non-interactive publicly verifiable secret sharing with thousands of parties. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 458–487. Springer (2022)
18. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_35
19. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 193–226. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38545-2_7
20. Goyal, P.: Private Information Retrieval with Access Control. Ph.D. thesis, Massachusetts Institute of Technology (2023)
21. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation. Acm Sigact News **32**(1), 60–65 (2001)
22. Ji, K., Zhang, B., Ren, K.: Fine-grained policy constraints for distributed point function. Cryptology ePrint Archive (2023)
23. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious prf with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 818–829 (2016)
24. Naor, M., Pinkas, B.: Computationally secure oblivious transfer. Journal of Cryptology **18**, 1–35 (2005)
25. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce. pp. 129–139 (1999)
26. Newman, Z., Servan-Schreiber, S., Devadas, S.: Spectrum: High-bandwidth anonymous broadcast. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). pp. 229–248 (2022)

27. Niksefat, S., Sadeghiyan, B., Mohassel, P., Sadeghian, S.: Zids: a privacy-preserving intrusion detection system using secure two-party computation protocols. The Computer Journal **57**(4), 494–509 (2014)
28. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Annual international cryptology conference. pp. 554–571. Springer (2008)
29. Servan-Schreiber, S., Beyzerov, S., Yablon, E., Park, H.: Private access control for function secret sharing. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 809–828. IEEE (2023)
30. Servan-Schreiber, S., Beyzerov, S., Yablon, E., Park, H.: Private access control for function secret sharing. In: 2023 IEEE Symposium on Security and Privacy. pp. 809–828. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179295
31. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
32. Stadler, M.: Publicly verifiable secret sharing. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 190–199. Springer (1996)
33. Vadapalli, A., Storrier, K., Henry, R.: Sabre: Sender-anonymous messaging with fast audits. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1953–1970. IEEE (2022)
34. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: Annual International Cryptology Conference. pp. 299–328. Springer (2022)

A Appendix

A.1 Proxy OT

A POT consists of the following algorithms [14] from non-interactive key exchange (NIKE) protocol allows two parties to generate a shared key based on their respective public keys (and without any direct interaction) [14]. In our scheme, n 1-out-of-256 OT should be run where n is the number of characters of the user's input. We use the construction of Naor and Pinkas [24] to efficiently reduce n invocations of the 1-out-of-256 OT to $8n$ invocations of 1-out-of-2 OT. So, $8n$ invocations of Proxy OT [14] are required. Every time, the parties need to run the setup algorithm to generate their key pairs. To reduce the key generation process, we designed a new protocol for choosing the keys related to the input of the user. We can use batch oblivious pseudorandom function (OPRF) [23] instead of running $8n$ NIKE. In more detail, in the Proxy OT the sender and the chooser obtain the outputs of the batch OPRF. We can assume that they are run in the offline phase and the sender and the chooser use the outputs of the offline phase. We can rewrite the scheme [14] as follows. Assume that $t = 8n$ where n is the length of the user's input.

– *Offline:*
  - The sender and the chooser share the random values $\mathbf{r}_{b,1}, \ldots, \mathbf{r}_{b,t} \xleftarrow{\$} \{0,1\}^*$, $\pi_j \xleftarrow{\$} \{0,1\}$, $b \in [0,1], j \in [1,t]$.
  - $(\mathbf{z}_{b,1}, \ldots, \mathbf{z}_{b,t}) \leftarrow \mathcal{F}_{\mathsf{OPRF}}(\mathbf{r}_{b,1}, \ldots, \mathbf{r}_{b,t})$

– *Online*
  - $\alpha_i \leftarrow \mathsf{Snd}(i, \mathbf{x}_{0,i}, \mathbf{x}_{1,i})$. In the i-th POT execution the sender, holding input $\mathbf{x}_{0,i}, \mathbf{x}_{1,i} \in \{0,1\}^{\kappa}$, runs this algorithm to generate a single encoded message α_i to be sent to the proxy. $\alpha_{\pi_i,i} = \mathbf{z}_{0,i} \oplus \mathbf{x}_{0,i}$, $\alpha_{1\oplus\pi_i,i} = \mathbf{z}_{1,i} \oplus \mathbf{x}_{1,i}$
  - $\beta_i \leftarrow \mathsf{Chs}(i, b_i)$. In the i-th POT protocol, the chooser, holding input $b_i \in \{0,1\}$, runs this algorithm to generate a single encoded message β_i to be sent to the proxy. $\beta_i \leftarrow (b_i \oplus \pi_i, \mathbf{z}_{b_i,i})$
  - $\mathbf{y}_i := \mathsf{Prx}(i, \alpha_i, \beta_i)$. In the i-th POT protocol, the proxy runs this algorithm using the sender message α_i and the chooser message β_i, and computes the value $\mathbf{y}_i = \mathbf{x}_{b_i}$. parse α_i as $(\alpha_{0,i}, \alpha_{1,i})$ and β_i as $(b'_i, \mathbf{z}'_i)$, $\mathbf{y}_i = \alpha_{b'_i,i} \oplus \mathbf{z}_{b'_i}$.

A.2 Proof of Theorem 4

Proof. The proof is based on the POT hybrid model. Hence, we consider three cases a adversary corrupts S_k, proxy $\mathcal{S}$ and user in the following.

– user *is corrupted.* Suppose that a PPT adversary $\mathcal{A}$ controls S_k. The simulator Sim behaves as follows that simulates the view of $\mathcal{A}$. Sim emulates $\mathcal{F}_{\mathsf{POT}}$ and obtains the keys used in POT's. Then, Sim garbles the matrix constructed from the server's keyword. The simulator inherits the property of the underlying POT protocol and therefore the adversary is unable to distinguish between the ideal world and real-world executions.

– proxy $\mathcal{S}$ *is corrupted.* Suppose that a PPT adversary $\mathcal{A}$ controls $\mathcal{S}$ and the simulator Sim behaves as follows that simulates the view of $\mathcal{A}$.
  - Sim emulates the functionality $\mathcal{F}_{\mathsf{POT}}$ and gets the input of the server S_k.
  - Sim runs $\mathcal{A}$ on the input w_k which $\mathcal{A}$ forwards it after receiving from S_k in the real execution. Then, Sim generates ch_{max} random keys for running OTs on w_k and sends them to the trusted authority.
  - Sim sends the server's input, w_k, to the trusted authority and gets back the evaluation of DFA on it, the set of corresponding states denoted by Γ_{w_k}.
  - Sim generates the garbled DFA matrix as the following steps which are for any character of input, and forwards it to $\mathcal{A}$. Let n denotes the length of w_k and $\kappa' = 2\kappa + 2\log|Q|$.
  - *Initialize:*
    * $\{s_1, s_2, \ldots, s_{nk}\} \xleftarrow{\$} \{1, \ldots, |Q|\}$, $start \leftarrow 0, i \leftarrow start$, $t \xleftarrow{\$} \{1, \ldots, inp\}$
  - *Start state:*
    * $s' \xleftarrow{\$} \{s_1, s_2, \ldots, s_{nk}\}$, $\mathbf{r} \xleftarrow{\$} \{0,1\}^{\kappa}$,
    * $\{\mathsf{K}_1, \mathsf{K}_2, \ldots, \mathsf{K}_{ch_{max}}\} \xleftarrow{\$} \{0,1\}^{\kappa'}$; generate ch_{max} random keys
    * $col \xleftarrow{\$} \{1, \ldots, out\}$, ${\mathbf{m}'_{\mathbf{0}}}^{col} \leftarrow (s' \| \mathbf{r} \| 0^{\kappa + \log|Q|}) \oplus \mathsf{K}_{col}$
    * ${\mathbf{m}'_{\mathbf{0}}}^{j} \xleftarrow{\$} 0^{\kappa'}$, $j = 1, \ldots, |Q|$, $j \neq col$
    * $\mathbf{a_0} \xleftarrow{\$} \{0,1\}^{\kappa}$
    * $t' \xleftarrow{\$} \{1, \ldots, inp\}$, $i \leftarrow s'$, ${\mathbf{pd}_i}^{t'} \leftarrow \mathbf{r}$
  - *Other states:*
    * $i \leftarrow s'$, ${\mathbf{pd}_i}^{t'} \leftarrow \mathbf{r}$, $col \xleftarrow{\$} \{1, \ldots, out\}$

* $\mathbf{gm_i}^{col} \leftarrow (t' \| s' \| \mathbf{r} \| 0^\kappa) \oplus \mathsf{K}_{col} \oplus \mathsf{PRG}(\mathbf{pd}_i{}^t)$
* $\mathbf{gm_i}^{j,1} \xleftarrow{\$} 0^{\kappa'}, \mathbf{gm_i}^{j,2} \xleftarrow{\$} 0^{\kappa'}, \ldots, \mathbf{gm_i}^{j,inp} \xleftarrow{\$} 0^{\kappa'}$, $j = 1, \ldots, |Q|$, $j \neq col$
* $\mathbf{a_i} \xleftarrow{\$} \{0,1\}^\kappa$, $t \leftarrow t'$, $i \leftarrow s'$, $\mathbf{pd}_i{}^{t'} \leftarrow \mathbf{r}$

We now prove that $\mathcal{A}$ cannot distinguish between its view during the real execution and its interaction with Sim. Suppose that a distinguisher $\mathcal{D}$ distinguishes the distributions of the real and ideal world transcripts with non-negligible probability. Let $\mathbf{gm}_i$ be in the i-th element of in the garbled matrix. Relying on a hybrid argument, we define the hybrid distributions $H_i \stackrel{\mathsf{def}}{=} (\mathbf{gm}_{1,re}, \ldots, \mathbf{gm}_{i,re}, \mathbf{gm}_{i+1,id}, \ldots, \mathbf{gm}_{z,id})$, $i = 0, \ldots, nk$ $z = |Q|out$. Note that H_0 is the view of $\mathcal{A}$ in the real execution and H_{nk} is his interaction with Sim. Hence, $\mathcal{D}$ distinguishes H_{i-1} from H_i for some i in polynomial time with non-negligible probability. But H_{i-1} and H_i are generated in the same way. Moreover, random strings for all values in row i are uniformly chosen except for one value that appears on the transit path of input. So, due to the security of PRG used in matrix construction, this contradicts the security of PRG. Hence we can argue that $H_{i-1} \stackrel{c}{\approx} H_i$. So, the view of $\mathcal{A}$ when interacting with Sim is the same as H_{nk}.

– *The server* S_k *is corrupted.* The only message from user is during the POT protocol execution, user's privacy against S_k follows from the privacy of POT used.