Proofs of Data Residency: Checking whether Your Cloud Files Have Been Relocated

Hung Dang, Erick Purwanto, Ee-Chien Chang
School of Computing, National University of Singapore
{hungdang,erickp,changec}@comp.nus.edu.sg

***Abstract*—While cloud storage services offer manifold benefits such as cost-effectiveness or elasticity, there also exists various security and privacy concerns. Among such concerns, we pay our primary attention to *data residency* – a notion that requires outsourced data to be *retrievable in its entirety from local drives of a storage server in-question*. We formulate such notion under a security model called *Proofs of Data Residency* (PoDR). PoDR can be employed to check whether the data is replicated across different storage servers, or combined with storage server geolocation to "locate" the data in the cloud. We make key observations that the data residency checking protocol should exclude all server-side computation and each challenge should ask for no more than a single atomic fetching operation. We illustrate challenges and subtleties in protocol design by showing potential attacks to naive constructions. Next, we present a secure PoDR scheme structured as a timed challenge-response protocol. Two implementation variants of the proposed solution, namely N-ResCheck and E-ResCheck, describe an interesting use-case of trusted computing, in particular the use of Intel SGX, in cryptographic timed challenge-response protocols whereby having the verifier co-locating with the prover offers security enhancement. Finally, we conduct extensive experiments to exhibit potential attacks to insecure constructions and validate the performance as well as the security of our solution.**

I. Introduction

The growth of information has made data-generation outpace storage availability [46]. This has given rise to cloud data storage models, as offered by various well-known cloud service providers [2], [6]. Cloud storage models have gained significant popularity, and offer manifold benefits including cost-effectiveness and elasticity. They also present data owners with a simple view of the outsourced files, abstracting away the underlying file-layout and/or storage mechanisms that they employ to maintain the data. While the abstraction is appealing, the lack of understanding of the underlying mechanisms adds to various security concerns on whether the service providers are upholding the service level agreement contract (SLA).

Various real-world incidents and application scenarios have demonstrated that those security concerns are realistic. A cloud crash disaster could permanently destroy the outsourced data [4]. Such a threat prompts the needs of testing for fault tolerance of the storage system, and checking whether the files are backed-up across multiple geographically separate storage servers [23]. In addition, various legislation and directives regulating the possessing and storage of data across national borders motivate paying attention to locations at which the files are maintained [36], [3], [5]. In view of these concerns, it is desired to have technical means that verify whether the files are indeed maintained in accordance with the agreements.

Existing works have studied techniques that allow data owners to obtain assurance on the locations of their cloud files. In particular, Gondree *et al.* [28] attempt to bind the storage to a geographical location, thus geolocating the data in the cloud. Bowers *et al.* [23] present a timed challenge-response protocol to check whether a file is distributed across multiple storage units. Benson *et al.* [20] investigate the correlation of network latency and geographical distance and employ a set of friendly landmarks to verify that files are replicated across multiple drives in a single datacenter. Nevertheless, due to the noisy network environment, it is still technically challenging to accurately and reliably geolocate the outsourced data.

In this paper, we take a different approach to address these problems. Instead of attempting to verify the geographical location of the data, we focus on a more modest goal of verifying the residency of the outsourced data in a server. We ask for a proof asserting the fact that an outsourced file F is indeed *maintained in its entirety* on the *local drives* of the server in-question. It is worth noting that the definition of data residency provides more information on the data's maintenance than just the retrievability of the data, which has been extensively studied under the notions of Proofs of Retrievability (PoR) and Provable Data Possession (PDP) [32], [17], [43], [42]. The data residency can be an integral component in auditing other contractual assurances. For instance, one can first employ existing host geolocating techniques [35], [30], [24] to geographically locate a storage server, and then attest the residency of the outsourced file on that server to affirm geolocation of the data. It can also be utilised to assess fault-tolerance degree of a storage system or to prove that the file is replicated at different geographically separate servers, by checking the residency of the file on each of the servers.

We formulate the notion of data residency under a security model called *Proof of Data Residency* (PoDR). Our model takes into consideration behaviour of storage devices and capabilities of dishonest storage providers (i.e. adversaries). Since the providers control the servers and network under their purview, they are able to derive a more accurate noise estimation, which could provide the adversary an advantage in evading data residency checking. In addition, the providers could potentially exploit parallelism, data compression techniques or hardware accelerations to influence the challenge-response latencies, which are the main sources of information

to be relied on in residency checking. In view of these challenges, we introduce a notion of *atomic fetching* operation – one which the prover must invoke in every challenge-response interaction – and consider a powerful adversary who can reduce all processing time for any challenge to the equivalent of a single atomic fetching, and able to sample the noise before making decisions.

We propose techniques to obtain proofs of data residency. A data residency checking is structured as a *timed challenge-response protocol*. Our solution adopts an authenticator-based PoR [34], [32] as an underlying cryptographic primitive to attest the file's retrievability. In addition, it assesses the response latencies incurred to establish the data residency. We discuss two implementation variants. The first variant, N-RESCHECK, allows a remote verifier to obtain proofs of data residency over the network, while the second variant, E-RESCHECK, necessitates a presence of an trusted unit on the server in-question. With the recent initiatives on trusted computing platform, most notably Intel SGX technology [8], it is of great interest to investigate a timed challenge-response protocol whereby having the verifier residing in an protected enclave on the prover's physical server enhances the security.

Our study suggests two general guidelines in the design of an efficient and secure PODR protocol. First, it is necessary to minimise or even eliminate the computation to be carried out by the prover. Preferably, during the verification process, the prover should only fetch and send data to the verifier. Previous works [28], [20] also advocate no server-side computation. Interestingly, their arguments are motivated by practical concerns on usability (since the cloud storage's API may not be extensive enough to support the required computation) and the needs to reduce server's computation load for cost-saving. In contrast, our observations are motivated by the security requirements. This guideline explains our choice of the authenticator-based PoR scheme as the underlying cryptographic primitive. Second, it is crucial to lower the response latencies incurred by an honest prover, even with an increase in performance overhead. This suggests fetching only a single data block of *small size* (e.g. as small as 64 bytes) for each response, which entails the use of very short authentication tags (say eight bits), in contrary to most known PoR constructions that utilise long tags [32], [43], [42]. We note that long authentication tags are necessary to thwart chosen-message attacks whereby the adversary has access to the verification oracle. Our application settings, on the other hands, limit the number of times the dishonest prover can invoke the verification oracle and thus allow the use of short authentication tags without compromising security.

We empirically show that for insecure constructions, the adversary can evade detection by over-clocking the computation or employing parallelism. Furthermore, we conduct extensive experiments to evaluate our proposed solution. The experiment results support the needs of small block size (around 64 bytes in all of our settings). Very low false acceptance rate and storage overhead can be achieved with authentication tags that are as small as eight bits. The experimental studies also demonstrate the level of improvement obtained by incorporating trusted computing. In particular, for the same performance requirements of 23% storage expansion (among which 21.5% due to error-erasure code and another 1.5% due to authentication tags) and audit size of 300 challenges, E-RESCHECK achieves an order of magnitude lower false acceptance rate (3.9×10^{-10} vs. 6.7×10^{-09}) and several orders of magnitude lower false rejection rate (2.6×10^{-22} vs. 7.3×10^{-08}) in comparison with N-RESCHECK. This illustrates an interesting use-case of trusted computing where security can be enhanced by having the verifier of a cryptographic protocol co-locating with the prover.

In summary, our paper makes the following contributions:

- We present a security definition of *Proofs of Data Residency* in a presence of a powerful adversary who is able to reduce the time taken for processing any challenge down to the equivalent of an atomic fetching operation, and able to foresee all timing measurements.
- We discuss and empirically show potential attacks on insecure PODR constructions.
- We propose a secure and efficient PODR protocol and analyse its security. We describe two implementation variants of the proposed protocol: N-RESCHECK and E-RESCHECK, illustrating an interesting use-case of trusted computing, in particular the use of Intel SGX, in cryptographic timed challenge-response protocols.
- We conduct extensive experiments to evaluate our solution, and show that the proposed PODR protocol obtains negligible false acceptance and false rejection rates with reasonable storage overhead and audit size.

The rest of this paper is organized as follows. We provide background on pertinent notions of PoR, geolocation and Intel SGX in Section II before stating our problem in Section III. Next, we present our definition of Proofs of Data Residency in Section IV. We discuss potential attacks on insecure constructions in Section V and propose a secure protocol in Section VI. Experimental evaluation is presented in Section VII while related works are surveyed in Section VIII. Finally, we conclude our work in Section IX.

II. PRELIMINARIES

In this section, we briefly provide background on the closely related notions of PoR and summarize key characteristics of Intel SGX technology.

A. Proofs of Retrievability

Initially proposed by Juels and Kaliski [32], PoR enables the data owner to audit the storage server on the data preservation. The main idea behind PoR is to encode the original data using a redundant encoding (such as the error-erasure Reed-Solomon code [39]), authenticate all the blocks of the encoded data before sending them to the storage server. Due to the redundant encoding, the storage provider has to discard or tamper with a considerable portion of the blocks to cause data loss. However, if a considerable portion of the blocks is lost, the verifier can detect this incident with overwhelming probability.

A PoR scheme is executed in a challenge-response fashion. The verifier $\mathcal{V}$ may issue a random challenge (which may contain one or various queries) at any time, and to which the prover $\mathcal{P}$ has to respond correctly to assert for its possession and the retrievability of the file. The first construction by Juels and Kaliski [32] has been followed by various variants [26], [22], [44], [45] and is also extended to the dynamic setting [43]. A similar notion known as Provable Data Possession (PDP) is proposed by Ateniese *et al.* [17]. It is commonly believed that PDP provides weaker security guarantees than PoR in a sense that even if the prover passes the PDP audit, there is still a non-negligible probability that the verifier cannot fully recover the original outsourced file [43].

B. Host Geolocation

While the notion of data residency concerns over the fact that the data is kept intact on local drives of a storage server, it implicitly assumes that the geographic location of the storage server is known to the verifier. Thus, also of interest are techniques to geographically locate an online party. Since machines/systems on the Internet can be uniquely identified by an IP address, this problem asks for a mapping from an IP address to a geographic location. It would have been trivial if the IP address system was designed to incorporate geographic information, unfortunately, it was not the case. Several proposals have been presented to address this problem [35], [30], [24]. Common among them are observations that major backbone Internet providers usually associate their host names with geographical clues, and that data travelling across the Internet are often routed via these backbone Internet providers' nodes. Moreover, a route that a data packet travels through can be identified using trace engines such as `Traceroute` utility [14]. When matching the intermediary computer nodes in the routing information of a packet against those of the backbone Internet providers, a target host (the destination of the packet in question) can be roughly located [24]. However, this technique alone does not offer fine granularity. When the packet is approaching its destination, it will be transferred using smaller networks to which geographical clues are not associated. At that granularity, the WHOIS servers [29] are queried to infer more precise location of the host. Other approaches rely on a premise that the latency in transmitting a packet between a pair of hosts is a function of the geographical distance among them, or a combination of partial IP-to-location and BGP prefix information to derive the target host's location [35].

C. Intel SGX

Intel SGX [8] is a set of extensions that provision the protected execution environments (aka trusted environments or enclaves). The TCB of such enclaves comprises solely of the processors and the code that the enclaves' owner places inside them, which is arguably minimal. Each enclave is associated with a region on physical memory, which we shall call enclave memory. All access to enclave memory are protected by the processor. In another word, code and data loaded to the enclaves cannot be disclosed or modified by the untrusted OS or any other processes/software; any attempt to read or write the enclave's memory by a non-enclave code will be blocked. On the other hand, enclave code may access enclave memory as well as memory outside of the enclave region (if the OS permits) [19]. Originally, memory pages can only be added to the enclave during its creation; however since revision 2 of the SGX specification, enclave pages can be added via a cooperation of the enclave and the (untrusted) OS [8] at any time during its lifetime. We note enclave code has to be loaded into the enclave during its creation.

Enclaves cannot directly execute OS-provided services such as I/O. In order to access those services, enclaves have to employ OCalls (calls executed by the enclave code to transfer the control to non-enclave code) and ECalls (API for untrusted applications to call in). These ECalls and OCalls constitute the enclave boundary interface, enabling a communication between the enclave code and the untrusted application to service OS-provided functions. Care should be taken on each and every ECall exposed to the untrusted application, as it may open up an attack surface to the protected execution environment.

SGX enables CPU-based attestation [15], enabling a remote verifier to check if a specific software has been loaded within the enclave by means of cryptography. Via the remote attestation mechanism, the verifier can establish shared secrets with the enclave, thus bootstrapping an end-to-end encrypted channel via which sensitive data can be communicated.

III. The Problem

A. Overview

We consider a model comprising of two entities. The *data owner* wishes to outsource a file F to a *storage server* and insists that her data is maintained locally at the storage server. A dishonest storage server has various economic incentives to violate the agreement and may move some of the data to another remote server. Hence, the data owner would like to periodically verify that *the file F can be retrieved in its entirety from data maintained on the server's local drives*. We refer to such verification as a *data residency checking* protocol. In data residency checking, the data owner plays a role of a *verifier* $\mathcal{V}$, while the storage server plays a role of a *prover* $\mathcal{P}$. Hereafter, we shall refer to the data owner as verifier, and storage provider as prover.

A data residency checking is structured as a timed challenge-response protocol. In another word, it consists of several challenge-response exchanges and for each response, $\mathcal{V}$ also captures the response latency (i.e. round trip time between the challenge and response). At the end of the protocol, $\mathcal{V}$ relies on the validity of the responses, as well as their latencies, to decide on accepting or rejecting the verification.

The retrievability of F can be checked using techniques in PoR [32], [42], whereas the assurance of storage locality relies on the response latency. Nevertheless, simply adopting a secure PoR scheme together with latency assessment does not necessarily provide the assurance on data residency, since a

dishonest server (i.e. adversary) could, through parallelism or over-clocking the processor, distort the latency measurements. Fortunately, the desired assurance is still possible, based on a premise that $\mathcal{P}$ has to invoke some atomic operations to prepare for each response, and such operations would take longer time when the data are stored remotely. The goal of our security model is to capture the above-mentioned factors.

B. Timing measurements

The response latency of a challenge is the round-trip-time of the challenge and response (i.e., the elapsed time between the moment the challenge is sent and the moment when the corresponding response is received). The latency consists of the following three portions:

- *Challenge-response transmission time*, which is incurred by transmission of the challenge and response between $\mathcal{V}$ and $\mathcal{P}$. In the trusted computing setting where both $\mathcal{V}$ and $\mathcal{P}$ reside in the same physical system, such transmission time is short. In the setting where $\mathcal{V}$ and $\mathcal{P}$ are connected in a networked environment, the time is significantly larger and subjected to higher level of noise.
- *Fetching time*, which is incurred when $\mathcal{P}$ fetches the required data from the storage. In cases where the prover fetches the data from another remote server, the fetching time include the transmission time between the prover and the remote server, and the time incurred by the remote server in loading the data from its storage device.
- *Computation time*, which is the total computation time taken by $\mathcal{P}$ in producing the response from the data fetched.

All these timings are probabilistic, and we call their distributions the environment profile $\mathcal{E}$.

C. Threats Model: Adversary's capability

We consider an adversary who is a dishonest prover, having complete control over the storage, server and network within its premises, and thus the adversary has the capability to reduce the response latency in various fashions:

Computation time. We consider adversaries who can speedup the computation time, for example via over-clocking or parallelism. Since it is difficult to bound the speedup factor which the adversaries can possibly achieve, we consider a strong adversarial model where the computation time is not included in the response latency for worst-case analysis. Note that such assumption does not imply arbitrary computation speedup by the dishonest prover, and we still require $\mathcal{P}$ to be polynomial time.

Fetching time. When the data is stored on the prover's local drives, the fetching time is simply that of a read from the local storage hardware. On the other hand, if the data to be fetched is stored remotely, the fetching time comprises the time taken to execute a read on the remote storage device, and the time required to transmit the data from the remote storage to the prover. A dishonest prover could apply various techniques such as data compression or distributed file system to reduce the storage loading, or the network transmission time, which in-turn reduces the fetching time. To account for this flexibility, we consider the fetching of a single byte as atomic, and give the adversary the capability of reducing the time taken to fetch any amount of data to the equivalent of fetching a single byte.

Noise Measurement. Due to the noisy environment, all the timing measurements are probabilistic. Nevertheless, a dishonest prover may be able to get a good estimate of the actual measurements. Such knowledge could help the dishonest prover to increase the chance of evasion. For example, let us consider an adversary who stores half of the data blocks in local drives, and the others in remote storage. If a challenge asks for a block stored in a remote storage, he could either retrieve it from the remote storage or forge the response. The knowledge of the actual response latency incurred by fetching the block from remote storage and that by reading it from local drives would benefit the adversary. In particular, if the former is faster than the latter (perhaps due to network congestion), the adversary will retrieve the correct block from remote storage; otherwise, he may choose to forge the block to meet the timing measurement constraint. In our adversarial model, to acknowledge the adversary's ability to accurately estimate the timing measurements, we assume the prover knows the actual measurements of all timings right in the beginning of the verification session.

D. Threats Model: Adversary's limitations

Limited access to verification oracle. We assume that the data owner is less forgiving to dishonest provers who forge the responses (instead of being late or admitting unable to produce the responses). While the adversary may argue that a response is not valid due to hardware failure, such event is highly unlikely since most storage and networking system has error detection/recovery mechanisms in-place. Hence, the adversary has a very small number of chances (or even none) for providing invalid responses. In other words, it has limited access to a verification oracle.

As mentioned earlier, one of our key observations is the enhancement brought by having very small block size, which entails the use of short authentication tags (say eight bits per tag). These short authentication tags are vulnerable to an adversary who has multiple accesses to the verification oracle, as discussed by Shacham *et al.* [42]. Since the adversary in our model only has limited accesses to the verification oracle, such attack vector is voided.

Atomic operation. A key assumption in our work is that of an atomic operation. Such atomic operation would take longer time when the data are stored remotely compared to when it is stored on the storage server's local drives. This assumption in-turn is based on the premise that there are no technically feasible and/or economically feasible mean for the dishonest

servers to reduce the time. In cases where the abovementioned premise is not met, unfortunately, the assurance provided by our schemes will not hold. Examples of those cases includes a dishonest prover who claims to use rotational disks for local drives, but employs flash storage (e.g., SSD) in a remote server and connects to it via an out-of-band communication channel such that the overall throughput outpaces that of the local drives. Nevertheless, it is arguably reasonable to assume that such out-of-band channel is not available and that the service providers are economically rational (i.e., it could not afford *arbitrarily* large and expensive resources in evading the data residency checking) and will employ storage devices of the same class (e.g., enterprise hard drives) on both local and remote storage (if any).

IV. Proofs of Data Residency

A. Setup and Audit Phases

A PoDR scheme is to be carried out in two phases, **Setup** and **Audit**:

- **Setup**: In the setup phase, $\mathcal{V}$ as a data owner generates a secret key sk based on the security parameter λ. Next, $\mathcal{V}$ encodes the file F into $\widetilde{F}$ using the secret key sk. Finally, $\mathcal{V}$ sends the encoded file $\widetilde{F}$ to $\mathcal{P}$, discards both F and $\widetilde{F}$, only keeps the secret key sk and some metadata needed for conducting audit.
- **Audit**: In the audit phase, $\mathcal{V}$ conducts a data residency checking by challenging $\mathcal{P}$ to prove that the original file F can be reconstructed from data maintained in its local drives. This phase comprises of two stages, *challenge-response* and *verification*.
  - *Challenge-response:* The verifier first obtains an environment profile $\mathcal{E}$ based on which she could assess the response latencies. The verifier $\mathcal{V}$ sends v challenges to the prover, and $\mathcal{P}$ replies with the corresponding responses. The challenges are sent one-by-one. Upon receipt of a response or a special symbol $\perp$, the verifier $\mathcal{V}$ proceeds by carrying out the following:
    1) Generates and sends the next challenge to $\mathcal{P}$. The verifier can choose not to send any challenge.
    2) Generates and sends a symbol $\perp$ back to itself, that will arrive at a time specified by $\mathcal{V}$. The verifier can choose not to send the symbol[1].

    Let $\langle q_1, \ldots, q_v \rangle$ be the v challenges sent, $\langle f_1, \ldots, f_v \rangle$ the corresponding responses, and $\langle t_1, \ldots, t_v \rangle$ their latency. Let us call v the audit size.
  - *Verification.* Based on the challenges, the corresponding responses and latencies $\langle t_1, \ldots, t_v \rangle$, together with the environment profile $\mathcal{E}$, $\mathcal{V}$ decides whether to accept $\mathcal{P}$ as passing the audit.

[1]The sending and receiving of the symbol $\perp$ enable the verifier to send the next challenge without waiting for response of previous challenge from the prover, if need be.

Overall, the algorithms in a PoDR scheme consists of: (1) the key generation and file encoding algorithms used during the setup, together with (2) the challenge generation algorithms, (3) the verification algorithm used in the audit. The scheme also implicitly requires an algorithm for the prover to generate the responses.

B. Security and Adversarial Model

We now formalise the capabilities and constraints of the adversary. First, let us define by T^{net}, T^{loc} and T^{rmt} three positive random variables, each follows a predefined distribution. The random variable T^{net} corresponds to the challenge-response transmission time, T^{loc} corresponds to fetching time of an honest prover in producing the response, and T^{rmt} corresponds to the fetching time when the data is loaded from a remote storage. The environment profile $\mathcal{E}$ is a description of the distribution of T^{net}, T^{loc} and T^{rmt}. The prover also has access to, but cannot influence, the environment profile $\mathcal{E}$.

Storage preparation during setup. During the setup phase, the prover applies transformation on the received encoded file $\widetilde{F}$, obtaining $\langle D, \widetilde{D} \rangle$. The portion D is to be kept in the local drives, whereas $\widetilde{D}$ is to be kept in the remote storage ($\widetilde{D}$ could be empty). The prover initialises a cache C of finite size.

Response generation during audit. For each challenge q_i, the prover can choose to compute the response from one of the three probabilistic algorithms R, $\widetilde{R}$ or $\hat{R}$. All of these algorithms have access to the cache C, but differ from each other in accessing to D and $\widetilde{D}$: R only reads from the local drives, $\widetilde{R}$ reads from both local and remote storage, and $\hat{R}$ does not read from any storage.

Given a challenge q_i, the prover independently draws three samples $(t_i^{\text{net}}, t_i^{\text{loc}}, t_i^{\text{rmt}})$ from the distributions T^{net}, T^{loc}, T^{rmt} respectively to obtain actual values of these three timings. Next, the prover decides to take one of the following actions:

1) Send $R(q_i, D, C)$ as response and set
$$t_i = t_i^{\text{net}} + t_i^{\text{loc}} + \delta_i;$$
2) Send $\widetilde{R}(q_i, \langle D, \widetilde{D} \rangle, C)$ as response and set
$$t_i = t_i^{\text{net}} + t_i^{\text{rmt}} + \widetilde{\delta}_i;$$
3) Send $\hat{R}(q_i, C)$ as response and set
$$t_i = t_i^{\text{net}} + \hat{\delta}_i.$$

where $\delta_i, \widetilde{\delta}_i$ and $\hat{\delta}_i$ are positive values chosen by the prover. By the above definition, the prover can foresee all the timing measurements and can influence the value of t_i by adding delays and choosing which algorithms he would use in preparing the response. The cache C is updated after the response is served.

Remarks. The above formulation implies a strong adversary who (1) has the knowledge of the actual time taken to read and transmit the data; (2) is able to produce the response as fast as the atomic loading operation[2]; and (3) is able to arbitrarily delay the response latency. As discussed in the threat model, it is necessary to consider such strong adversary since the dishonest prover would have full control of both the local and remote servers.

C. Security definitions

Given the profile $\mathcal{E}$, we say that a PoDR scheme is ($\mathcal{E}$, ψ)-secure if, for any prover who passes the verification with probability at least ψ, there is a polynomial time algorithm to reconstruct the original file F from D – a portion of data that the prover stores locally (except with negligible probability of failure). The randomness is taken over the random decisions made by the probabilistic algorithms, and the sampling of the timings.

For a PoDR scheme and a profile $\mathcal{E}$, we call the smallest upper bound on ψ' such that the scheme is $(\mathcal{E}, \psi')$-secure the *false acceptance rate* (denoted by ψ). We call the probability that the honest prover, who keeps entire $\widetilde{F}$ in its local drives, fails the verification the *false rejection rate* (denoted by γ).

V. Potential Attacks

In this section, we consider two data residency checking protocols that incorporate latency measurements with well-known PoR schemes [42], [32] in a straightforward manner, and illustrate how a dishonest prover who has relocated significant portion of the data to remote storages, to an extent that the original file cannot be reconstructed from its local drives, can evade detection.

We investigate an ability to evade detection of an adversary $\mathcal{A}$ by comparing the two distributions: $T_{\mathcal{P}}$, the distribution of response latencies incurred by an honest prover $\mathcal{P}$, and $T_{\mathcal{A}}$ - that of the response latencies incurred by $\mathcal{A}$. If the cumulative distribution function (CDF) of $T_{\mathcal{A}}$ *stochastically dominates* $T_{\mathcal{P}}$, that is,

$$Pr(T_{\mathcal{A}} \leq t) \geq Pr(T_{\mathcal{P}} \leq t) \quad \forall t$$

then it is possible for $\mathcal{A}$ to add intentional delays so that these two distributions are identical, successfully evading detection.

A. SW-PoR based data residency checking

Protocol. We first consider a data residency checking constructed on top of the PoR scheme by Shacham and Waters (SW-PoR) [42]. In this PoR scheme, each challenge asks for v data blocks and their associated homomorphic authentication tags. The response are aggregated from the requested data blocks, resulting in a much smaller size. In a SW-PoR based residency checking protocol, the verifier $\mathcal{V}$ measures the response latencies, and accepts the prover as passing the audit if the responses are valid (with respect to the SW-PoR scheme) and the response latencies are within an expected threshold.

Dishonest Prover. We consider two adversaries who relocate the data to three remote storage servers and attempt to reduce response latency by speeding up the computation time required to generating the response. The first adversary – denoted by $\mathcal{A}_{OC}$ – over-clocks its processor in order to evade the detection. $\mathcal{A}_{OC}$ carries out the following steps in upon receiving the challenge from the verifier:

1) The local server redirects the challenge to the three remote servers.
2) The three remote servers concurrently load the data, and send them to the local server.
3) The local server over-clocks its processor to aggregate the data.

The second adversary – denoted by $\mathcal{A}_{DQ}$ – parallelises the aggregation in the following steps:

1) The local server redirects the challenge to the three remote servers.
2) The three remote servers concurrently load the data, aggregate them and send the intermediate results to the local server.
3) The local server aggregates the received intermediate results.

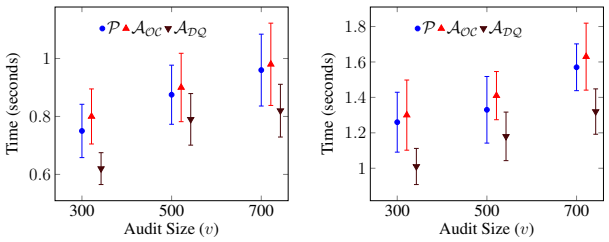
We conduct experimental studies to inspect the response latencies of the honest prover in comparison with those of the two adversaries. In these experiments, provers compute the responses using a vCPU Intel Xeon Family running at base clock speed of 2.5GHz, except for $\mathcal{A}_{OC}$ who over-clocks its processor, running at Turbo Boost speed of 3.3GHz [1].

Empirical results. We vary the block sizes (number of group elements in each data block) as well as the number of data block requested (i.e., audit size) in each challenge. We observe that the response latencies of the three provers generally follow normal distributions, each with different mean and standard deviation. We depict these distributions in Figure 1 by showing their means and standard deviations. To give a better intuition on the adversaries' ability to evade latency measurements, we show in Figure 2 CDFs of their response latencies in experiments where audit size are 700 blocks, with block size of 160 and 320 group elements. As can be seen from the figure, the CDFs of $\mathcal{A}_{DQ}$'s latency measurements stochastically dominate those of the honest prover. Hence, $\mathcal{A}_{DQ}$ can evade the detection by intentionally introducing delays to the response time. Although the CDFs of $\mathcal{A}_{OC}$'s latency measurements do not stochastically dominate $\mathcal{P}$'s, they are similar and thus it requires challenge of significant size in order to detect $\mathcal{A}_{OC}$'s violation of the SLA.

B. JK-PoR based residency checking

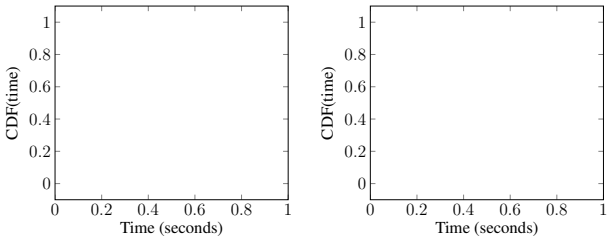
Protocol. One possible mitigation for the previous attack is to adopt a PoR scheme in which the prover performs virtually no computation in executing the residency checking, such as

[2]We stress that the prover's algorithms are still polynomial time.



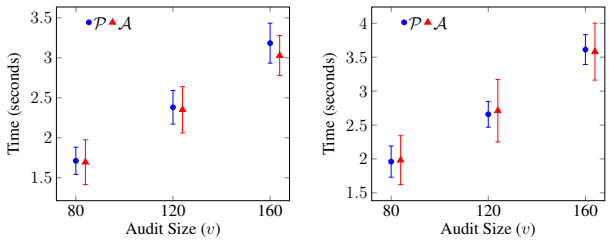
(a) Block size: 160 (b) Block size: 320

Fig. 1: Response latencies incurred $\mathcal{P}$, $\mathcal{A}_{\mathcal{OC}}$ and $\mathcal{A}_{\mathcal{DQ}}$ in SW-PoR based residency checking. The error bars represent one standard deviation.



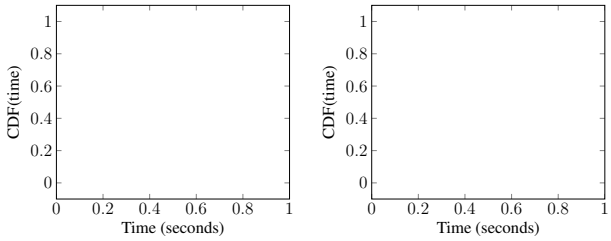
(a) Block size: 160 (b) Block size: 320

Fig. 2: CDF of the response latencies incurred by $\mathcal{P}$, $\mathcal{A}_{\mathcal{OC}}$ and $\mathcal{A}_{\mathcal{DQ}}$ in SW-PoR based residency checking.



(a) 512-byte block (b) 1024-bytes block

Fig. 3: Response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in in JK-PoR based residency checking. The error bars represent one standard deviation.



(a) 512-bytes block (b) 1024-bytes block

Fig. 4: CDF of the response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in JK-PoR based residency checking.

the authenticator-based PoR [32], [34]. In this scheme, the data owner pre-processes the file F using an error-erasure code to create $\widetilde{F}$, partitions $\widetilde{F}$ into m blocks, and appends a MAC under secret key sk to each of them before outsourcing them to the storage server. During the residency checking, the verifier issues a single request that asks for $v \ll m$ randomly chosen data blocks (the value of v is determined by the security setting of the scheme) and measures the latency incurred by the storage provider in delivering all those requested blocks.

Dishonest Prover. Although it is no longer possible to speedup the response latency by over-clocking its processor or employing parallelism, a dishonest storage provider can still reduce the latency by distributing the fetching of the requested blocks. With sufficient number of remote storage servers, the reduction of fetching time can offset the additional latency incurred by accessing the remote storage.

We empirically study the effectiveness of the dishonest prover. In our experiments, the honest prover $\mathcal{P}$ follows the protocol and keeps the user's data in its own local drives, while the dishonest prover $\mathcal{A}$ distributes the data blocks to five different remote servers[3], and pulls data blocks from these servers in parallel to the local server upon requested. Each data block is appended with a 160-bit MAC. The storage servers are equipped with commodity storage hardware whose read latency ranges from 12 to 15ms on average.

[3]Average round-trip time of transmitting a 64-byte packet between $\mathcal{A}$ and these servers is 6.5m.

Empirical results. We vary the number of blocks requested in each audit from 80 to 160, as well as the block size (512 and 1024 bytes), and observe that the response latencies of $\mathcal{P}$ and $\mathcal{A}$ generally follow normal distributions, each with different mean and standard variation. We show the means and standard deviations of these distributions in Figure 3. We also depict in Figure 4 their CDFs for audits of size 160 blocks. When the block size is 1024 bytes, although we do not have stochastic dominances, the two CDFs are similar. With block size of 512 bytes, the CDF of $\mathcal{A}$'s s latency measurements stochastically dominates that of $\mathcal{P}$, implying it can always evade the detection.

VI. Proposed Construction

In this section, we present our construction for PoDR. Our construction is built on top of the authenticator-based PoR by Juels et al. [32], but with very short authentication tags. We first give an overview of the setup and audit phases. We propose two implementation variants for the residency checking, a network-based implementation N-ResCheck, and a trusted computing-based implementation E-ResCheck. E-ResCheck illustrates an interesting use-case of trusted computing, where having the verifier of a cryptographic protocol co-locating with the prover enhances the security. Table I summarises notations that are used throughout the rest of the paper.

TABLE I: Summary and descriptions of the notations that are used throughout the paper. Group I are parameters decided in the setup phase. Group II are parameters and variables involved in the audit phase. Group III are the security metrics of our construction.

	Notation	Description
I	n	number of blocks in the original file F
	s_0	F's block size
	c	expansion rate due to error-erasure code
	m	number of encoded blocks; $m = (1 + c) \times n$
	s	authenticated block size; $s = s_0 + b$
	b	bit length of authentication tags (MACs)
	h	total file expansion factor; $h = (m \times s)/(n \times s_0)$
II	v	audit size (i.e. number of challenge-responses)
	d	latency threshold
	l	late delivery threshold
	q_i	i^{th} challenge
	f_i	i^{th} response
	t_i	measured latency of i^{th} response
III	ψ	false acceptance rate
	γ	false rejection rate

A. Setup

The original file F is divided into n blocks where the size of each block is a parameter to be determined. The data owner applies standard error-erasure code (such as the Reed-Solomon code [39]) on F, generating $\widetilde{F}$. The encoded file $\widetilde{F}$ consists of $m = (1+c) \times n$ blocks ($c > 0$) such that knowledge of any n blocks is sufficient to reconstruct F. We refer to the ratio n/m as *code rate*. $\widetilde{F}$ is identified by a particular file handle and each block in $\widetilde{F}$ is indexed by an unique integer $i \in [1..m]$ which is referred to as *block ID*. Every encoded data block in $\widetilde{F}$ is appended with a b-bit MAC of the block's content and ID under the secret key sk. After entrusting $\widetilde{F}$ to the storage provider, the data owner deletes all local copies, keeping only the secret key sk and some metadata for verification.

B. Audit

To begin the residency checking, $\mathcal{V}$ first obtains an environment profile $\mathcal{E}$ (i.e. description of the distribution of T^{net}, T^{loc} and T^{rmt}) and decides on three parameters:

- v: The audit size, which is the number of blocks that she would like to challenge $\mathcal{P}$.
- d: The latency threshold, which is the threshold of a response being declared late.
- l: The late delivery threshold, which is the number of late responses (whose latency exceeds d) that $\mathcal{V}$ is willing to tolerate.

In Section VII, we investigate how these parameters are to be chosen.

Next, $\mathcal{V}$ executes the residency checking procedure detailed in Algorithm 1. The algorithm utilises three functions. At the start, INITIATEQUERY(v) chooses v block IDs at random[4], each is a challenge asking the prover for the corresponding data block. The challenge are then sent one-by-one. The function REQUEST(q_i) is an interaction between $\mathcal{V}$ and $\mathcal{P}$ whereby $\mathcal{V}$ issues the challenge q_i, and waits for the requested block f_i from the prover. If the prover provides an invalid response (to be checked by the function ISVALID(sk, f_i)), the verifier determines that the response is forged and rejects the audit. On the other hand, if all of the responses are valid, the verifier will rely on the number of late responses (w.r.t the latency threshold d) to make the decision. If such number exceeds the late delivery threshold l, $\mathcal{V}$ rejects the audit.

[4]This is not inconsistent with the description of the Audit phase in Section IV. Since the block IDs are chosen at random and independent of one another, choosing them all at once or at different times would not affect the randomness. In addition, note that there is no sending and receiving of the special symbol $\perp$, in this residency checking procedure, the next challenge can only be sent after response for the previous challenge is received.

Algorithm 1 Residency Checking

```
1:  procedure RESIDENCYCHECKING(v, d, l)
2:      Q ← INITIATEQUERY(v)
3:      late ← 0; forged ← false;
4:      for each q_i ∈ Q do
5:          t_i, f_i ← REQUEST(q_i);
6:          if ISVALID(sk, f_i) then
7:              if t_i > d then
8:                  late ← late + 1;
9:              end if
10:         else
11:             forged ← true;
12:         end if
13:     end for
14:     if forged or (late > l) then
15:         Reject;
16:     else
17:         Accept;
18:     end if
19: end procedure
```

The parameters are chosen to meet the security and performance requirements, in particular the false acceptance rate (ψ), false rejection rate (γ) and total file expansion factor (h). The three parameters b, c and s are to be decided during the setup phase. The audit size v and the two thresholds d, l can be determined during the audit phase, or predetermined so that they are the same for all audit sessions. The parameters setting also depends on the environment profile $\mathcal{E}$. In practice, $\mathcal{V}$ can obtain information on $\mathcal{E}$ using various mechanisms, depends on the implementation details. We shall discuss two variants in the following.

C. Network-based Implementation (N-RESCHECK)

The first implementation variant, N-RESCHECK, assumes that the verifier $\mathcal{V}$ and the prover $\mathcal{P}$ communicate over the network. In this variant, the environment profile $\mathcal{E}$ contains information on network status. This information can be obtained using various tools and techniques [7], [13], [31]. The latency observed by $\mathcal{V}$ accounts for the data fetching time and challenge-response transmission time.

Recall that guaranteeing the delivery of challenges and responses is necessary, for $\mathcal{P}$ has to respond to every challenge. Our implementation employs the reliable TCP [38] for

transmission of challenges and responses, although it incurs higher latency variance in comparison with other protocols such as UDP [37], which suffers from packet loss. We note that it is possible to design a residency checking protocol which supports packet loss of known rate, to which UDP can be adopted, so as to lower the latency variance. However, as discussed previously, such variant introduces difficulties in differentiating dishonest prover who relocates the data from the one who discards some of the blocks.

The communication cost of N-RESCHECK is reasonable. For a residency checking of size v on $\widetilde{F}$ which consists of m s-byte blocks, the overall communication cost is $(8s + \log m) \times v$ *bits*. As we shall show later in Section VII, an optimal choice of block size is 64 bytes and that of challenge size v ranges from 250 to 400. With these parameters, the overall communication cost for verifying the residency of a 1GB file is only a few KBs.

Limitations. The assumption that N-RESCHECK makes on the ability of $\mathcal{V}$ to obtain information regarding the network status at audit time may not always be feasible. In addition, the measured latency inevitably includes the transmission time of the challenges and responses, adding noise to the measurement and thus having a certain impact on the security of the residency checking. To mitigate these limitations, we discuss in the next section another implementation variant relying on a presence of a trusted unit co-locating with the drives on the storage server. Such trusted unit can be provisioned by various implementation mechanisms, for example by utilizing the recently released Intel SGX processors [12].

D. Trusted computing-based Implementation (E-RESCHECK)

The second implementation – E-RESCHECK – entrusts a trusted unit on the prover's storage server (i.e. the trusted unit and the drives are both installed on the same server) to carry out the residency checking. Such unit is responsible for provisioning a protected execution environment (aka enclave). Hereafter, we shall refer to this protected enclave as *Verifying Enclave (*VE*)*. The prover can neither tamper with VE operations, nor change the code and data loaded to it without being detected by $\mathcal{V}$. However, $\mathcal{P}$ can supply inputs for VE.

In E-RESCHECK, the environment profile $\mathcal{E}$ contains information on house keeping operations at the OS level on $\mathcal{P}'$s server, which arguably can be accurately estimated. VE, representing the verifier, conducts a residency checking as specified in Algorithm 1. Unlike the previous variant, the latency measured by VE accounts only for the fetching time of the prover, excluding altogether the network time required for transmission. Without the potentially noisy factor, E-RESCHECK offers more reliable measurements, and thus is more secure.

We treat the trusted unit as an abstraction so that it can be realised by various mechanisms. Our implementation provisions VE using Intel Skylake processors with SGX Enabled BIOS support [12]. The code running inside VE – the verification code – can be written by $\mathcal{V}$, or by any other party but vetted by $\mathcal{V}$. In another word, the verifier can be assured of the correctness of the verification code. Once VE is created, the verification code is loaded into VE by an untrusted party (e.g. $\mathcal{P}$). While the untrusted party may load a wrong code into the enclave, there exists an attestation mechanism [15] allowing $\mathcal{V}$ to check if the correct code is loaded. Moreover, this attestation mechanism also allows the verifier and VE to establish shared secrets, which enable secure channel for their communication (e.g. for the verifier to sends the secret key to VE or for VE to send the residency checking result).

The Intel SGX specifications are well aligned with our protocol and threat model. In specific, enclaves cannot directly access OS-provided services (which are not trusted in the thread models of SGX). They need to make OCall to an interface routine to asks the untrusted application to handle those services [9]. In our context, the fetching of the requested block is performed by the prover, who is also untrusted. The VE issues a query for a requested block by making an OCall to the prover's untrusted application, which then retrieves the block and makes an ECall to pass it as input parameter to VE. Since this ECall is invoked by the untrusted party, the verification code needs to be written with care so that no attack window is exposed. We refer readers to Intel SGX programming reference for further details on coding guideline for programming enclave code [10], [9].

The response latency is measured by VE as a duration between an OCall and the corresponding ECall which passes the requested block into VE. While getting a trusted source for absolute time in SGX is challenging, there exists mechanisms to measure relative time with respect to a reference point [47]. We note that absolute time is not necessary in our setting, since what VE measures is an elapsed time between to particular moments, to which relative time with respect to the same stable reference point is sufficient.

Effect of block size on security. We highlight the effect of the block size on the overall security. Rotational drives, in general, are partitioned into sectors of 512 bytes[5]. These sectors are physically aligned on the hardware device. When a data block is written to disk, it may span across multiple sectors, which are not necessarily physically contiguous. Reading such data block may require multiple seeks, depending upon the (relative) position of the sectors on the disk. This results in substantial variance in atomic fetching time. On the other hand, if the data block fits entirely in one physical sector, only a single seek is required and thus the atomic fetching time is less varied. To eliminate noise in timing measurements, it is desired to have blocks of small size so that each data block fits in a physical disk sector with high probability. We exhibit the implication of the block size on security in greater details in Section VII. Previous works on data geolocation [20], [28]

[5]Although hard drives with Advanced Format (AF) are divided into sectors exceeding 512 bytes, we shall rely on the 512-byte sector format. The security of our model is not affected when the storage devices use the advanced format. However, if the protocols is designed with AF sector size, the security becomes malleable on system equipped with legacy 512-byte sector-based HDDs.

did not take into consideration mechanisms and behaviours of storage hardware with respect to the block size, resulting in an oversight of the strong affect that the block size has on the protocol's security.

E. Security Analysis

The level of false acceptance/rejection rate of the proposed protocol depends on various parameters, including the environment profile $\mathcal{E}$, the audit size v (number of challenges), the bit length of the MACs b, the expansion rate of the error-erasure code c, and the two thresholds d and l. Also recall that during the setup phase, the original data F of n blocks is encoded into $\widetilde{F}$ of $(1+c)n$ blocks such that knowledge of any n encoded blocks is sufficient to reconstruct F.

False acceptance rate. Let us consider an adversary $\mathcal{A}$ who keeps $n-1$ blocks of $\widetilde{F}$ on its local drives and the remaining blocks in a remote storage. We denote the first portion by D, and the other by $\widetilde{D}$. Clearly, the original file F cannot be reconstructed from D. We want to determine the acceptance rate of this adversary, which in turn gives a bound on the false acceptance rate.

Consider a challenge q_i asking for a block f_i, we say that it is a *hit* if one of the following two conditions holds:

1) f_i is in $\widetilde{D}$ and the latency $t_i^{\text{net}} + t_i^{\text{rmt}} > d$;
2) f_i is in D and the latency $t_i^{\text{net}} + t_i^{\text{loc}} > d$,

where t_i^{net} is the transmission time of the q_i and f_i, t_i^{loc} is the fetching time of f_i if it is stored locally, and t_i^{rmt} is its fetching time if stored remotely.

For a challenge that is a hit, the adversary has two choices. If the adversary chooses to load the response from the storage, then the response will certainly arrive late and contribute one count towards the number of late responses permitted by the late delivery threshold l. On the other hand, if the adversary chooses to forge the response, then the probability that the response is valid is $2^{-b} + \mu(\lambda)$ where $\mu()$ is some negligible function on the security parameter λ. Note that it is possible that the transmission time t_i^{net} already exceeds the threshold d, and thus the response will definitely be late even if the adversary chooses to forge the response.

Let *Hit* be the number of hits among the v random challenges. Given the set of hit challenges, the adversary $\mathcal{A}$ chooses l of them where the responses are to be computed from the storage, and the rests by forging. In choosing this set of l hits challenges, the adversary will first select those whose transmission time already exceeds l. For the remaining $\textit{Hit} - l$ challenges, $\mathcal{A}$ forges the response. Note that such $\mathcal{A}$ is optimal in the sense that all other choices lead to a lower or equal probability of acceptance.

The probability that a challenge is hit can be derived from the environment profile $\mathcal{E}$, the latency threshold d and the expansion rate c. Clearly *Hit* follows binomial distribution. Furthermore, the probability that all x forged responses are valid is $2^{-bx} + \mu(\lambda)$. Hence, the probability that $\mathcal{A}$ being accepted is at most

$$\Pr(\textit{Hit} \leq l) + \sum_{x=1}^{v-l} \Pr(\textit{Hit} = x + l) \cdot (2^{-bx} + \mu(\lambda))$$

The above is not an equality because we omit the cases where more than l of challenges already have transmission time exceeding d. Moreover, although the derivation is based on a specific adversary $\mathcal{A}$, it also serves as an upper bound of the false acceptance rate. There is no lost of generality since, if the original file F cannot be constructed from D, then there is no more than $n-1$ blocks in D.

The cache C (see Section IV-B) kept by the prover could have some, but minor, effects on the false acceptance rate. This is because the number of challenge collisions (i.e. two challenges asking for the same block) is small, due to the short audit session, and the fact that challenges are randomly generated. Hence, in this security analysis, we ignore the effect of the cache and only consider setting where the cache size is 0.

False rejection rate. It is easy to derive the probability γ whereby the honest prover, who keeps all the data locally, fails to pass the verification. Let α be the probability the requested block arrives later than the threshold d (i.e. $t_i^{\text{net}} + t_i^{\text{loc}} > d$). Hence, the false rejection rate is:

$$\gamma = \sum_{j=l+1}^{r} \binom{r}{j} \alpha^j (1-\alpha)^{r-j}$$

False acceptance rate of PoR. For comparison, we consider the false acceptance rate ϵ_{DL} of an adversary who keeps only $n-1$ blocks in local storage and discards the rest:

$$\epsilon_{DL} \leq \left(\frac{2^b + c}{(1+c)2^b} \right)^r + \mu(\lambda)$$

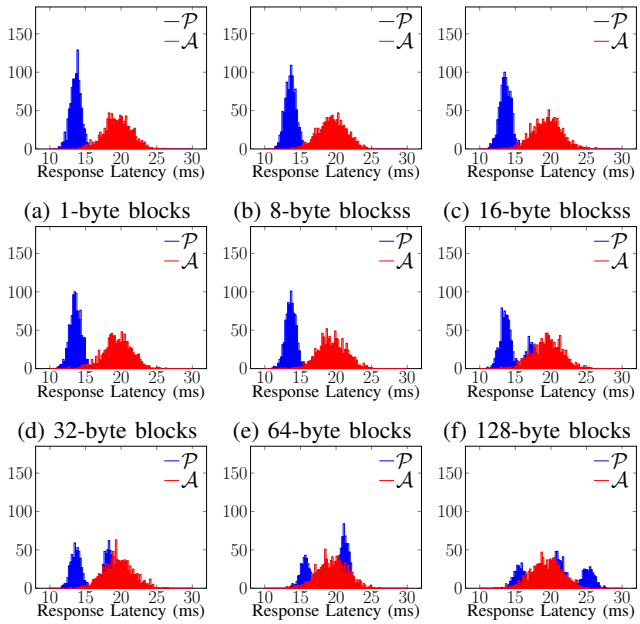
Hence, if the integrity of the data is compromised, it will be detected with an overwhelming probability $1 - \epsilon_{DL}$.

VII. Evaluations

In this section, we conduct experimental studies to evaluate the performance and security of our residency checking construction. In details, we investigate the effect of block size s, the MAC length b, the audit size v and choice of the late delivery threshold l on the false acceptance and false rejection rates ψ and γ, respectively.

A. Setup

In our experiments, the honest prover $\mathcal{P}$ stores the data as a whole in its local drives, while the dishonest prover $\mathcal{A}$ relocates the data blocks by splitting large blocks whose size are larger than 64 bytes into 64-byte segments and

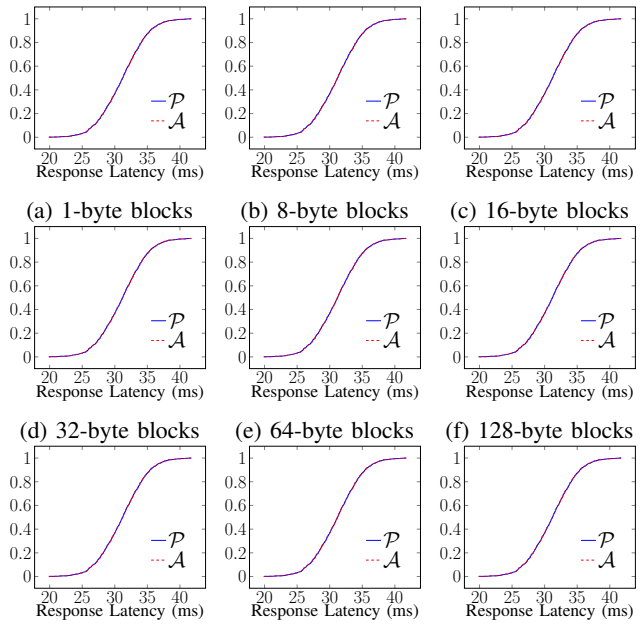


(a) 1-byte blocks (b) 8-byte blockss (c) 16-byte blockss

(d) 32-byte blocks (e) 64-byte blocks (f) 128-byte blocks

(g) 256-byte blocks (h) 512-byte blocks (i) 1024-byte blocks

Fig. 5: Histograms of the response latencies incurred by honest prover $\mathcal{P}$ and dishonest prover $\mathcal{A}$ in E-RESCHECK with respect to different block sizes. $\mathcal{P}$ stores the blocks as-is in its local drives, whereas $\mathcal{A}$ splits large blocks into 64-byte segments and stores all the data at remote storages.



(a) 1-byte blocks (b) 8-byte blocks (c) 16-byte blocks

(d) 32-byte blocks (e) 64-byte blocks (f) 128-byte blocks

(g) 256-byte blocks (h) 512-byte blocks (i) 1024-byte blocks

Fig. 6: CDFs of response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in E-RESCHECK with respect to different block sizes.

distributing them to remote storage servers[6], and retrieves them (simultaneously if possible) upon $\mathcal{V}$'s requests.

All experiments are conducted on Ubuntu 14.04 commodity systems equipped with quad-core Intel Skylake processors with SGX Enabled BIOS support, 1TB hard drives with I/O latency ranging from 12-15ms on average and 1GB Ethernet cards. $\mathcal{P}$ and $\mathcal{A}$ are represented by different programs running on the same physical system. In N-RESCHECK, the provers and the verifier are located across countries[7], while in E-RESCHECK, the verification enclave VE resides on the same physical system which hosts $\mathcal{P}$ and $\mathcal{A}$. VE is provisioned using Intel SGX SDK for Linux [11]. Unless stated otherwise, all experiments are repeated for 100 times and the average results are reported.

B. *Effect of block size (s)*

As discussed earlier, a large block may be scattered across physical disk sectors, leading to higher and more varied fetching time. Even worse, this potentially exposes an attack vector whereby the adversary splits a large block into several smaller segments so that it can retrieve them simultaneously in an attempt of speeding up the fetching time. On the other hand, unnecessarily small block size implies more blocks to be handled, resulting in more involved housekeeping operations and extra storage overhead incurred by the authentication tags. In this first set of experiments, we would like to confirm the effect of block size on security of our constructions, and investigate the optimal block size for efficiency.

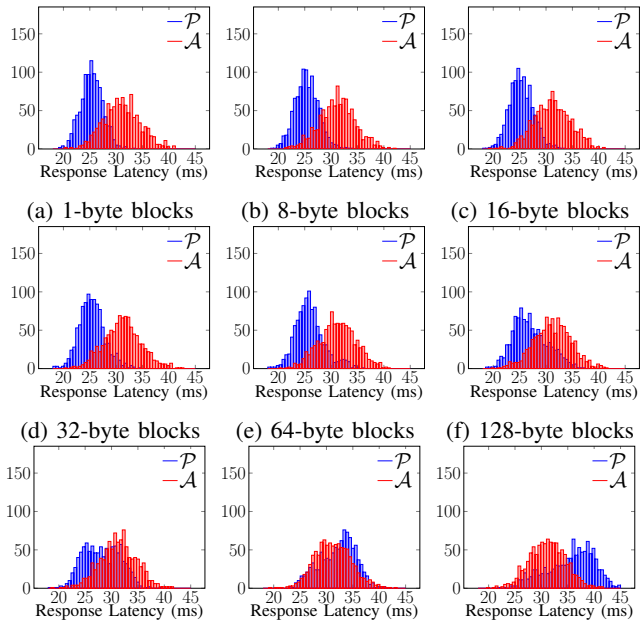
We vary the block size from one to 1024 bytes and measure the response latencies incurred by an honest prover $\mathcal{P}$ and an adversary $\mathcal{A}$. We report the results in Figures 5, 6 for E-RESCHECKand 7, 8 for N-RESCHECK.

Figure 5 shows histograms of 1000 response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in E-RESCHECK, with respect to different block sizes. As can be seen, the block size has great implication on fetching time. To be more specific, when the block size ranges from one to 32 bytes, the fetching times of $\mathcal{P}$ follow a normal distribution with mean 12.93ms and standard variation of 0.73. As the block size increases, blocks are more likely to span across physical sectors, increasing the variance in fetching time. Figure 5f, 5g and 5h suggest that the fetching times for blocks of sizes 128 to 512 bytes can be classified into two groups, each follows a normal distribution with different mean (12.93ms and 19.03ms where block size varies from 64 to 256 bytes, and 12.93ms and 21.52ms with 512-byte blocks). This is even worse when the block size is increased to 1024-bytes. The fetching times for 1024-byte blocks are divided into three different groups (Figure 5i). Recall that the fetching times are desired to be uniform, so as to have a reliable latency assessment. Given such requirement, large blocks size (e.g. larger than 64 bytes) are clearly not suitable for security in our protocols.

The response latencies of $\mathcal{A}$ follow a normal distribution, with mean 19.58ms and standard variation 2.71ms. For blocks that are larger than 64 bytes, $\mathcal{A}$ splits them to 64 byte segments

[6]Average round-trip time of transmitting a 64-byte packet between $\mathcal{A}$ and these servers is 6.5ms.

[7]Average round-trip time of transmitting a 64-byte packet between them is 12.7ms.

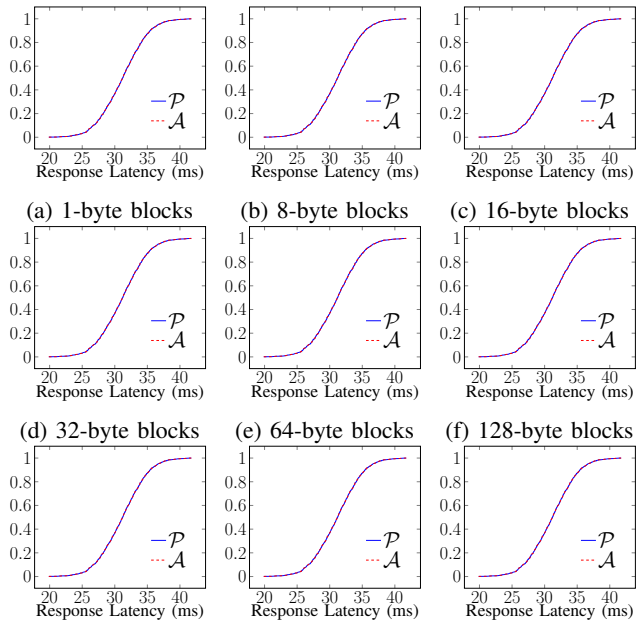


(a) 1-byte blocks (b) 8-byte blocks (c) 16-byte blocks

(d) 32-byte blocks (e) 64-byte blocks (f) 128-byte blocks

(g) 256-byte blocks (h) 512-byte blocks (i) 1024-byte blocks

Fig. 7: Histograms of response latencies incurred by honest prover $\mathcal{P}$ and dishonest prover $\mathcal{A}$ in N-RESCHECK with respect to different block size. $\mathcal{P}$ stores the blocks as-is in its local drives, while $\mathcal{A}$ splits large blocks into 64-byte segments and stores all the data at remote storages.

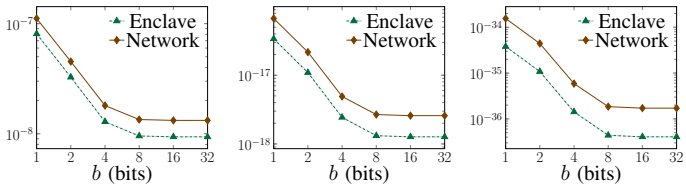


(a) 1-byte blocks (b) 8-byte blocks (c) 16-byte blocks

(d) 32-byte blocks (e) 64-byte blocks (f) 128-byte blocks

(g) 256-byte blocks (h) 512-byte blocks (i) 1024-byte blocks

Fig. 8: CDFs of response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in N-RESCHECK with respect to different block sizes.



(a) c = 10% (b) c = 20% (c) c = 40%

Fig. 9: Effect of MAC length on false acceptance rate. The audit size is set to $v = 300$ challenges, and late delivery threshold (l) set to eight.

and retrieves them in parallel in order to speed up the fetching time. This explains why $\mathcal{A}$'s fetching times for large blocks are not divided into different groups as those of $\mathcal{P}$. Comparing across figures, one can see that as the block size increases, differentiating latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ becomes more problematic, potentially resulting in higher false acceptance and rejection rates. It even seems impossible when 1024-byte blocks are used.

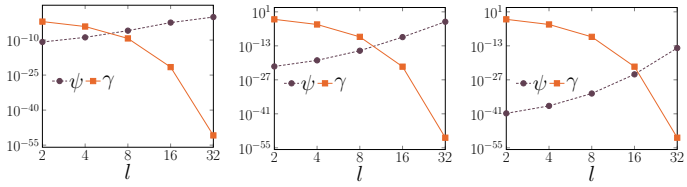
To get a better intuition on the effect of the block size on the ability to distinguish $\mathcal{P}$ and $\mathcal{A}$ based on response latencies, we show in Figure 6 CDFs of their response latencies in E-RESCHECK, also with respect to different block sizes. As the block size approaches 512 bytes, CDFs of $\mathcal{P}'$s response latencies stop dominating those of $\mathcal{A}$, suggesting complications in distinguishing the latencies of the honest prover from those of the adversary.

The effect of the block size on the response latency in N-RESCHECK (depicted in Figure 7) is also noticeable, but not as evident as in E-RESCHECK. This is because the response latencies observed by $\mathcal{V}$ are the accumulation of fetching time and challenge-response transmitting time. The latter component is relatively similar for all the block sizes considered in our experiments. The response latencies of $\mathcal{P}$ follow normal distributions, with means ranging from 24.32ms to 30.34ms and standard variation varying from 1.81ms to 2.52ms depending on the block sizes, while response latencies of $\mathcal{A}$ follow normal distribution with mean approximately 31.22ms and standard variation oscillating around 3.18ms. Similar to the E-RESCHECK, it is harder to distinguish honest and dishonest provers based on the response latencies as the block size increases, and becomes impossible when the block size reaches 512 bytes.

We demonstrate in Figure 8 CDFs of $\mathcal{P}$ and $\mathcal{A}$'s response latencies. Figures 8h and 8i especially show CDFs of the adversary's response latencies dominating those of the honest prover, implying it has significant advantage in disguising its response latencies and thus its behaviours as honest one. From the results of this experiment set, it is apparent that the block size has strong impact on the security of our protocols. A too large block size would lead to failure in detecting adversarial behaviours. We recommend the block size of 64 bytes for both E-RESCHECK and N-RESCHECK, and use this block size in all subsequent experiments.

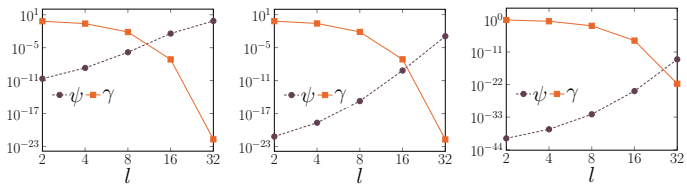
C. Effect of MAC length (b)

In the second set of experiments, we examine the effect of MAC length on the false acceptance rate ψ, with respect



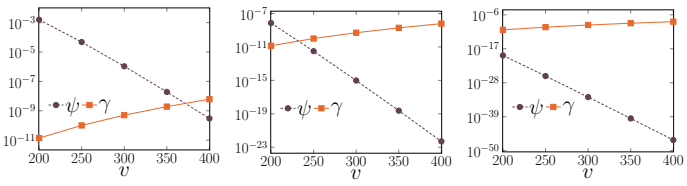
(a) c = 10% (b) c = 20% (c) c = 40%

Fig. 10: Effect of the late delivery threshold l on the security in E-RESCHECK. MAC length is set to eight bits, and audit size is $v = 300$ challenges.



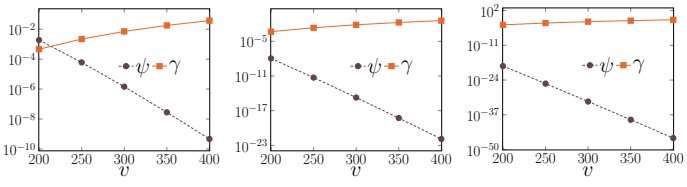
(a) c = 10% (b) c = 20% (c) c = 40%

Fig. 11: Effect of the late delivery threshold l on the security in N-RESCHECK. MAC length is set to eight bits, and audit size is $v = 300$ challenges.



(a) c = 10% (b) c = 20% (c) c = 40%

Fig. 12: Effect of audit size v on the security in E-RESCHECK. MAC length is set to eight bits, and late delivery threshold is set to eight.



(a) c = 10% (b) c = 20% (c) c = 40%

Fig. 13: Effect of audit size v on the security in N-RESCHECK. MAC length is set to eight bits, and late delivery threshold is set to eight.

to different values of c ranging from 10% to 40%. The late delivery threshold is set to five, and the audit size is $v = 300$ challenges.

Figure 9 shows the experiment results. ψ drops exponentially – by at least an order of magnitude – when b increases from one to four bits. To be more specific, ψ reduces from 10^{-7} to 10^{-8} when $c = 10\%$ in N-RESCHECK or 3.8×10^{-35} to 1.4×10^{-36} in E-RESCHECK when $c = 40\%$. The reduction becomes less evident as b approaches eight bits. Further increasing b does not result in better false acceptance rate. With eight bits MAC and the block size is set at 64 bytes as suggested in the previous set of experiments, the expansion rate due to authentications tags is as small as 1.5%.

In addition, when comparing across figures, it is evident that larger error-erasure code's expansion rate leads to lower false acceptance rate and hence better security. It is also worth noting that the false acceptance rates of E-RESCHECK are consistently smaller than those of N-RESCHECK, suggesting E-RESCHECK offers better security guarantee.

We also note that the use of short authentication tags (e.g. eight bits MAC) does not compromise the ability to detect an adversary who incurs data loss (i.e. keeping less than n data blocks). For example, with parameter setting of $c = 40\%$, $v = 300$ and eight bits MAC, the probability that such adversary escapes the detection is less than 2^{-142}.

D. Effect of late delivery threshold (l)

In the third set of experiments, we fix the MAC length at eight bits, the audit size at $v = 300$ challenges, and investigate the effect of late delivery threshold l on false acceptance rate ψ and the false rejection rate γ.

Figure 10 shows the results for E-RESCHECK, while Figure 11 for N-RESCHECK. As l increases from two to 32, the false rejection rate γ drops exponentially – by upto 22 orders of magnitude for N-RESCHECK and almost 50 orders of magnitude for E-RESCHECK. This suggests it is possible to make the scheme more tolerable to environment noise.

However, increasing l leads to the growth of the false acceptance rate ψ. For both implementations, ψ grows by eight to 16 orders of magnitude when l increases from two to 16, depending on the code rate of the error-erasure code in use. For both implementations, when $c = 10\%$ and l is set to 32, ψ raises upto 0.7.

As observed in our experiments, we suggest l be set to eight, attaining γ as small as 5×10^{-10}, while still keeping ψ smaller than 10^{-6} even for $c = 10\%$. We note that for the same parameter settings, ψ drops exponentially when c increases. For examples, in E-RESCHECK, ψ reduces by upto 30 orders of magnitude when c increases from 10% to 40%.

E. Effect of audit size (v)

In the last set of experiments, we study the effect of audit size v on the overall security. We fix the MAC length at eight bits, late delivery threshold is eight and examine how v effects ψ and γ. The results are reported in Figure 12 (for E-RESCHECK) and 13 (for N-RESCHECK). For E-RESCHECK, ψ reduces by eight to 26 orders of magnitude when v varies from 200 to 400. Likewise, the reduction in N-RESCHECK is similar. This suggests that we can make the false acceptance rate ψ arbitrarily small by increasing the audit size (i.e. issuing more challenges). Though expanding the audit size leads to larger communication costs in N-RESCHECK, the actual increase is only in KBs, which is reasonable. Note

that E-RESCHECK does not require transferring the challenges and responses over the network, thus incurring no network communication overhead.

Nevertheless, we observe that as v expands from 200 to 400 challenges, γ increases from 1.3×10^{-11} to 6.1×10^{-9} (almost $450\times$) in E-RESCHECK and from 4.6×10^{-4} to 3.7×10^{-2} (by $80\times$) in N-RESCHECK. While the increment of γ in E-RESCHECK is much larger than that in N-RESCHECK, the former witnesses the false rejection rate of only 6.1×10^{-9}, several order of magnitude smaller than the corresponding value of the latter. The reason for such increases is because larger audit size leads to greater exposure to the environment noise; and the noise introduced by network transmission is much greater than that of the house keeping operations at OS level in the E-RESCHECK.

It is evident across all experiments that E-RESCHECK is superior to N-RESCHECK. It offers better false acceptance and rejection rates, incurs no network communication overhead, and is less exposed to the environment noise.

VIII. RELATED WORKS

Proofs of Retrievability. Proofs of retrievability were first proposed by Juels and Kaliski [32], and have been followed by various works [42], [26], [43], [22], [44], [45]. While these works address similar problems – auditing a remote and untrusted storage server on data preservation – they differ in their security models. A closely related technique is PDP, initially discussed by Ateniese *et al.* [17]. While PDP assures that most (but not necessarily all) of the data are stored, PoR offers a stronger guarantee which enables a feature called *extraction*, allowing the data owner to retrieve her file in its entirety. Related to these two notions are *memory checking* [21] by Blum *et al.* and *sublinear authenticators* by Naor and Rothblum [34]. Later on, the notions of PoR and PDP are also extended to dynamic settings [43], [27]. While there are various efficient constructions in the literature [32], [42], none of them has taken the location of data into consideration. Our notion of PODR attains a proof that the original file F is retrievable in its entirety from data stored locally at the storage provider's server.

Timed Challenge-Response Protocols. Timed challenge-response protocols have been studied in various application scenarios. Bowers *et al.* [23] present a remote assessment of fault tolerance based on measuring the time taken by a server to respond to a read request for a collection of file blocks. In such an assessment, it is assumed that network latency can be accurately estimated and deemed as a constant. Our model makes a more reasonable assumption that the network latency is probabilistic, only its distribution can be determined.

Benson *et al.* [20] and Gondree *et al.* [28] discuss techniques to bind the storage to a geographical location, and discuss various feasibility issues. Gondree *et al.*[28] propose a framework that employs a set of known landmarks to verify the storage geolocation. Benson *et al.* investigate the correlation of network latency and geographical distance, and suggest the use of such technique in verifying whether data are indeed replicated in geographically separated datacenters. Our construction, on the other hand, focuses on verifying whether the original data can indeed be reconstructed from local drives of the server-in-question. Another difference is that while those proposals advocate minimising server-side computation due to practical concerns on usability and for cost-saving, we discuss such requirement from security perspective. Further, we stress the impact of block size on the security of the protocol, which has not been studied in previous works.

Locality of Storage. Incentives for storing data locally have also been discussed by recent new cryptocurrencies proposals [33], [41]. Essentially, these proposals require miners to construct a proof of retrievability during the mining, which in turn is designed to encourage miners to store data locally as opposed to outsourcing them to a remote storage. While these works share with ours a concern on storage location, they only incentivise local preservation of the data but do not enforce such requirement. PODR, on the other hand, imposes local preservation of the data and offers an auditing mechanism to detect provers who do not follow the stipulation.

Protected Execution Environment. Various works have relied on trusted computing to provision the protected execution environment for secure services, such as query processing [18], [16], MapReduce computation [25] and data analytic [40]. By making a realistic assumption on the presence of the trusted environment, these works are able to offer security with efficiency and at scale. Besides the trusted execution environment, our construction employs a co-location of verifier and the prover, which is made feasible by trusted computing primitives, to enhance security.

IX. CONCLUSION

We have defined the security definition of Proofs of Data Residency. PODR enables the data owner to obtain a proof that the file F is retrievable in its entirety from local drives of a storage server in-question. PODR can be an integral component in auditing contractual assurances. In particular, it can be combined with host geolocating to affirm geolocation of the data, or utilised to access fault tolerance of a storage system, by checking the residency of the files at different separate storage servers. We show potential attacks on insecure constructions and propose a secure PODR scheme. The two implementations of the proposed construction, N-RESCHECK and E-RESCHECK, illustrate an interesting use-case of trusted computing, where having the verifier of a cryptographic protocol co-locating with the prover enhances the security.

The focus of this paper has been on static settings where the data owner does not frequently update F. It would be an interesting future work to extend our construction to support dynamic data updates.

REFERENCES

[1] "Amazon EC2 instances," https://aws.amazon.com/ec2/instance-types/.
[2] "Amazon S3," https://aws.amazon.com/s3/.
[3] "Australian privacy act," http://www.austlii.edu.au/au/legis/cth/consol_act/pa1988108/.

[4] "Business Insider. Amazon′s cloud crash disaster permanently destroyed many customers data," http://www.businessinsider.com/amazon-lost-data-2011-4?IR=T&r=US&IR=T.
[5] "Data protection directive," http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=URISERV%3Al14012.
[6] "Google Drive," https://www.google.com/drive/.
[7] "hping," http://www.hping.org/.
[8] "Intel SGX," https://software.intel.com/en-us/sgx.
[9] "Intel SGX Enclave Writer's Guide," https://software.intel.com/sites/default/files/managed/ae/48/Software-Guard-Extensions-Enclave-Writers-Guide.pdf.
[10] "Intel SGX programming reference," https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.
[11] "Intel SGX SDK for Linux," https://github.com/01org/linux-sgx.
[12] "Intel Skylake processor," http://ark.intel.com/products/codename/37572/Skylake.
[13] "PsPing," https://technet.microsoft.com/en-us/sysinternals/psping.aspx.
[14] "Traceroute," http://linux.die.net/man/8/traceroute.
[15] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in *HASP*, 2013.
[16] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase," in *CIDR*, 2013.
[17] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *CCS*, 2007.
[18] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," in *TKDE*, 2014.
[19] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," in *OSDI*, 2014.
[20] K. Benson, R. Dowsley, and H. Shacham, "Do you know where your cloud files are?" in *CCSW*, 2011.
[21] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," *Algorithmica*, 1994.
[22] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *CCSW*, 2009.
[23] K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "How to tell if your cloud files are vulnerable to drive crashes," in *CCS*, 2011.
[24] G. Connolly, A. Sachenko, and G. Markowsky, "Distributed traceroute approach to geographically locating ip devices," in *IDAACS*, 2003.
[25] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang, "M^{2}R: Enabling stronger privacy in mapreduce computation," in *USENIX Security*, 2015.
[26] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Theory of cryptography*, 2009.
[27] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS*, 2009.
[28] M. Gondree and Z. N. Peterson, "Geolocation of data in the cloud," in *CODASPY*, 2013.
[29] K. Harrenstien, M. K. Stahl, and E. J. Feinler, "NICNAME/WHOIS," *RFC-954*, 1985.
[30] C. Houri, "Method and systems for locating geographical locations of online users," 2003, uS Patent 6,665,715.
[31] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *ACM SIGCOMM*, 2002.
[32] A. Juels and B. S. Kaliski Jr, "PORs: Proofs of retrievability for large files," in *CCS*, 2007.
[33] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *IEEE S&P*, 2014.
[34] M. Naor and G. N. Rothblum, "The complexity of online memory checking," in *FOCS*, 2005.
[35] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for internet hosts," in *SIGCOMM*, 2001.
[36] Z. N. Peterson, M. Gondree, and R. Beverly, "A position paper on data sovereignty: The importance of geolocating data in the cloud." in *HotCloud*, 2011.
[37] J. Postel, "User datagram protocol," 1980.
[38] ——, "Transmission control protocol," 1981.
[39] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. SIAM*, 1960.
[40] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich, "VC3: Trustworthy data analytics in the cloud," in *IEEE S&P*, 2015.
[41] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, "Retricoin: Bitcoin based on compact proofs of retrievability," in *ICDCN*, 2016.
[42] H. Shacham and B. Waters, "Compact proofs of retrievability," *Journal of cryptology*, 2013.
[43] E. Shi, E. Stefanov, and C. Papamanthou, "Practical dynamic proofs of retrievability," in *CCS*, 2013.
[44] E. Stefanov, M. van Dijk, A. Juels, and A. Oprea, "Iris: A scalable cloud file system with efficient integrity checks," in *ACSAC*, 2012.
[45] J. Xu and E.-C. Chang, "Towards efficient proofs of retrievability," in *ASIACCS*, 2012.
[46] I. N. Yezhkova, "Worldwide and U.S. enterprise storage systems forecast update, 2015-2019. White Paper," 2015.
[47] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town Crier: An authenticated data feed for smart contracts," in *CCS*, 2016.