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 exist 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 are 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 that 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.

1. INTRODUCTION

The growth of information has made data-generation outpace storage availability [39]. This has given rise to cloud data storage models, as offered by various well-known cloud service providers [4]. Cloud storage models have gained significant popularity, and offer manifold benefits including cost-effectiveness and elasticity. They present data owners with a simple view of the outsourced files, abstracting away underlying file-layout and storage mechanisms. 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).

ASIA CCS '17, April 02-06, 2017, Abu Dhabi, United Arab Emirates

DOI: http://dx.doi.org/10.1145/3052973.3053016

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 [2]. Such a threat prompts the needs of testing for fault tolerance of the storage system [17]. In addition, various legislation and directives regulating the possessing and storage of data across national borders advocate paying attention to locations at which the files are maintained [30, 1, 3]. 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 discussed technical means to audit cloud storage providers on how the outsourced data are maintained. Bowers *et al.* tested fault tolerance of the storage system by checking if files are replicated across different drives. Gondree *et al.* [22] and Benson *et al.* [15] attempted to "geolocate" the data in the cloud. Nevertheless, due to the noisy network environment, it is still technically challenging to attain accurate and reliable assurances.

In this paper, we take a different approach to address these problems, focusing on a more modest goal of verifying *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 proof of data residency provides *more* assurance 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) [26, 13]. Further, attesting data residency can be an integral component in auditing contractual assurances, for which existing technical means appear insufficient. For instances, one can first geographically locate a storage server in-question [29, 24], and then attest the residency of the outsourced file on such server to affirm geolocation of the data. Moreover, one can also assess replications of the data at different geographically separate servers by checking the residency of the file on each of the servers simultaneously.

We formulate the notion of data residency under a security model called *Proof of Data Residency*, taking into consideration behaviours of storage devices and capabilities of dishonest storage providers (i.e. adversaries). The adversaries could potentially derive accurate estimation of the network noise, and 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 – which the prover must invoke in every challenge-response interaction – and consider a powerful adversary that can reduce processing time of any challenge to the equivalent of a single atomic fetching, and fully aware of the network noise.

We propose techniques to attest data residency. A data residency checking is structured as a *timed challenge-response protocol*. Our solutions adopt an authenticator-based PoR [28, 26] as an underlying cryptographic primitive to attest the file's retrievability, and assess the response latencies to establish the data residency. To this end, we discuss two implementation variants. The first variant, namely N-ResCheck, conducts the data residency checking over the network, while the second variant, dubbed E-ResCheck, necessitates the presence of a trusted unit on the server in-question. With recent initiatives on trusted computing primitives, especially Intel SGX technology being available on commodity systems [6], it is interesting to investigate the security of a timed challenge-response protocol wherein the verifier resides in the protected enclave on the prover's physical server.

Our study suggests two general guidelines in the design of an efficient and secure PoDR protocol. First, it is necessary to minimise the computation carried out by the prover. Preferably, during the verification process, the prover should only fetch and send data to the verifier. Previous works [22, 15] also advocated 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 cost-savings. 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. Secondly, it is crucial to lower the response latencies incurred by an honest prover. This suggests that each challenge should only ask for *one* data block and that the size of data blocks in use should be small (say 64 bytes). We empirically demonstrate that protocols which fail to adhere to these suggestions are likely susceptible to evasion. The requirement on small block size further entails the use of short authentication tags (say 16 bits) so as to keep the storage expansion factor reasonable. Readers may wonder if this will raise security concerns wherein short authentication tags are vulnerable to chosen-message attacks. We argue that these attacks are irrelevant in our application settings, for the adversary has only limited access to the verification oracle. Moreover, one can always reduce the probability of successful attacks by increasing the number of challenges (see Section 7.5). We elaborate on effects of block size and authentication tag length in Section 7.

Our empirical studies show that for insecure constructions, the adversary can evade detection. The experiment results on our proposed solution support the need of small block size (e.g., 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 16 bits. The experiments also demonstrate significant security improvement obtained by incorporating trusted computing. In particular, for the same performance requirements of 24% storage expansion (among which 21% due to error-erasure code and another 3% due to authentication tags) and audit with 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 to N-ResCheck. This illustrates an interesting use-case of trusted computing where having the verifier of a cryptographic protocol co-locating with the prover enhances security.

In summary, our paper makes the following contributions:

- We present the 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 fully aware of the network noise.
- 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 2 before stating our problem in Section 3. Next, we present our definition of Proofs of Data Residency in Section 4. We discuss potential attacks on insecure constructions in Section 5 and propose a secure protocol in Section 6. Experimental evaluation is presented in Section 7 while related works are surveyed in Section 8. Finally, we conclude our work in Section 9.

2. PRELIMINARIES

In this section, we briefly provide background on the related notions of PoR and host geolocation, as well as summarize key characteristics of Intel SGX technology. We defer more detailed discussion of these notions to Appendix B.

Proof of retrievability [26] enables data owners to audit the storage server on the intactness of their outsourced files. Prior to outsourcing the data to the cloud, the data owner encodes her original data using a redundant encoding (such as the error-erasure Reed-Solomon code [33]), and authenticates all the blocks of the encoded data. In order to assert the retrievability of her data, the data owner engages the storage provider in a challenge-response protocol, checking for the authenticity of λ blocks, where λ is a security parameter. Due to the redundant encoding, the storage provider has to discard or tamper with a considerable portion of the blocks to cause data loss. Should such incident happen, it will be detected by the verifier's "spot-checking" with overwhelming probability.

The notion of data residency implicitly requires knowledge of the storage server's geographic location. Host geolocation techniques [29, 24, 18] enable the verifier to obtain such information. One approach is to match the intermediary nodes in the routing information of a packet against those of the backbone Internet providers (whose geolocation is known) to locate a target host (the destination of the packet in question) [18]. Other approaches rely on latencies in transmitting a packet between a pair of hosts to approximate geographical distance among them, or a combination of partial IP-to-location and BGP prefix information to derive the target host's location [29].

Intel SGX [6] is a set of extensions that provision protected execution environments (aka trusted environments or enclaves). The trusted processor preserves the confidentiality and integrity of the code and data loaded into the enclave against the untrusted OS or any other processes/software by blocking any non-enclave code's attempt to read or write the enclave's memory.

3. THE PROBLEM

3.1 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 are 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 other remote servers. 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. 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 [26, 36], 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.

3.2 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 subject 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 includes 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 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 $\mathcal{E}$.

3.3 Threats Model: Adversary's capability

We consider an adversary that is a dishonest prover, having complete control over the storage, server and network within its premises. That is, the adversary is able to reduce the response latency in various fashions:

Computation time. The adversary could speedup the computation time, for example via over-clocking or parallelism. Since it is difficult to bound the speedup factor which the adversary can possibly achieve, we consider an adversarial model wherein the computation time is not included in the response latency for worst-case analysis. Note that this does not imply arbitrary computation speedup by the adversary, and we still require the prover $\mathcal{P}$ to be polynomial time.

Fetching time. When the data are 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, the adversary may be able to get a good estimate of the actual measurements. Such knowledge can help the adversary in increasing the chance of evasion. For example, let us consider an adversary who keeps half of the data blocks in local drives, and stores the rest of the data blocks in a 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 adversary knows the actual measurements of all timings right in the beginning of the verification session.

3.4 Threats Model: Adversary's limitations

Limited access to verification oracle. The adversary attempting to forge the authentication tags could exploit the verifier as the verification oracle. We assume that the data owner could tolerate a few delayed or missing responses to, she would not accept invalid responses whose authentication tags do not checkout. While the adversary may argue that a response is not valid due to hardware failure, such event is highly unlikely, for most storage and networking system have error detection/recovery mechanisms in-place. Therefore, the adversary has few chances of 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 small block size, which entails the use of short authentication tags (say 16 bits per tag) in order to keep the storage expansion factor reasonable. While short authentications are vulnerable to chosen-message attacks wherein the adversary has unlimited accesses to the verification oracle [36], such attacks are irrelevant in our application settings, for the adversary has only limited access to the verification oracle

Atomic operation. A key assumption in our work is that of an atomic operation, which would take longer time when the data are stored remotely compared to when they are stored on the storage server's local drives. This assumption in-turn is based on a premise that there are no technically feasible and/or economically feasible means for the dishonest servers to reduce the time. In cases where the above mentioned premise is not met, unfortunately, the assurance provided by our schemes will not hold. Examples of those cases include an adversary 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 would 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).

4. PROOFS OF DATA RESIDENCY

4.1 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 the 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 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. Generate and send the next challenge to $\mathcal{P}$. $\mathcal{V}$ can choose not to send any challenge.
  2. Generate and send to itself the special symbol $\perp$ which will arrive at a time specified by $\mathcal{V}$. $\mathcal{V}$ can choose not to send the symbol $\perp$[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. We 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.

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

4.2 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 are loaded from the remote storage. The environment profile $\mathcal{E}$ is a description of the distributions 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 a 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 in their access to D and $\widetilde{D}$: R only reads from the local drives, $\widetilde{R}$ reads from both local and remote storages, and $\hat{R}$ does not read from any storage.

[1]Recall the next challenge can only be sent upon receipt of the response of the previous one or the special symbol $\perp$, this allows $\mathcal{V}$ to send the next challenge without waiting for the response.

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 algorithm it would use in preparing the response. Nevertheless, it cannot speed-up the timings further than what dictated by $\mathcal{E}$. The cache C is updated after every the response.

Remarks. The above formulation implies a strong adversary that (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 an atomic loading operation[2]; and (3) is able to arbitrarily delay the response. As discussed in the threat model, it is necessary to consider such strong adversary since the adversary would have full control of both the local and remote servers.

4.3 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 audit 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 audit the *false rejection rate* (denoted by γ).

5. POTENTIAL ATTACKS

In this section, we consider two data residency checking protocols that incorporate latency measurements with well-known PoR schemes [36, 26] in a straightforward manner, and briefly discuss 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 report detailed attacks in Appendix C.

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

5.1 SW-PoR based data residency checking

The first protocol is constructed on top of the PoR scheme by Shacham and Waters (SW-PoR) [36]. The audit asks for v data blocks and their associated homomorphic authentication tags. The prover passes the audit if the response is valid (with respect to the SW-PoR scheme) and the response latency is within an expected threshold.

The protocol's logic is based on a premise that should the dishonest prover have located significant portion of the data to remote storages, the extra time taken to retrieve the requested data would make the overall latency exceed the expected threshold. Nevertheless, such a premise fails to consider a possibility that the dishonest prover can still evade the expected threshold by speeding up the time taken to compute the response (i.e., aggregating the requested data blocks). In particular, it can over-clock its processor or have the remote servers concurrently compute the intermediate values and then aggregate the intermediate values into the final response. We conduct experimental studies to illustrate feasibility of these attacks, and report the results in Appendix C.1.

5.2 JK-PoR based residency checking

One possible mitigation for the previous attack is to eliminate computation time from the response latency, adopting the authenticator-based PoR [26, 28]. In this scheme, the data owner encodes her file using a redundant encoding and authenticate all the blocks of the encoded data. During the residency checking, the verifier issues a single request that asks for v randomly chosen data blocks (v is a security parameter) and measures the latency incurred by the storage provider in delivering all those requested blocks.

Although the computation time is eliminated from the response latency, an adversary 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 such evasion strategy and report the results in Appendix C.2.

6. 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. [26]. We first give an overview of the setup and audit phases. Next, 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, wherein having the verifier of a cryptographic protocol co-locating with the prover enhances the security. Finally, we analyse the security of our construction.

6.1 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 [33]) on F, generating $\widetilde{F}$. The encoded file $\widetilde{F}$ consists of $m = (1+c) \times n$ encoded data 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 identi-

fied by an unique file handle $\widetilde{F}_{ID}$ and each block in $\widetilde{F}$ is indexed by a particular integer $i \in [1..m]$. The data owner appends to every encoded data block $\widetilde{f}_i$ in $\widetilde{F}$ a b-bit MAC of its content, file handle and index under her secret key sk, obtaining $f_i \leftarrow \widetilde{f}_i||HMAC(sk, \widetilde{f}_i||\widetilde{F}_{ID}||i)$, where $HMAC()$ is a keyed-hash function that returns b-bit output and $||$ means concatenation. After entrusting the data to the storage provider, the data owner can delete all local copies, keeping only the secret key sk and some metadata for verification and reconstructing F from the outsourced blocks.

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, q_i, 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
```

6.2 Audit

To begin the residency checking, $\mathcal{V}$ first obtains an environment profile $\mathcal{E}$ (i.e. description of the distributions 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 an expected latency that a valid response should meet.
- l: The late delivery threshold, which is the number of late responses (whose latency exceeds d) that $\mathcal{V}$ is willing to tolerate.

We investigate how these parameters are to be chosen in Section 7.

Next, $\mathcal{V}$ executes the residency checking procedure detailed in Algorithm 1. The algorithm utilises three functions. InitiateQuery(v) chooses v block indices at random[3], each is a challenge asking the prover for the corresponding data block. The challenges are to be 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 ($q_i \leftarrow \widetilde{F}_{ID}||i$), receives f_i from $\mathcal{P}$, and at the same time measures the response latency t_i. $\mathcal{V}$ extracts $\tilde{f}_i$ from f_i and computes $HMAC(sk, \widetilde{f}_i||q_i)$, obtaining a b-bit MAC of $\tilde{f}_i||q_i$ under sk. $\mathcal{V}$ then compares it against f_i's authentication tag (i.e., IsValid(sk, q_i, f_i)), rejecting the audit if they are inconsistent[4]. 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 call the decision. If such number exceeds the late delivery threshold l, $\mathcal{V}$ rejects the audit.

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 parameter 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.

6.3 N-ResCheck Implementation

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 [10, 25]. The latency observed by $\mathcal{V}$ accounts for the data fetching time and challenge-response transmission time.

Guaranteeing the delivery of challenges and responses is necessary, for $\mathcal{P}$ has to respond to every challenge. Our implementation employs the reliable TCP [32] for transmission of challenges and responses, despite its higher latency variance compared to other protocols such as UDP [31], which suffers from packet loss. Although it is possible to design a residency checking protocol which supports packet loss of known rate, such variant would introduce 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 7, 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,

[3]This is not inconsistent with the description of the Audit phase in Section 4. Since the block indices 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.

[4]Should the authentication tag be computed solely from $\widetilde{f}_i$, $\mathcal{P}$ will be able to rightfully replace one block with another that has the same tag and destroy the former. Since the authentication tag in our protocol also covers the block ID, which is chosen at random during the audit, such incident will be detected with high probability.

we discuss in the next section another implementation variant that relies on 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 [9].

6.4 E-ResCheck Implementation

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), which we shall refer to as *Verifying Enclave (*`VE`*)*. $\mathcal{P}$ can neither tamper with `VE` operations, nor change the code and data loaded to it without being detected by $\mathcal{V}$. However, it can supply inputs for `VE`.

In E-ResCheck, the environment profile $\mathcal{E}$ contains information on housekeeping 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.

While 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 [9]. Unlike special trusted unit hardware such as IBM secure processor [5], these SGX-enabled processors are now widely available in commodity systems. The code running inside `VE` – the verification code – can be written by $\mathcal{V}$, or by any other party and vetted by $\mathcal{V}$ to ensure its correctness. Further, remote attestation mechanism [12] allows $\mathcal{V}$ to check if the correct code is loaded into `VE`. This mechanism also allows the verifier and `VE` to establish shared secrets, which enables secure channel for their communication (e.g. for $\mathcal{V}$ to send the secret key to `VE` or for `VE` to send the residency checking result to $\mathcal{V}$).

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 ask the untrusted application to handle those services. 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 [7].

While getting a trusted source for absolute time in SGX is challenging, it is possible to measure relative time with respect to a reference point [40]. We note that absolute time is not necessary in our setting, for the response latency measured by `VE` is an elapsed duration between an OCall and the corresponding ECall passing the requested block into `VE`, 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 7. Previous works [15, 22, 17] did not take into consideration mechanisms and behaviours of storage hardware with respect to the block size, resulting in an oversight of the strong effect that the block size has on security.

6.5 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 file F of n blocks is encoded into $\widetilde{F}$ of $m = (1+c)n$ 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 the 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

[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.

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 hits, $\mathcal{A}$ chooses l challenges to which it will respond by reading data from the storage. These l challenges are chosen with priority given to those whose transmission time already exceeds d. For the remaining $Hit - l$ challenges, $\mathcal{A}$ forges the responses. 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

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

The above is not an equality because we omit the cases wherein more than l challenges have transmission time exceeding d. Although the derivation is based on a specific adversary $\mathcal{A}$, it serves as an upper bound of the false acceptance rate. There is no loss of generality, for the fact that the original file F cannot be constructed from D implies there must be less than $n - 1$ blocks in D.

Recall that the challenges are randomly generated, it follows that the number of challenge collisions (i.e. those that ask for the same block) would be very small, thus the cache C (see Section 4.2) kept by $\mathcal{P}$ only has minor effects on the false acceptance rate. Hence, in this security analysis, we can safely ignore the effect of the cache and consider the setting where the cache size is zero.

False rejection rate. Let γ denote the probability that the honest prover keeping 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. $t_i^{\text{net}} + t_i^{\text{loc}} > d$). The false rejection rate of an audit with v challenges is:

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

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:

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

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

7. EVALUATIONS

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.

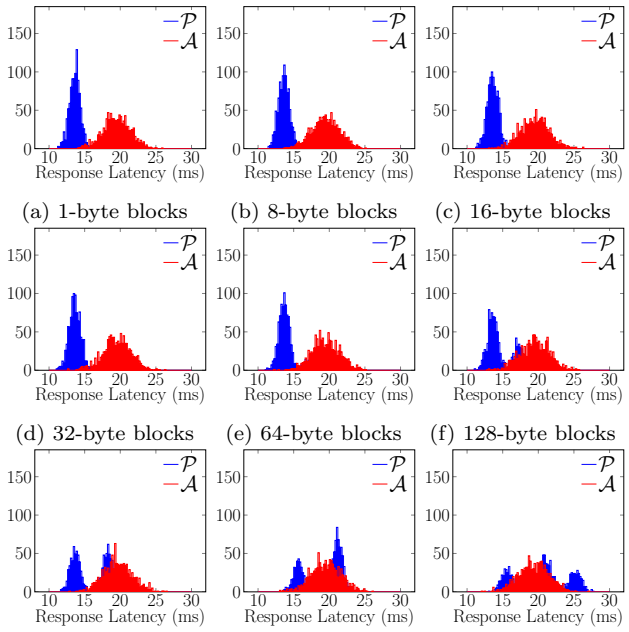


Figure 1: Histograms of the response latencies incurred by honest prover $\mathcal{P}$ and dishonest prover $\mathcal{A}$ in E-ResCheck with respect to different block sizes. $\mathcal{P}$ stores the blocks as-is in its local drives, whereas $\mathcal{A}$ splits large blocks into 64-byte segments and stores all the data at remote storages.

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

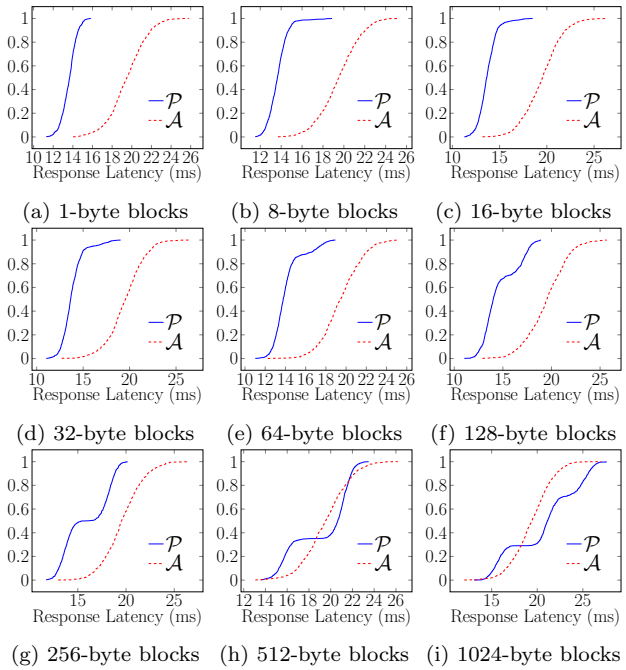
All experiments are conducted on Ubuntu 14.04 commodity systems equipped with quad-core Intel Skylake processors with SGX Enabled BIOS support, 1TB hard drives with I/O latency ranging from 12-15ms on average and 1GB Ethernet cards. $\mathcal{P}$ and $\mathcal{A}$ are represented by different programs 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 very physical system which hosts $\mathcal{P}$ and $\mathcal{A}$. `VE` is provisioned using Intel SGX SDK for Linux [8]. Unless stated otherwise, all experiments are repeated for 100 times and the average results are reported.

7.2 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 simul-

[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.

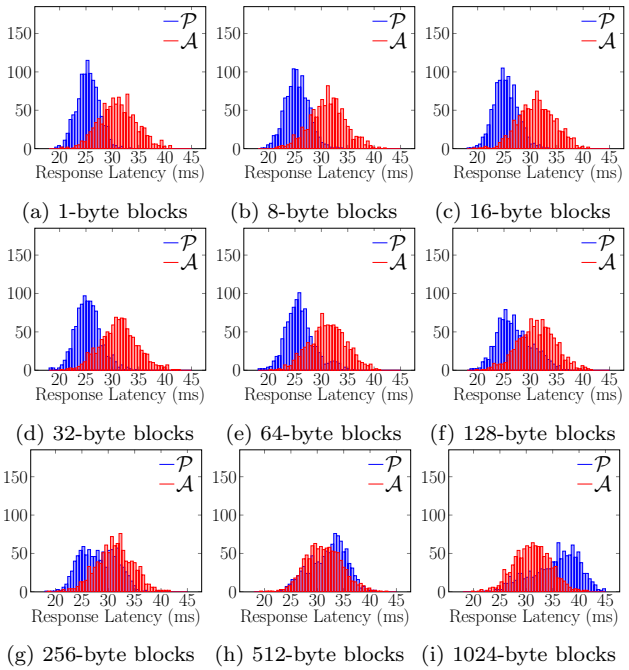


(a) 1-byte blocks (b) 8-byte blocks (c) 16-byte blocks

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

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

Figure 2: CDFs of response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in E-ResCheck with respect to different block sizes.



(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

Figure 3: 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.

taneously in an attempt to speed up the fetching time, evading the latency threshold. 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.

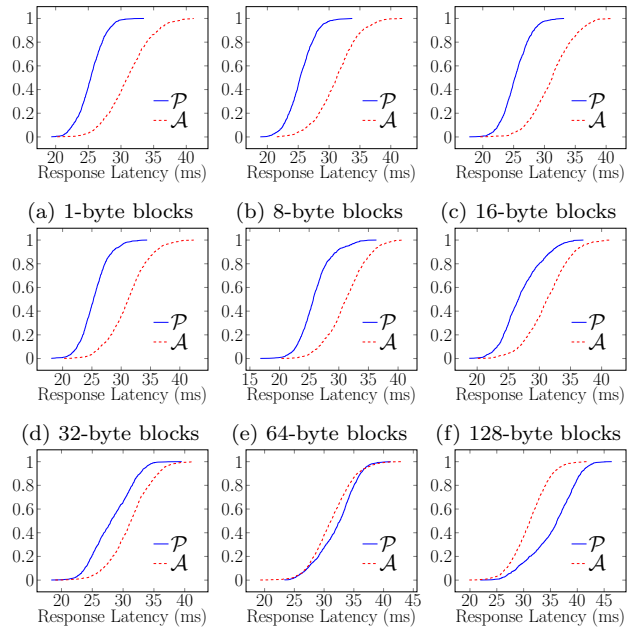
We vary the block size from one to 1024 bytes and measure the response latencies incurred by an honest prover $\mathcal{P}$ and an adversary $\mathcal{A}$. We report the results in Figures 1, 2 for E-ResCheck and 3, 4 for N-ResCheck.

Figure 1 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 deviation of 0.73. As the block size increases, blocks are more likely to span across physical sectors, increasing the variance in fetching time. Figure 1f, 1g and 1h 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 1i). Recall that the fetching times are desired to be uniform, so as to have a reliable latency assessment. Given such requirement, large block sizes (e.g. larger than 64 bytes) are clearly not suitable for security in our protocol.

The response latencies of $\mathcal{A}$ follow a normal distribution, with mean 19.58ms and standard deviation 2.71ms. For blocks that are larger than 64 bytes, $\mathcal{A}$ splits them to 64 byte segments 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 2 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 3) is noticeable, but not as evident as in E-ResCheck, for the response latency observed by $\mathcal{V}$ is the accumulation of the fetching time and the challenge-response transmitting time. The latter component is almost 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 deviations varying from 1.81ms to 2.52ms. The response latencies of $\mathcal{A}$ follow normal distribution with mean approximately 31.22ms and standard deviations oscillating around 3.18ms. Similar to E-ResCheck implementation, distinguishing response latencies incurred by honest and dishonest provers in N-ResCheck becomes more difficult as the block size increases, and even impossible when the block size reaches 512 bytes.

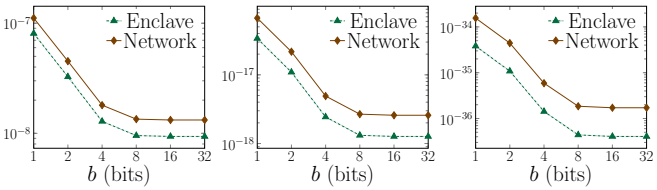


(a) 1-byte blocks (b) 8-byte blocks (c) 16-byte blocks

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

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

Figure 4: 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%

Figure 5: 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.

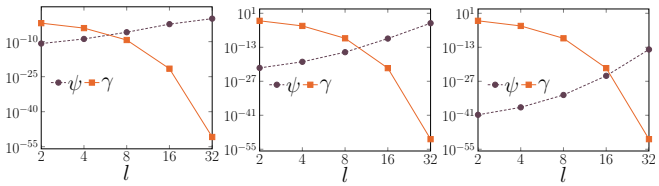
We demonstrate in Figure 4 CDFs of $\mathcal{P}$ and $\mathcal{A}$'s response latencies. Figures 4h and 4i especially show that CDFs of the adversary's response latencies dominate those of the honest prover, implying it has significant advantage in disguising its response latencies and thus its behaviours 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.

7.3 Effect of MAC length (b)

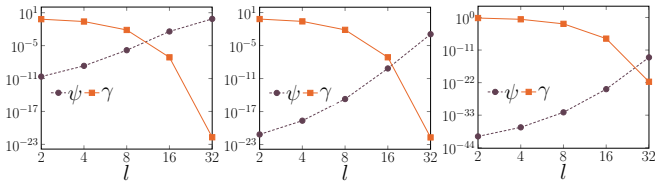
In the second set of experiments, we vary c from 10% to 40% and examine the effect of MAC length on the false acceptance rate ψ. The late delivery threshold is set to five, and the audit size is 300 challenges.

Figure 5 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 16 bits. Further increasing b does not result in better false acceptance rate. With 16 bits MAC and the block size is set to 64 bytes as suggested in the previous set of experiments, the expansion rate due to authentications tags is as small as 3%.



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

Figure 6: Effect of the late delivery threshold l on the security in E-ResCheck. MAC length is set to 16 bits, and audit size is $v = 300$ challenges.



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

Figure 7: Effect of the late delivery threshold l on the security in N-ResCheck. MAC length is set to 16 bits, and audit size is $v = 300$ challenges.

Comparing across figures, it is evident that larger c leads to lower false acceptance rate and hence better security. Moreover, the false acceptance rate of E-ResCheck is consistently smaller than that of N-ResCheck, suggesting E-ResCheck offers better security guarantee.

We note that the use of short authentication tags (e.g. 16 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 16 bits MAC, the probability that such adversary escapes the detection is less than 2^{-145}.

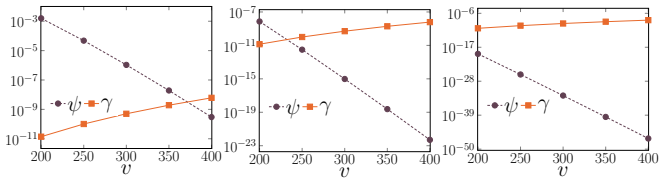
7.4 Effect of late delivery threshold (l)

In the third set of experiments, we fix the MAC length at 16 bits, the audit size at v at 300 challenges, and investigate the effect of late delivery threshold l on the false acceptance rate ψ and the false rejection rate γ.

Figure 6 shows the results for E-ResCheck, and Figure 7 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.

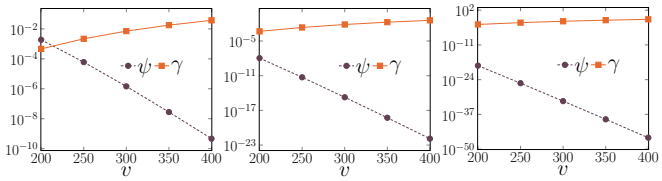
However, increasing l leads to the growth of the false acceptance rate ψ. For both implementations, ψ grows by eight to 16 orders of magnitude when l increases from two to 16, depending on the code rate of the error-erasure code in use. In particular, when $c = 10\%$ and the late delivery threshold is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

We suggest the late delivery threshold l to be set to eight, attaining γ as small as 5×10^{-10}, while still keeping ψ smaller



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

Figure 8: Effect of audit size v on the security in E-ResCheck. MAC length is set to 16 bits, and late delivery threshold is set to eight.



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

Figure 9: Effect of audit size v on the security in N-ResCheck. MAC length is set to 16 bits, and late delivery threshold is set to eight.

than 10^{-6} even for $c = 10\%$. We note that for the same parameter setting, 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%.

7.5 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 16 bits, late delivery threshold at eight and examine how v effects ψ and γ. The results are reported in Figure 8 (for E-ResCheck) and 9 (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×) in E-ResCheck and from 4.6×10^{-4} to 3.7×10^{-2} (by 80×) 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 orders 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 housekeeping 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.

8. RELATED WORKS

Proofs of Retrievability. Proofs of retrievability were first proposed by Juels and Kaliski [26], and have been followed by various works [36, 16, 38]. 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.* [13], assuring that most (but not necessarily all) of the data are stored. Later on, the notions of PoR and PDP are also extended to dynamic settings [37, 21]. While there are various efficient constructions in the literature [26, 36], none of them has taken the location of data into consideration. 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.* [17] presented a remote assessment of fault tolerance based on measuring the response latency of 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 assumes that the network latency is probabilistic, only its distribution can be determined.

Gondree *et al.*[22] proposed a framework that employs a set of known landmarks to verify the storage geolocation. Benson *et al.* [15] investigated the correlation of network latency and geographical distance, and suggested the use of such technique in verifying replications of the data across geographically separated datacenters. Our construction, on the other hand, focuses on verifying residency of the data on the server-in-question. Moreover, 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 cryptocurrency proposals [27, 35]. These proposals require constructing 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 instead of enforcing such requirement. PoDR, on the other hand, imposes local preservation of the data and offers an auditing mechanism to detect storage providers 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 [19, 34, 14]. 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.

9. 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, namely N-ResCheck and E-ResCheck, illustrate an interesting use-case of trusted computing, wherein having the verifier of a cryptographic protocol co-locating with the prover enhances the security.

The focus of this work has been on a static setting 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.

Acknowledgements

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Corporate Laboratory@University Scheme, National University of Singapore, and Singapore Telecommunications Ltd.

10. REFERENCES

[1] Australian privacy act. http://www.austlii.edu.au/au/legis/cth/consol_act/pa1988108/.

[2] 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.

[3] Data protection directive. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=URISERV%3Al14012.

[4] Google Drive. https://www.google.com/drive/.

[5] Ibm 4764 pci-x cryptographic coprocessor. http://www-03.ibm.com/security/cryptocards/pcixcc/overview.shtml.

[6] Intel SGX. https://software.intel.com/en-us/sgx.

[7] Intel SGX programming reference. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[8] Intel SGX SDK for Linux. https://github.com/01org/linux-sgx.

[9] Intel Skylake processor. http://ark.intel.com/products/codename/37572/Skylake.

[10] PsPing. https://technet.microsoft.com/en-us/sysinternals/psping.aspx.

[11] Traceroute. http://linux.die.net/man/8/traceroute.

[12] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. Innovative technology for cpu based attestation and sealing. In *HASP*, 2013.

[13] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *CCS*, 2007.

[14] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *OSDI*, 2014.

[15] K. Benson, R. Dowsley, and H. Shacham. Do you know where your cloud files are? In *CCSW*, 2011.

[16] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: Theory and implementation. In *CCSW*, 2009.

[17] 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.

[18] G. Connolly, A. Sachenko, and G. Markowsky. Distributed traceroute approach to geographically locating ip devices. In *IDAACS*, 2003.

[19] T. T. A. Dinh, P. Saxena, E.-C. Chang, B. C. Ooi, and C. Zhang. M^2R: Enabling stronger privacy in mapreduce computation. In *USENIX Security*, 2015.

[20] Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In *Theory of cryptography*. 2009.

[21] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia. Dynamic provable data possession. In *CCS*, 2009.

[22] M. Gondree and Z. N. Peterson. Geolocation of data in the cloud. In *CODASPY*, 2013.

[23] K. Harrenstien, M. K. Stahl, and E. J. Feinler. NICNAME/WHOIS. *RFC-954*, 1985.

[24] C. Houri. Method and systems for locating geographical locations of online users, 2003. US Patent 6,665,715.

[25] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM SIGCOMM*, 2002.

[26] A. Juels and B. S. Kaliski Jr. PORs: Proofs of retrievability for large files. In *CCS*, 2007.

[27] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In *IEEE S&P*, 2014.

[28] M. Naor and G. N. Rothblum. The complexity of online memory checking. In *FOCS*, 2005.

[29] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *SIGCOMM*, 2001.

[30] 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.

[31] J. Postel. User datagram protocol. 1980.

[32] J. Postel. Transmission control protocol. 1981.

[33] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 1960.

[34] 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.

[35] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai. Retricoin: Bitcoin based on compact proofs of retrievability. In *ICDCN*, 2016.

[36] H. Shacham and B. Waters. Compact proofs of retrievability. *Journal of cryptology*, 2013.

[37] E. Shi, E. Stefanov, and C. Papamanthou. Practical dynamic proofs of retrievability. In *CCS*, 2013.

[38] J. Xu and E.-C. Chang. Towards efficient proofs of retrievability. In *ASIACCS*, 2012.

[39] I. N. Yezhkova. Worldwide and U.S. enterprise storage systems forecast update, 2015-2019. White Paper. 2015.

[40] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town Crier: An authenticated data feed for smart contracts. In *CCS*, 2016.

APPENDIX

A. NOTATION TABLE

A table of notations that are used throughout the paper is shown in Table 1.

Table 1: Summary and descriptions of the notations that are used throughout the paper. Group I are parameters to be 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

B. RELATED NOTIONS

B.1 Proofs of Retrievability

Proof of retrievability [26] enables the data owner to audit the storage server on the data preservation. In PoR protocols, the data owner encodes the original data using a redundant encoding (such as the error-erasure Reed-Solomon code [33]), authenticates 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 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 [26] has been followed by various variants [20, 38] and is also extended to the dynamic setting [37]. A similar notion known as Provable Data Possession (PDP) is proposed by Ateniese *et al.* [13]. 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 [37].

B.2 Host Geolocation

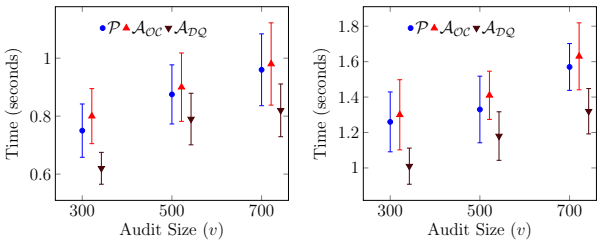
While the notion of data residency concerns over the fact that the data are kept intact on local drives of a storage server, it implicitly assumes that the geographic location of the storage server is known to the verifier. Thus, also of interest are techniques to geographically locate an online party. Since machines/systems on the Internet can be uniquely identified by IP addresses, 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 [29, 24, 18]. 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 [11]. 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 [18]. 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 [23] 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 [29].

B.3 Intel SGX

Intel SGX [6] is a set of extensions that provision the protected execution environments (aka trusted environments or enclaves). The TCB of such enclaves comprises solely 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 accesses to enclave memory are protected by the processor. In another word, code and data loaded to the enclave 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). 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 [6] at any time during its lifetime. We note that the enclave code has to be loaded into the enclave during its creation.

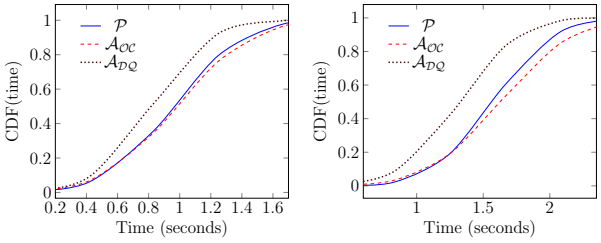
Enclaves cannot directly execute OS-provided services such as I/O. In order to access those services, enclaves have to employ OCalls (calls executed by the enclave code to transfer the control to non-enclave code) and ECalls (API for untrusted applications to call in). These ECalls and OCalls constitute 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, enabling a remote verifier to check if a specific software has been loaded within the enclave by means of cryptography. Via such 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.



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

Figure 10: 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

Figure 11: CDF of the response latencies incurred by $\mathcal{P}$, $\mathcal{A}_{\mathcal{OC}}$ and $\mathcal{A}_{\mathcal{DQ}}$ in SW-PoR based residency checking.

C. VULNERABLE CONSTRUCTIONS AND POTENTIAL ATTACKS

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* that of $T_{\mathcal{P}}$, that is,

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

then $\mathcal{A}$ could intentionally add delays so that these two distributions are identical, thus successfully evading detection.

C.1 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) [36]. In this PoR scheme, the audit asks for v data blocks and their associated homomorphic authentication tags. The response is 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 latency, and accepts the prover as passing the audit if the response is valid (with respect to the SW-PoR scheme) and the response latency is 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 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.

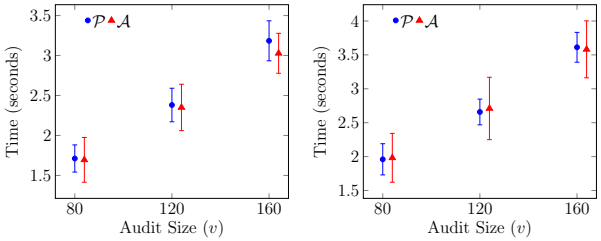
We conduct experimental studies to inspect the response latencies of the honest prover in comparison with those of the two adversaries. In these experiments, provers compute the responses using a vCPU Intel Xeon Family running at base clock speed of 2.5GHz, except for $\mathcal{A}_{OC}$ who over-clocks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

Empirical results. We vary the block size (number of group elements in each data block) as well as the number of data blocks 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 10 by showing their means and standard deviations. To give a better intuition on the adversaries' ability to evade latency measurements, we show in Figure 11 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 times. Although the CDFs of $\mathcal{A}_{OC}$'s latency measurements do not stochastically dominate $\mathcal{P}$'s, they are similar and thus it requires challenges of significant size in order to detect $\mathcal{A}_{OC}$'s violation of the SLA.

C.2 JK-PoR based residency checking

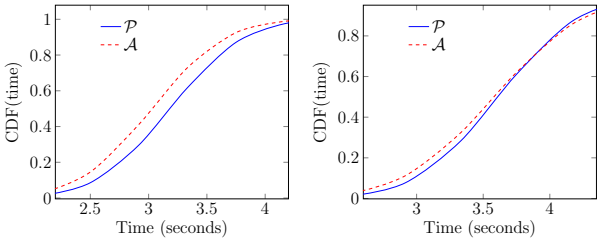
Protocol. One possible mitigation for the previous attack is to adopt a PoR scheme in which the prover performs virtually no computation in executing the residency checking, such as the authenticator-based PoR [26, 28]. In this scheme, the data owner pre-processes the file F using an error-erasure code 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



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

Figure 12: 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

Figure 13: CDF of the response latencies incurred by $\mathcal{P}$ and $\mathcal{A}$ in JK-PoR based residency checking.

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[8], 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.

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 12. We also depict in Figure 13 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.

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