SEEDS: Secure Extensible Enclave Decentralized Storage for Secrets

Stefanie Dukovac
RCS Lab, University of Waterloo
smdukova@uwaterloo.ca

Emil Tsalapatis
RCS Lab, University of Waterloo
emil.tsalapatis@uwaterloo.ca

Ali José Mashtizadeh
RCS Lab, University of Waterloo
ali@rcs.uwaterloo.ca

Abstract

Applications that use passwords or cryptographic keys to authenticate users or perform cryptographic operations rely on centralized solutions. Hardware Trusted Platform Modules (TPMs) do not offer a way to replicate material, making access in a distributed environment difficult. Meanwhile, remote services require a constant network connection and are a central point of failure. Admin access to such a service means total and permanent compromise of the secrets.

We present SEEDS, an SGX based secure decentralized multi-user store for cryptographic secrets. SEEDS prevents secrets from leaking even against attackers with access to user credentials by providing an API to use keys without reading them. SEEDS tolerates long network partitions and uses CRDTs to reconcile state at reconnection, making it a viable mechanism for local authentication.

SEEDS provides a base for distributed SGX capable applications through an expressive policy engine. It represents both all data and metadata like users as entries in a key value store, both of which are governed by policy. SEEDS also allows admins to add functionality to existing enclaves by composing new API calls out of existing SEEDS operations.

We demonstrate how to use SEEDS as a secrets manager with two applications. We present a decentralized and highly available alternative to LDAP plus Kerberos. We also descirbe a software U2F token implementation.

ACM Reference Format:
Stefanie Dukovac, Emil Tsalapatis, and Ali José Mashtizadeh. 2022. SEEDS: Secure Extensible Enclave Decentralized Storage for Secrets. In *5th Workshop on System Software for Trusted Execution (SysTEX '22), February 28, 2022, Lausanne, CH*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

SysTEX '22, February 28, 2022, Lausanne, CH

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Managing secrets like cryptographic keys securely while keeping them accessible is challenging in untrusted systems. Users need keys for operations like message signing, communication, encryption, and authentication. Attackers that compromise a system can read and exfiltrate a user's keys, which allows the attacker to impersonate the user and intercept communications.

Users cannot prevent these attacks because most systems hold keys unencrypted in memory. Even if the keys are encrypted at rest, they must be decrypted before being used. An adversary that has gained access to the system can inspect user memory and intercept the user's sensitive data. It is well known that popular password managers running on a user's device are vulnerable to this attack [3].

One solution is to use dedicated hardware such as a Trusted Protection Module (TPM) [4] to securely manage secrets on a single machine. TPMs have a rich but restricted API and do not solve the problem of users with multiple devices, requiring users to register new credentials on each device. Users often own several computers, tablets, and phones and need to access their secrets across these devices. Cloud based solutions such as Cloud Key Store (CKS) [18] rely on network connectivity. Losing internet access renders the TPM unavailable even for local authentication, possibly preventing the user from using their own machine.

Alternative designs such as cTPM [11] do not require constant connection. cTPM accesses data from the local machine and forwards updates to the cloud. This system however depends on a pre-shared key between TPMs and the cloud. The key must be burned into the TPM during manufacturing. A single compromise of the cloud infrastructure permanently compromises the TPM. cTPM, along with CKS, also imposes limitations on the size of each secret they store.

Trusted execution environments (TEEs) ensure integrity and confidentiality even against attackers with full system read and write privileges. TEEs such as Intel SGX [2] and ARM TrustZone [1] extend existing CPU architectures to provide a secure virtual processor and memory. TEEs ensure data confidentiality by making application memory inaccessible, even to privileged software, so attackers cannot inspect or overwrite it during execution.

TEEs also provide code integrity to local and remote applications through attestation. Attestation uses Public Key

	Single Node[7, 15]	Distributed[5]	CKS [18]	cTPM[11]	SEEDS
Security					
Multi-User Access Control	○	○	●	●	●
Denial of Service	○	○	○	○	●
Compromised User	-	-	●	○	●
Compromised Host	●	●	●	◐	◐
Functionality					
Replication	○	●	◐	◐	●
Disconnected Operations	-	○	○	◐	●
Arbitrary Data	●	●	○	○	●

Table 1. Comparison of TEE and TPM related secrets managers. Single node TEEs are accessible only locally. Distributed TEE systems do not ensure availablity and do not support multiple users. CKS and cTPM use a cloud machine that is a single point of failure.

Infrastructure (PKI) to verify the enclave is running a specific program and on genuine hardware. TEEs are widely available in commodity hardware including smartphones, tablets, and laptops.

Typical TEE based systems for local storage do not provide the same robust guarantees that TPMs do. Compromising a TPM is only feasible by breaking the module itself. With TEEs, intercepting a user's credentials is enough to allow an attacker access to the secrets. The attacker can then exfiltrate them and continue using them even after being found out. Compromising an admin also leaks all secrets in the enclave.

At the same time, having a way to extract the contents of an enclave provides a straightforward update mechanism. Updates are necessary for adding new functionality to the enclave. Developers typically simply create a new enclave, and insert into it the data from the old one. This workflow is impossible if not even admins can extract enclave data.

Existing TEE based secrets managers also sacrifice availability for consistency. This choice makes sense for systems that aim for high performance and act as distributed databases. However, it complicates the use of the manager for local authentication e.g., as a replacement to LDAP. A network failure prevents a legitimate user from logging in. Secrets managers on the other hand have high availability requirements, especially if used for local authentication.

We present SEEDS, an Intel SGX based secrets manager that balances strong security guarantees with maintainability and usability. Users deploy SEEDS on all their devices to automatically replicate cryptographic secrets. Cryptographic operations, e.g., key signing and authentication, occur in the enclave to prevent exposure. SEEDS prevents leakage against attackers with user or even admin credentials.

SEEDS provides a policy engine for developers to easily define applications with. All data and metadata is represented as key value pairs in a KV store, with policies determining available operations. For example, private keys can be restricted to signing when authorized by a specific principal, but never read. Policies determine which API calls are allowed for each pair, including policies themselves.

Admins safely add new API calls at runtime in the form of scripts executed in the enclave by an interpreter. Such scripts are compositions of existing calls that are prevented by policies from leaking data. SEEDS propagates API changes to all enclaves in the cluster automatically.

SEEDS uses an eventually consistent TEE based distributed system design and thus supports offline operation. We describe how we solve the complications that arise from looser consistency guarantees regarding revocation and synchronization. We use *strong eventual consistency* to ensure updates converge when devices come online. We use Conflict-Free Replicated Data Types (CRDTs) to reconcile conflicts.

We demonstrate SEEDS' flexibility by building two applications. First, we replace a centralized LDAP+Kerberos server with a distributed, secure, and highly available service with a SEEDS based NSS/PAM module. Second, we use SEEDS to build a replicated password manager with support for PKI authentication methods.

2 Overview

SEEDS internally represents all data and metadata as pairs in a key-value (KV) store. Each key is of the format "type.user.ID", for the type of data, user it belongs to, and unique ID. Each piece of data has a specific type corresponding to the data it represents. For example, a private key has a different type from a symmetric key or a monotonic counter.

SEEDS provides a separate API for each type, defining valid operations. For example, the private key type has as an API of (`generate`, `decrypt`, `activate/deactivate`, `delete`). Notice that there is no call to directly read the key. Since SEEDS only allows access using the API, the keys are inaccessible from outside the enclave by all accounts.

The same rules hold for the metadata types, like those for machines or users. For example, the metadata for each machine in SEEDS is represented as a KV pair of type 'machine'. The API for this type is (`add_machine`, `remove_machine`).

Each key has an associated policy that further restricts operations on the key. Policies are allowlists that define the API calls allowed by each user for this pair. A possible use of policies is, for example, to temporarily revoke the private key a user by preventing all API calls to it. Policies support multi-user environments and prevent compromised users from damaging or leaking other users' keys. As we show

in Section 3, policies are flexible enough to prevent admins themselves from reading secrets.

Providing confidentiality even in the face of admin compromises limits our capabilities to update the enclave code. Typically an admin updates enclaves by migrating all the data to a new trusted instance which supports the updated API. This is, e.g., the case with CKS [18]. For SEEDS, supporting such an upgrade path would open it up to attack by an adversary with admin privileges. Such an attacker could 'update' SEEDS to a new enclave which then allows for the exfiltration of the data.

SEEDS allows developers to dynamically define new API calls from existing ones. Developers create new operations by composing scripts that exclusively existing API calls. For example, we can create a new SEEDS call that finishes a TLS handshake and which uses the existing APIs for PKI and symmetric keys. The call authenticates the remote host using the local's private key, then generates the new symmetric key and inserts it to SEEDS. All operations happen fully withing the enclave, and SEEDS enforces existing policies at each step. This limitation prevents a malicious admin from creating a new call that leaks otherwise inaccessible secrets.

Each node in SEEDS stores a full copy of the key-value store and exchanges updates with other nodes opportunistically. Updates are exchanged using TLS connections terminated inside the enclave. If the device loses connectivity, SEEDS reestablishes communications when connectivity is restored and merges updates with other nodes.

SEEDS implements strong eventual consistency that ensures all nodes converge and presents applications with a consistent view. Strong eventual consistency allows nodes to survive network partitions, offline operations, and even DoS attacks. SEEDS specifically relies on conflict-free replicated data types (CRDTs) [9] (see section 3) to achieve strong eventual consistency.

2.1 Threat Model

SEEDS runs on SGX-capable user devices and allows local applications to submit commands. Communication happens thorugh untrusted IPC or a locally attested channel between an SGX application and a SEEDS enclave.

The threat model assumes a powerful adversary that can control any machine in the SEEDS cluster. As a result, the system can fail to respond to enclave requests, respond incorrectly, and issue requests to the enclave. The adversary can read main memory by performing memory dumps or DMA operations but cannot read or write to enclave memory.

All untrusted software can be subverted, including the untrusted code of SEEDS. We assume that there are no exploitable software bugs within the trusted code that can be used to influence enclave code execution. Additionally, the adversary can interfere with network traffic by modifying, dropping, delaying, and reordering packets.

In addition, the attacker can get a hold of user credentials. With these credentials the attacker can impersonate a user to the SEEDS enclave, and make calls on their behalf. The attacker still cannot directly access the enclave, but only submit API calls on behalf of a valid user.

The attacker can also get a hold of admin credentials, essentially user credentials with heightened privileges. They can then remove temporarily remove machines and users from the enclave, or prevent users from accessing their keys. They can also delete enclave state.

2.2 Security Goals

The security goal of SEEDS is to protect the confidentiality of user secrets from system and user compromise. All SEEDS functions should be secure from any malicious software on the host. We ensure this by using SGX to prevent an attacker from reading or writing enclave data from a compromised untrusted host.

SEEDS should also prevent an attacker from exfiltrating data even when they have access to user credentials. An attacker might use enclave secrets, but should not be able to read them out. That is, the damage of the attack should be limited to the scope and privileges of that user.

Additionally, an attacker should not be able to launch a replay or rollback attack on SEEDS. Replay attacks subvert a program by executing previously authorized operations, while rollback attacks revert state. Hence replaying any requests between an application and SEEDS or updates between SEEDS replicas should not execute or influence the state in any way.

Since updates are applied immediately at the local replica, an adversary can interfere with a replica and prevent it from discovering new updates. As a result, the disconnected replica will service local requests with stale data. SEEDS should be able to gracefully continue execution after the partitions merge back. Moreover, SEEDS should be usable for local authentication during such a partition.

We do not protect against side-channel [19, 26] or speculative execution [10, 13] attacks as they are out of the scope of this work. Existing research has developed side-channel countermeasures and tools to assist in detecting potential speculative execution bugs using code instrumentation and static analysis [21, 22]. Finally, we do not protect against denial of service (DoS) attacks [14] that prevent the enclave from executing.

3 Design

Below we describe in more detail three elements of the SEEDS platform: policies, extensible calls, and CRDTs.

API and Policies Table 2 shows the SEEDS API. SEEDS provides applications with a key value store and a rich set of cryptographic protocols. The cryptographic primitives operate on passwords, symmetric keys, public-private keys, and

Operation	Passwords	Symmetric Keys	PKI	Counters	General
`get`	✓			✓	✓
`put`	✓	✓	✓	✓	✓
`delete`	✓	✓	✓	✓	✓
`inc/dec`				✓	
`sign/verify`			✓		
`encrypt/decrypt`		✓	✓		
`authenticate`	✓	✓	✓		
`generate`	✓	✓	✓		
`hmac`	✓	✓			
`key exchange`			✓		

Table 2. The SEEDS API by field type. The policy can restrict operations further conditional on the appropriate user credentials. We support multiple cryptographic algorithms for each operation. The policy and machines are stored in the key-value store itself as a reserved portion of the namespace.

counters. We chose these primitives because they allow us to support a wide range of application. The list is expandable.

As shown in the table, certain types of data by default can be used through the API but not read. This includes PKI and symmetric keys, which can be used for authentication and encryption/decryption. We allow reading passwords from the enclave by default, but users can prevent reading individual passwords using policies.

SEEDS represents every piece of information as a tuple in the key value store. This includes configuration data like access policies. A policy's key is the UID of the KV pair it corresponds to, and its value is an allowlist encoded as a list of strings `uid.permissions`. The UID can refer to a particular user or be `ANY` if it refers to all users.

Policies are themselves data in the KV store, so they themselves are governed by policies. To prevent recursion, administrators can define certain policies to be immutable. Such policies become irrevocably read-only, so they do not need policies themselves.

This scheme allows us among other things to turn data read-only even for admins. Let us take for example a password `PW` owned by user `U`. Figure 1 shows how to prevent even admins from reading `PW`. The policy P1 for `PW` includes just a string `U.{authenticate, hmac}`. The policy P2 for P1 is `ANY.{read}`, and is immutable. Policy P1 thus prevents all users from directly reading the password, while P2 prevents modifications even by admins to P1.

Safe Extensibility SEEDS allows developers to extend the API by adding new cryptographic protocols using an

```
passwd.U.PW = "deadbeef"
P1 = RW for PW allow U.{authenticate, hmac}
P2 = RO for P1 allow ANY.{read}
```

Figure 1. An example SEEDS policy prevent reading out a password. `P1` applies to all users and allows only user `U` to access `PW`, and that only to authenticate or generate an HMAC. The password is not readable. `P2` turns `P1` read-only to prevent an attacker from adding back read permissions. `P2` is immutable, and so is `P1` because of `P2`.

embedded interpreter. This interpreter allows small scripts which amount to chains of API calls to happen fully inside the enclave. This in turn allows SEEDS to provide more complex functionality and abstract away more granular operations.

SEEDS allows developers to implement cryptographic protocols that use SEEDS data, APIs, and additional byte and string manipulation functions. The protocol runs inside the enclave without exposing intermediate results to the untrusted system. SEEDS uses a simple language that defines calls as sequences of SEEDS operations and arithmetic.

```
u2f_authenticate(credentials, appID, challenge) {

    counter = seeds.inc(credentials, counter);
    presentBit = 1;
    response = concat(appID, presentBit, counter,
              challenge);
    hash = sha256(response);
    signature = seeds.sign(U2FKeyID, hash);

    return (counter, signature);
}
```

Figure 2. The dynamically loaded SEEDS function for authenticating using the U2F protocol. The user sends their credentials, the unique ID of the application requesting authentication, and the remote's challenge. The function signs the authentication response and atomically increments the token's counter.

New API calls look like scripts that include other calls. These calls run with the privileges of the user that invoked them, and do not have direct access to the KV store. They mechanism thus cannot be used by a malicious admin to undermine the security model of SEEDS.

SEEDS imposes severe performance and memory limitations to the interpreter. For example, we do not support recursive calls or `OCALL`s, and do not use more than a few kilobytes of memory. This is because API calls are expected to combine together cryptographic operations, not do any kind of data processing.

As an example, we create a new API call specifically for U2F authentication out of a monotonic counter primitive.

The U2F protocol uses a monotonic counter when sending authentication requests to prevent replay attacks. The protocol concatenates this counter with the application ID and the challenge, and hashes the result. It then signs the hash with the authenticator key and sends the result out. All the steps in the process are simple and in fact the algorithm has no flow control. We can thus implement the authentication function as a new API call that uses SEEDS' already existing PKI and monotonic counter data types.

Distributed Updates SEEDS has relaxed consistency guarantees to ensure availability regardless of connectivity. The relaxed consistency guarantees do not affect the system in practice. Updates are relatively uncommon because users create new keys infrequently. Moreover, most users are typically using one or more connected machines at the same time and they mostly update their own data. Thus concurrent updates from different users are rare or impossible.

Machines use point to point communication that terminates in the enclave for updates. Enclaves combine SGX attestation with the TLS handshake [16], embedding the enclave report in the x509 certificate. Enclaves prevent forking attacks by inspecting each other's global version vector (see below) during TLS setup to ensure they are not stale.

SEEDS reconciles state updates between machines using conflict-free replicated data types (CRDTs). SEEDS uses state based CRDTs that send all of enclave state instead of just the operations to make reconciling state easier. SEEDS has CRDT operations that correspond to the KV store's `put/get/delete` operations.

These three operations are not commutative, so we define a deterministic resolution strategy to ensure the CRDTs converge. A `put` supersedes concurrent `delete`s, and concurrent `put`s use last-writer-wins semantics. We define 'last' using the last-time-modified timestamp attached to each KV entry.

SEEDS also offers a monotonic counter data type that is trivial to implement as CRDT. We currently use it exclusively for the authentication token application (see Section 4).

Last-writer-wins semantics are problematic if two machines independently insert a new key value pair into the store. Even though the concurrent entries are unrelated, one will overwrite the other due to a name collision. SEEDS assigns a unique name using the local machine ID and a timestamp to ensure there are no key conflicts.

Conflict Resolution SEEDS uses Optimized OR Sets (Opt-OR Sets) [9], for CRDTs. Regular OR Sets [24] define last-writer-wins semantics using tags whose total size grows indefinitely. Opt-OR sets retain these semantics and ensure bounded space usage without garbage collection (GC [27]). GC methods require acknowledgements and ordered message delivery, both incompatible with SEEDS' model.

Opt-OR sets need a local version vector for each key value pair on each replica, and a global one for the whole key value store. SEEDS does updates by sending out the store's state all at once as a single message. Because of state based replication the order of updates does not matter. SEEDS uses each key value pair's local vector to enforce the CRDT's semantics when merging.

The version vector and timestamp increase the space usage of each pair, but the total space used is small so the overhead is acceptable. Similarly, the bandwidth cost for sharing state based updates of the entire store is small, It is also incurred only when SEEDS sends out updates. This happens infrequents due to its relaxed consistency guarantees.

4 Applications

We have been developing two applications on top of SEEDS to explore these ideas. First, we built a decentralized user management system to replace the traditional LDAP plus Kerberos architecture. Second, we built a password manager that uses SEEDS to replicate passwords and keys between a user's devices.

4.1 Decentralized Alternative to LDAP and Kerberos

Lightweight Directory Access Protocol (LDAP) and Kerberos are popular protocols for managing user accounts and login credentials. Often they are used together where LDAP manages account information and Kerberos manages credentials. Each machine is issued keys that give it read-only access to LDAP and Kerberos to prevent making account information widely available. Correctly configuring these services is difficult. These services are also a single point of failure and susceptible DoS attacks.

To improve performance and allow offline operation admins often use a persistent cache on clients that store password hashes and account information. Many programs will trigger LDAP calls, e.g., running `ls` requires mapping user and group IDs to names that result in costly calls to the LDAP service if the information is not cached. By persisting the cache, users can continue to use their laptops while offline.

SEEDS Decentralized User Management Service. We built a decentralized user management service using SEEDS. To add a machine to the cluster, the administrator initializes the SEEDS enclave and pairs it with any existing machine. During this process, SEEDS generates a unique host private key and registers it with the cluster, similar to administrators configure LDAP and Kerberos. SEEDS will synchronize the entire user database on to the host.

Each host has a full replica of the database allowing machines to operate offline for some duration. Even though the service is decentralized we still use a centralized administration model where only specific users can add and remove hosts, users and groups from the system.

Our user management service consists of a command-line tool for setup and administration. On each host, we install our Pluggable Authentication Modules (PAM) and Name Service Switch (NSS) modules. These modules use SEEDS to implement several C library and PAM functions.

Administrators can use our command-line tool for initializing the host and adding it to an existing cluster, and managing users and groups.

The PAM module, `libpam_seeds.so`, implements the PAM authentication and account APIs to allow machines to login by validating passwords or hardware tokens against the SEEDS database. Applications like `login` and `su` use PAM to call into the enclave and verify credentials

The NSS module, `libnss_seeds.so`, provides the POSIX C library with access to user and group information including a user's UID, home directory, shell, group membership. NSS provides access to all the information a POSIX system would typically query from the `passwd` and group files. The NSS API allows several other files in the `/etc/` directory to be served to hosts and this could be trivially added.

4.2 SEEDS Password Manager and Virtual Token

We built a password manager based on SEEDS that replicates credentials between a user's multiple devices. Adding, updating and removing accounts will be replicated to all the users devices assuming they have connectivity. The SEEDS password manager stores passwords and public-private keys. PKI keys can be used as a software U2F token.

The password manager has a software Universal 2nd Factor (U2F) token. The U2F protocol provides two-factor authentication (2FA) to protect user accounts even when an attacker successfully steals their password.

Most U2F tokens are physical devices which means that losing them locks a user out of their accounts. Users usually need to register multiple U2F tokens for each account and store them in separate places which is time consuming and a burden for many.

To address these practical issues we built U2F support into our password manager to emulates a U2F token through SEEDS. This simplifies the use of U2F to prevent phishing or stealing passwords but does not provide a physical button due to limitations in SGX. These U2F tokens are available on all devices a user owns, allowing the user to not worry about registering and securing multiple physical tokens. They protect the U2F authentication process, essentially a PKI signature, with the use of a password or pin.

We use `libcuse` to emulate a U2F HID USB device that looks like a physical token to applications. Web browsers and OpenSSH use the standard `libfido2` library to authenticate the user against the software token. No code changes to applications or `libfido2` is necessary.

Thwarting Replay Attacks An issue with the SEEDS U2F token is the reconciliation of counter updates. U2F devices use monotonic counters to detect stolen credentials. If the U2F counter is stored in the key-value store, the highest counter should always be used. Our counter CRDT is monotonically increasing even in the case of conflicts.

However, due to the asynchronous nature it is possible to use the token from two machines against the same service, resulting in the service believing a key may have been compromised. Services typically alert the user of this potential issue. If the user runs SEEDS across all their devices they are unlikely to physically travel between two machines that have no connectivity to each other. Carrying any device running SEEDS would prevent such a case.

5 Related Work

There has been extensive research on storage systems based on SGX[6, 8, 17, 25]. These systems are performance oriented and suppose connectivity between the nodes of the system. They thus employ strong consistency at the cost of availability. Moreover, these systems use SGX to prevent a system compromise from allowing an attacker to access sensitive data. They do not protect against an attacker with access to the right credentials from exfiltrating data.

TPMs have also been implemented as an abstraction in software. cTPM [12] uses a remote machine as a TPM, assuming a preshared key between the remote machine and the local node. Compromising the remote machine permanently compromises the preshared key. fTPM implements TPM-like functionality using firmware and ARM TrustZone, but is easily applicable to SGX.

CKS [18] provides an interface similar to that of a TPM, providing encryption and decryption capabilities using a key. CKS allows among others to monitor the uses of the keys it contains, and so includes the cloud server on every operation. This in turn means that it cannot provide high availability. Moreover, CKS faces the issue that an attacker with access to user credentials can abuse the enclave update feature. The attacker can 'update' the enclave to a version they have created, which in turns directly exports the secrets. SEEDS aims to be resistant to such compromise.

The use of interpreters in enclaves is well-established practice. Twine compiles code to WebAssembly before running it in the enclave [20]. MapReduce for SGX uses a Lua interpreter to run user defined map and reduce scripts at runtime [23]. We adopt similar techniques, to permanently expand the enclave API instead.

6 Conclusion

We present SEEDS, an SGX based decentralized, highly available secrets manager with an expressive cryptographic and policy API. SEEDS prevents attackers from extracting secrets from the enclave even they compromise not just the system and intercept user credentials. SEEDS at the same time distributed and highly available even on devices with intermittent connectivity. We demonstrate the system's usefulness by using it to secure three security sensitive applications,

References

[1] [n.d.]. Arm TrustZone Technology. https://developer.arm.com/ip-products/security-ip/trustzone. Accessed: 2021-12-15.

[2] [n.d.]. Intel Software Guard Extensions. https://www.intel.ca/content/www/ca/en/architecture-and-technology/software-guard-extensions.html. Accessed: 2021-12-15.

[3] [n.d.]. Password Managers: Under the Hood of Secrets Management. https://www.ise.io/casestudies/password-manager-hacking/. Accessed: 2021-12-05.

[4] [n.d.]. Trusted Platform Module Library Part 3: Commands. https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf. Accessed: 2022-01-10.

[5] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Vijay Nagarajan, Pramod Bhatotia, et al. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 {USENIX} Annual Technical Conference ({USENIX} {ATC} 21)*. USENIX Association, 65–79.

[6] Maurice Bailleu, Dimitra Giantsidi, Vasilis Gavrielatos, Do Le Quoc, Vijay Nagarajan, and Pramod Bhatotia. 2021. Avocado: A Secure In-Memory Distributed Storage System. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 65–79. https://www.usenix.org/conference/atc21/presentation/bailleu

[7] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. {SPEICHER}: Securing lsm-based key-value stores using shielded execution. In *17th {USENIX} Conference on File and Storage Technologies ({FAST} 19)*. 173–190.

[8] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. SPEICHER: Securing LSM-based Key-Value Stores using Shielded Execution. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*. USENIX Association, Boston, MA, 173–190. https://www.usenix.org/conference/fast19/presentation/bailleu

[9] Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, and Sérgio Duarte. 2012. An optimized conflict-free replicated set. *arXiv preprint arXiv:1210.3368* (2012).

[10] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 991–1008. https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[11] Chen Chen, Himanshu Raj, Stefan Saroiu, and Alec Wolman. 2014. cTPM: A cloud {TPM} for cross-device trusted applications. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*.

[12] Chen Chen, Himanshu Raj, Stefan Saroiu, and Alec Wolman. 2014. cTPM: A Cloud TPM for Cross-Device Trusted Applications. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. USENIX Association, Seattle, WA. https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/chen

[13] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*. 142–157. https://doi.org/10.1109/EuroSP.2019.00020

[14] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution* (Shanghai, China) *(SysTEX'17)*. Association for Computing Machinery, New York, NY, USA, Article 5, 6 pages. https://doi.org/10.1145/3152701.3152709

[15] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. 2019. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–15.

[16] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863* (2018).

[17] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: Policy Enhanced Secure Object Store. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 25, 17 pages. https://doi.org/10.1145/3190508.3190518

[18] Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. 2018. Keys in the clouds: auditable multi-device access to cryptographic credentials. In *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 1–10.

[19] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.

[20] Jämes Ménétrey, Marcelo Pasin, Pascal Felber, and Valerio Schiavoni. 2021. Twine: An Embedded Trusted Runtime for WebAssembly. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 205–216. https://doi.org/10.1109/ICDE51399.2021.00025

[21] Shirin Nilizadeh, Yannic Noller, and Corina S Pasareanu. 2019. DifFuzz: differential fuzzing for side-channel analysis. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 176–187.

[22] Oleksii Oleksenko, Bohdan Trach, Mark Silberstein, and Christof Fetzer. 2020. SpecFuzz: Bringing Spectre-type vulnerabilities to the surface. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 1481–1498.

[23] Rafael Pires, Daniel Gavril, Pascal Felber, Emanuel Onica, and Marcelo Pasin. 2017. A Lightweight MapReduce Framework for Secure Processing with SGX. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 1100–1107. https://doi.org/10.1109/CCGRID.2017.129

[24] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. *A comprehensive study of convergent and commutative replicated data types.* Ph.D. Dissertation. Inria–Centre Paris-Rocquencourt; INRIA.

[25] Jörg Thalheim, Harshavardhan Unnibhavi, Christian Priebe, Pramod Bhatotia, and Peter Pietzuch. 2021. Rkt-Io: A Direct I/O Stack for Shielded Execution. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 490–506. https://doi.org/10.1145/3447786.3456255

[26] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2421–2434. https://doi.org/10.1145/3133956.3134038

[27] Gene TJ Wuu and Arthur J Bernstein. 1984. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*. 233–242.