Microshard™ Technical White Paper

Why Microshard™ Data?

Fragmenting data into tiny elements which are spatially dispersed and intermixed with other fragments (Microshard data) has numerous advantages in security and compliance. The fragment size can be chosen to statistically reduce the frequency or even eliminate the possibility of sensitive data and contextual metadata existing. This applies to both data at rest and, when the fragments are routed over multiple network paths, data in flight.

An attacker intercepting Microshard data has no way to put the pieces back together because, by definition, they have an incomplete set.

This is contrasted with encryption, in which the full set of data is compromised and needs to be unscrambled. Unscrambling data requires time and compute power. Reconstituting data fragments requires most or all of the data fragments, something the attacker cannot obtain without compromising all possible storage locations everywhere. We have effectively turned the attacker's challenge from a time and compute power based problem to a time and compute power and spatial problem.

Encryption may slow an attacker down, but Microshard data protection persists over time. Faster computers won't help an attacker, not even quantum computers. You can't unscramble data that you don't have.

Additionally, the option to ensure that no single (or statistically meaningful) set of data fragments contains a full element of sensitive data has compliance benefits. Once the Microshard data elements no longer contain such information, the files in which they sit are no longer sensitive. Much like tokenized data no longer needs to be protected by policies governing sensitive information, Microshard data similarly has stripped the files of the elements that made them sensitive. There is no need to treat files that contain no meaningful, sensitive data as if they contained such information.

Microshard data fragments aren't sensitive and need not be treated as such by policy. Files containing these data fragments aren't valuable as they can't be "cracked". We have reduced the attack surface from the edge of all data (large) to the single point capable of reassembling the data into its original form.

The size of the data fragments is set by policy to ensure that you have the level of fragmentation security you need for your data.

SHARDSECURE™

Inside the Engine

While running a system to Microshard data has obvious benefits, there are a few hurdles to overcome in building the technology. Chief among these are performance, streaming support, and system size. ShardSecure™ overcomes all of them.

Performance actually has 2 dimensions; throughput and latency. Throughput measures the I/O bandwidth of data as it's being read from (or written to) storage, while latency measures the delay before receiving (or storing) the first byte.

Sharding for performance has been around for many years, notably used in storage systems like RAID. In essence, you can compensate for slow throughput on disks by breaking data into blocks and writing those blocks in parallel to different physical drives. While our focus is security, and our shard size much smaller as a result, the concept of reading and writing Microshard data fragments in parallel allows us to compensate for any amount of overhead associated with fragmenting and managing the data. As long as we have a handful of destination files to read and write in parallel, we can overcome throughput concerns.

Need even more throughput without increasing remote file? No problem. We scale horizontally. You can add additional appliances and replicate their state. This also provides fault tolerance should an appliance fail.

The second part of performance is latency, or "time to first byte". Any inline system will add some latency, our goal is to minimize this as a percentage of the total time. To do this we make extensive use of compression of the pointers that refer to where data fragments are stored. Why? Because moving around small amounts of pointer data is faster than moving around large amounts of pointer data. We also do prefetching, caching, and leave connections to remote files open (to minimize the time for connection establishment)…all in the name of optimizing latency. The result is that we add just a few mS of latency to the first byte; far less than the typical time associated with accessing data at cloud storage providers (and similar to network latency for internal storage access).

Another challenge involves streaming data support. While it would be easy to operate at a "file level", waiting until an entire file has been transferred before fragmenting (write) or reassembling (read), that has two obvious disadvantages. First, latency would be huge on large files, creating unacceptable performance. Second, some applications open a file and rarely close it (databases, logging systems, etc. operate like this). If we didn't start working until the file was closed by the application, we might never start! We understand that any system unable to handle streams of data has limited value and we've designed ours for streaming from the start. ShardSecure's intense focus on high throughput, low latency, and optimization around data flows allows us to deliver streaming data.

Our final challenge is to minimize the resources needed to keep track of what could be billions of data fragments. Unlike encryption systems, we have pointers to each fragment of data so we can reassemble files and streams on the fly. Imagine if the average data fragment size was 10 bytes and the pointers to track where each fragment was stored were 16 bytes. Why 16 bytes? We have a remote system name or IP address, a file or object on that system, and an index into the file or object…for each fragment. The issue when every 10 bytes of data requires 16 bytes of information to locate it is that the storage requirements for the pointer database is 1.6 times the size of the actual data. Said differently, securing 100TB of data would require an appliance with 160TB of disk to store the pointers! We need the average pointer size to be a fraction of the average data fragment size. Our objective here is under 2 bytes to store a typical pointer.

ShardSecure's solution is to compress the pointers (compression also provides some of the performance benefits previously described). We use a variety of techniques, including tokenizing portions of the pointers, and using relative indexing to describe offsets within remote files and object. The result? Increased entropy (frustrating attackers) while dramatically shrinking the pointers.

Securing the Security

Remember that one of the methods we use for pointer compression was tokenization? That helped minimize pointer storage requirements and helped improve performance. It has a third benefit; securing the ShardSecure system itself against compromise. Should attackers get a copy of the ShardSecure pointer database, and know our algorithms for the computed portions of the pointers, they still wouldn't be able to find the remote data fragments. A key element of each pointer (the remote host's address) has been replaced by a token. An attacker might learn that a data fragment was in location 7, but without the mapping file to tell them where location 7 is (perhaps a file at Google Drive or a bucket at Amazon S3), they cannot reassemble the data. We persist our token mapping file in a different location from our database for added security.

In addition to tokenizing our pointers, access to the ShardSecure appliance (whether physical, virtual, or hosted) can be limited and secured from "front door" attacks while the appliance secures data from "side door" attacks. Protecting a single entry point is far easier than protecting access to all data everywhere. This is particularly true when the data to be protected from unauthorized access is created dynamically by different groups in your organization. Rather than tracking and protecting each file, you simply limit access to the fixed "front door" of the ShardSecure appliance.

As mentioned earlier, we have dramatically shrunk the attack surface from all data everywhere to a single front door of an appliance. In doing so, we can focus controls on this one point without having to worry about where large data sets may sit remotely (such as cloud environments).

Dropping It In

Having created a compact, high-performance, and streaming data capable Microshard engine, the next obvious question is "How can I use it with my existing infrastructure and apps?" ShardSecure has stuck to a very simple but powerful design philosophy; we look like a disk. Rather than trying to be application aware, and risking future incompatibilities, our engine is packaged as a virtual storage appliance.

If you run a database, we look like the underlying storage. If you have a custom app, we look like the underlying storage. It's that simple.

Our approach means that any data you can write to a disk can be protected by Microsharding it. Applications don't need to be modified in any way to secure their data. We support a set of storage interfaces that are typical of a NAS (Network Attached Storage) appliance. You can mount your virtual disk through NFS or as a Microsoft share (SMB). We also provide a RESTful API over HTTP that can be easily used by cloud services and internal applications. In short, you simply drop in the ShardSecure appliance, mount it like a disk, and you are protected.

SHARDSECURE™

Remote Storage

While our front door emulates network-based storage, the secured side of our appliance places Microshard data in files and objects on real storage. We support a number of destinations to place the "remote data" including:

1) Local disk
2) NFS drives
3) Microsoft SMB shares
4) Amazon EFS (via NFS)
5) FUSE
6) Amazon S3
7) Google Cloud Storage
8) Microsoft Azure Object Storage
9) IBM Cloud Object Storage
10) Box
11) Dropbox

ShardSecure has modularized our system to easily add additional types of storage providers. This means you can use the storage providers you have today and can be confident that you will be able to easily add or migrate to new ones in the future.

Implementation Examples

The ShardSecure virtual storage appliance can be deployed on-prem or hosted at a cloud service provider such as Amazon's AWS. Either way, it runs in a Linux VM. The choice of where to place your virtual appliance depends on where your applications run, your data flows and how you choose to protect the appliance itself. Some example use cases follow, along with possible deployment configurations.

Internal files

To secure data stored within your firewall, a ShardSecure appliance would be deployed on-prem. All data can stay inside your firewall, or you can choose to place some Microshard fragments at remote cloud providers.

Cloud storage

With internally hosted applications and users, a ShardSecure appliance would typically be hosted on-prem, but could be run at a remote cloud provider. Placing the appliance inside of your firewall allows for maximum security.

SHARDSECURE™

Cloud storage with cloud applications

When cloud based applications connect to cloud storage, a cloud hosted ShardSecure appliance makes the most sense. You will still limit access to the appliance's front door, and would typically place your host mapping file (to tokenize pointers) inside of your firewall. This provides the best balance of security and performance in an all-cloud world.

Cloud file sharing with partners

Securely sharing large files with trusted partners presents unique challenges and opportunities. Leveraging cloud storage is an obvious solution, but the underlying data needs to be protected from side door attacks (in a way more robust over time than encryption). A cloud-hosted ShardSecure appliance meets this need.