The Encrypted Elephant in the Cloud Room

Lori MacVittie, 2012-04-04

#infosec Encrypting data in the cloud is tricky and defies long held best practices regarding key management. New kid on the block Porticor aims to change that.

Anyone who's been around cryptography for a while understands that secure key management is a critical foundation for any security strategy involving encryption. Back in the day it was SSL, and an entire industry of solutions grew up specifically aimed at protecting the key to the kingdom – the master key. Tamper-resistant hardware devices are still required for some US Federal security standards under the FIPS banner, with specific security protections at the network and software levels providing additional assurance that the ever important key remains safe.

In many cases it's advised that the master key is not even kept on the same premises as the systems that use it. It must be locked up, safely, offsite; transported via a secure briefcase, handcuffed to a security officer and guarded by dire wolves. With very, very big teeth.

No, I am not exaggerating. At least not much. The master key really is that important to the security of cryptography.

That's why encryption in the cloud is such a tough nut to crack. Where, exactly, do you store the keys used to encrypt those Amazon S3 objects? Where, exactly, do you store the keys used to encrypt disk volumes in any cloud storage service?

Start-up Porticor has an answer, one that breaks (literally and figuratively) traditional models of key management and offers a pathway to a more secure method of managing cryptography in the cloud.

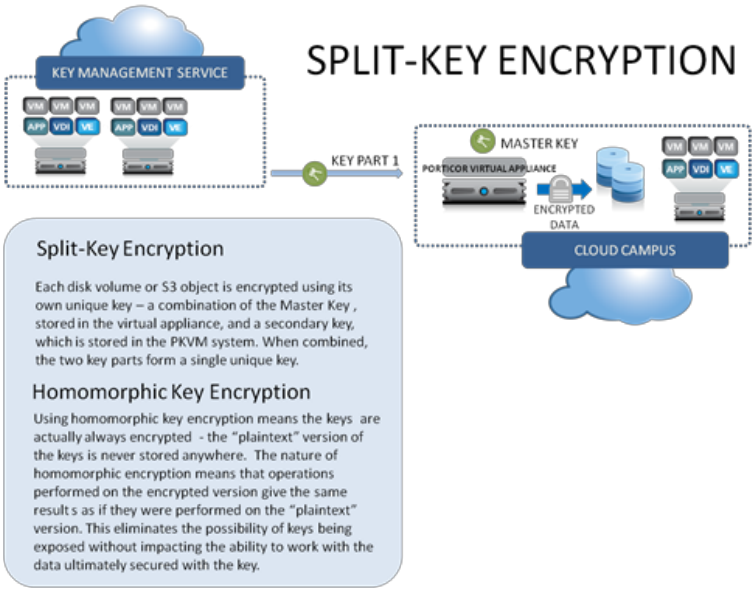
SPLIT-KEY ENCRYPTION

Porticor is a combination SaaS / IaaS solution designed to enable encryption of data at rest in IaaS environments with a focus on cloud, currently available on AWS and other clouds. It's a combination in not just deployment model – which is rapidly becoming the norm for cloud-based services – but in architecture, as well.

To alleviate violating best practices with respect to key management, i.e. you don't store the master key right next to the data it's been used to encrypt – Porticor has developed a technique it calls "Split-Key Encryption."

Data encryption comprises, you'll recall, the execution of an encryption algorithm on the data using a secret key, the result of which is ciphertext. The secret key is the, if you'll pardon the pun, secret to gaining access to that data once it has been encrypted. Storing it next to the data, then, is obviously a Very Bad Idea™ and as noted above the industry has already addressed the risk of doing so with a variety of solutions. Porticor takes a different approach by focusing on the security of the key not only from the perspective of its location but of its form.

The secret master key in Porticor's system is actually a mathematical combination of the master key generated on a per project (disk volumes or S3 objects) basis *and* a unique key created by the Porticor Virtual Key Management™ (PVKM™) system. The master key is half of the real key, and the PVKM generated key the other half. Only by combining the two – mathematically – can you discover the true secret key needed to work with the encrypted data.



The PVKM generated key is stored in Porticor's SaaS-based key management system, while the master keys are stored in the Porticor virtual appliance, deployed in the cloud along with the data its protecting.

The fact that the secret key can only be derived algorithmically from the two halves of the keys enhances security by making it impossible to find the actual encryption key from just one of the halves, since the math used removes all hints to the value of that key. It removes the risk of someone being able to recreate the secret key correctly unless they have both halves at the same time. The math could be a simple concatenation, but it could also be a more complicated algebraic equation. It could ostensibly be different for each set of keys, depending on the lengths to which Porticor wants to go to minimize the risk of someone being able to recreate the secret key correctly.

Still, some folks might be concerned that the master key exists in the same environment as the data it ultimately protects. Porticor intends to address that by moving to a partially homomorphic key encryption scheme.

HOMOMORPHIC KEY ENCRYPTION

If you aren't familiar with homomorphic encryption, there are several articles I'd encourage you to read, beginning with "Homomorphic Encryption" by Technology Review followed by Craig Stuntz's "What is Homomorphic Encryption, and Why Should I Care?" If you can't get enough of equations and formulas, then wander over to Wikipedia and read its entry on Homomorphic Encryption as well.

Porticor itself has a brief discussion of the technology, but it is not nearly as deep as the aforementioned articles.

In a nutshell (in case you can't bear to leave this page) homomorphic encryption is the fascinating property of some algorithms to work both on plaintext as well as on encrypted versions of the plaintext and come up with the same result. Executing the algorithm against encrypted data and then decrypting it gives the same result as executing the algorithm against the unencrypted version of the data.

So, what Porticor plans to do is apply homomorphic encryption to the keys, ensuring that the actual keys are no longer stored anywhere – unless you remember to tuck them away someplace safe or write it down. The algorithms for joining the two keys are performed on the encrypted versions of the keys, resulting in an encrypted symmetric key specific to one resource – a disk volume or S3 object.

The resulting system ensures that:

- No keys are ever on a disk in plain form
- Master keys are never decrypted, and so they are never known to anyone outside the application owner themselves
- The "second half" of each key (PVKM stored) are also never decrypted, and are never even known to anyone (not even Porticor)
- Symmetric keys for a specific resource exist in memory only, and are decrypted for use only when the actual data is needed, then they are discarded

This effectively eliminates one more argument against cloud – that keys cannot adequately be secured.

In a traditional data encryption solution the only thing you need is the secret key to unlock the data. Using Porticor's split-key technology you need the PVKM key and the master key used to recombine those keys. Layer atop that homomorphic key encryption to ensure the keys don't actually exist anywhere, and you have a rejoined to the claim that secure data and cloud simply cannot coexist.

In addition to the relative newness of the technique (and the nature of being untried at this point) the argument against homomorphic encryption of any kind is a familiar one: performance. Cryptography in general is by no means a fast operation and there is more than a decade's worth of technology in the form of hardware acceleration (and associated performance tests) specifically designed to remediate the slow performance of cryptographic functions. Homomorphic encryption is noted to be excruciatingly slow and the inability to leverage any kind of hardware acceleration in cloud computing environments offers no relief. Whether this performance penalty will be worth the additional level of security such a system adds is largely a matter of conjecture and highly dependent upon the balance between security and performance required by the organization.

Related blogs & articles:

- Getting at the Heart of Security in the Cloud
  Threat Assessment: Terminal Services RDP Vulnerability
  The Cost of Ignoring 'Non-Human' Visitors
  Identity Gone Wild! Cloud Edition
-

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com