Pragmatic Key Management for Data Encryption

Version 1.0
Released: September 13, 2012

Securosis, L.L.C. http://securosis.com

Author's Note

The content in this report was developed independently of any sponsors. It is based on material originally posted on the Securosis blog but has been enhanced and professionally edited.

Special thanks to Chris Pepper for editing and content support.

Licensed by Thales e-Security

Thales e-Security is a leading global provider of data protection solutions – delivering high assurance data encryption and key management solutions to the financial services, manufacturing, government, retail, healthcare, and technology sectors. The company has a 40-year track record of protecting sensitive corporate and government information across a wide range of technology areas including PKI, credential management, payment processing, network encryption, and many more. Thales e-Security solutions reduce the cost and complexity associated with the use of cryptography in today's traditional, virtualized, and cloud-based infrastructures, helping organizations reduce risk, demonstrate compliance, enhance agility, and pursue strategic goals with greater confidence. The company is represented in over 90 countries around the world. For more information, visit www.thales-esecurity.com.

Copyright

This report is licensed under the Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 license.

http://creativecommons.org/licenses/by-nc-nd/3.0/us/

Table of Contents

Introduction

Few terms strike as much dread in the hearts of security professionals as **key management**. Those two simple words evoke painful memories of massive PKI failures, with millions spent to send encrypted email to the person in the adjacent cube. Or perhaps it recalls the head-splitting migraine you got when assigned to reconcile incompatible proprietary implementations of a single encryption standard. Or memories of half-baked product implementations that worked fine in isolation on a single system, but were effectively impossible to manage at scale. And by scale, I mean "more than one".

Over the years key management has mostly been a difficult and complex process. This has been aggravated by the recent resurgence in data encryption – driven by regulatory compliance, cloud computing, mobility, and fundamental security needs.

Fortunately, encryption today is not the encryption of yesteryear. New techniques and tools remove much of the historical pain of key management – while also supporting new and innovative uses.

We also see a change in how organizations approach key management – a move toward practical and lightweight solutions.

In this paper we will explore the latest approaches for pragmatic key management. We will start with the fundamentals of crypto *systems* rather than encryption *algorithms*, what they mean for enterprise deployment, and how to select a strategy that suits your particular project requirements.

The historic pain of key management

Technically there is no reason key management needs to be as hard as it has been. A key is little more than a blob of text to store and exchange as needed. The problem is that everyone implements their own methods of storing, using, and exchanging keys. No two systems worked exactly alike, and many encryption implementations and products didn't include the features needed to use encryption in the real world – and still don't.

Many products with encryption features supported only their own proprietary key management – which often failed to meet enterprise requirements in key management lifecycle areas such as key generation, rotation, storage, backup, and destruction or important security and compliance needs like separation of duties and reporting. Encryption is featured in many different types of products but developers who plug an encryption library into an existing tool have (historically) rarely had enough experience in key management to produce refined, easy to use, and effective systems.

On the other hand, some security professionals remember early failed PKI deployments that cost millions and provided little value. This was at the opposite end of the spectrum – key management deployed for its own sake, without thought given to how the keys and certificates would be used or even what they would be protecting.

Why key management isn't as hard as you think it is

As with most technologies, key management has advanced significantly since those days. Current tools and strategies offer a spectrum of possibilities, all far better standardized and with much more robust management capabilities.

We no longer have to deploy key management with an all-or-nothing approach, either relying completely on local management or on an enterprise-wide deployment. Increased standardization (potentially powered in part by KMIP, the Key Management Interoperability Protocol) and improved, enterprise-class key management tools make it much easier to fit deployments to requirements.

Products that implement data encryption now tend to include better management features, with increased support for external key management systems when those features are insufficient. We now have smoother migration paths which support a much broader range of scenarios.

I am *not* saying life is now perfect. There are plenty of products that still rely on poorly implemented key management and don't support standards or other ways of integrating with external key managers, but fortunately they are slowly dying off or being fixed due to constant customer pressure. Additionally, dedicated key managers often support a range of non-standards-based integration options for those laggards.

It isn't always great, but it is much easier to mange keys now than even a few years ago.

The new business drivers for encryption and key management

These advances are driven by increasing customer use of, and demand for, data encryption. We can trace this back to 3 primary drivers:

- Expanding and sustained regulatory demand for encryption. Encryption has always been hinted at by a variety of regulations, but it is now mandated in industry compliance standards (most notably the Payment Card Industry Data Security Standard – PCI-DSS) and certain government regulations. Even when it isn't mandated, most breach disclosure laws reduce or eliminate the need to publicly report loss of client information if the lost data was encrypted.
- Increasing use of cloud computing and external service providers. Customers of cloud and other hosting providers want to protect their data when they give up physical control of it. While the provider often has better security than the customer, this doesn't reduce our visceral response to *someone else* handling our sensitive information.
- The increase in public data exposures. While we can't precisely quantify the growth of actual data loss, it is certainly far more public than it has ever been before. Executives who previously ignored data security concerns are now asking security managers how to stay out of the headlines.

> *Note: For simplicity I will often use "encryption" instead of "cryptographic operation" through this paper. If you're a crypto geek, don't get too hung up… I know the difference – it's for readability.*

More enforcement of more regulations, increasing use of outsiders to manage our data, and increasing awareness of data loss problems, are all combining to produce the greatest growth the encryption market has seen in a long time.

Key management isn't just about data encryption (but that's our focus here)

Before we delve into how to manage keys, it is important to remember that cryptographic keys are used for more than just encryption, and that there are many different kinds of encryption.

Our focus in this report is on *data encryption* – not digital signing, authentication, identity verification, or other crypto operations (although many of these issues apply to any implementation of key management). We will not spend much time on digital certificates, certificate authorities, or other signature-based operations. Instead we will focus on data encryption, which is only one area of cryptography.

Much of what we see is as much a philosophical change as improvement in particular tools or techniques. I have long been bothered by people's tendency to either indulge in encryption idealism at one end, and or dive into low-level details that don't practically affect security at the other end – both reinforced by our field's long-running ties to cryptography. But with the new pressures to encrypt more information in more places (while keeping auditors happy), we are finally seeing much more focus on pragmatic implementation.

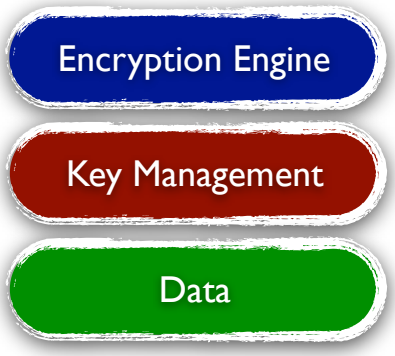
Next we will cover the major components of an encryption system, and how they affect key management options. We will follow up with the four major key management strategies, and suggestions for how to pick the right one for your requirements.

Understanding Data Encryption Systems

One of the common problems in working with encryption is getting caught up with the intimate details of things like encryption algorithms, key lengths, cipher modes, and other minutiae. Not that these details aren't important – depending on what you're doing they might be critical – but in the larger scheme of things these aren't the aspects most likely to trip up your implementation. Before we get into different key management strategies, let's take a moment to look at crypto systems at the macro level. We will stick to data encryption for this paper, but these principles apply to other types of cryptosystems as well.

The three components of a data encryption system

Three major components define the overall structure of an encryption system:



- **The data:** The object or objects to encrypt. It might seem silly to break this out, but the security and complexity of the system are influenced by the nature of the payload, as well as by where it is located or collected.
- **The encryption engine:** The component that handles the actual encryption (and decryption) operations.
- **The key manager:** The component that handles keys and passes them to the encryption engine.

In a basic encryption system all three components are likely located on the same system. Take personal full disk encryption (the default you might use on your home Windows PC or Mac) – the encryption key, data, and engine are all kept and run on the same hardware. Lose that hardware and you lose the key and data – and the engine, but that isn't normally relevant.

But once we get into SMB and the enterprise we tend to split out the components for security, management, reliability, and compliance.

Building a data encryption system

Where you place these components defines the structure, security, and manageability of your encryption system. Here are a few common examples:

Full Disk Encryption

Our full disk encryption example above isn't the sort of approach you would want to take for an organization of any size greater than 1. All major FDE systems do a good job of protecting the key if the device is lost, so we aren't worried about

security too much from that perspective, but managing the key on the local system means the system is much less manageable and reliable than if all the FDE keys are stored together.

Enterprise-class FDE manages the keys centrally – even if they are also stored locally – to enable a host of more advanced functions; including better recovery options, audit and compliance, and the ability to manage hundreds of thousands of systems.

Database encryption

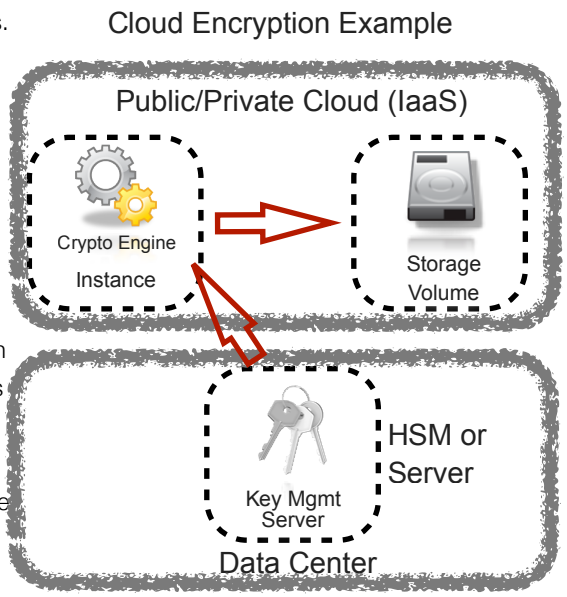
Let's consider another example: database encryption. By default, all database management systems (DBMS) that support encryption do so with the data, the key, and the encryption engine all within the DBMS. But you can mix and match those components to satisfy different requirements.

The most common alternative is to pull the key out of the DBMS and store it in an external key manager or HSM (Hardware Security Module). This can protect the key from compromise of the DBMS itself, and increases separation of duties and security. It also reduces the likelihood of lost keys and enables extensive management capabilities – including easier key rotation, expiration, and auditing.

But the key *could* be exposed to someone on the DBMS host itself because it *must* be stored in memory before it can be used to encrypt or decrypt. One way to protect against this is to pull *both* the encryption engine and key out of the DBMS. This could be handled through an external proxy, but more often custom code is developed to send the data to an external encryption server or appliance. Of course this adds complexity and latency.

Cloud encryption

Cloud computing has given rise to a couple of additional scenarios.



- To protect an Infrastructure as a Service (IaaS) storage volume running at an external cloud provider you can place the encryption engine in a running instance, store the data in a separate volume, and use an external key manager which could be a hardware appliance connected through VPN and managed in your own data center.
- To protect enterprise files in an object storage service like Amazon S3 or RackSpace Cloud Files, you can encrypt them on a local system before storing them in the cloud – managing keys either on the local system or with a centralized key manager. While some of these services support built-in encryption, they typically store and manage the key themselves, which means the provider has the (hopefully purely theoretical) ability to access your data. But if you control the key and the encryption engine the provider cannot read your files.

Backup and storage encryption

Many backup systems today include some sort of an encryption option, but the implementations typically offer only the most basic key management. Backup up in one location and restoring in another may be a difficult prospect if the key is stored only in the backup system.

Additionally, backup and storage systems themselves might place the encryption engine in any of a wide variety of locations – from individual disk and tape drives, to backup controllers, to server software, to inline proxies.

Some systems store the key with the data – sometimes in special hardware added to the tape or drive – while others place it with the engine, and still others keep it in an external key management server.

Between all this complexity and poor implementations by storage vendors, I tend to see external key management used for backup and storage more than for just about any other data encryption usage.

Application encryption

Our last example is application encryption. One of the more secure ways to encrypt application data is to collect it in the application, send it to an encryption server or appliance (or embed it in the application code), and then store the encrypted data in a separate database. The keys themselves might be on the encryption server, or could even be stored in yet another system. The separate key store offers greater security, simplifies management for multiple encryption appliances, and helps keep keys safe for data movement – backup, restore, and migration/synchronization to other data centers.

In each of these examples we see multiple options for where to place the components – with the corresponding tradeoffs in security, manageability, and other aspects. This list isn't comprehensive but should give you a taste of the different ways these bits and pieces can be mixed and matched for different data encryption systems.

Next we will build on this to define the four major key management strategies, followed by recommendations for how to pick the right options to suit our needs.

The Four Key Management Strategies

In the last section we covered the components of data encryption systems and ran through some common examples. Now it's time to move on to key management itself, and dig into the four different key management strategies.

We need to start with a discussion of the differences between encryption operations and key management; then we will detail the different enterprise-level strategies.

The differences between key management and encryption operations

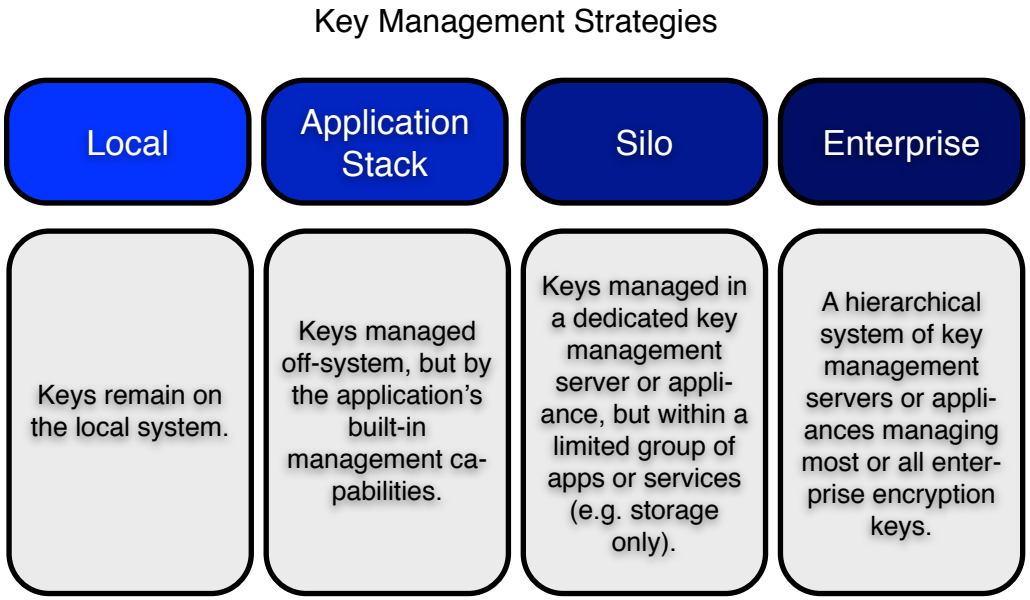
As we focus on data encryption across the organization rather than isolated applications of basic encryption, it is time to spend a moment on what we mean when we discuss key management vs. encryption operations.

Every data encryption operation involves a key, so there is *always* a key to manage, but a full-fledged management system is the most important aspect of building a multipart encryption system.

Many data encryption systems don't bother with "real" key management – they only store keys locally, and users never interact with the keys directly. For example, if you encrypt data with a passphrase using one of the many common command-line tools available, the odds are good that you don't do anything with the key beyond choosing an encryption algorithm and key length. Super-simple implementations don't bother to store the key at all – it is generated as needed from the passphrase. In slightly more complex (but still relatively simple) cases the key is actually stored with the data, protected by a series of other keys which are still generated from passphrases.

There is a clear division between this and the enterprise model, where you actively manage keys. Key management involves separating keys from data for increased flexibility and security. It does *not* require you to move keys to an external system, but that is one of the more important options. You can have multiple keys for the same data, the same key for multiple files, key backup and recovery, and many more choices.

There are four main approaches to managing data encryption keys within an organization. These apply to individual cryptosystems, to various different kinds of applications, and to larger and more complicated cryptography systems. Many of them also apply to other kinds of encryption operations, such as digital signatures and certificates, but we aren't concerned with those for this paper.



Local key management

This option is the closest to doing nothing at all for key management. Keys are all managed locally (on a single system or a cluster of systems), with all key functions handled within a single application.

Local key management is actually quite common, even though it isn't always the best idea. Common examples include:

- Full disk encryption managed by a single user (*e.g.,* Bitlocker or FileVault without tying into a key management server)
- Transparent database encryption
- Building encryption into an application server
- Basic backup encryption
- File server or SAN/NAS encryption

In each of these cases all keys can be managed locally – in which case any key rotation, backup/restore, or auditing also must be built into the local system, but more often these capabilities are simply nonexistent.

Local key management isn't necessarily bad, in particular isolated scenarios. For example, if you back up your data unencrypted, or with a system that uses its own keys, there may be no reason to worry about managing local keys. But for anything serious – including anything with compliance requirements – relying on local key management is asking for trouble.

Application stack key management

This refers to separating the keys from the local system and managing them within a multi-instance application. Whatever software stack/system you run manages its own keys for its own client software.

Full disk encryption is one of the most common enterprise examples. A central management server handles configuration and keys for all encrypted laptops and desktops that use that vendor's software. This key management system is never used for anything else, such as databases, but may manage other data encryption features supported by the product

(including file/folder encryption). All important key management functions, including administrative and recovery keys, rotation, backup/restore, and audit, are built into the application stack's key manager.

Application stacks are closed, proprietary systems that may involve distributed instances. They implement encryption within their own software components, and aren't designed to manage it externally. That said, many support use of external key management.

Other typical uses include email encryption, some backup encryption tools, and even enterprise Digital Rights Management – DRM is implemented through cryptography.

Application stack key management is totally suitable when it meets the particular requirements of the situation. When encryption is the key function of a product, as with full disk encryption, this approach often works perfectly – with no need for additional key management. On the other hand, when encryption is merely a feature of an existing product, key management is often minimal at best – typified by encryption products bolted onto existing backup systems.

Silo key management

So far the two strategies we have discussed keep the keys within a single system or application stack. The next couple strategies introduce a new component: a dedicated key management system.

> ***Silo key management*** uses a dedicated key manager, while ***application stack key management*** relies on key management features built into your application software.

When local or application stack key management is inadequate, it's time to bring in a tool specifically to address the problem. Move keys outside the application and integrate dedicated key management with one or more application stacks. This used to be incredibly difficult, but more and more products (both commercial and free software / Open Source) now support key management standards that make it much easier to use external management. Before standards we had to either rely on the vendor to provide proprietary hooks, or reverse engineer the entire thing.

We call this "silo" key management since a key manager is typically used for one or more applications managed within a single organizational silo. For example, you might have one silo to manage full disk encryption keys, another for database and application keys, and another for storage keys. Each of these silos is managed independently, on different hardware/software.

A variety of dedicated key management options are available – including hardened hardware appliances (Hardware Security Modules, HSMs), software, virtual appliances, and even Software as a Service (SaaS). We are focusing on key management strategies rather than products, so we won't go into all the various features and functions, but suffice it to say they tend to have far more robust capabilities (and often stronger security) than all but the best application stack tools. Aside from all the added functionality of an external service, the external service can manage keys for multiple different application stacks. This can be important for unifying auditing/reporting and meeting other compliance requirements.

Key management tools also reduce the overhead and complexity of encryption operations – especially for application and database encryption, where application stack management often isn't available. Using APIs and plugins, your developers and DBAs don't need to reinvent the wheel; something very few people – including crypto experts – manage to do securely. This approach also removes keys from the systems involved when they aren't needed, further benefiting security. Hardened encryption engines that link up with external key managers offer high-security modes, where they do things like pull the key down for a single operation, use it, and then overwrite the key's memory addresses to completely eliminate it from the system.

Not to give away the next section, but if a data encryption feature or software doesn't include centralized key management, or your application stack key management software doesn't provide all the functions you need, it's time to move up to a dedicated key management service. The most common places we see these are backup and storage encryption, application and database encryption, and data encryption for cloud services. But at this level we are still dealing with a somewhat-isolated silo versus an enterprise strategy.

Enterprise key management

Building on silo key management, enterprise key management adds a "manager of managers" to centralize most or all key management within the organization. We are focused on data encryption key management, but this may also encompass keys for other operations such as certificate management. The manager adds features to broaden its scope; such as improved separation of duties; integration and management of other dedicated key managers; the ability to segregate keys and users based on role, use, etc.

While an organization might have a collection of different key management services in different silos, enterprise key management ties them all together with central administration and management. Practically speaking there will probably still be some silos, but this strategy embraces and manages keys for at least most encrypted data.

Although enterprise key management has been discussed for years, it is more of a vision organizations strive towards and used by relatively few enterprises. We predominantly see it within financial services, retail, and other companies with need for distributed encryption operations across application silos. In some cases, we even see it used to exchange keys between organizations.

Enterprise key management can also play a key role in use cases beyond data encryption- especially certificate, identification, and signing functions where keys and certificates need to work across different systems, applications, or even functions. We'll go into more selection details in the next section, but organizations should take a look at enterprise key management if they anticipate ever needing cross-domain/silo keys, such as those used to encrypt data in an application, but then decrypt or use it in other applications, databases, or analysis tools.

To recap, the four key management strategies are:

- Manage keys locally
- Manage keys within an single application stack with a built-in key management feature
- Manage keys for a silo using an external key management service/server/appliance, separate from the data and application stacks
- Coordinate management of (most or all) keys across the enterprise with a centralized key management tool

Next we will talk about how to choose your strategy, and when to switch between these options.

Choosing Your Key Management Strategy

In our last section we covered the four enterprise key management strategies. Here's how to pick the right strategy for your organization.

To recap, there are four key management strategies:

- Local management
- Application stack management
- Silo management
- Enterprise key management

As much as I would like to drag this out into a long and complex assessment process, it's actually fairly simple:

- You should never use local key management for anything other than development, testing, and one-off applications. About the only thing I use it for is some personal encryption, and not even much of that.
- Stick with application stack management if it meets your needs, but this generally only works for encryption-oriented products such as full disk encryption, email, and a couple other cases. By 'needs' I mean everything from basic manageability and auditing/reporting all the way through administrator separation of duties, key rotation/backup/restore, multi-location key synchronization and replication, and all sorts of other requirements beyond the scope of this series.
- When local and application stack won't work, building a silo with a key management service is the way to go.
- Full enterprise key management is nice to have, but not something to focus on at the start.

If you do stick with application stack management but need a key manager for another project, it is often worthwhile to transition your applications over to the silo key manager; once you have a key manager you might as well take advantage of it for backup, restore, redundancy, and other management features.

Here are some of the reasons to move up the stack, especially from application stack management to an external key manager:

- If you require more robust reporting (especially for compliance) than is included in your application stack's key management. For example, some compliance initiatives require secure (segregated) logging and reporting not only of all key operations (like key rotations), but also any administrator access to keys or key management functions.

- If you're concerned about key backup and restore processes, especially access to older archived data. For example, if you encrypt a backup you might need to access it up to seven years later, even if you've upgraded or changed which backup software you use.
- If you need greater granularity in administrative control, such as split keys or m-of-n administrator keys for high-level access (where you need something like 3 of 4 potential administrators to all provide their key before allowing an action).
- Not all application stack key management can handle re-keying of older data during a key rotation, which may be a requirement.
- When you need to manage keys across different applications that aren't fully integrated into the application stack (this is also often the first step towards enterprise key management).
- When you need to synchronize or replicate keys across locations and this isn't supported by the application.

This isn't an exclusive list, but some of the more common reasons we see for moving to external key management in data encryption implementations.

The key is to think strategically. Once you start managing multiple encryption applications, you will eventually move into some sort of dedicated key manager. To build a key management service in a silo, pick a platform that will grow as you increase usage – even if the first deployment is narrowly scoped. People often start with a single application, database, or storage encryption project – a silo where key management is poor or doesn't exist. But don't choose purely based on immediate requirements – pick something that meets your immediate needs and can expand into other areas, for example by providing a backup key manager for disk encryption.

We see two common problems when people build key management strategies. The first is that they *don't* build strategically. Everyone buys or builds key management for each project, rather than offering and taking advantage of a central service whenever possible. On the other end of the spectrum, organizations obsess over implementing enterprise key management but forget to properly manage their silos and projects.

We see the best success when organizations plan strategically and then grow into broader key management. Practically speaking, this typically starts with a single project using a dedicated key manager, which is then expanded and leveraged for other complementary projects. It's fine to keep some application stack managers, and it's okay to have key managers in their own silos when there is no need to plug them into something larger. For example, you don't necessarily need to have both your database encryption and full disk encryption projects report up to a single enterprise key manager.

We have mentioned this before, but sweet spots which may justify moving up to a key manager include:

- Backup encryption, due to a mix of longevity needs and very limited key management implementations in the backup products themselves.
- Database encryption, since most database management systems only include the most rudimentary key management, and rarely the ability to centrally manage keys across different database instances or segregate keys from the database administrators.
- Application encryption, which nearly always relies on a custom encryption implementation and, for security reasons, should separate key management from the application itself.
- Cloud encryption, due to the high volume of keys and variety of deployment scenarios faced.

In all four areas we tend to see strong need for encryption but weak key management.

To recap: avoid local management; application stack managers are fine when they meet your needs; step projects up to external key managers when it makes sense for the project; expand coverage over time; and stick with one platform for cleaner management when feasible. Key management and how you structure your crypto system both matter more than the encryption engine itself. We haven't discussed key manager selection criteria (fodder for a future report); but it should be obvious that deployment is easier when products support standards, include good APIs and plugins, and play well out of the box with common platforms and software.

You should now have a much better idea of how data encryption *systems* work, the different strategies for managing encryption keys, and how to pick the best one for your organization.

Who We Are

About the Author

Rich Mogull, Analyst and CEO

Rich has twenty years of experience in information security, physical security, and risk management. He specializes in data security, application security, emerging security technologies, and security management. Prior to founding Securosis, Rich was a Research Vice President at Gartner on the security team where he also served as research co-chair for the Gartner Security Summit. Prior to his seven years at Gartner, Rich worked as an independent consultant, web application developer, software development manager at the University of Colorado, and systems and network administrator. Rich is the Security Editor of TidBITS, a monthly columnist for Dark Reading, and a frequent contributor to publications ranging from Information Security Magazine to Macworld. He is a frequent industry speaker at events including the RSA Security Conference and DefCon, and has spoken on every continent except Antarctica (where he's happy to speak for free — assuming travel is covered).

About Securosis

Securosis, L.L.C. is an independent research and analysis firm dedicated to thought leadership, objectivity, and transparency. Our analysts have all held executive level positions and are dedicated to providing high-value, pragmatic advisory services.

We provide services in four main areas:

- Publishing and speaking: Including independent objective white papers, webcasts, and in-person presentations.
- Strategic consulting for end users: Including product selection assistance, technology and architecture strategy, education, security management evaluations, and risk assessments.
- Strategic consulting for vendors: Including market and product analysis and strategy, technology guidance, product evaluations, and merger and acquisition assessments.
- Investor consulting: Technical due diligence including product and market evaluations, available in conjunction with deep product assessments with our research partners.

Our clients range from stealth startups to some of the best known technology vendors and end users. Clients include large financial institutions, institutional investors, mid-sized enterprises, and major security vendors.

Securosis has partnered with security testing labs to provide unique product evaluations that combine in-depth technical analysis with high-level product, architecture, and market analysis.