Cracking the Confusion: Encryption and Tokenization for Data Centers, Servers, and Applications

Version 1.0
Released: March 11, 2015

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 Vormetric

Vormetric (@Vormetric) is the industry leader in data security solutions that protect data-at-rest across physical, big data and cloud environments. Vormetric helps over 1500 customers, including 17 of the Fortune 30, to meet compliance requirements and protect what matters — their sensitive data — from both internal and external threats. The company's scalable Vormetric Data Security Platform protects any file, any database and any application's data —anywhere it resides — with a high performance, market-leading solution set.

Learn more at http://vormetric.com

Copyright

Table of Contents

The New Age of Encryption

Data encryption has long been part of the information security arsenal. From passwords, to files, to databases, we rely on encryption to protect our data in storage and on the move. It's a foundational element in any security professional's education. But despite its long history and deep value, adoption inside data centers and applications has been relatively — even surprisingly — low.

Today we see encryption growing at an accelerating rate in data centers, for a confluence of reasons. A trite way to summarize them is "compliance, cloud, and covert affairs". Organizations need to keep auditors off their backs; keep control over data in the cloud; and stop the flood of data breaches, state-sponsored espionage, and government snooping (even by their own governments).

Thanks to increasing demand we have a growing range of options, as vendors and even free and Open Source tools address this opportunity. We have never had more choice, but with choice comes complexity — and outside your friendly local sales representative, guidance can be hard to come by.

For example, given a single application collecting an account number from each customer, you could encrypt it in any of several different places: the application, the database, or storage — or use tokenization instead. The data is encrypted (or substituted), but each place you might encrypt raises different concerns. What threats are you protecting against? What is the performance overhead? How are keys managed? Does it all meet compliance requirements?

This paper cuts through the confusion to help you pick the best encryption options for your projects. In case you couldn't guess from the title, our focus is on encrypting in the data center: applications, servers, databases, and storage. Heck, we will even cover cloud computing (IaaS: Infrastructure as a Service), although we covered it in depth in another paper. We will also cover tokenization and discuss its relationship with encryption.

We won't cover encryption algorithms, cipher modes, or product comparisons. We will cover different high-level options and technologies, such as when to encrypt in the database vs. in the application, and what kinds of data are best suited for tokenization. We will also address key management, essential platform features, and how to tie it all together.

Understanding Encryption Systems

When security professionals first learn about encryption the focus is generally on keys, algorithms, and modes. We learn the difference between symmetric and asymmetric, and spend a lot of time talking about Bob and Alice.

Once you start working in the real world your focus needs to change. Those fundamentals are still important but now you need to put them into practice as you implement an *encryption system* — a combination of technologies that actually protects data. Even the strongest crypto algorithm is worthless if the system around it is full of flaws.

Before we go into specific scenarios let's review the basic concepts underlying encryption systems; these will guide decisions on which encryption options to use.

The Three Components of a Data Encryption System

When encrypting data, especially in applications and data centers, knowing how and where to place these pieces is incredibly important, and mistakes here are one of the most common causes of failure. We use all our data at some point, so understanding where the exposure points are, where the encryption components reside, and how they tie together, all determine how much actual security we end up with.

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

- **Data:** The object or objects to encrypt. It might seem silly to break this out, but the security and complexity of the system depend on the nature of the payload, as well as where it is located or collected.
- **Encryption Engine:** This component handles actual encryption (and decryption) operations.
- **Key Manager:** This handles keys and passes them to the encryption engine.

In a basic encryption system all three components are likely located on the same system. As an example take personal full disk encryption (the built-in tools you might use on your home Windows PC or Mac): the encryption key, data, and engine are all stored and used on the same hardware. Lose that hardware and you lose the key and data — and the engine, but that isn't generally a concern for FDE. (Neither is the key, usually, because it is protected with another key, or passphrase, that is *not* stored on the system — but if the system is lost while running, with the key in memory, that becomes a problem). For data centers these components are likely to reside on different systems, increasing complexity and security concerns over how they work together.

Building an Encryption System

In a straightforward application we normally break out the components — such as the encryption engine in an application server, the data in a database, and key management in an external service or appliance.

Or for a legacy application we might instead enable *Transparent Database Encryption* (TDE) for the database, with the encryption engine and data both on the same server but key management elsewhere.

All data encryption systems are defined by where these pieces are located — which, even assuming everything works perfectly, control how well the data can be protected. We will discuss the different layers of data encryption in the next section, but at a high level they are:

- In the application where you collect the data.
- In the database that holds the data.
- In the files where data is stored.
- On the storage volume (typically a hard drive, tape, or virtual storage) where the files reside.

All data flows through that stack (sometimes skipping applications and databases for unstructured data). Encrypt at the top and the data is protected all the way down, but this adds complexity to the system and isn't always possible. Once we start digging into the specifics of different encryption options you will see that defining your requirements almost always naturally leads you to select a particular layer, which then determines where to place the components.

The Three Laws of Data Encryption

Years ago we developed the *Three Laws of Data Encryption* as a tool to help guide the encryption decisions listed above. When push comes to shove, there are only three reasons to encrypt data:

1. If the data moves, physically or virtually.
2. To enforce separation of duties beyond what is possible with access controls. Usually this only means protecting against administrators, because access controls can stop everyone else.
3. Because someone says you must. We call this "mandated encryption".

Here is an example use of the Laws. Let's say someone tells you to "encrypt all the credit card numbers" in a particular application. We will further say the reason is to prevent loss of data in case a database administrator account is compromised, which eliminates our third reason.

The data isn't necessarily moving, but we want separation of duties to protect the database even if someone steals administrator credentials. Encrypting at the storage volume layer wouldn't help, because a compromised administrative account still has access within the database. Encrypting the database files alone wouldn't help either, because for the database to work authorized users work inside the database where the files are no longer encrypted.

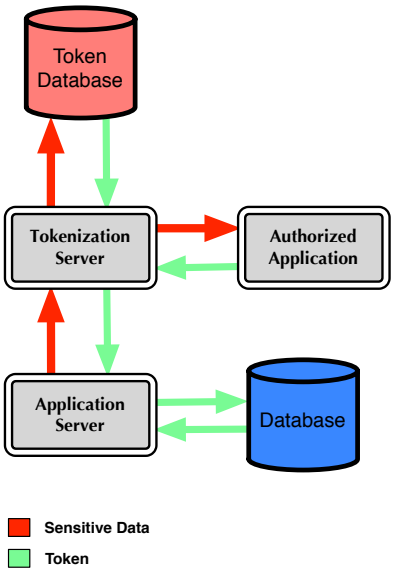
Encrypting within the database is an option, depending on where the keys are stored (they must be outside the database) and some other details which we will get to later. Encrypting in the application definitely helps because that is completely outside the database. But in either case you still need to know when and where an administrator could potentially access decrypted data.

That's how it all ties together. Know why you are encrypting, then where you can potentially encrypt, then how to position the encryption components to achieve your security objectives.

Tokenization and Data Masking

Two alternatives to encryption are sometimes offered in commercial encryption tools: tokenization and data masking. We will spend more time on them later, but for now we will simply define them:

- **Tokenization** replaces a sensitive piece of data with a random piece of data that can fit the same format (such as by looking like a credit card number without actually being a valid credit card number). The sensitive data and token are then stored together in a highly secure database for retrieval under limited conditions, and the non-sensitive token is available for use throughout the system.
- **Data Masking** replaces sensitive data with random data, but the two aren't stored together for later retrieval. Masking can be a one-way operation such as generating a test database, or a repeatable operation such as dynamically masking a specific field for an application user based on permissions.



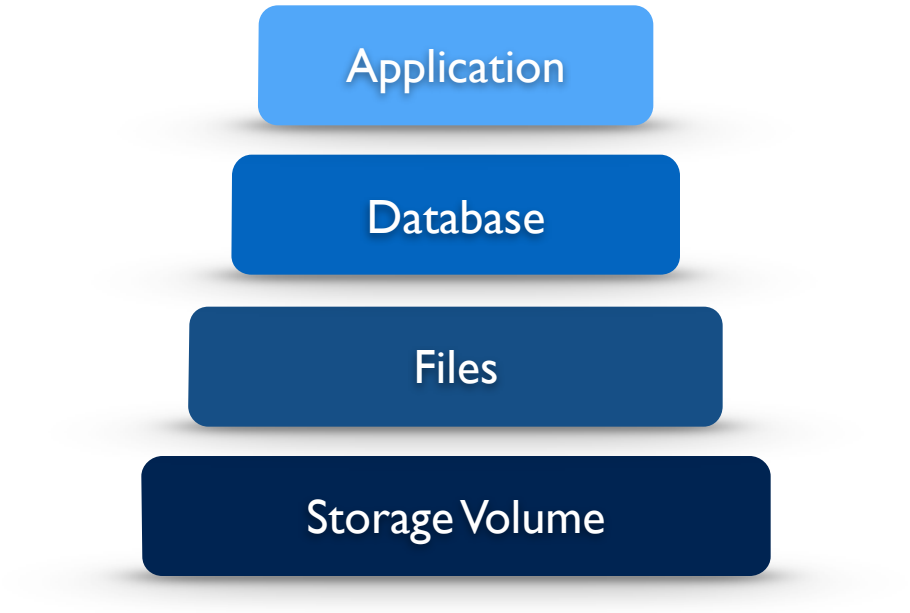
For more information on tokenization vs. encryption you can, read our paper.

That covers the basics of encryption systems. Our next section goes into details of the encryption layers above before delving into key management, platform features, use cases, and a decision tree for picking the right option.

Encryption Layers

You can picture enterprise applications as a layer cake: applications sit on databases, databases on files, and files are mapped onto storage volumes. You can use encryption at each layer in your application stack: within the application, in the database, on files, or on storage volumes. *Where* you use an encryption engine dominates security and performance. Higher up the stack can offer stronger security, with higher complexity and performance cost.

There is a similar tradeoff with encryption engine and key manager deployments: more tightly coupled systems offer less complexity, but also weaker security and reliability. Building an encryption system requires a balance between security, complexity, and performance. Let's take a closer look at each layer and the various tradeoffs.



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 an encryption library embedded in the application), and then store the encrypted data in a separate database. The application has full control over who sees what so it can secure data without depending on the security of the underlying database, file system, or storage volumes. The keys themselves might be on the encryption server or could be stored in yet another system. The separate key store increases security, simplifies management of multiple encryption appliances, and helps keep keys safe for data movement: backup, restore, and migration/synchronization to other data centers.

Database Encryption

Relational database management systems (RDBMS) typically offer two encryption options: transparent and column. In the layer cake above columnar encryption occurs as applications insert data into the database, whereas transparent encryption occurs as the database writes data out. Transparent encryption is applied automatically to data before it is

stored at the file or disk layer. In this model encryption and key management happen behind the scenes, without the user's knowledge, and without requiring application programming. The database management system handles encryption and decryption operations as data is read (or written), ensuring *all* data is secured, and offering very good performance. When you need finer control over data access you can encrypt single columns, or tables, within the database. This approach offers the advantage that only authenticated users of encrypted data are able to gain access, but it requires changing database or application code to manage encryption operations. With either approach there is less burden on application developers to build a crypto system, but slightly less control over who can access sensitive data.

Some third-party tools also offer transparent database encryption by automatically encrypting data as it is stored in files. These tools aren't part of the database management system itself, so they can work with databases that don't support TDE directly; they provide greater separation of duties for database administrators, as well as better protection for file-based output like reports and logs.

File Encryption

Some applications, such as payment systems and web applications, do not use databases; instead they store sensitive data in files. Encryption can be applied transparently as the data is written to files. This type of encryption is either offered as a third-party add-on to the file system, or embedded within the operating system. Encryption and decryption are transparent to both users and applications. Data is decrypted when a user requests a file, after they have authenticated to the system. If the user does not have permission to read the file, or has not provided proper credentials, they only get back useless encrypted data. File encryption is commonly used to protect "data at rest" in applications that do not include encryption capabilities — including legacy enterprise applications and many big data platforms.

Disk/Volume Encryption

Many off-the-shelf disk drives and Storage Area Network (SAN) arrays include automatic data encryption. Encryption is applied as data is written to disk, and decrypted for authenticated users/applications when requested. Most enterprise-class systems hold encryption keys locally to support encryption operations, but rely on external key management services to manage keys and provide advanced key services such as key rotation. Volume encryption protects data in case drives are physically stolen. Authenticated users and applications have access to unencrypted data.

Tradeoffs

In general, the further "up the stack" you deploy encryption, the more secure your data is. The price of that extra security is more difficult integration, usually in the form of application code changes. Ideally we would encrypt all data at the application layer and fully leverage user authentication, authorization, and business context to determine who can see sensitive data. In the real world, the code changes required for this level of precise control often pose insurmountable engineering challenges or are cost prohibitive. Surprisingly, transparent encryption often perform *faster* than application-layer encryption, even with larger data sets. The tradeoff is moving high enough "up the stack" to address relevant threats while minimizing the pain of integration and management. Later in this series we will walk you through the selection process in detail.

Key Management Options

As mentioned back in our opening, the key (pun intended — please forgive us) to an effective and secure encryption system is proper placement of the components. The one that most influences the overall system is the key manager.

You *can* encrypt without a dedicated key manager. We know numerous applications that take this approach. We also know numerous applications that break, fail, and get breached. You should nearly always use a dedicated external key management option; they break down into four types.

- An **HSM or other hardware key management appliance**. This provides the highest level of physical security. It is the most common option in sensitive scenarios such as financial services. The HSM or appliance runs in your data center, and you always need more than one for backup. Lose access and you lose your keys. A *hardware root of trust* is the most secure option, and all those products also include hardware acceleration of cryptographic operations to improve performance.
- A **key management virtual appliance**. A vendor provides a pre-configured virtual appliance (instance) for you to run where you need it. This reduces costs and increases deployment flexibility, but isn't as secure as tamperproof hardware. If you decide to go this route use a vendor who takes exceptional memory protection precautions, because there are known techniques for pulling keys from memory in certain virtualization scenarios. A virtual appliance cannot offer the same physical security as a physical server, but they are hardened and support more flexible deployment options — you can run them within a cloud or virtual datacenter. Some systems also allow you to use a physical appliance as the hardware root of trust for your keys, and then distribute keys to virtual appliances to improve performance in distributed scenarios (for virtualization or simply cost savings).
- **Key management software**, which can run either on a dedicated server or within a virtual/cloud server. The difference between software and a virtual appliance is that you install the software yourself rather than receiving a hardened and configured image. Otherwise software offers the same risks and benefits as a virtual appliance, assuming you harden the server as well as the virtual appliance.
- **Key Management Software as a Service (SaaS)**. Multiple vendors now offer key management as a service specifically to support public cloud encryption. This also works for other kinds of encryption, including private clouds, but most usage is for public clouds.

Client Access Options

Whatever deployment model you choose, you need some way of getting keys where they need to be, when they need to be there, for cryptographic operations.

Clients (whatever needs the key) usually need support for the following core functions for a complete key management lifecycle:

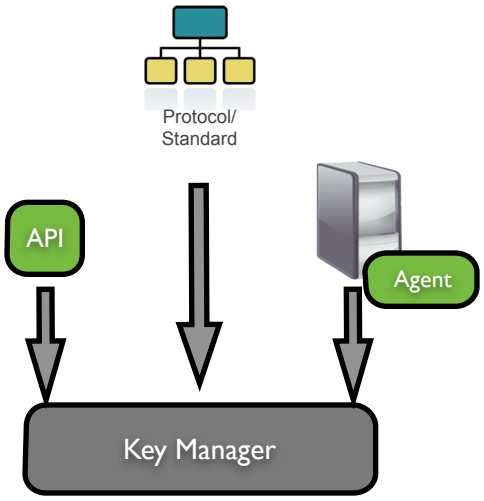
- Key generation
- Key exchange (gaining access to the key)

- Additional key lifecycle functions, such as expiring or rotating a key

Depending on what you are doing, you will allow or disallow functions under different circumstances. For example you might allow key exchange for a particular application, but not allow the application any other management functions (such as generation and rotation).

Access is managed in one of three ways, and many tools support more than one:

- **Software Agent:** A dedicated agent handles client key functions. These are generally designed for specific use cases — such as supporting native full disk encryption, specific backup software, various database platforms, and so on. Some agents may also perform cryptographic functions for additional hardening, such as wiping the key from memory after each use.
- **Application Programming Interfaces:** Many key managers are used to handle keys from custom applications. An API allows you to access key functions directly from application code. Keep in mind that APIs are not all created equal — they vary widely in platform support, programming languages supported, simplicity or complexity of API calls, and the functions accessible via the API.
- **Protocol & Standards Support:** The key manager may support a combination of proprietary and open protocols. Various encryption tools support their own protocols for key management, and like software agents, the key manager may include support — even if it is from a different vendor. Open protocols and standards are also emerging but not yet in wide use, and may be supported.



We have written a lot about key management in the past. To dig deeper take a look at Pragmatic Key Management for Data Encryption and Understanding and Selecting a Key Management Solution.

Additional Platform Features and Options

The encryption engine and the key store are the major functional pieces of an encryption platform, but any data center encryption solution will include supporting systems which are important, both for overall management and to tailor the solution to fit the application infrastructure. We frequently see the following major features and options to support customer needs:

Central Management

For enterprise-class data center encryption you need a central location, to define both what data to secure and key management policies. Management tools provide a window onto what data is encrypted, and a place to set usage policies for cryptographic keys. You can think of this as governance of the entire crypto ecosystem — including key rotation policies, integration with identity management, and IT administrator authorization. Some products even provide the ability to manage remote cryptographic engines and automatically apply encryption as data is discovered. Management interfaces have evolved to enable both security and IT management to set policy without cryptographic expertise. The larger and more complex your environment, the more critical central management becomes, so you can control your environment without making it a full-time job.

Format Preserving Encryption

Encryption protects data by scrambling it into an unreadable state. Format Preserving Encryption (FPE) also scrambles data into an unreadable state, but retains the *format* of the original data. For example if you use FPE to encrypt a 9-digit Social Security Number, the encrypted result would be 9 digits as well. All commercially available FPE tools use variants of AES encryption, which remains nearly impossible to break, so the original data cannot be recovered without the key. The principal reason to use FPE is to avoid re-coding applications and re-structuring databases to accommodate encrypted (binary) data. Both tokenization and FPE offer this advantage. But encryption obfuscates sensitive information, while tokenization removes it entirely to another location. Should you need to propagate copies of sensitive data while still controlling occasional access, FPE is a good option. Keep in mind that FPE is still encryption, so sensitive data is still present.

Tokenization

Tokenization is a method of replacing sensitive data with non-sensitive placeholders: tokens. Tokens are created to look exactly like the values they replace, retaining both format and data type. Tokens are typically 'random' or semi-random values that look like the original data but lack any intrinsic value. For example a token that looks like a credit card number cannot be used to submit credit card transactions. Its *only* value is as a reference to the original value stored in the token server that created and issued the token. Tokens are usually swapped in for sensitive data stored in relational databases and files, allowing applications to continue to function without changes while removing the risk of a data breach. Tokens may even include elements of the original value to facilitate processing. Tokens may be created from 'codebooks' or one time pads; these tokens are still irreversible but retain a mathematical relationship to the original, blurring the line between

random numbers and FPE. Tokenization has become a very popular and effective way to reduce exposure of sensitive data.

Masking

Like tokenization, masking replaces sensitive data with similar non-sensitive values. And like tokenization, masking produces data that looks and acts like the original data but doesn't pose a risk of exposure. But masking solutions go one step further, protecting sensitive data elements while maintaining the value of the aggregate data set. For example we might replace real user names in a file with names randomly selected from a phone directory, skew each person's date of birth by some number of days, or randomly shuffle salaries between employees in a database column. This enables many reports and analytics can continue to run and produce meaningful results, while the database as a whole is protected. Masking platforms commonly take a copy of production data, mask it, and then move the copy to another server. This is called static masking or "Extract, Transform, Load" (ETL for short).

A recent variation is called "dynamic masking": masks are applied in real time, as data is read from a database or file. With dynamic masking the original files and databases remain untouched; only delivered results are changed, on-the-fly. For example, depending on the requestor's credentials, a request might return the original (real, sensitive) data, or a masked copy. In the latter case data is dynamically replaced with a non-sensitive surrogate. Most dynamic masking platforms function as a 'proxy' — something like a firewall, using redaction to quickly return information without exposing sensitive data to unauthorized requesters. Select systems offer more intelligent randomization, tokenization, or even FPE.

The lines between FPE, tokenization, and masking are blurring as new variants emerge. But tokenization and masking variants offer superior value when you don't want sensitive data exposed but cannot risk application changes.

Top Encryption Use Cases

Encryption, like most security, is only adopted in response to business need. It may be a need to keep corporate data secret, protect customer privacy, ensure data integrity, or satisfy a compliance mandate, that requires data protection — but there is always a motivating factor driving companies to encrypt. The principal use cases have changed over the years, but those are all still common.

Databases

Protecting data stored in databases is a top use case across mainframes, relational, and NoSQL databases. The motivation may be to combat data breaches, keep administrators honest, support multi-tenancy, satisfy contractual obligations, or even comply with privacy laws. Perhaps surprisingly, database encryption is a relatively new phenomenon. Database administrators historically viewed encryption as carrying unacceptable performance overhead, and data security professionals viewed it as a redundant control — only effective if firewalls, identity management, and other security measures *all* failed. Only recently has the steady stream of data breaches shattered this false impression. Combined with continued performance advancements, varied deployment options, and general platform maturity, database encryption no longer carries its old stigma. Today data sprawls across hundreds of internal databases, test systems, and third-party service providers; so organizations use a mixture of encryption, tokenization, and data masking for the best mix of protection.

The two best options for encrypting a database are encrypting data fields in the application before sending to the database, and Transparent Database Encryption. Some databases support field-level encryption, but the primary driver for database encryption is usually to restrict database administrators from seeing specific data, which prevents relying on the database's own encryption capabilities.

TDE (either the database feature or an external tool) is best to protect this data in storage. It is especially useful if you need to encrypt a lot of data, and for legacy applications where adding field encryption isn't feasible.

For more information see Understanding and Selecting a Database Encryption or Tokenization Solution.

Cloud Storage

Encryption is the main data security control for cloud computing. It enables organizations to maintain control over data security, even in multitenant environments. If you encrypt data, and control the keys, even your cloud provider cannot access it.

Unfortunately cloud encryption is generally awkward for SaaS, but there are decent options for integrating encryption into PaaS, and excellent ones for IaaS. The most common use cases are encrypting storage volumes associated with applications, encrypting application data, and encrypting data in object storage. Some cloud providers are even adding options for customers to manage their own encryption keys, while the provider encrypts and decrypts the data within the platform (we call this Bring Your Own Key).

For details see our paper on Defending Cloud Data with Infrastructure Encryption.

Compliance

Compliance is a principal driver of encryption and tokenization sales. Some obligations, such as PCI, explicitly require it, while others encourage encryption by providing a "safe harbor" provision in case encrypted data is lost. Typical policies cover IT administrators accessing data, users issuing *ad hoc* queries, retrieval of "too much" information, or examination of restricted data elements such as credit card numbers. So compliance controls typically focus on privileged user entitlements (what users can access), segregation of duties (so administrators cannot read sensitive data), and the security of data as it moves between application and database instances. These policies are typically enforced by the applications which process users requests, limiting access (decryption) according to policy. Policies can be as simple as allowing only certain users to see certain types of data. More complicated policies build in fraud deterrence, limit how many records specific users are allowed to see, and shut off access entirely in response to suspicious user behavior. In other use cases, where companies move sensitive data to third-party systems they do not control, data masking and tokenization have become popular choices for ensuring sensitive data does not leave the company at all.

Payments

The payments use case deserves special mention; although commonly viewed as an offshoot of compliance, it is more a backlash — an attempt to avoid compliance requirements altogether. Before data breaches it was routine to copy payment data (account numbers and credit card numbers) anywhere they could possibly be used, but now each copy carries a burden of security and oversight, which costs money — lots of it. In most cases where payment data was available, it was not actually required, but usage patterns based around payment data became so entrenched that removal would break applications. For example merchants do not need to store — or even see — customer credit card numbers for payment, but many of their IT systems were designed to depend on credit card numbers.

In the payment use case the idea is to remove payment data wherever possible, and thus the threat of data breach, in order to reduce audit responsibility and cost. Here tokenization, Format Preserving Encryption, and masking have come into their own: removing sensitive payment data, and along with it most need for security and compliance. Industry organizations like PCI and regulatory bodies have only recently embraced these technical approaches for compliance scope reduction, and more recent variants (including Apple Pay merchant tokens) also improve user data privacy.

Applications

Every company depends on applications to one degree or another, and these applications process data critical to the business. Most applications, whether 'web' or 'enterprise', leverage encryption. Encryption capabilities may be embedded in the application or bundled with the underlying file system, storage array, or relational database system.

Application encryption is selected when fine-grained control is needed, to encrypt select data elements, and to only decrypt information as appropriate for the application — not merely because recognized credentials were provided. This granularity of control comes at a price — it is more difficult to implement, and changes in usage policies may require application code changes, followed by extensive validation and testing.

The operational costs can be steep but this level of security is essential for some applications — particularly financial applications. For other types of applications, simply protecting data "at rest" (typically in files or databases) with transparent encryption at the file or database layer, is generally sufficient.

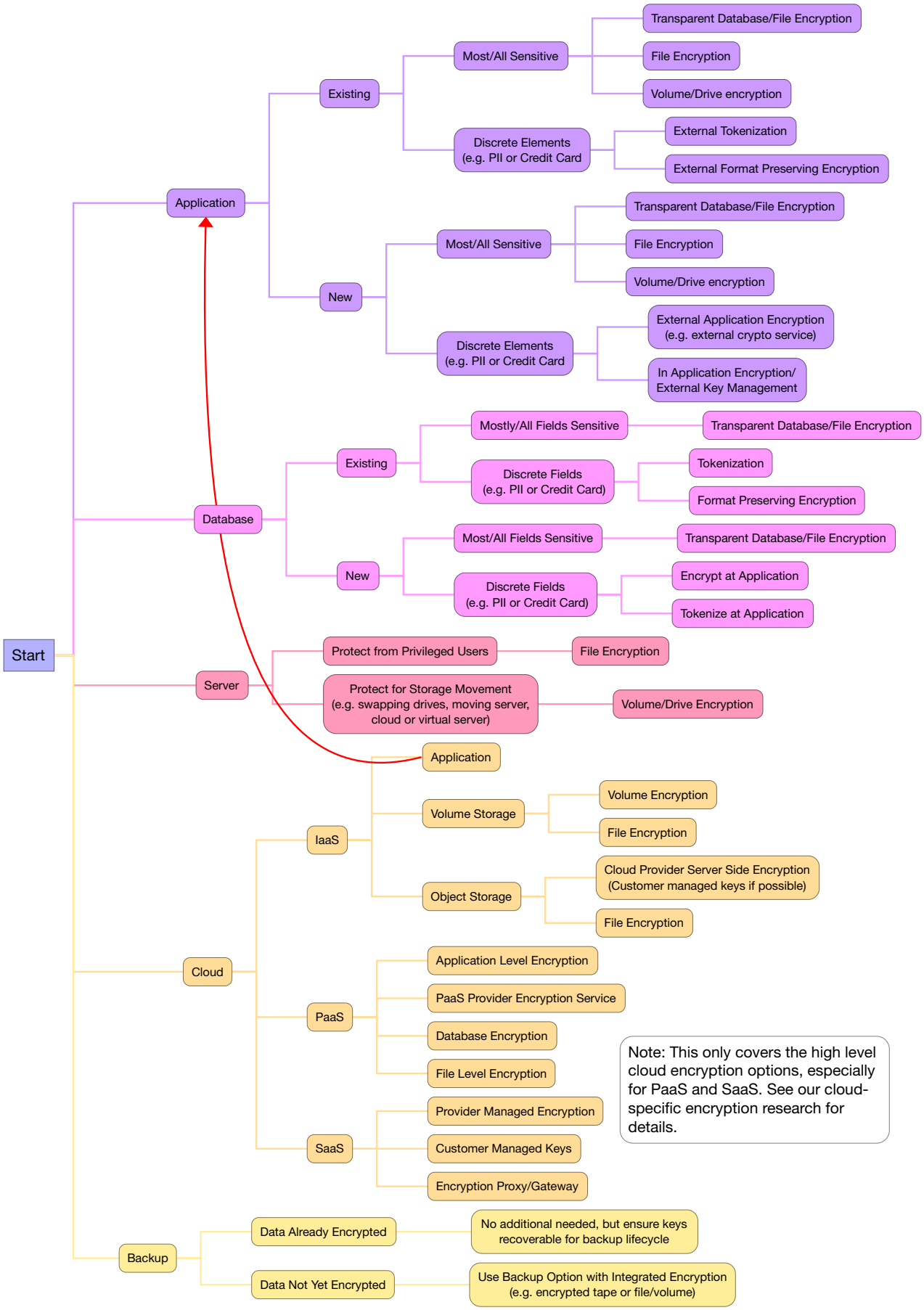
Choosing the Best Option

There is no way to cover all the myriad factors in picking a specific encryption option in a (relatively) short paper like this, so we compiled a visual decision tree to at least get you into the right bucket.

Here are a few notes on the decision tree.

- This isn't exhaustive but it should get you looking at the right set of technologies.
- You will always want secure external key management.
- In general, for discreet data you should encrypt as high up in the stack as possible. When you don't need as much separation of duties, encrypting lower may be easier and more cost effective.
- For both database and cloud encryption, in a few cases we recommend you encrypt in the application instead.
- When we list multiple options, the order of preference is top to bottom.
- As you use this tree keep the *Three Laws* in mind; they will help you assess the security of your choices.

Once you understand how encryption systems work, the different layers where you can encrypt, and how they combine to improve security (or not), it is usually relatively easy to pick the right approach.


Start
Application
Existing
Most/All Sensitive
Transparent Database/File Encryption
File Encryption
Volume/Drive encryption
Discrete Elements (e.g. PII or Credit Card
External Tokenization
External Format Preserving Encryption
New
Most/All Sensitive
Transparent Database/File Encryption
File Encryption
Volume/Drive encryption
Discrete Elements (e.g. PII or Credit Card
External Application Encryption (e.g. external crypto service)
In Application Encryption/ External Key Management
Database
Existing
Mostly/All Fields Sensitive
Transparent Database/File Encryption
Discrete Fields (e.g. PII or Credit Card)
Tokenization
Format Preserving Encryption
New
Most/All Fields Sensitive
Transparent Database/File Encryption
Discrete Fields (e.g. PII or Credit Card)
Encrypt at Application
Tokenize at Application
Server
Protect from Privileged Users
File Encryption
Protect for Storage Movement (e.g. swapping drives, moving server, cloud or virtual server)
Volume/Drive Encryption
Cloud
IaaS
Application
Volume Storage
Volume Encryption
File Encryption
Object Storage
Cloud Provider Server Side Encryption (Customer managed keys if possible)
File Encryption
PaaS
Application Level Encryption
PaaS Provider Encryption Service
Database Encryption
File Level Encryption
SaaS
Provider Managed Encryption
Customer Managed Keys
Encryption Proxy/Gateway
Note: This only covers the high level cloud encryption options, especially for PaaS and SaaS. See our cloud-specific encryption research for details.
Backup
Data Already Encrypted
No additional needed, but ensure keys recoverable for backup lifecycle
Data Not Yet Encrypted
Use Backup Option with Integrated Encryption (e.g. encrypted tape or file/volume)

Conclusions

We have never had more options for encrypting data across applications, databases, servers, storage, and pretty much the entire datacenter (and the cloud). Although choosing the right approach *can* be confusing, once you understand how encryption systems are built, map that to your security objectives, and then look at the architectural options… the answer is usually straightforward.

Keep in mind that you don't make these decisions in a vacuum. You likely already have a variety of encryption projects in production, and even some encryption appliances or key management tools. Ideally you will leverage existing investments as much as possible, to reduce both capital and operational overhead. And if you are just getting started, try to pick a platform that is flexible to support future needs.

The hard part of these projects is to architect and implement the encryption technology; and integrate it into your data center, application, or cloud service. That is where our other encryption research can be useful, and the following reports should help:

- Understanding and Selecting a Key Management Solution
- Pragmatic Key Management for Data Encryption
- Understanding and Selecting a Database Encryption or Tokenization Solution
- Defending Cloud Data with Infrastructure Encryption
- Understanding and Selecting a Tokenization Solution
- Understanding and Selecting Data Masking Solutions

Who We Are

About the Authors

Rich Mogull, Analyst and CEO

Rich has twenty years experience in information security, physical security, and risk management. He specializes in cloud security, data security, application security, emerging security technologies, and security management. He is also the principle course designer of the Cloud Security Alliance training class and actively works on developing hands-on cloud security techniques. Prior to founding Securosis, Rich was a Research Vice President at Gartner on the security team. 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.

Adrian Lane, Analyst/CTO

Adrian Lane is a Senior Security Strategist with 25 years of industry experience. He brings over a decade of C-level executive expertise to the Securosis team. Mr. Lane specializes in database architecture and data security. With extensive experience as a member of the vendor community (including positions at Ingres and Oracle), in addition to time as an IT customer in the CIO role, Adrian brings a business-oriented perspective to security implementations. Prior to joining Securosis, Adrian was CTO at database security firm IPLocks, Vice President of Engineering at Touchpoint, and CTO of the secure payment and digital rights management firm Transactor/Brodia. Adrian also blogs for Dark Reading and is a regular contributor to Information Security Magazine. Mr. Lane is a Computer Science graduate of the University of California at Berkeley with post-graduate work in operating systems at Stanford University.

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.