What is a data privacy vault? Why do you need it?

skyflow

Contents

Abstract

Authored by Manish Ahluwalia, Field CTO, Skyflow

Increased security and privacy concerns are forcing financial technology companies to consider how they deal with what is often their most valuable asset -- their data. This requires a difficult balancing act where security and regulatory needs conflict with business goals: security requires locking down data while fintech companies want to use that same data to derive insights in an agile manner.

In this white paper, we will examine the types of data protection concerns that apply to your data. While you have a lot of data, only a small amount of it is PII -- personally identifiable information -- that can be used to identify an individual. This part of your data should be isolated (logically or physically) from the rest of your data and managed differently.

During this discussion, we will frequently use an analogy with a secrets store / secrets vault like Hashicorp Vault, or AWS Secrets Manager. You have a lot of configuration data, but only a small portion of it is sensitive and is treated specially -- by storing just that portion in a vault. Similarly, you have a lot of data, but only some of it is sensitive -- has data protection obligations.

We will examine the data protection problem and explore the technical requirements for a solution. Finally, we will iteratively build an architectural pattern -- the data privacy vault -- that aims to maximize data usage while reducing the security and compliance risk from doing so.

Mo' data, mo' problems

You have lots of sensitive financial data stored in various data stores -- operational and analytic, backup copies, temporary tables, and so on. You have a variety of systems and third party integrations that need this data in order to function. Very likely, your data has a mix of various types of data elements:

1. **Elements that are highly sensitive (i.e. have security / privacy / compliance concerns)**
2. **Elements that have no confidentiality concerns (i.e. data that is publicly available, like comments posted to a public forum, or aggregated data)**
3. **The rest, which lay in between those two extremes**

Which data elements fall in which category depends on a number of factors.

Some elements, like a person's name, are obviously sensitive -- names are classified like PII under GDPR. A breach of this data would trigger notification obligations and penalties in some jurisdictions. Other elements vary -- IP addresses, for example, are considered sensitive in some jurisdictions, not so in others.

Your systems need access to sensitive data in order to meet some use cases. For instance, you need email addresses for CRM or social security numbers (SSNs) to perform credit checks. The critical thing to note here is that while a lot of systems may be accessing the data stores and records that store this sensitive information, many, if not most, will not really need access to the underlying data. They will either be simply passing this information along or will be ignoring the sensitive fields. Other elements are not sensitive in and of themselves but would be sensitive if combined with identifying data. For instance, a credit score of 678 doesn't mean anything much by itself, but if you know this is Alice's credit score, now you have information about Alice that she might not be happy to have you lose in a breach. Your system will most certainly have a ton of such data elements that are crucial to performing your core business functions. It's important to note that if this data were not associated with identifiable information, this may not remain sensitive information. So if you are breached and you lose the information 'Alice has a credit score of 678', you have breach notification obligations, but if you lose the information 'User 35 has score 678', without simultaneously leaking the identity of user 35 then you do not have breach notification obligations.[1]

Determining your data protection obligations is a complex issue that we urge you to learn as much as possible about. You will need to consider applicable legal and regulatory constraints, contractual obligations, brand reputation impacts, insurance requirements, and reasonable cost-benefit tradeoffs. As a fintech company, you are likely storing data which is subject to very stringent financial regulations, particularly payment card information, adding another layer of complexity.

For now, we will assume that you have figured out what data elements need protection and what kinds of access you can or must allow to enable your business to function.

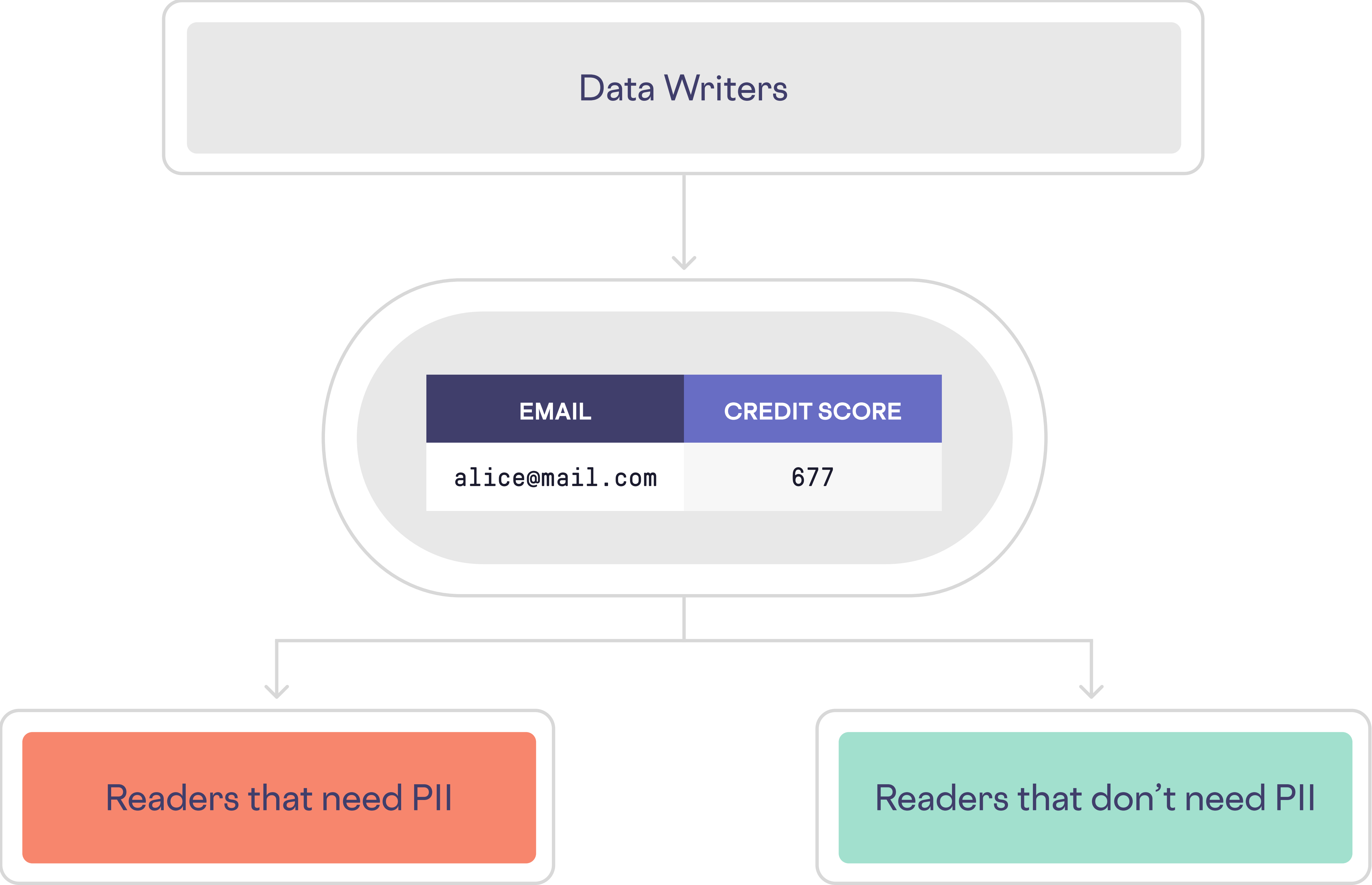
[1] If you have sufficient non-identifying personal information, you can often 'triangulate' and identify the individual with a combination of non-identifying information. This makes categorizing what is and isn't sensitive information a difficult judgement call. Breach notification obligations on the other hand are very specific as to when they are triggered.

So how do we start to deal with our data protection problem?

Let’s look at an analogy with configuration information. Most likely, you treat the hostname of your backend server (not sensitive information, maybe even public info) differently from the login credentials to the server. You store the credentials in a secrets vault and carefully manage access to them.

By analogy, there are limited use cases that actually need Alice’s name, social security number, or email address. The majority of use cases may touch User 35’s data but don’t need to know that User 35 is Alice or what her SSN is. Therefore if you ‘protect’ Alice’s personal data, replace references to Alice with random(ish) pointers and allow limited access to the real data, you have greatly reduced the size of the problem of securing your company and bringing it into compliance.

Let’s say we have the following highly abstracted and data-centric picture of your architectural components:



EMAIL	CREDIT SCORE
alice@mail.com	677

In this case, your database clearly contains sensitive information and is a security concern. But so are the ‘Readers that don’t need PII’ because they needlessly have access to sensitive information.

Continued

The goal is to minimize exposure without sacrificing functionality at a reasonable cost. There are multiple ways to address this, but it helps to figure out what an ideal solution's characteristics would be. Let's make a list of requirements for the solution:

1. **Functional Equivalence:** Your system should not lose access to the data it needs to do its job.

2. **Security and Privacy Support:** Your system should:
    a. Encrypt the data at rest and during transmission
    b. Provide authn/authz capabilities so exactly the intended users of the sensitive data can access it
    c. Provide auditing for access to sensitive data and its administration
    d. Provide data life-cycle management capabilities

3. **Surface Minimization:** Your system should reduce the need for data to be extracted in the clear out of the store, thereby reducing the surface area that needs to be hardened. This is done in the following ways:
    a. Tokenization: Systems and processes that need to work with data without needing access to its contents can do so via holding only tokens that represent the data. Only authorized processes can detokenize the tokens to retrieve the sensitive data.
    b. Policy-constrained partial detokenization: Users of the data who only need partial access to the data data should get access only to that part of the data
    c. Execute workflow on tokens: Systems that need to execute a workflow on the sensitive data underlying a token should be able to execute the workflow on the token without needing to detokenize and retrieve the sensitive data first.

4. **Storage and Processing Overhead:** The solution should impose a minimal reasonable performance overhead.

5. **Cost effectiveness:** The solution should be easy to integrate and operate.

Continued

These requirements are important for the reasons outlined. However, you get your real power from combinations of these:

- Functional Equivalence + Security means that Security and Functionality are not a zero-sum game. You should look for solutions that let you have both. For instance, your solution may ask you to encrypt your customer's zip codes, but that means you now can't group by zip code, then you have a zero-sum situation.

  You could solve the above dilemma by decrypting your phone numbers before grouping or by providing your analytics code decryption keys for the phone numbers. While this does solve the problem, it goes against the 'Surface Minimization' requirement.

- The 'Policy-constrained partial detokenization' is a requirement that helps reduce your privacy risk without sacrificing functionality. Say you need to compute the average income -- this process does not really need to know the individual incomes, only be able to count and sum them. Another example would be when your customer service agents need to access the last four digits of SSN and that's all they should be able to access, not the full SSN that is masked at the agent's client.

What are the alternatives?

Let's take a look at various alternatives and compare them against the requirements outlined earlier.

Get out of the business of acquiring and storing user data

No, seriously! In many cases, you can free yourself of the concerns and obligations associated with storing a data element by restructuring your business or by forming partnerships so you don't actually store it. For instance, if you were helping users find a mortgage, instead of asking for detailed financial information to provide users with tailored recommendations you could just list a bunch of general mortgage leads. The first approach requires collecting sensitive data, the second doesn't. Typically, more data might be more problems, but you want that data so you can provide more value -- Notorious B.I.G. will happily sing all the way to the bank about how Mo' money causes Mo' problems.

As another example, you can take credit card information to process payment, opening your site up to PCI obligations, or you could use a solution like Paypal or Stripe to process payments, freeing you of those concerns while still allowing you the same capabilities.

While this solution eliminates security concerns and storage requirements and has a low cost of ownership, it severely hobbles your functionality. At the end of the day, you will almost certainly find that to be effective, you have to work with data, including sensitive data.

So let's get back to the problem of securing the sensitive data you're left with.

Harden the systems that read, write, store, or transmit data

In theory, we could harden all the systems that touch any sensitive data and make sure they work in a compliant and secure manner. In practice, this approach is not reasonable since there is a lot of continually changing surface area to harden, most of which doesn't need to be burdened with the additional cost of dealing with sensitive data. For example, it makes sense to harden the CRM system, but if you want to bucket users by credit score, do you really want to harden the entire analytics pipeline to the same standards as the CRM system?

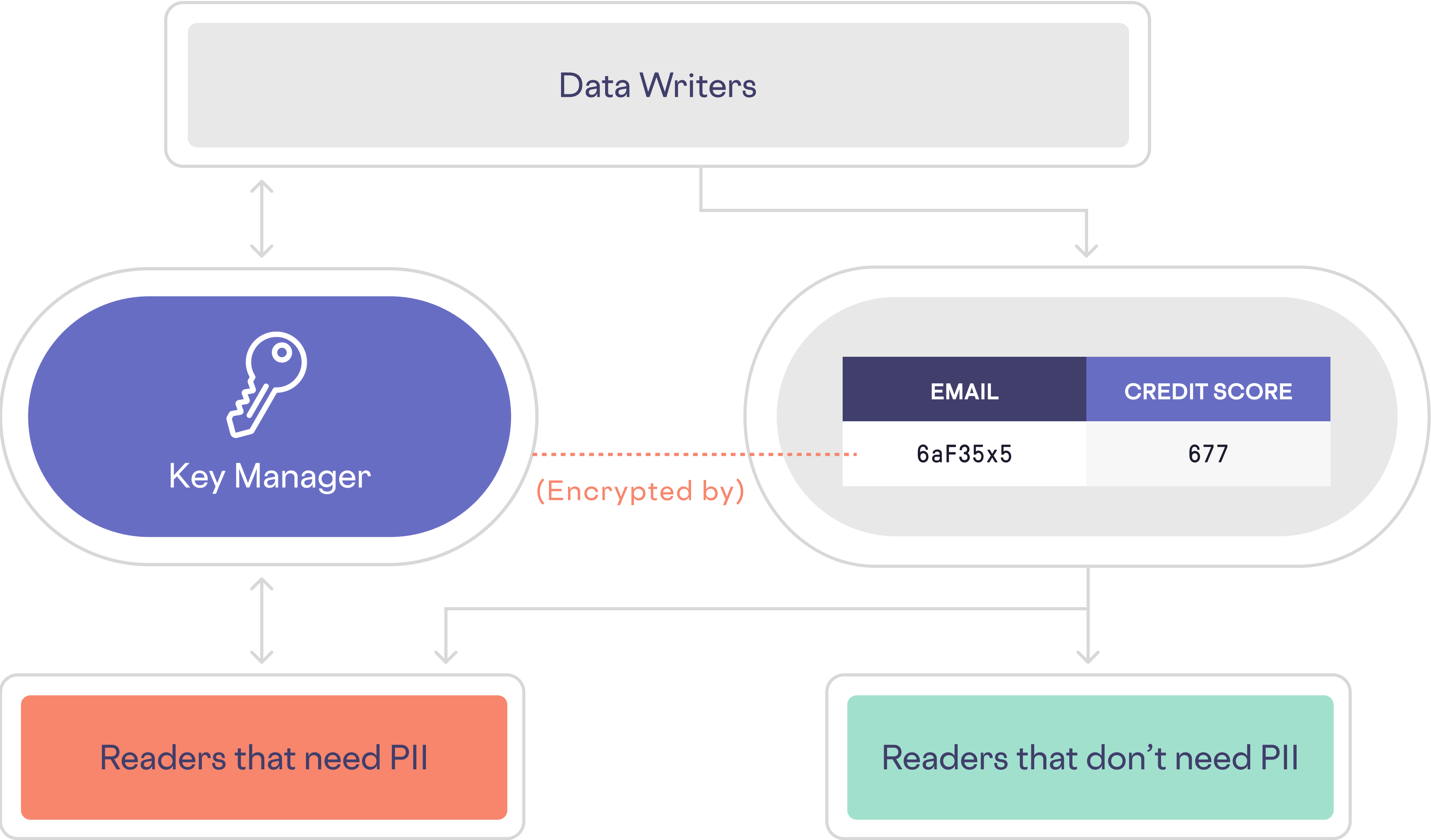
This approach would leave your functionality and storage capability unchanged but it would not meet any data storage security requirements. Moreover, by requiring you to harden all systems that touch data, this approach imposes an unreasonable implementation and vigilance cost. Hardening a large number of large, possibly legacy systems that were not built with security and privacy in mind requires a lot of time and expertise. Keeping all of them hardened as your engineering organization grows is even harder and will slow development. It is easier to (logically or physically) separate the data of concern so you reduce the amount of surface area that must be hardened and systems that must be made privacy compliant.

Continued

Encrypt the sensitive data

We could encrypt the sensitive data at rest and as it moves through the system. We would give a privileged set of readers the keys. Thus only the systems that really need access to the data would have the access while the others would see only ciphertext. In this case, a breach to the database or a leak in the analytics pipeline would not have regulatory impact -- although of course, it could still negatively impact your company's public image and credibility

It would look like this:



EMAIL	CREDIT SCORE
6aF35x5	677

In this case, sensitive data is encrypted before being written. Access to read the ciphertext is not restricted. Systems that need the sensitive data will be given access to the keys so they can decrypt the ciphertext and access the sensitive data. Other systems would only see the non-sensitive data in the clear. Functionality is therefore not disrupted.

The storage (size and performance) overhead of encryption is non-zero but negligible. A good key management and encryption system would do an adequate job of reliable, performant encryption. It would provide good audit logs as to when keys were accessed, maybe even who decrypted certain elements.

This solution would also provide satisfactory data lifecycle management capabilities. For instance, if a user were to ask for their personal information to be deleted from your systems (a required capability in some jurisdictions), you can delete it from the primary system easily -- just delete Alice's record or her encrypted data. The problem here is that data often ends up making copies of itself in various other databases, data warehouses, logs, and other places. So you do not only need to delete Alice's encrypted data from your initial database, but also other places that may have stored *encrypted* versions of the data.

Continued

Encrypt each user’s data with a key tied to that user

The last problem can be solved by modifying the above solution by encrypting each user’s data with a per-user key. When the user wants their data deleted, we just delete the per-user key and all of this user’s encrypted data is effectively deleted. This meets all of our requirements, except for ‘policy-constrained partial detokenization’ and ‘execute workflow on tokens’. Let’s look at these in more detail.

Suppose you want to send the user an email nudge. You have followed all the steps so far and your user-nudge service (or microservice) will pull the user’s encrypted information from the database, decrypt it with the user’s key and then use this information to send the email using a 3rd party email provider. This exposes the user’s sensitive data to only the user-nudge service . This seems reasonable, but is it the best possible solution?

Let’s take another look at our analogy with secrets management. If you have a secret key with which you need to perform encryption and decryption, you would store the secret in your secret vault and extract it only when you need to perform the encryption. Or, even better, you could have your secret storage mechanism do the encryption for you without ever having to extract the secret key from storage.

Ideally, we want the same kind of capability for our data. Our data protection solution should be able to execute the nudge-user workflow by directly communicating with the email provider. Only the email provider needs to know the user’s sensitive data, while our user-nudge-service only needs to initiate workflows when needed, it doesn’t need the sensitive data. The ideal solution would minimize the exposure of the sensitive data (or the underlying encryption keys!)
to systems that don’t need access to them.

Or take yet another use case. You want your customer service agents’ portal to be able to access users’ SSN for verification purposes but only see the last 4 digits of it. In our current solution, your portal would have to extract the full SSN and then perform the redaction. In an ideal world you would authorize your portal to only read the redacted data. This is not possible with an off-the-shelf encryption system.

Encryption is therefore a good solution but doesn’t meet all of our needs. To solve this last gap, let’s turn to our final alternative.

Continued

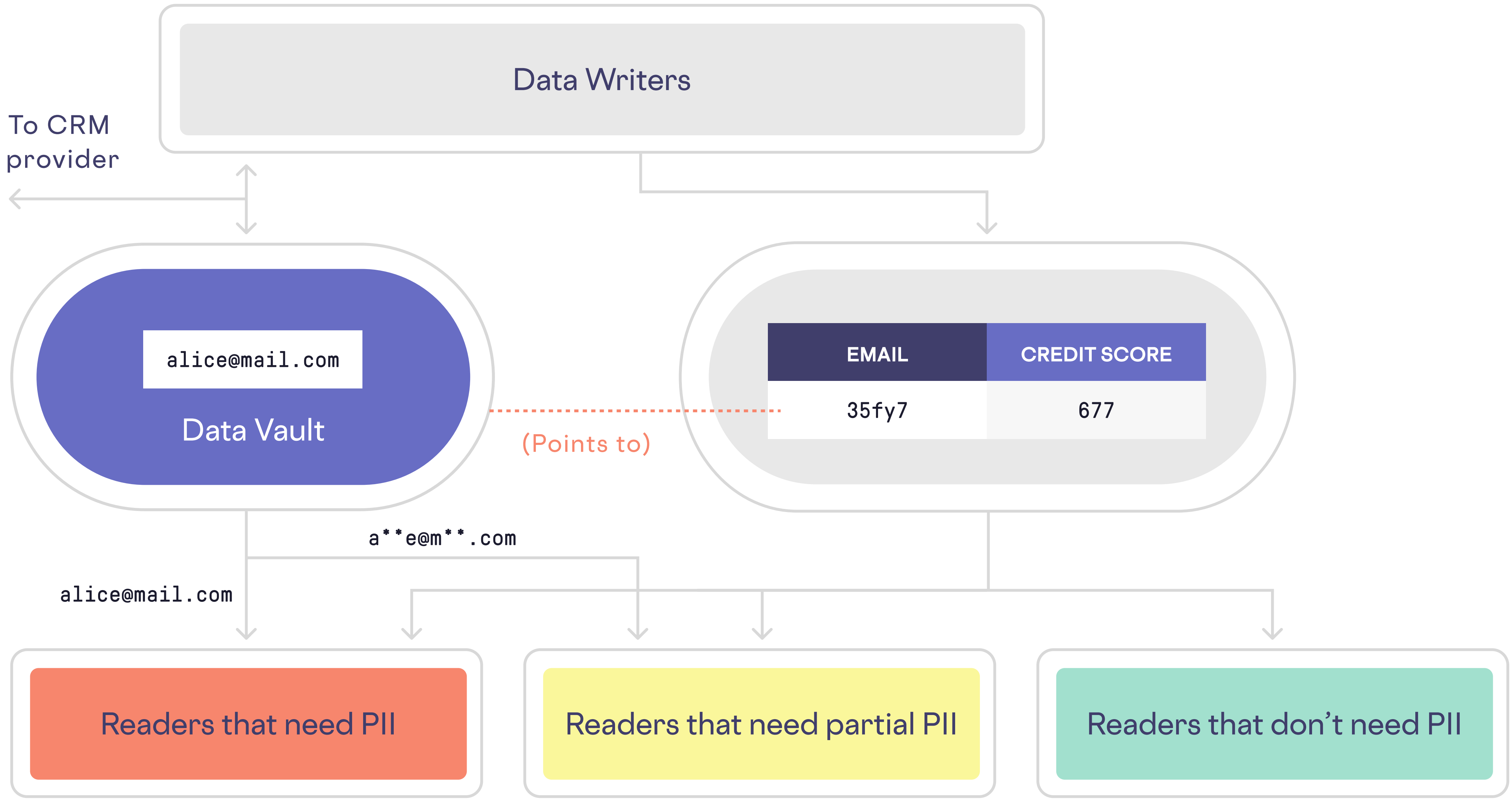
Use a data vault[2]

In this case, we replace the key manager or crypto engine with a full-fledged data vault. A data vault is something that gives you all the goodies that the encryption solution above did (indeed, any reasonable implementation of a data vault will make heavy use of encryption), but it is built to solve the problems encryption alone can't.

You store the sensitive data in the data vault and replace the sensitive data with pointers returned to you by the vault. Only authorized readers are permitted to access data from the vault (i.e. dereference the pointer to retrieve the underlying data). Others can access only the ciphertext, or a partially or fully redacted version of the data.

More importantly, the vault allows you to execute workflows on the data without you having to extract data from the vault. Going back to our user-nudge-service example, if we wanted to send an email to a user, instead of extracting the email address from the vault in order to send the email, we could initiate the 'send-email' workflow directly from the vault, which would initiate the operation with our CRM provider without our system ever having direct access to the email address.

Our architecture now looks like this:



EMAIL	CREDIT SCORE
35fy7	677

Similarly, the customer service agents' portal would be given access to only the redacted SSN (or email) by the vault.

This solves the problems we couldn't solve with straight-up encryption.

[2] There is a data warehousing architectural pattern of the same name. What follows is a distinct concept.

Continued

In order to meet all the requirements we had laid out above, you should also demand the following capabilities from a data vault you build or buy:

1. **The data vault should appear to your systems as an external database -- it should provide similar interfaces, performance characteristics, and management capabilities.**
2. **It should provide the following security capabilities:**
    a. **maximum encryption without sacrificing functionality**
    b. **authn / authz of data accessors**
    c. **auditing**
3. **It should provide secure deletion capabilities -- when you delete the data from the vault, all copies of the pointer to the data in the vault become junk.**

How does one design a vault to meet all of these requirements? That's a topic we will explore in subsequent white papers.

Side-by-side comparison

Criterion	Don’t have the data	Harden Systems	Encrypt Sensitive Data	Use a Data Vault
Fuctional Equivalence	🟠	🟢	🟢	🟢
Security & Privacy				
Encryption	n/a	🟠	🟢	🟢
Authn/Authz	n/a	🟠	🟢	🟢
Auditing	n/a	🟠	🟢	🟢
Data-Life Cycle	n/a	🟠	🟢	🟢
Surface Minimization				
Tokenization	n/a	🟠	🟢	🟢
Policy constrained partial detokenization	n/a	🟠	🟠	🟢
Tokenized workflows	n/a	🟠	🟠	🟢
Storage / Processing cost	n/a	🟢	○	○
Total cost of Ownership	🟢	🟠	🟢	🟢

Does anyone use a data vault?

You may be wondering here if this architectural pattern of a data vault is used in real life. If it is, how come you don't see this used more commonly?

Many technologies and large companies use this architectural pattern. When you make a payment through Google Pay, Apple Pay, or Samsung Pay, you are relying on a vault to secure and tokenize your credit card while still keeping it usable seamlessly.

For uses more general than credit card numbers, Netflix[3] has built several data vaults, as have Goldman Sachs[4], and Adyen[5] to name just a few.

As you may guess, building and operating a data vault takes substantial resources. In future white papers, we'll show you some of the cool stuff Skyflow has done to build a data vault as a service. If you don't have the resources that these engineering giants have to build your own data vault, reach out to us.

> "Skyflow gives us a secure data vault, with customizable access controls and PCI compliance built in. It frees my team to focus on shipping features."
>
> - Ed Cortis, Chief Technology Officer at Unifimoney.

[3] "Skyflow has taken the best ideas of data vaults built at companies like Netflix and delivered them as a service with a simple Stripe-like API. They can solve use cases that most vault efforts struggle with!" - Jitender Aswani, Head of Data and Engineering at Moveworks, Former Head of Security/Privacy Engineering at Netflix

[4] "At Barclay's Bank and at Goldman Sachs - we relied on the data vault architecture to meet our most critical security needs for sensitive data." - Boe Hartman CTO NomiHealth, Former CTO of Goldman Sachs

[5] From where I copied the Mo Data, Mo Problems section title

About Skyflow

Founded in 2019, Skyflow is a data privacy vault for sensitive data. The company was founded by former Salesforce executives Anshu Sharma and Prakash Khot to radically transform how businesses handle users' financial, healthcare, and other personal data that powers the digital economy. Skyflow is based in Palo Alto, California, with offices in Bangalore, India. For more information, visit skyflow.com or follow on Twitter and LinkedIn.

About the Author

Manish Ahluwalia has over 2 decades of experience in the software industry, with over 10 years in information-security. Most recently he was running security for NerdWallet. He currently works to help Skyflow's customer's find the right architecture for their data protection needs.

skyflow