Sriram Kunjathur

BLOG · TECH

# Securing your personal data via tokenization

Securing your personal data is more important than ever. Sriram Kunjathur, Developer, Merchant Data Infrastructure at Adyen tells you everything about data tokenization and how it can help you secure your personal data and solve data security issues.

October 1, 2019 · 9 Minutes

At Adyen we take privacy and data security very seriously, and it is at the heart of everything that we build. As a global payment platform, we handle a lot of Personally Identifiable Information or PII, which is any information that identifies a person. We use this information to drive many of our powerful tools such as anti-money laundering, identity verification (commonly referred to as KYC), fraud detection and more.

Being part of the greater payment industry, which moves around trillions of dollars every year, we are built on layers of trust. Our customers trust Adyen and we need to make sure that we are always in control of data security.

We are headquartered in the EU and with many of our customers doing business in the EU (with EU-based users); GDPR compliance is crucial. We also hold a European Banking license, which brings with it several degrees of compliance and regulatory requirements over and above other types of business that handles Personal Data.

Being a payment service provider means that we are a visible and valuable target to cyber criminals. We work continuously to make sure that we're handling our data with care. We are constantly evaluating ourselves, minimizing risks at every step of the way with the ultimate goal of protecting our customers and their user's data.

In this blog post, we'll look at the challenges of securely managing data, and how we built tools to scale secure data management.

powerful, but with great power comes great responsibility. That responsibility translates to two major problems — scalably handling all of the data (you can't have different teams managing data differently), and handling it securely.

Addressing this responsibility starts with having a threat model. A threat model is a representation of an application (Adyen platform) and all potential security considerations and implications (data, architecture, malicious actors, etc). Adyen manages an extensive threat model.

Our threat model includes entities trying to either disrupt Adyen's ability to provide service, or entities attempting to extract data from Adyen, such as by compromising a database. A compromised database could mean real impact to real people, as well as serious consequences to our business.

We include how compromised data could be used into our threat model:

- Extracting credit card information to commit fraud.
- Stealing a victim's identity.
- Sophisticated future cyber attacks based on compromised data.

Keeping the data safe is not enough — it has to scale. What works for a handful of transactions should work for a billion transactions without missing a beat. The system should be able to generate reports, detect fraud, handle payments, provide real-time monitoring, and keep powering all the other products in our system without any down-time.

We also have to have a system that can handle heavy reading and writing. A system tuned to handle a write-heavy operation, such as making a payment, might not work for a read-heavy operation such as report generation.

Lastly, we have a "scale 20x" standard when designing and building a system. If a potential solution can't scale 20x, we won't pursue it.

### A new human right for the 21st century?

Along with the Right to privacy, jurisprudence and legislation in Europe and across the world is starting to identify a human right - the "right to be forgotten" - in law to allow "any person living a life of rectitude ... that right to happiness which includes a freedom from unnecessary attacks on his character, social standing or reputation." to quote some relevant case law. As part of our obligation to comply with the GDPR, we need to be able to technically support mechanisms that respect an individual's right to be forgotten.

With the challenges of security, performance, scalability and compliance requirements established, we need to address how we store PII.

## Encrypt all the things! Or not...

Everyone knows that storing passwords in plain text is a really bad idea. But what about PII? The damage that can be caused by leaked PII is just as bad as a stolen password, and potentially much worse — think identity theft.

Best practice password storage dictates using one-way cryptographic hashes with per password unique salt. This makes "stored" passwords difficult to compromise even with large a corpus of potential passwords. But this one-way approach typically does not work for PII data which needs to be passed back in the clear for display, reporting or matching processes.

business goals or generate reports? Reports need to contain representative data which can be used to visualize trends without compromising PII. Providing a report with a bunch of encrypted data is a bad idea for a couple of different reasons:

1. How do you make sure that there isn't someone out there studying these cipher-texts and actively trying to reverse your encrypted data? Your data might be safe today, but in five years all the accumulated cipher-text could be a treasure trove for analysis.
2. How do you effectively "shorten" some data elements to put in your report? An email address that is 10 characters long will have a cipher-text shorter than an email with 20 characters. You can't just chop up cipher-text, because someone is going to expect that the cipher-text can be decrypted by some downstream process, which will fail the day you assume that you can just chop up the cipher-text.

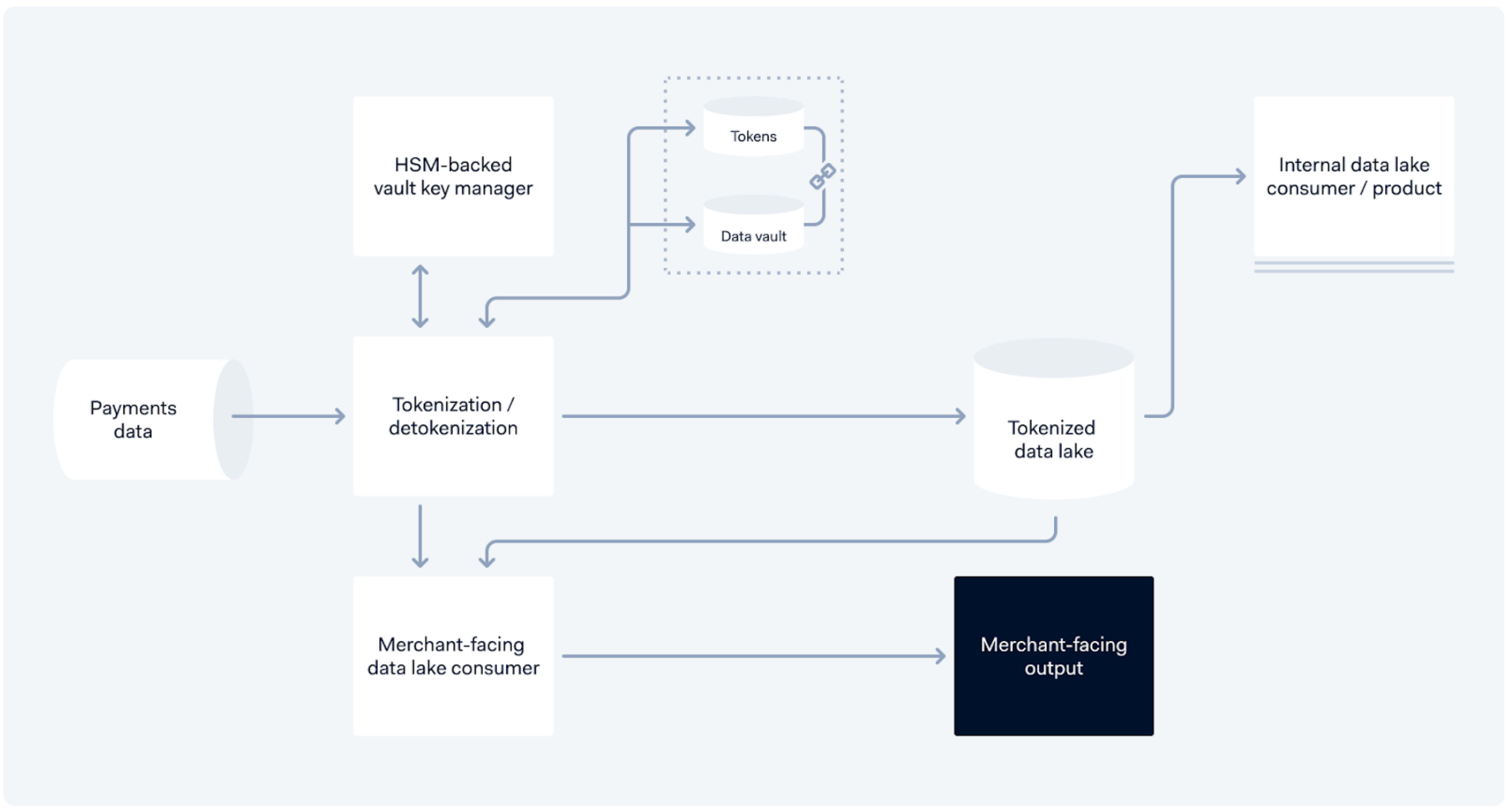
There are also issues with getting additional features you might need, like:

1. Knowing which key was used to encrypt the data.
2. Being able to identify when the data was encrypted.
3. Redacting the data at a later date.
4. Providing an easy way to show different views on the data to different clients.

In order to overcome these problems and more, we use a different strategy called tokenization. Instead of returning cipher-texts to the customer or our downstream systems, we assign identifiers to the data and return that instead. These identifiers point to metadata that allows us to track various properties about the token itself as well as the data behind the token. Choosing a good identifier format and generation algorithm ensures that we can also scale in an environment which has multiple active tokenization services that operate concurrently in a high-availability striping or load-balancing structure.

## Secure all the things with data tokenization!

Token generation at Adyen is implemented in a fairly straight-forward way:


HSM-backed vault key manager
Tokens
Data vault
Internal data lake consumer / product
Payments data
Tokenization / detokenization
Tokenized data lake
Merchant-facing data lake consumer
Merchant-facing output

**Tokenization service data flow**

PII data (this could be a name, email address, credit card details, etc) coming in from the left

The token then replaces the PII elements in the data, and stored in the Tokenized Data Lake, to be used by downstream processes.

When a Merchant-Facing consumer, such as a report builder, requires this data, it is read from the Tokenized Data Lake, and the generated report is detokenized using the Detokenization service, and the final report with actual data is generated.

Our internal services also require data, but they can now use the Tokenized data instead of the raw data, thus keeping the PII data safe.

This approach has several advantages, described below:

## Token values are context-free

The UUIDs do not represent, in themselves, any information about what the underlying data is, which means that these tokens can be freely shared in our platform without risk of attacks such as enumeration, rainbow tables, collision or reversing, thereby creating an opaque reference to the data.

## Tokens are contextual

"*Wait, you said they were context-free?*"

While a token's value is context-free, a token itself has a context which identifies for what purpose it was generated. Tokens generated for Merchant A will be different from tokens generated for Merchant B for the same data. This ensures that any compromise at Merchant A does not impact the tokens and data held for customers of Merchant A at other merchants.

## Fast generation and guaranteed uniqueness

Our payment systems that handle the firehose of data are distributed, thus we cannot have a single point of token generation, which cannot scale up for the continuously increasing volume of tokens we need to generate. If one tokenization instance becomes unavailable, we need to still keep the platform running — including tokenization — while we fix it. The only way to be fault tolerant while scaling up is to be able to keep the system consistent while being distributed, and using UUIDs is the first step towards that. UUIDs have a one-in-17 billion chance of collision, and are generated using hardware entropy.

## Compliant with regulations

Merchants can request Adyen to remove data held for a customer by providing relevant context information for the customer. As the process of reversing a token is controlled by the system that generates it, we can easily erase the tokens for the context on our platform, removing the possibility of the information being retrieved successfully.

## Let it go, or retention periods

PII should not be held longer than is necessary, but deleting data is sometimes hard — even finding all copies can be tough in large businesses. Tokenization services help, because the data is typically centralized in a vault. And since the data is stored in the vault under a rotating protection key, reversing is only possible for as long as the protection key is still active. Removing an older protection key by maintaining a retention policy on the key effectively makes the encryption irreversible.

text into lexemes which are then indexed into searchable parts (we know that this is overly simplified but you should get the idea). Exact match search is easily achievable with the tokenization solution: we can directly match tokens.

## Everything comes at a cost

### Tokens are poor for wildcard search

Fuzzy search and wildcard search is not trivial with a tokenization-based system. Since there is no link between the data and its token, searching for *Pasport* or *Passpor** can't find *Passport*, which raises questions around how to index and search data fuzzily or provide auto-complete and query-correction reliably.

### Consistency is hard

Adyen handles payments around the world — and our customers are also global. A person queuing to buy a coffee in San Francisco could also be buying a new dress from a fancy French online boutique at the same time. Our system needs to have a consistent view of this data internally to ensure that fraud risks, customer recognition and all the other things that people expect today can be managed effectively at speed across geographic scale. Generation of UUIDs needs to somehow be consistent globally, while also being random.

### Choosing a Database for the future isn't easy

We design our systems keeping all parts of our platform in mind — development, security, infrastructure and the business. Tokens based on UUIDs might solve the issue of distributed generation, but the underlying storage needs to scale elegantly. Our choice of database must make sense across several metrics for us:

1. Is the database easy to understand and develop with? Our code needs to be understandable at 3AM under stress, and a database that makes that process easy goes a long way.
2. Can it handle a high throughput? At the moment, we're looking at an average of 8000 incoming unique tokenization requests a second. As mentioned before we need to scale to 20X, which means nearly **160K requests per second**.
3. Will our Infrastructure teams be stressed? The database system should easily rebalance load as we notice hotspots in our data, and scale up by just throwing more hardware as we notice our data requirements grow.
4. Will the system be fast and stay fast? Latency per request should not go up as we get more data — our payment systems are engineered to be fast, and our choice of database should let us stay fast when we are handling 20 times as much data.

## Where do we go from here?

We are currently working on solving a set of interesting and hard problems, including:

1. Providing data to data scientists in a way that allows them to build and train models without compromising on our data security commitments.
2. Supporting wildcard search is a very challenging problem without sufficient context.
3. Scaling up to 20X without increasing the burden on our DBAs. Adyen likes to scale up everything, including the efficiency of our staff, so scaling a system 20X should definitely

If solving problems like these sounds interesting to you, reach out to us.

### Technical careers at Adyen

We are on the lookout for talented engineers and technical people to help us build the infrastructure of global commerce!

Check out developer vacancies

### Developer newsletter

Get updated on new blog posts and other developer news.

Subscribe now

TECH

**Adyen Tech Radar: engineer's autonomy to the next level**

May 31, 2022 · 7 Minutes

TECH

**Fighting PostgreSQL write amplification with HOT updates**

May 10, 2022 · 10 Minutes

TECH

**Revamp of the API credentials screens**

March 24, 2022 · 3 Minutes

See all

## Sign up for the newsletter

| |
|---|
| First Name* |
| Last Name* |

Country*
Select...

Je ne suis pas un robot
reCAPTCHA
Confidentialité - Conditions

Subscribe

By submitting this form, you acknowledge that you have reviewed the terms of our Privacy Statement and consent to the use of data in accordance therewith.

| About | Products | Resources | Platform | Want to stay up to date? |
|---|---|---|---|---|
| Careers | Payments | Documentation | Infrastructure | Email * |
| Press & Media | Risk management | Support | Licenses | I confirm that I have read Adyen's Privacy Policy and I agree to the use of my data in line therewith. |
| Investor Relations | Authentication | Blog | Terms & Conditions | |
| Partners | Issuing | Newsletter | Legal | |
| Contact | Pricing | | | |

Global (English)